Нестандартни задачи

Тук ще намерите нестандартни задачи, подбрани от най-различни източници: събеседвания за работа като програмист, изпити в различни образователни институции, ученически състезания и др. Много от задачите отдавна са се превърнали във фолклор.

Логически и математически задачи

1) Монета в ролята на генератор на случайни числа.

Как с помощта на монета ще генерирате случайни числа в интервала $[0\ ;\ 1]$? Докажете, че генерираните числа са равномерно разпределени.

Решение: Задачата може да се реши по два начина, привидно различни, но равносилни по същество.

Първи начин: Заимстваме идеята на решението от двоичното търсене. Разделяме интервала $[0\ ;\ 1]$ на две равни части, т.е. $[0\ ;\ {}^1/_2]$ и $[{}^1/_2\ ;\ 1]$. Хвърляме монетата: ако се падне ези, избираме лявата половина, т.е. $[0\ ;\ {}^1/_2]$; ако се падне тура, избираме дясната половина, т.е. $[{}^1/_2\ ;\ 1]$. Повтаряме описаната процедура с новия инетрвал, като всеки път делим интервала на две половинки. Така след определен брой хвърляния интервалът ще се смали дотолкова, че ще можем да го считаме за едно число (например левия край).

П р и м е р : Нека се е паднало следното: ези, тура, ези, ези, тура. На тази редица от хвърляния съответства следната редица от интервали: $[0\ ;\ 1]$, $[0\ ;\ {}^1/_2]$, $[{}^1/_4\ ;\ {}^1/_2]$, $[{}^1/_4\ ;\ {}^3/_8]$, $[{}^1/_4\ ;\ {}^5/_{16}]$, $[{}^9/_{32}\ ;\ {}^5/_{16}]$. Взимаме левия край на последния получен интервал, т.е. ${}^9/_{32} = 0{,}28125$. Това е генерираното случайно число. С повече хвърляния на монетата се получава число с повече цифри, т.е. по-голям брой различни числа.

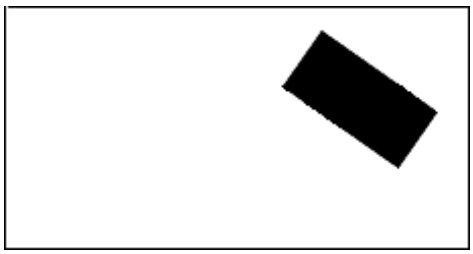
Втори начин: Образуваме двоично число с N цифри след запетаята (преди запетаята стои цифрата 0). Хвърляме монетата N пъти. При всяко хвърляне съответната цифра става 0, ако се падне ези, или 1, ако се падне тура.

П р и м е р : Нека се е паднало следното: ези, тура, ези, ези, тура. На тази редица съответства числото $0{,}01001_{(2)} = {}^9/_{32} = 0{,}28125_{(10)}$.

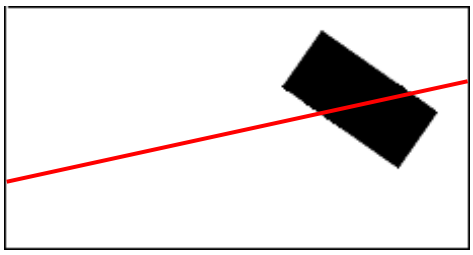
Получените случайни числа са равномерно разпределени, защото всяко от тях може да бъде генерирано с една и съща вероятност: ${}^1/_2{}^N$.

2) Разделяне на двойка правоъгълници.

От правоъгълна торта е изрязано правоъгълно парче. Страните на парчето не са непременно успоредни на страните на тортата. Как с един праволинеен разрез да разделим остатъка от тортата на две равни части?



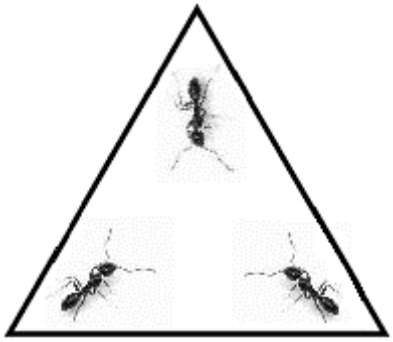
Решение: Права разделя правоъгълник на две равни части тогава и само тогава, когато минава през неговия център. Ето защо правата през центровете на двата правоъгълника (тортата и парчето, изрязано от нея) разделя всеки от тях, а значи и остатъка, на две равни части.



Втори (нестандартен) начин: Да направим хоризонтален разрез през средата на височината на тортата, т.е. да я срежем на две по-тънки торти! Това може да се счита за изобретателно решение с неочакван ход или за опит да избягаме от истинската задача.

3) Три мравки.

В триъгълен съд има три мравки —
по една във всеки ъгъл.
Мравките се придвижват едновременно
и независимо една от друга,
като всяка от тях тръгва към
случайно избран ъгъл от другите два.
Каква е вероятността да има
по една мравка във всеки ъгъл
след преместването?



Решение: За да има по една мравка във всеки ъгъл след прехода, то трябва мравките да се движат в една и съща посока (или всички по часовника, или всички обратно на часовника); общо два благоприятни случая. Всяка мравка избира между два ъгъла, затова всички възможни изходи са $2 . 2 . 2 = 8$. Следователно търсената вероятност е $^{2}/_{8} = {}^{1}/_{4} = 0{,}25 = 25\%$.

4) Двама пазачи.

а) Престъпник е осъден на затвор, но му дават шанс да се отърве от присъдата. Завеждат го в стая, където има две врати и двама пазачи. Едната врата води на свобода, а другата води към затвора. Всеки от двамата пазачи знае коя врата накъде води, но единият пазач винаги лъже, докато другият винаги казва истината. Това е известно на престъпника, но той не знае нито кой пазач лъже, нито коя врата води към свободата (обаче пазачите знаят и двете неща). Осъденият има право на един въпрос от вида "да или не?" към единия от пазачите, след което трябва да избере през коя врата да мине. Какъв въпрос би помогнал на осъдения да излезе на свобода със сигурност?

Решение: Да си мислим, че имаме две устройства, които преобразуват битове. Двете устройства са еднакви на външен вид. Едното устройство запазва подадения сигнал (ако на входа бъде подадена 0, то на изхода се появява 0; а ако на входа бъде подадена 1, то и на изхода се появява 1). Другото устройство обръща сигнала (т.е. превръща 0 в 1 и 1 в 0). Можем ли с тяхна помощ да разберем какъв е бил неизвестен за нас входен сигнал, при условие че имаме право само на едно измерване?

Този проблем е същият като задачата за пазачите. Не знаем кой пазач лъже и не знаем кое устройство обръща сигнала. Можем обаче да сглобим двете устройства: да свържем изхода на едното с входа на другото. Има два начина за сглобяване, защото не знаем кое устройство е първо (те са еднакви на външен вид). Но това няма значение: и в двата случая полученото сложно устройство обръща сигнала: 0 — 0 — 1 и 1 — 1 — 0 в единия вариант; 0 — 1 — 1 и 1 — 0 — 0 в другия вариант. Така с едно измерване узнаваме какъв е бил неизвестният входен сигнал: обратен на изходния.

Аналогично, трябва някак си да "сглобим" пазачите, т.е. единият от тях да бъде питан за отговора на другия. Осъденият може да посочи едната врата и да попита единия пазач: "Какво ще отговори твоят колега на въпроса дали тази врата води към свободата?" Ако запитаният каже "не", значи посочената врата води към свободата; а ако каже "да", значи вратата води към затвора.

б) Ами ако пазачът е един?

Решение: Същата стратегия работи и в този случай: караме пазача да прецени сам собствения си отговор. Ако пазачът лъже, се получава двойно отрицание и в крайна сметка пак излиза истината! Осъденият посочва една врата и задава на пазача следния въпрос:

"Какво ще отговориш, ако те попитам дали тази врата води към свободата?"

От теоретична гледна точка този подход е безупречен. Изглежда, че никой никога не може да излъже, което не е така на практика. Възможно е пазачът да не разбере въпроса правилно: той може например да си помисли, че това е разтегната, учтива формулировка (нещо като форма на любезност) и да реши, че го питат просто "Дали тази врата води към свободата?". Дори да е разбрал въпроса правилно, пазачът може нарочно да даде подвеждащ отговор. Един истински лъжец несъмнено би постъпил точно тъй. Лъжците от логическите задачи обикновено не са толкова изобретателни: те се придържат строго към законите на логиката и затова действията им са автоматични. (Ето защо можахме по-горе да заменим пазачите с устройства за преобразуване на сигнали.) Следователно решението на задачата е правилно от теоретична гледна точка, но може да е неприложимо в действителност.

5) Трудно изчисление.

Ако $A = 1$, $B = 2^A$, $C = 3^B$, . . . , $Z = 26^Y$, то пресметнете стойността на израза $(X–A) \cdot (X–B) \cdot (X–C) \ldots (X–Z)$.

Заб. На пръв поглед изглежда, че задачата не може да се реши на ръка, но в действителност тя е елементарна, трябва само малко наблюдателност. По-трудно е да се реши с груба сила, т.е. на компютър, тъй като се получава препълване: стойностите на някои променливи, напр. Z, са огромни числа.

Решение: Последният множител е $X–Z$, предпоследният множител е $X–Y$, а множителят преди него е $X–X = 0$. Щом един от множителите е нула, то и цялото произведение е нула.

6) Намиране на фалшива монета с едно теглене.

Имаме осем буркана, пълни с монети. Всички монети тежат по 10 г освен една кутия, в която монетите са по 9 г. Разполагаме с електронна везна, която показва теглото. Как с едно теглене да определим буркана с леките монети?

Решение: Взимаме една монета от първия буркан, две от втория и т.н. до осмия буркан, от който взимаме осем монети. Следователно сме взели общо 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 = 36 монети. Ако всички избрани монети тежаха по 10 грама, то везната щеше да покаже тегло 360 грама. Но тъй като всяка фалшива монета е с 1 грам по-лека, то разликата между 360 грама и показаното тегло е равна на броя на взетите фалшиви монети, който от своя страна е равен на номера на буркана с фалшиви монети. Например, ако везната покаже 357 грама, то фалшивите монети са в третия буркан.

7) Разбъркване на карти.

Средно колко разбърквания са необходими за стандартна колода с 52 карти, преди да се получи подреждане, което никога досега не се е срещало в историята на човечеството? (Приемаме, че при разбъркване на картите всички възможни подредби са равновероятни.)

Решение: В момента на земята живеят около 7 милиарда души, а в средата на XX век са били около 3 милиарда души, т.е. населението на планетата се увеличава бързо. През по-голямата част от историята населението е било сравнително малобройно. Общият брой хора, живели някога на света, е по-малък от 100 милиарда. Ако всеки от тях е направил през живота си един милион разбърквания на карти (което е явно завишена оценка), то през цялата история на човечеството са се появили не повече от 100 милиарда по 1 милион = 10^{17} различни подреждания. Този брой изглежда голям, но е нищожен спрямо броя на всички възможни подреждания, който е равен на $52! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot 50 \cdot 51 \cdot 52 \approx 10^{68}$. Вероятността едно случайно подреждане на картите (например следващото) да се е появявало вече в историята на човечеството е нищожна: $10^{17} / 10^{68} = 10^{-51}$. Тоест с почти пълна сигурност още първото разбъркване ще доведе до подреждане на картите, което не се е срещало никога в историята на човечеството.

8) Японска задача.

Тази задача е била дадена като входен тест в японска детска градина за даровити деца.

Имаме съответствие, получено по някакво тайно правило. Примери за съответствието са показани в таблицата вдясно.

Отгатнете тайното правило. По-конкретно, какво съответства на 45678, т.е. 45678 → ?

42	→	1
1337	→	0
669	→	3
1882	→	4
688	→	5
12345	→	1
67890	→	5
123	→	0
456	→	2
789	→	3

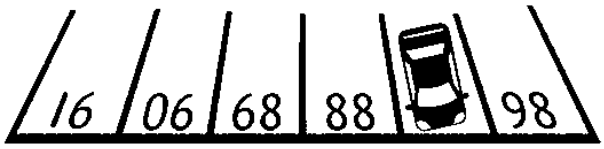
Решение: Трябва да забележим, че числото отдясно показва колко на брой са заградените участъци в цифрите отляво. Във всяка от цифрите 0, 4, 6 и 9 има по точно един заграден участък; в цифрата 8 има два такива участъка; а цифрите 1, 2, 3, 5 и 7 не заграждат нито един. В цифрите на числото 45678 има общо четири заградени участъка, т.е. 45678 → 4.

9) Място за паркиране.

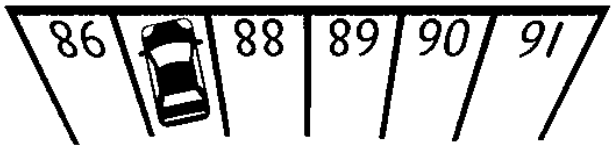
Китайски тест за прием в първи клас:

Кой номер е мястото, на което е паркирала колата?

Отговорете в рамките на 20 секунди.



Решение: На пръв поглед не се забелязва каквато и да е закономерност в номерацията на местата за паркиране: 16 , 06 , 68 , 88 , ? , 98. Задачата изисква досетливост: номерата са написани така, че да бъдат разчитани лесно от шофьорите, които искат да паркират. Като застанем обратно, т.е. на мястото на шофьора, се получава следната картинка:



И така, мястото е номер 87 (или *L*8 — както се вижда от първоначалната позиция).

10) Самоописваща се редица.

Кой е следващият член на редицата
1 , 11 , 21 , 1211 , 111221 , 312211 , 13112221 , 1113213211 , ?

Решение: Всеки член от втория нататък описва предходния член, като за всяка група от последователни еднакви цифри се изписва броят им и стойността им.

П р и м е р : шестият член описва петия:
три единици, две двойки, една единица, т.е. 312211.

Вече е ясно, че следващият член на редицата описва последния даден: три единици, една тройка, една двойка, една единица, една тройка, една двойка, две единици, т.е. 31131211131221.

11) Три електрически крушки.

Намирате се в стая с три ел. ключа, свързани с три ел. крушки в друга стая, която не можете да видите. Всеки ключ е свързан с точно една крушка и обратно. Как ще разберете кой ключ с коя крушка е свързан, ако веднъж отишли в стаята с крушките, не можете да се върнете в стаята с ключовете?

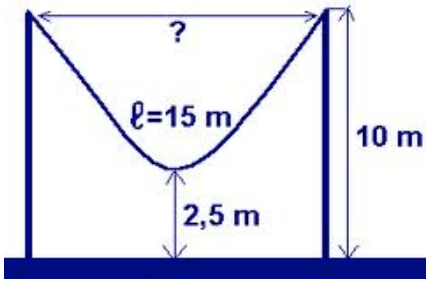
Решение: Задачата е от времето на ел. крушките с нажежаема жичка. Те не само светят, но и се нагряват от електрическия ток. Затова включваме първия ел. ключ за около пет минути, след това го изключваме, веднага включваме втория и влизаме в стаята с крушките. Докосваме двете несветещи крушки. Едната от тях ще е топла (тъй като е светила досега); тя е свързана с първия ключ. Студената несветеща крушка е свързана с третия ключ. Светещата крушка е свързана с втория ключ.

Това решение като че ли не работи добре с халогенни крушки. 

Заб. Без някаква хитрост от този род няма как да решим задачата: очевидно разделянето на крушките по признака “свети – не свети” носи само един бит информация, който би бил достатъчен за задача с две крушки и два ключа. Но при три крушки и три ключа вече имаме нужда от втори бит информация, например “топло – студено”.

12) Ще стигне ли въжето?

Въже с дължина 15 метра е окачено в горните краища на два стълба, всеки от които с височна 10 метра. Най-ниската точка на въжето е на височина 2,50 метра над земята. Въз основа на тези данни намерете разстоянието между стълбовете.



Решение: По принцип формата на въжето, а оттам и най-ниската му точка се определят посредством методите на висшата математика. Числените данни в тази задача обаче са подбрани така, че тя може да се реши с елементарни средства. Като извадим 2,50 метра от 10 метра, виждаме, че отначало въжето се спуска със 7,50 м, докато стигне до най-ниската си точка, а после се издига пак със 7,50 метра. Следователно 7,50 + 7,50 = 15 метра са нужни за спускане и издигане на въжето, с което цялата му дължина е изразходвана за движение във вертикална посока. Не остава никаква дължина за хоризонтално движение. Следователно двата стълба са разположени непосредствено един до друг, т.е. разстоянието между тях е нула.

Заб. Дължината на въжето (15 метра) е избрана доста удачно. Ако въжето беше по-късо, то задачата нямаше да има решение: описаната ситуация щеше да бъде невъзможна (дължината на въжето нямаше да стигне дори само за спускането и издигането). Ако пък въжето беше по-дълго от 15 метра, то задачата щеше да има решение, но то нямаше да бъде толкова елементарно; щеше да се наложи да използваме формули от висшата математика.

13) Букет.

В букет цветя
всички освен две са рози,
всички освен две са маргаритки,
всички освен две са лалета.

Колко цветя има в букета?

Решение:

От условието следва, че розите, маргаритките и лалетата са равен брой (и този брой е с две по-малък от броя на всички цветя в букета). Ако махнем например розите, ще останат две цветя (защото всички освен две са рози). Тези две цветя са лалета и маргаритки, а щом са поравно, значи в букета има едно лале и една маргаритка. Следователно и розата е само една.

И така, букетът се състои от три цветя — роза, лале и маргаритка.

14) Коя е отровната гозба?

За сватбата на краля главният готвач приготви хиляда гозби. Току-що кралските съгледвачи съобщиха на негово величество, че някой е сипал отрова в една от гозбите. Нещо повече, те знаят, че действието на отровата започва да се забелязва един час след поемане на храната, а до сватбения пир остава само малко повече от час — тоест има време само за един опит!

За щастие, в тъмницата се намират много затворници, върху които може да се изпробва действието на смеси от една или повече гозби. Затворниците не са получавали свястна храна от седмици, така че ще изядат всякакъв буламач — дори смес от всичките хиляда ястия. По това, кои затворници покажат признаци на отравяне до един час, кралят ще може да определи коя гозба е отровна. Сега той се чуди колко най-малко затворници са нужни, за да може със сигурност да определи отровната гозба.

Примерно решение е на всеки от общо хиляда затворници да се даде по точно една от гозбите (на всеки затворник — различна гозба), но има решение с далеч по-малък брой. Тоест иска се стратегия с минимален брой използвани затворници, а не с минимален брой умрели.

Решение:

Отговорът е десет затворници: $\lceil \log_2 1000 \rceil = 10$, понеже $2^{10} = 1024,\ 2^9 = 512.$

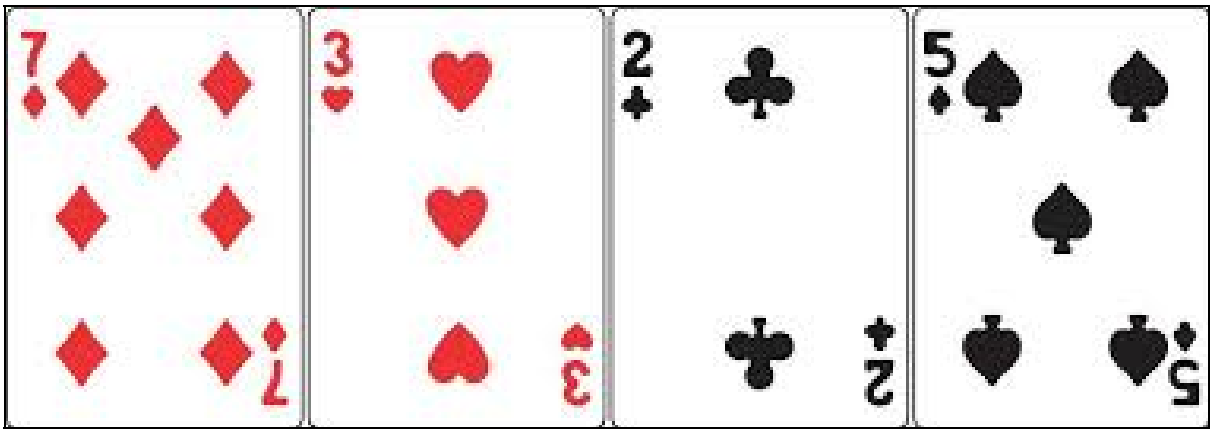
Нагледно можем да представим схемата на опита с помощта на двоичния запис на числата. Използваме т.нар. битови маски. Номерираме всичките хиляда ястия с числата от 0 до 999 (вместо с числата от 1 до 1000). Номерираме и избраните затворници с числата от 0 до 9 (вместо от 1 до 10).

	З а т в о р н и ц и									
Я с т и я	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
ястие № 0	0	0	0	0	0	0	0	0	0	0
ястие № 1	0	0	0	0	0	0	0	0	0	1
ястие № 2	0	0	0	0	0	0	0	0	1	0
ястие № 3	0	0	0	0	0	0	0	0	1	1
ястие № 4	0	0	0	0	0	0	0	1	0	0
. . . . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
ястие № 998	1	1	1	1	1	0	0	1	1	0
ястие № 999	1	1	1	1	1	0	0	1	1	1

Всеки затворник получава смес от онези гозби, чиито номера съдържат единица в колонката на затворника. Например затворник № 0 получава смес от гозбите с нечетен номер. След час събираме онези степени на двойката, за показатели на които служат номерата на затворниците, проявили признаци на отравяне. Например, ако се отровят затворниците № 1, № 2, № 5, № 6, № 7, № 8 и № 9, то отровното ястие има номер $1111100110_{(2)} = 2^1 + 2^2 + 2^5 + 2^6 + 2^7 + 2^8 + 2^9 = 998_{(10)}$. Ако не се отрови никой затворник, то отровно е първото ястие (т.е. № 0).

15) Петата карта.

Двама фокусници изнасят следното представление. От стандартна колода с 52 карти човек от публиката избира пет и ги дава на единия фокусник. Илюзионистът разглежда петте карти, избира една от тях, а останалите четири съобщава на колегата си в определен ред. По това вторият фокусник познава коя е петата (премълчаната) карта!



Каква стратегия биха могли да имат двамата илюзионисти, за да определят еднозначно петата карта по дадените четири? Четирите карти се съобщават в определен ред от единия фокусник на другия, но не може да се подсказва по никакъв друг начин (чрез интонация, жестове и т.н.). Тоест търси се подход, основан единствено на логика, а не на някакъв вид измама.

Решение:

Двамата илюзионисти трябва да се договорят (още преди началото на фокуса) за някаква наредба на картите: коя е първа, коя — втора, коя — трета и т.н. Например по стойност: A < 2 < 3 < 4 < 5 < 6 < 7 < 8 < 9 < 10 < J < Q < K; а тези, които са равни по стойност, се сравняват по боя: ♣ < ♦ < ♥ < ♠. Тоест **A ♣** < **A ♦** < **A ♥** < **A ♠** < **2 ♣** < **2 ♦** < **2 ♥** < **2 ♠** < . . . < **K ♥** < **K ♠**. Тогава всеки четири карти, дадени в определен ред, образуват пермутация на числата 1, 2, 3, 4. Така например, четворката, показана на картинката (т.е. **7 ♦** , **3 ♥** , **2 ♣** , **5 ♠**) съответства на пермутацията (4, 2, 1, 3), защото седмица каро е най-голямата, тройка купа е втора по големина, двойка спатия е най-малка (т.е. първа), а петица пика е трета поред: **2 ♣** < **3 ♥** < **5 ♠** < **7 ♦**. От друга страна, всяка пермутация си има номер. Например можем да подредим пермутациите като числа: 1) 1234; 2) 1243; 3) 1324; 4) 1342; и т.н. до 24) 4321, защото всички пермутации на четири елемента са 4**!** = 24. При това положение пермутацията 4213 ще бъде № 21, т.е. четирите карти в реда, в който са дадени, посочват двайсет и първата карта от останалите. (В действителност конкретната наредба няма значение. Важно е само да бъде договорена предварително. Разбира се, добре е тя да бъде максимално лесна за помнене.) Проблемът е, че остават 52 – 4 = 48 карти, а пермутациите са само 24 на брой, т.е. недостатъчно за различаване на оставащите 48 карти.

Понеже 48 **:** 24 = 2, то нужен е още един бит информация, т.е. съобщение от вида “да – не”. Този един бит очевидно няма как да бъде предаден чрез съобщените карти, значи трябва да бъде предаден предварително.

Тук трябва да бъде отчетен изборът на първия фокусник. Той не бива да избира петата карта произволно, а трябва да я избере така, че да сведе неопределеността до 24 карти (вместо до 48). Тъкмо тази информация за начина, по който първият илюзионист избира коя от петте карти да премълчи, е допълнителният бит, известен предварително на колегата му, тъй като двамата са договорили схемата още преди представлението. Да видим как това може да стане на практика.

Най-проста е следната схема: щом първият фокусник получава пет карти, а боите са само четири, то поне една боя се повтаря (т.е. има поне две карти от съответната боя). Ако се повтарят няколко бои, то първият илюзионист избира една от тях произволно. Ако от избраната боя има повече от две карти (тоест три, четири или всичките пет), то фокусникът избира две от тези карти произволно. Едната от тях съобщава, а другата премълчава. Необходимо е двамата фокусници да се договорят коя карта ще бъде премълчавана.

Да смятаме, че картите от всяка боя са наредени в кръг, т.е. след попа идва асото. Първият фокусник избира да съобщи онази от двете карти, за която разстоянието от нея до другата карта е по-малко. Например, ако едната карта е шестица, а другата — десетка (от същата боя), то първият илюзионист ще съобщи шестицата, а ще премълчи десетката: разстоянието от шестицата до десетката е равно на 4 единици (седмица, осмица, деветка, десетка), а разстоянието от десетката до шестицата е 9 единици (вале, дама, поп, асо, двойка, тройка, четворка, петица, шестица). Естествено, сборът от двете разстояния е 13 (винаги, а не само в този пример), защото всички карти от една боя са тринадесет на брой. Оттук следва, че по-малкото разстояние не надхвърля 6.

И така, начинът на първия фокусник да избере едната от двете карти е тъкмо липсващият бит информация: след като колегата му знае как става този избор, неопределеността намалява наполовина. За целта е достатъчно онази от двете карти от една боя, която е избрана за съобщаване (в примера по-горе това беше петицата), да бъде съобщена първа. Останалите три карти кодират разстоянието от нея до премълчаната карта (броено нагоре). Понеже това разстояние е най-много 6, а три карти може да бъдат подредени по 3**!** = 6 различни начина, то вторият фокусник разполага с всички нужни сведения, за да познае петата (премълчаната) карта.

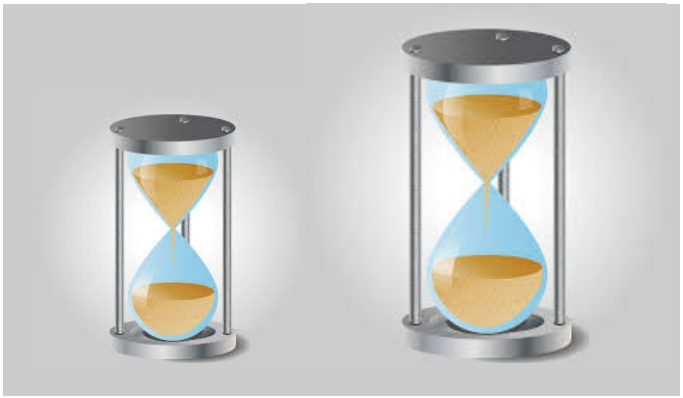
Пермутациите на три елемента се подреждат (т.е. номерират се) като трицифрени числа, т.е. 1) 123; 2) 132; 3) 213; 4) 231; 5) 312; 6) 321.

П р и м е р : Зрителят връчва на единия фокусник следните пет карти: **К ♥** , **2 ♠** , **7 ♠** , **7 ♦** , **А ♦**. Фокусникът избира произволно едната от двете повтарящи се бои, например пиката. Понеже разстоянието от двойката до седмицата е 5 единици (тройка, четворка, петица, шестица, седмица), а от седмицата до двойката е 8 единици (осмица, деветка, десетка, вале, дама, поп, асо, двойка), то илюзионистът избира да премълчи седмицата пика, а да съобщи двойката пика. При това той съобщава първо двойката пика, а след нея — останалите три карти (асо каро, седмица каро и поп купа) в точно определен ред, който трябва да кодира разстоянието от 5 единици. Тъй като петата пермутация е (3, 1, 2), то от трите оставащи карти първо се съобщава третата, най-голямата (т.е. поп купа), после се съобщава първата, най-малката (т.е. асо каро), най-накрая — втората, средната по големина (т.е. седмица каро). Окончателно, илюзионистът съобщава на колегата си четири карти в следния ред: **2 ♠** , **К ♥** , **А ♦** , **7 ♦** . Вторият фокусник, получавайки редицата, я разшифрова така: боята на първата карта му показва боята на петата карта, т.е. петата карта е пика. Другите три карти (поп купа, асо каро, седмица каро) образуват пермутацията (3, 1, 2), защото попът е най-голямата, а асото — най-малката от трите карти. Тази пермутация е № 5, което задава разстоянието до неизвестната карта. Затова вторият фокусник тръгва от двойка (първата карта) и брои нагоре 5 единици: тройка, четворка, петица, шестица, седмица. Така стига до **7 ♠** .

П р и м е р : Ако вторият фокусник получи ръката от началото на задачата (**7 ♦** , **3 ♥** , **2 ♣** , **5 ♠**), по боята на първата карта ще разбере, че петата карта е каро. Останалите три карти образуват пермутацията (2, 1, 3), защото двойката е най-малка, а петицата — най-голяма. Тази пермутация е № 3, т.е. петата карта е на 3 единици нагоре от първата карта (седмицата): осмица, деветка, десетка. Окончателно, петата карта е **10 ♦** .

16) Пясъчни часовници.

С два пясъчни часовника трябва да отмерите точно 9 минути, и то считано от този миг, при условие че пясъкът в единия часовник изтича точно за 4 минути, а в другия — за 7 минути.



Решение: Пускаме двата часовника. След 4 мин. обръщаме малкия. След още 3 мин. (общо 7 мин.) големият часовник изтича, а на малкия му остава 1 мин. Обръщаме големия часовник. След 1 мин. (осмата) малкият часовник изтича. Сега в долната част на големия часовник се е събрал пясъкът, изтекъл през последната минута (осмата). Обръщаме големия часовник и след като този пясък изтече, ще е изминала още една минута (деветата).

17) Колко карти са с лицето нагоре?

Ели и Станчо играят на следната игра. В началото Станчо взима колода карти (52 на брой), като всички карти са с лицето надолу. Ели изгася лампите. (Не, това е логическа задача и не продължава така, както си мислите.) В тъмното Станчо обръща някои от картите с лицето нагоре, като накрая връща обърнатите карти в купчината. След тази процедура някои карти гледат нагоре, а останалите — надолу. Станчо може да избере колко и кои карти да обърне, а също и къде в колодата да ги постави. Момичето не вижда нищо от този процес.

След това Станчо подава колодата карти на Ели и ѝ казва колко на брой карти е обърнал. Тя трябва да раздели колодата на две купчинки, като също има право да обръща картите (колкото и които пожелае). Тъй като е тъмно, тя не знае кои от картите са обърнати нагоре; знае само колко на брой са те.

Накрая лампите биват светнати и ако в двете купчинки равен брой карти гледат нагоре, то Ели печели. В противен случай печели Станчо. Намерете печеливша стратегия за Ели.

Решение: Нека Станчо е обърнал K карти с лицето нагоре (това е числото, което Станчо казва на Ели). Тогава K карти гледат нагоре, а другите $52-K$ гледат надолу. Стратегията на Ели е да раздели колодата на две купчинки, като отброи K карти в едната купчинка (а в другата остават $52-K$ карти). Ако във втората купчинка X карти са с лицето нагоре, то в първата купчинка с лицето нагоре са останалите $K-X$ карти, обърнати от Станчо. Ели не знае числото X, но то не ѝ е нужно. Достатъчно е да обърне отброените K карти. Тогава в първата купчинка онези X карти, които досега са били с лицето нагоре, вече ще гледат надолу; а останалите $K-X$ карти, които досега са гледали надолу, вече ще гледат нагоре. Така и в двете купчинки ще има по $K-X$ карти с лицето нагоре, т.е. равен брой такива карти, и Ели ще спечели.

П р и м е р : Нека Станчо е обърнал 10 карти с лицето нагоре. Той казва това число на Ели, след което тя изтегля 10 карти от произволно място. Нека например 3 от тях са с лицето нагоре (Ели не знае този брой). Следователно сред останалите 42 карти има 10 – 3 = 7 обърнати с лицето нагоре (този брой също е неизвестен на Ели). Но тя обръща всичките 10 карти, така че сега 3 от тях са с лицето надолу, а останалите 7 — с лицето нагоре. Така във всяка от двете купчинки (едната с 10, другата с 42 карти) има по равен брой обърнати с лицето нагоре (по 7).

Заб. Числото 52 не е съществено: предложеното решение остава в сила за колода с произволен брой карти.

18) Цветни шапки.

Неколцина осъдени на смърт получили последен шанс да се спасят. Казали им, че на другия ден ще бъдат подложени на изпитание. На главата на всеки от тях ще бъде сложена цветна шапка — червена, зелена или синя. Всеки осъден ще вижда цветовете на шапките на всички други осъдени, но не и на своята. След това осъдените ще бъдат питани един по един какъв е цветът на собствената им шапка, като ще имат право да отговарят само с една дума: "червен", "зелен" или "син". Които познаят цвета на шапката си, ще бъдат помилвани, а които сбъркат, ще бъдат екзекутирани.

Осъдените имали на разположение цялата нощ, за да измислят стратегия, чрез която да се спасят възможно най-много от тях. Накрая измислили такава стратегия, която гарантира спасение на всички без първия запитан, комуто стратегията не гарантира успех, но и не намалява вероятността за спасение (която е $^1/_3$ при случаен отговор). Каква е била стратегията на осъдените?

Решение: Първо, каквато и стратегия да изберат осъдените, тя не може нито да увеличи, нито да намали шансовете на първия осъден: за него вероятността за правилен отговор е винаги $^1/_3$, защото единственото, което той знае, е цветът на шапките на останалите затворници, а това няма връзка с цвета на неговата собствена шапка.

Обаче отговорът на първия запитан може много да помогне на другите осъдени. Той може да им съобщи сбора от цветовете на техните шапки по модул 3. Нека например всяка червена шапка има стойност 0, всяка зелена да има стойност 1, а всяка синя — стойност 2 (това са тъкмо остатъците при деление на 3). Първият осъден събира числата (цветовете) на всички други шапки (т.е. всички, които вижда), дели сбора на 3 и съобщава получения остатък (0, 1 или 2) чрез съответния цвят. Сега всеки от другите осъдени знае сбора на цветовете на всички шапки (без шапката на първия), а вижда всички шапки без своята, така че лесно може да намери неизвестното събираемо.

П р и м е р : Нека осъдените са получили шапки със следните цветове:

1 2 1 1 0 2

Първият осъден събира числата (цветовете) на всички останали шапки, т.е. 1 + 2 + 1 + 1 + 0 + 2 = 7, което дава остатък 1 при деление на 3. Затова първият осъден казва "зелен" (т.е. 1). В конкретния случай той не познава цвета на шапката си и бива екзекутиран.

Вторият осъден събира цветовете, които вижда (без шапката на първия), т.е. 2 + 1 + 1 + 0 + 2 = 6. Понеже 6 при деление на 3 дава остатък 0, а целият сбор е 1 (това вторият осъден го е научил вече от отговора на първия), то шапката на втория осъден има цвят, който допълва 0 до 1, т.е. 1 (0 + 1 = 1), значи зелен. Затова вторият осъден казва "зелен" и се спасява.

Третият осъден събира цветовете, които вижда (без шапката на първия), т.е. 1 + 1 + 1 + 0 + 2 = 5. Понеже 5 при деление на 3 дава остатък 2, а целият сбор е 1 (това третият осъден го е научил вече от отговора на първия), то шапката на третия осъден има цвят, който допълва 2 до 1, т.е. 2 (2 + 2 = 4, което дава остатък 1 при деление на 3), значи син. Затова третият осъден казва "син" и се спасява.

Шестият осъден събира цветовете, които вижда (без шапката на първия), т.е. 1 + 2 + 1 + 1 + 2 = 7. Понеже 7 при деление на 3 дава остатък 1, а целият сбор е 1 (това шестият осъден го е научил вече от отговора на първия), то шапката на шестия осъден има цвят, който допълва 1 до 1, т.е. 0 (1 + 0 = 1), значи червен. Затова шестият осъден казва "червен" и се спасява.

19) Винарска изба (първи вариант).

Брат и сестра наследяват винарска изба с ценни вина. В нея има четен брой бутилки, наредени в редица. Братът и сестрата се споразумяват да си поделят бутилките по следния начин: започвайки със сестрата (все пак е дама), те ще се редуват да взимат по една бутилка от редицата, като всеки от тях, когато е на ред, ще може да избира между най-лявата и най-дясната от останалите бутилки.

Съществува ли стратегия, която да гарантира на сестрата не по-малко от половината от общата стойност на бутилките вино?

Решение: Сестрата може, ако иска, да вземе всички бутилки, стоящи на четна позиция: за целта тя трябва първия път да вземе най-дясната бутилка, а след това да повтаря действията на брат си (т.е. да взима от същия край, от който е взел бутилка той последния път). Аналогично, сестрата може да вземе всички бутилки, стоящи на нечетна позиция: първия път взима най-лявата бутилка, после повтаря действията на брат си.

И така, бутилките се разделят на две множества — такива, които стоят на четно място, и такива, които стоят на нечетно място. Видяхме, че само от сестрата зависи кое от двете множества ще остане за нея. Избирайки множеството с по-голяма сборна цена, сестрата със сигурност ще получи бутилки на обща стойност, не по-малка от половината от стойността на всички бутилки във винарската изба.

20) Омагьосани предмети.

Редица съдържа N предмета, наглед еднакви, но единият е истински, а другите са негови образи (миражи или холограми). Искаме да хванем истинския предмет. Имаме право да докосваме предметите един по един в произволен ред. Ако докоснем истинския предмет, то миражите ще изчезнат и ние ще сме постигнали целта си. Ако пипнем мираж, то след дръпването на ръката истинският предмет разменя мястото си с миража отляво или отдясно. Търсим стратегия, чрез която със сигурност да уловим истинския предмет, независимо как се движи и независимо къде се намира отначало.

Решение: Да номерираме позициите от 1 (най-лявата) до N (най-дясната). Една интуитивна стратегия е първо да докоснем № 1, после № 2 и т.н., докато стигнем до № N. Тя работи само ако отначало истинският предмет се намира на нечетна позиция.

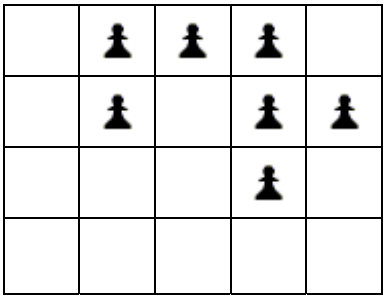
Но ако е бил на четна позиция, например на № 2, то след като докоснем холограмата на № 1, истинският предмет може да се премести на № 1, а след това винаги да отива в позицията, която току-що сме докоснали. В този случай няма да уловим истинския предмет чрез описаната стратегия.

Забелязваме, че на всяка стъпка истинският предмет сменя четността на позицията си. Установихме, че ако отначало той стои на нечетна позиция, то чрез горната стратегия ще успеем да го хванем. Ако сме стигнали до № N и не сме хванали истинския предмет, то следва, че в началото е бил на четна позиция и някъде се е разминал с нас. Следва още, че всеки път досега сме докосвали позиция с четност, различна от четността на позицията на истинския предмет. Затова трябва еднократно да запазим четността си (а през това време истинският предмет ще промени своята четност). Ето защо докосваме позиция № N още веднъж (истинският предмет може току-що да е отишъл там), след което продължаваме с № N–1, N–2, N–3 и т.н. до № 1. Сега вече всеки път докосваме позиция със същата четност като четността на позицията на истинския предмет, така че няма как да се разминем с него, а от друга страна, все повече стесняваме областта, където той може да се намира, докато най-сетне го хванем.

И така, възможна стратегия е да докосваме предметите в следния ред: № 1 , № 2 , ... , № N–1 , № N , № N , № N–1 , ... , № 2. Не е нужно да стигаме до № 1, тъй като в най-лошия случай ще уловим истинския предмет на позиция № 2.

21) Скачащи пешки.

В някои от клетките на правоъгълна дъска с m реда и n колони има пешки. Във всяка клетка има най-много по една пешка. Двама играчи се редуват да местят пешките. На всеки ход може да бъде преместена само една пешка,



и то само наляво или надолу. Пешката може да бъде преместена в избраната посока или с една клетка, или с три клетки. Пешката не може да напуска дъската и не може да стъпва върху клетка, заета от друга пешка. Преместване с три клетки е възможно само ако двете прескочени клетки са заети от други пешки (прескачане). Когато пешка попадне в долния ляв ъгъл, тя изчезва (сваля се от дъската). Печели играчът, който направи последния ход, тоест играчът, който опразни дъската. По дадена начална позиция определете кой от играчите има печеливша стратегия и каква е тя.

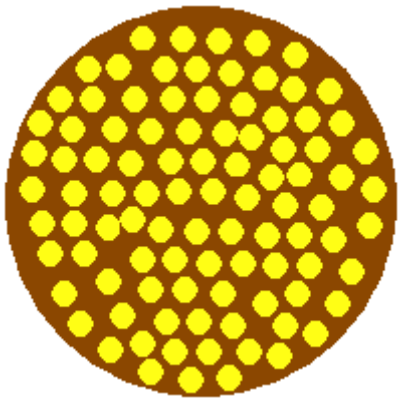
П р и м е р : На дадената фигура са възможни следните движения:
— пешката на първи ред, втори стълб — само с една клетка наляво;
— пешката на първи ред, трети стълб — само с една клетка надолу;
— пешката на първи ред, четвърти стълб — с три клетки наляво или надолу;
— пешката на втори ред, втори стълб — с една клетка наляво или надолу;
— пешката на втори ред, четвърти стълб — само с една клетка наляво;
— пешката на втори ред, пети стълб — само с една клетка надолу;
— пешката на трети ред, четвърти стълб — с една клетка наляво или надолу;

Решение: Да номерираме редовете отдолу нагоре с числата 0, 1, 2, ... , $m - 1$, а стълбовете — отляво надясно с числата 0, 1, 2, ... , $n - 1$. Така i-тата пешка има координати (r_i, c_i), $i = 1, 2, 3, \ldots, k$, където k е броят на пешките. Всеки ход превръща позицията от губеща за текущия играч в печеливша за другия играч и обратно — от печеливша за текущия играч в губеща за другия играч, ако текущият играч избере правилен ход. От друга страна, да разгледаме сумата $S = \sum_{i=1}^{k} r_i + \sum_{i=1}^{k} c_i$. На всеки ход тя също се променя: намалява или с 1, или с 3. Следователно на всеки ход сумата S променя своята четност. Следователно четността на сумата S може да служи за признак дали позицията е печеливша. По-точно, ако дъската е празна, то тази позиция е губеща за текущия играч (противникът му току-що е извадил последната пешка), а $S = 0$, т.е. четна сума съответства на губеща позиция, а нечетна сума — на печеливша позиция. Ясно е, че ако позицията е губеща, то печеливша стратегия няма. Ако пък позицията е печеливша, то играчът трябва да играе така, че да промени четността на S. Обаче това се постига с произволен ход. Значи няма значение какъв ход ще направи играчът.

22) Игра с монети върху кръгла маса.

Върху кръгла маса с радиус 10 см двама състезатели играят следната игра. Редувайки се, на всеки ход слагат по една монета с радиус 1 см на масата. Монетите са много на брой и всичките са еднакви. Важат следните правила:

1) Монетите не бива да се застъпват (може да се допират).
2) Монетите трябва да лежат на масата изцяло (не бива да стърчат навън от масата).
3) Губи играчът, който не може да постави монета.



Кой от играчите има печеливша стратегия и каква е тя?

Решение:

Първият играч има печеливша стратегия: на първия си ход слага монета в центъра на масата, после играе симетрично на ходовете на втория играч. (Размерите нямат значение, стига монетите да са по-малки от масата.)

23) Смяна на караула.

Дванадесет войници трябвало да дежурят общо 12 часа — всеки по един час. Обаче преди началото на дванадесетчасовата смяна един от войниците бил пратен на друго място и останалите единадесет трябвало да поемат и неговото дежурство. Войниците нямат друг уред за отчитане на времето освен часовник с две стрелки — часова и минутна. Как да си разпределят дежурството?

Решение:

Въпросът всъщност е как 12-часова смяна да бъде разделена поравно между 11 войници. Очевидно всеки трябва да дежури по $^{12}/_{11}$ часа = $1\,^{1}/_{11}$ часа. Проблемът е как да отмерим $^{1}/_{11}$ от часа: един час съдържа 60 минути, а пък 60 не се дели на 11.

Трудността изчезва, след като забележим, че часовата и минутната стрелка съвпадат точно през $^{12}/_{11}$ часа. Причината е, че те се движат в една посока и периодично се настигат, при което за 12 часа минутната стрелка прави дванадесет обиколки, а часовата — само една. Тоест за 12 часа минутната стрелка настига часовата 11 пъти — разликата в броя на обиколките. Понеже стрелките се движат равномерно, то времето между две последователни настигания е точно $^{12}/_{11}$ часа.

И така, решението е отначало войниците да нагласят часовника на 12:00, а после да се сменят при всяко съвпадане на стрелките.

24) Пирати делят съкровище.

Петима пирати A, B, C, D и E току-що са открили сандък със 100 жълтици. Сега те се чудят как да си поделят съкровището.

Пиратите имат строга подредба по старшинство: A е по-старши от B, който е по-старши от C, който е по-старши от D, който пък е по-старши от E.

В света на пиратите важат следните правила за разпределяне на намерено имане. Първо най-старшият пират предлага разпределение на монетите. Пиратите (включително предложилият) гласуват за или против предложеното разпределение. Ако поне половината пирати гласуват "за", то бива прието. В противен случай предложението се отхвърля, а пиратът, който го е направил, бива изхвърлен зад борда, след което се прави ново предложение (от пирата, който следва по старшинство) и ново гласуване. Този процес продължава, докато някое предложение бъде прието. Пиратите взимат своите решения въз основа на следните три мотива:

1. Първо (най-важно): всеки пират иска да оцелее.
2. Второ: стига да оцелее, всеки пират гледа да спечели повече жълтици.
3. Трето: ако това, дали пиратът ще гласува "за" или "против", не влияе на оцеляването и печалбата му, то той гласува "против".

Пиратите не си вярват (има защо!) и не правят обещания и договорки помежду си, но всеки знае, че останалите се ръководят от същите мотиви. Какво ще бъде крайното разпределение на съкровището, ако пиратите действат възможно най-рационално?

Решение: Интересното при тази задача е, че отговорът е много далеч от това, което подсказва интуицията. Човек очаква, че пиратът A ще даде голяма част от монетите (дори всичките) на другите пирати от страх да не го изхвърлят. Но се оказва, че оптималната стратегия на A е съвсем различна.

Да разгледаме задачата отзад напред. Какво се случва, ако са останали само пиратите D и E? Тогава D ще предложи 100 монети за себе си и 0 за E, предложението ще бъде прието с гласове 1 на 1. Получава се следното разпределение на монетите: { D: 100, E: 0 }.

Сега да видим какво става, ако има трима останали пирати — C, D и E. Пиратът C знае, че ако бъде изхвърлен, D ще предложи на E нула монети. Следователно C може да предложи 1 монета на E и да запази останалите 99 за себе си. E ще е доволен, защото ще получи 1 монета вместо 0 (както би било, ако гласува против C), следователно гласува за предложението на C и се получава мнозинство: 2 гласа срещу 1. Така разпределението в този случай е $\{$ C: 99, D: 0, E: 1 $\}$.

С четирима пирати (B, C, D и E) нещата не са особено по-различни. Пиратът B знае какво ще се случи, ако го изхвърлят зад борда, и основава стратегията си на това. Трябва му още един човек, който да гласува за него, за да има мнозинство, следователно може да подкупи D с една монета, на което D ще е доволен. Така разпределението ще бъде $\{$ B: 99, C: 0, D: 1, E: 0 $\}$.

Тъй като логиката на A е безпогрешна, той ще знае тези неща и ще предложи разпределението $\{$ A: 98, B: 0, C: 1, D: 0, E: 1 $\}$, с което ще спечели гласовете на C и E. Това е и отговорът на задачата.

25) Бързо преминете моста!

Четирима души A, B, C и D трябва да преминат по мост, чиято ширина позволява едновременното придвижване само на двама души. Четиримата имат общо само един фенер, а мостът е опасен и не може да бъде преминат в тъмното. За съжаление, батерията на фенера е на привършване и фенерът ще изгасне след 17 минути. Знае се, че A може да прекоси моста за 1 минута, B — за 2 минути, C — за 5 минути, D — за 10 минути. Когато двама души вървят заедно, те се движат със скоростта на по-бавния. Как четиримата ще преминат моста, преди да свърши батерията на фенера?

Решение:

1) A и B преминават моста за 2 минути.
2) A се връща за 1 минута и оставя фенера на C и D.
3) C и D преминават моста за 10 минути.
4) B взима фенера и се връща при A за 2 минути.
5) A и B преминават моста за 2 минути.

Така четиримата изразходват общо 2 + 1 + 10 + 2 + 2 = 17 минути.

26) Затворници и кутии с имена.

Сто затворници (именувани S_1, S_2, ... , S_{100}, като $S_i \neq S_j$ при $i \neq j$) очакват да ги екзекутират следващата сутрин. Тъмничарите казват на затворниците, че утре ще им бъде даден последен шанс за спасение.

В една стая ще бъдат наредени в редица по случаен начин 100 кутии, всяка от които съдържа име на затворник (и всяко име се среща точно по веднъж). Един по един затворниците ще бъдат извеждани от общата килия и пускани в стаята, където всеки затворник ще може да отвори не повече от 50 кутии. Целта на всеки затворник ще е да открие името си сред 50-те отворени кутии. След излизането на всеки затворник от стаята с кутиите те ще бъдат затваряни и оставяни така, както са били преди влизането му.

Затворниците ще бъдат пуснати на свобода само ако всички намерят имената си в кутиите. Ако дори един не успее, всички ще бъдат екзекутирани.

Затворниците имат нощта да измислят стратегия, с която да отварят кутиите, но след като веднъж бъдат изведени от килията, няма да могат да общуват.

Има ли стратегия, даваща голяма вероятност за спасение на затворниците?

Решение: Най-простата стратегия е всеки да отваря кутии по случаен начин. Вероятността да открие името си е 50 на 100, т.е. ½, което не изглежда зле. Да, но трябва всички затворници да намерят името си, за да не бъдат екзекутирани. Следователно вероятността за успех е едва $(½)^{100} \approx 10^{-30}$, което е много, много малка вероятност.

Има обаче по-добра стратегия. Нека за простота заменим всяко име с номер: т.е. заменяме S_1 с 1, S_2 с 2, . . . , S_{100} със 100. Дадено е, че имената са различни, следователно няма проблем със замяната. Така можем да считаме, че кутиите съдържат числата от 1 до 100 включително, разбъркани в произволен ред, т.е. случайна пермутация.

Нека затворникът с номер K отваря първо K-тата поредна кутия. Ако вътре стои числото K, т.е. името на затворника, то всичко е наред: успял е да намери името си. А ако в кутията стои число, различно от K, например L ? Тогава затворникът при втория си опит отваря L-тата поредна кутия и т.н., докато открие името си или докато изчерпи позволените 50 опита.

Всяка пермутация се разбива на съвкупност от непресичащи се цикли. При втората стратегия затворникът с номер K ще намери името си само ако цикълът, съдържащ K, е с дължина, по-малка или равна на 50. Следователно затворниците ще се спасят, ако и само ако пермутацията не съдържа цикъл с дължина, по-голяма от 50.

Да видим каква е вероятността да се случи това. Ако има цикъл с дължина, по-голяма от 50, то той е единствен, защото 50 е половината на 100 (броя на всички затворниици).

Броят на пермутациите, съдържащи цикъл с дължина n, се пресмята така: $C_{100}^{n} = \dfrac{100!}{n!\,(100-n)!}$ е броят на начините, по които можем да изберем кои n елемента от всичките 100 участват в цикъла. По-нататък, самите тези n елемента могат да бъдат разместени в цикъла по $(n-1)!$ начина, а останалите $(100-n)$ елемента се разместват (в цикъл или не) по $(100-n)!$ начина. Така броят на всички начини е равен на

$$(n-1)!\,(100-n)!\,C_{100}^{n} = \frac{\cancel{(n-1)!}\;100!\;\cancel{(100-n)!}}{\cancel{n!}\;\cancel{(100-n)!}} = \frac{100!}{n}.$$

Сумираме по n от 51 до 100: следва, че $\displaystyle\sum_{n=51}^{100} \frac{100!}{n}$ е броят на всички пермутации на 100 елемента, съдържащи цикъл с дължина над 50. Но $100!$ е броят на всички пермутации на 100 елемента (със или без дълъг цикъл). Следователно вероятността затворниците да попаднат на такава пермутация е равна на

$$\frac{1}{\cancel{100!}} \sum_{n=51}^{100} \frac{\cancel{100!}}{n} = \sum_{n=51}^{100} \frac{1}{n} = \sum_{n=1}^{100} \frac{1}{n} - \sum_{n=1}^{50} \frac{1}{n} \approx (\gamma + \ln 100) - (\gamma + \ln 50) =$$

$$= \ln 100 - \ln 50 = \ln \frac{100}{50} = \ln 2 = 0,693147 \approx 0,69 = 69\%.$$ (Точната стойност на сбора $\displaystyle\sum_{n=51}^{100} \frac{1}{n}$ е $0,688172$ и съвпада с приближението с точност до стотни.)

И така, вероятността за съществуване на дълъг цикъл (с дължина над 50) е приблизително 69 % — това е всъщност вероятността затворниците да бъдат екзекутирани. Следователно вероятността да се спасят е 100 % – 69 %, т.е. 31 %, което е много повече, отколкото при първата стратегия.

Вижда се, че броят на затворниците (100) не оказва съществено влияние на решението. Всяко достатъчно голямо число би свършило същата работа.

27) Разделяне на златна верижка.

Майстор златар има златна верижка, съставена от седем звена. Майсторът наема работник за седем дена, като се уговарят работникът да получава по едно звено в края на всеки ден. На колко най-малко части майсторът трябва да раздели верижката, че да може да плаща на работника според уговорката?

Решение: Нека частите са N. Всяка част е или у работника, или у майстора (две възможности). За всичките N части възможностите са 2^N, т.е. могат да се образуват най-много 2^N различни суми (вкл. нулата). Понеже у работника може да се намира всяка от сумите 0, 1, 2, 3, 4, 5, 6 и 7 (общо осем на брой), трябва $2^N \geq 8$, т.е. $N \geq 3$. Значи майсторът трябва да раздели верижката на поне три части. Наистина, три части стигат: ако техните дължини са степени на двойката (1, 2 и 4), то с тях може да се образува всеки сбор от 0 до 7 вкл.

28) Неравномерно горящи фитили.

Разполагате с два фитила, всеки от които изгаря за 60 минути, но гори неравномерно: половината от фитила може да изгори например за 5 минути, а другата половина — за 55 минути. Имате също запалка, която можете да ползвате произволен брой пъти. Как ще отмерите точно 45 минути?

Решение: Палим единия фитил от двата края, другия фитил — от единия край. Чакаме да изгори първият фитил (30 минути). В този миг от втория фитил ще са изгорели 30 минути и ще има за още 30. Палим го от другия край, което ще съкрати оставащото време на 15 минути. Общо: 30 + 15 = 45 минути.

29) Коварна гатанка.

Кое е това, което се среща веднъж във всяка минута, два пъти във всеки момент и нито веднъж в сто години?

Решение: Буквата "м" се среща веднъж във всяка "минута", два пъти във всеки "момент" и нито веднъж в "сто години". Трудността се състои в това да преминем от говорене за вещите (*de re*) към говорене за думите (*de dicto*). Двусмислени (но по друг начин) могат да бъдат и картините: сходни образи (напр. заекът и патето) се възприемат като различни заради разположението.

30) Скъсана тениска.

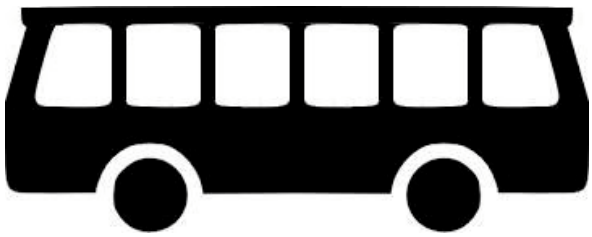
Колко дупки има в тази тениска?

Решение: Дупките са осем:
— две отпред (скъсаното);
— две отзад (щом фонът се вижда през тениската, значи тя е скъсана не само отпред, но и отзад);
— две на всеки ръкав (за ръцете);
— една отгоре (за главата);
— една отдолу (на кръста).

31) Автобус.

Вчера в София снимах автобус, който тъкмо пресичаше на червено. Не разбирам от фотография, затова не можах да хвана в кадър светофара, нито пътя, а само автобуса.

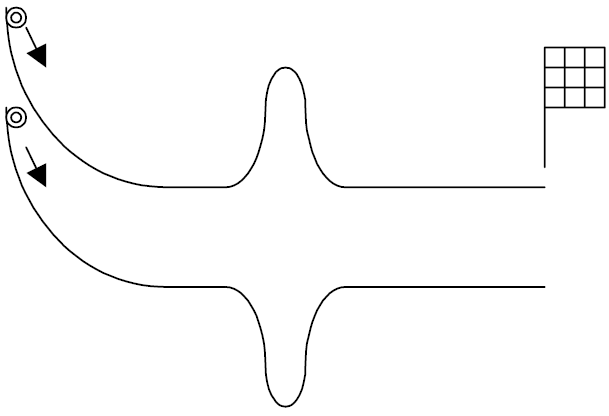
Накъде се движи автобусът?



Решение: Автобусът се движи наляво. Иначе щяха да се виждат вратите, а в момента се виждат само прозорци. (Това важи за държави като България с дясноориентирано движение.)

32) Две търкалящи се топки.

Две топки се търкалят по две писти без начална скорост, без триене и без съпротивление на въздуха. На пътя на първата топка има възвишение, а на пътя на втората топка има също такава вдлъбнатина. (Вдлъбнатината е огледален образ на възвишението.) Коя топка първа ще стигне финала? Коя топка ще има по-голяма крайна скорост?



Решение: Двете топки ще имат една и съща крайна скорост — това следва от закона за запазване на енергията. Скоростта на горната топка е по-малка по време на преодоляване на възвишението, затова нейната средна скорост е по-малка от крайната скорост. Скоростта на долната топка пък е по-голяма по време на преминаване на вдлъбнатината, затова нейната средна скорост е по-голяма от крайната скорост. Следователно първа ще пристигне топката с по-голяма средна скорост, т.е. долната топка.

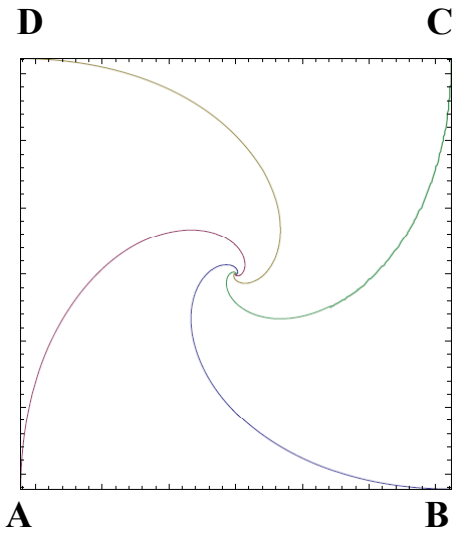
33) Четири таблетки.

Четири таблетки изглеждат напълно еднакво — по форма, размери, тегло, на цвят и на вкус. В действителност таблетките са от два различни флакона — по две таблетки от всеки флакон. След малко трябва да пиете по една таблетка от двата вида. Как ще спазите предписанието?

Решение: Чупим таблетките надве и взимаме по половинка от всяка таблетка. Или ги счукваме на прах, смесваме ги и изпиваме половината от сместа.

34) Бръмбари.

Във всеки от върховете на квадрат е застанал по един бръмбар. Бръмбарите едновременно започват да се движат — всеки се насочва към съседа си по посока на часовниковата стрелка (т.е. A към D, D към C, C към B, B към A). Четирите бръмбара ще се срещнат в центъра на квадрата, след като всеки от тях опише по една спирала, започваща от съответния връх (разбира се, спиралите са еднакви). Намерете дължината на всяка спирала, т.е. дължината на пътя, изминат от всеки бръмбар.

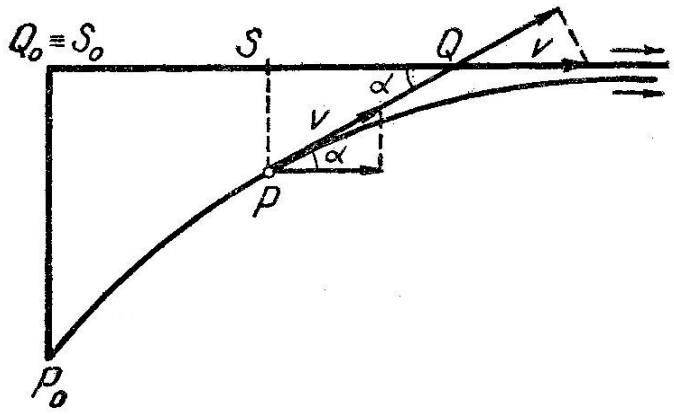


Решение: Да разгледаме например бръмбарите A и D. Скоростта на A сочи право към D, а скоростта на D сочи право към C, т.е. перпендикулярно на скоростта на A. Затова скоростта на D нито помага, нито пречи на сближаването на A и D, т.е. това сближаване зависи само от скоростта на A. Следователно сближаването на A и D протича (от гледна точка на A) така, сякаш D не се движи (по-точно, разстоянието между A и D намалява по същия начин, както ако D беше неподвижен). Ето защо A ще стигне до D за същото време (и със същата скорост), а значи ще измине и същия път, както ако D не се движеше. Но в последния случай A би изминал път, равен на дължината на страната на квадрата. От направените разсъждения следва, че също такъв път ще бъде изминат и когато D се движи. И така, бръмбарът A ще измине път, равен на дължината на страната на квадрата. Същото важи и за другите бръмбари. Всяка спирала е дълга колкото страната на квадрата.

Поради симетрията във всеки миг бръмбарите се намират във върховете на четириъгълник с равни страни и прави ъгли, т.е. във върховете на квадрат. Този квадрат се върти по часовниковата стрелка и непрекъснато се смалява, докато се свие в точка в центъра на първоначалния квадрат.

Уравнението на траекторията (спиралата) може да се получи с помощта на висшата математика, но то не е нужно за решаването на задачата.

35) Гонитба.



Кораб P се движи на север и забелязва право пред себе си на разстояние 6 мили кораб Q, движещ се на изток. Тогава корабът P започва да гони Q, държейки курс право към него (докато Q продължава пътя си на изток). Корабите се движат с постоянни и равни скорости, така че корабът P никога няма да настигне Q, но ще успее да смали разстоянието. Колко ще бъде разстоянието между двата кораба след достатъчно дълго време?

Решение: Тази задача прилича малко на предишната по това, че единият обект държи курс към другия. Има обаче съществена разлика: скоростите на двата обекта не са постоянно перпендикулярни (а само в началния момент). Затова решението на предишната задача е неприложимо в тази.

Да означим с v скоростта на всеки от корабите, α е ъгълът между посоките на техните скорости, S е ортогоналната проекция на P върху траекторията на Q. Точката S преследва точката Q със скорост $v \cdot \cos\alpha$, а в същото време Q бяга от S със скорост v. Следователно разстоянието SQ расте със скорост $v \cdot (1 - \cos\alpha)$.

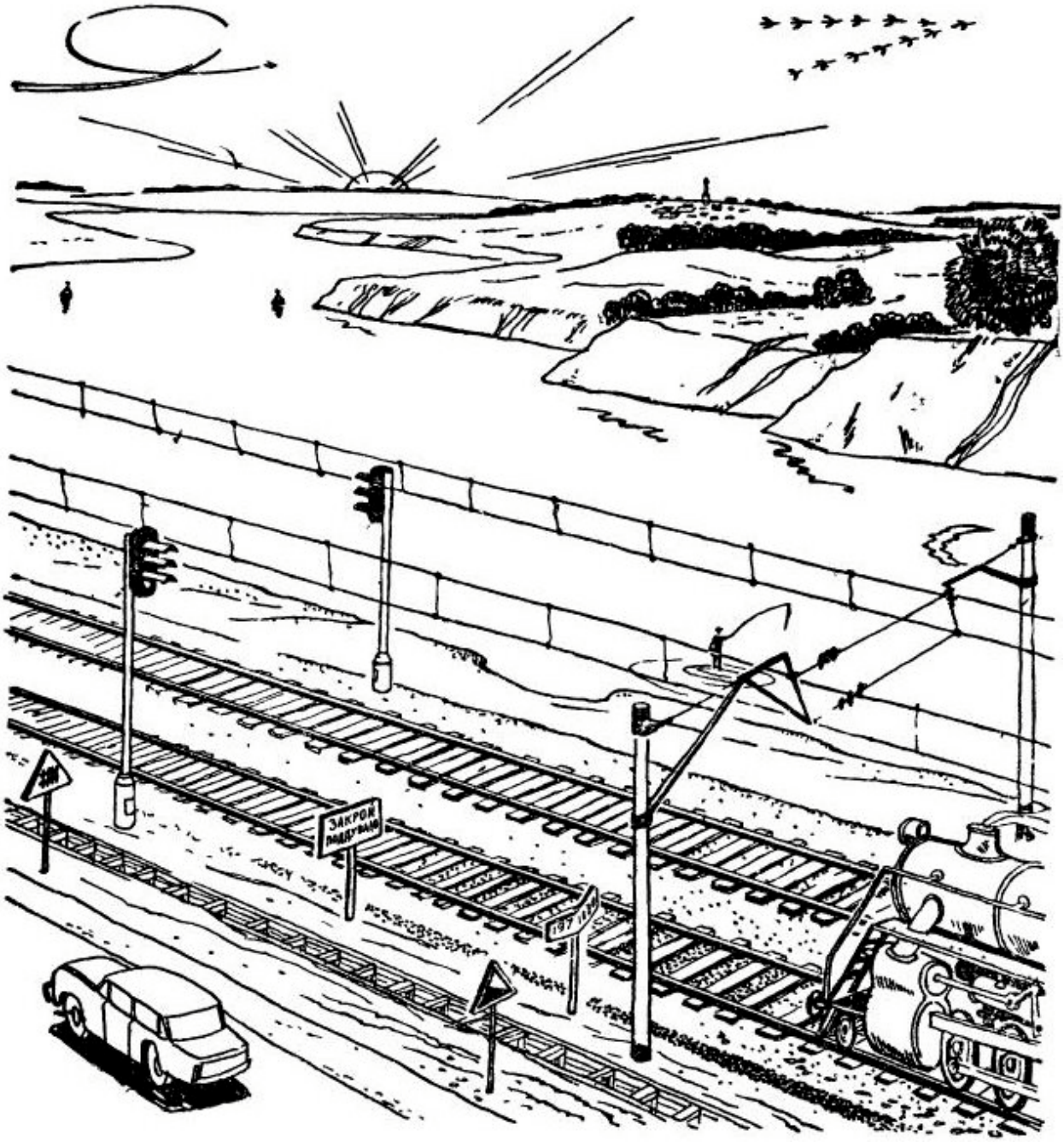
Проекцията на скоростта на кораба Q върху правата PQ има големина също $v \cdot \cos\alpha$, тоест корабът Q се отдалечава от P със скорост $v \cdot \cos\alpha$, като същевременно P се приближава към Q със скорост v. Следователно разстоянието PQ намалява със скорост $v \cdot (1 - \cos\alpha)$.

И така, разстоянието PQ намалява със същата моментна скорост, с която разстоянието SQ расте. Следователно техният сбор PQ + SQ не се изменя. Отначало $Q_0 \equiv S_0$, така че $Q_0 S_0 = 0$, а $P_0 Q_0 = 6$ мили по условие, затова PQ + SQ = 6 мили в началния момент, а значи и във всеки момент, тъй като този сбор не се променя (въпреки че всяко от събираемите се мени с течение на времето). След достатъчно дълго време точките P и S могат да се считат за съвпадащи, затова PQ = SQ, откъдето $PQ + SQ = 2 \cdot PQ = 6$ мили, тоест PQ = 3 мили. Това е търсеното разстояние.

Заб. Вижда се, че числовите данни в тази задача не са съществени (просто крайното разстояние PQ е равно на половината от началното). Съществено е условието за равенство на скоростите на двата кораба. Идеята на решението може да се обобщи за случая на неравни, но постоянни скорости: разглежда се сборът $PQ + k \cdot SQ$ с подходящо избран коефициент k, зависещ от двете скорости (по-точно, от тяхното отношение).

36) Наблюдателни ли сте?

На рисунката е изобразена измислена местност. Следва списък с въпроси, чиито отговори могат да бъдат получени с помощта на картинката, която съдържа достатъчно подсказки.



1) Много време ли остава до новолунието? 2) Скоро ли ще настъпи нощта?
3) Кое годишно време е? 4) Накъде тече реката? 5) Плавателна ли е тя?
6) Бързо ли се движи влакът? 7) Отдавна ли е минал предишният влак?
8) Автомобилът дълго ли ще се движи успоредно на железопътната линия?
9) За какво трябва да се подготви шофьорът на автомобила?
10) Има ли наблизо мост? 11) Има ли летище в района?
12) Лесно ли спират насрещните влакове? 13) Духа ли вятър?

Решение:

1) До новолунието остава малко време: месецът старее (отражението му се вижда във водата). Когато лунният сърп е извит като “**С**”, той старее, а когато е извит като буквата “**Р**”, расте.

Това следва от факта, че месечното движение на Луната около Земята се извършва от запад на изток.

Земята се завърта около оста си за едно денонощие и прави една обиколка около Слънцето за една година. Луната също се върти около оста си и обикаля около Земята за един месец. Тези въртеливи движения се извършват в една посока — обратно на часовника, ако гледаме от Северния полюс.

2) Тук всъщност се пита дали слънцето изгрява, или залязва. Старият месец се вижда при изгрев, а новият — при залез, значи сега слънцето изгрява, така че нощта няма да настъпи скоро.

Това правило следва от факта, че месечното движение на Луната около Земята се извършва от запад на изток. Тъй като Земята се върти около оста си също от запад на изток, то това придава на небесните тела видимо движение по небосвода от изток на запад. Следователно месечното движение на Луната е в обратна посока на денонощното, т.е. всяко денонощие тя изостава малко от Слънцето във видимото си движение. Когато “старее”, тя се приближава към Слънцето по небето (по-точно, то я настига), затова тя изгрява и залязва преди него. Старият месец не може да бъде видян в небето по залез-слънце, защото вече е залязъл, но може да се види сутрин, тъй като изгрява първи.

При новолуние Слънцето изпреварва Луната по небето, а после започва да изгрява и залязва преди нея. Следователно новият месец не може да бъде видян по изгрев-слънце, тъй като още не е изгрял, но може да се види вечер, понеже залязва по-късно.

3) Слънцето изгрява право пред нас от изток. Ятото птици лети надясно, т.е. на юг — към топлите страни. Значи е есен.

4) В Северното полукълбо десният бряг на реките е по-стръмен. Следователно реката тече от нас към хоризонта.

Това се обяснява чрез т.нар. сила на Кориолис. Различните точки от земната повърхност се въртят с различна скорост около оста на Земята. Най-бързо се върти екваторът, а най-бавно — точките близо до полюсите. Телата, които се отдалечават от екватора, се отклоняват на изток, защото по инерция запазват по-високата скорост, с която са се движили отначало. Обратно: телата, приближаващи се към екватора, се отклоняват на запад, защото по инерция запазват по-ниската скорост, с която са се движили отначало. С други думи, всички тела, които не се движат точно на изток или на запад, се отклоняват от посоката си, сякаш им действа някаква сила (тъкмо тази инерчна сила се нарича сила на Кориолис). Телата се отклоняват (спрямо посоката на своето движение) надясно в Северното полукълбо и наляво — в Южното. Това важи както за летящи тела, така и за въздушни и водни маси, в това число и за реките. Затова в Северното полукълбо реките подмиват десния бряг (а в Южното — левия).

5) Реката е плавателна: виждат се шамандури.

6) Влакът е спрял: свети най-долната светлина на светофора — червената.

7) Предишният влак е минал оттук скоро. Той се намира в следващия участък от жп линията, поради което е включен забранителният сигнал.

8) Автомобилът няма дълго да се движи успоредно на жп линията: има пътен знак за железопътен прелез.

9) Шофьорът трябва да се приготви за спускане, т.е. трябва да натисне спирачката, за да намали скоростта: има предупредителен пътен знак за предстоящо стръмно спускане.

10) Вероятно наблизо има мост: поставен е жп знак, който задължава машиниста да затвори вратичката на котела.

Заб. Надписът е на руски, т.е. местността наистина е в Северното полукълбо (това се предполага в някои от разсъжденията по-горе).

11) В небето се вижда следа от самолет, направил лупинг, а правенето на фигури от висшия пилотаж е разрешено само в близост до летище.

12) Знакът близо до жп линията показва, че насрещните влакове изкачват наклон, така че не е трудно да спрат.

13) Духа вятър: димът на парния локомотив се стеле, а влакът, както знаем, не се движи.

37) Рожден ден.

Алберт и Бернард току-що са се запознали с Шерил и искат да научат кога има рожден ден. Шерил им дава списък с десет възможни дати:
15. май, 16. май, 19. май;
17. юни, 18. юни;
14. юли, 16. юли;
14. август, 15. август и 17. август.

След това Шерил казва на Алберт месеца, в който е родена, а на Бернард — числото от месеца.

— Не знам кога е рожденият ден на Шерил, обаче знам със сигурност, че и Бернард не знае — заявява Алберт.

— Отначало не знаех кога е рожденият ден на Шерил, но вече знам — отговаря Бернард.

— Сега и аз знам кога е родена Шерил — казва Алберт.

Коя от десетте дати е рожденият ден на Шерил?

Решение: Първия път Алберт, естествено, не знае кога е рожденият ден, защото, който и месец да е чул, ще има поне две дати, между които да се колебае. Алберт обаче е сигурен, че и Бернард не знае рождения ден, а това значи, че числото, казано от Шерил на Бернард, се съдържа в поне две от десетте дати. Такива са числата 14, 15, 16 и 17. Понеже Алберт не знае числото, но е сигурен, че то е измежду тези, то месецът, чут от Алберт, съдържа само дати с тези числа (а не съдържа числата 18 и 19). Следователно месеците май и юни отпадат, т.е. рожденият ден е през юли или през август.

От репликата на Бернард следва, че тази информация му е достатъчна, за да определи рождения ден. Следователно числото, което е чул, се съдържа само в една от петте дати за юли и август. С други думи, числото 14 отпада, тъй като има две дати с него (14. юли и 14. август). Остават възможностите 16. юли, 15. август и 17. август, от които Бернард е избрал правилната, като е знаел числото от месеца.

Последната реплика на Алберт означава, че той може да познае рождения ден на Шерил сред тези три дати, като знае само месеца. Но ако месецът беше август, то Алберт би се колебал между 15. август и 17. август. Значи месецът е юли, а не август.

Окончателно, рожденият ден на Шерил е 16. юли.

Алгоритмични задачи

При този тип задачи трябва да измислите алгоритъм или структура от данни, която да се справя ефективно с даден проблем. Понякога има хитрост, чието откриване води до решение на задачата чрез стандартен алгоритъм.

1) Идеалният президент трябва да е популярен сред хората, но да познава възможно най-малко от тях, за да бъде независим.

Имаме n канидат-президенти, номерирани с числата от 1 до n. Техните познанства са зададени чрез предиката `know(i: int; j: int): bool`, който връща `true`, ако № i познава № j, и `false` — в противен случай. Забележете, че отношението не е непременно симетрично: възможно е № i да познава № j, но № j да не познава № i. Идеален кандидат за президент е човек, когото всички други познават, но той не познава никого от тях.

Съставете алгоритъм, който отпечатва номерата на идеалните кандидати за президент с минимален брой извиквания на предиката `know`.

Решение: Очевидно не може да има двама или повече идеални кандидати (защото те трябва хем да се познават, хем да не се познават). Тоест или има точно един идеален кандидат, или няма нито един.

Най-просто е да проверим за всеки човек дали е идеален кандидат. Правим два вложени цикъла: с външния обхождаме кандидатите, а с вътрешния проверяваме всеки кандидат отговаря ли на изискванията.

```
isPerfect(i,n: int): bool
for j ← 1 to n
  if (j ≠ i) and (know(i,j) or not know(j,i))
    return false
return true

findPerfect(n: int): int // -1 <=> няма идеален кандидат
for i ← 1 to n
  if isPerfect(i,n)
    return i
return -1
```

Този алгоритъм не е достатъчно бърз, защото, ако например кандидатите № 1, 2, 3, … , $\left\lfloor \frac{n}{2} \right\rfloor$ познават всички, а № $\left\lfloor \frac{n}{2} \right\rfloor + 1$, $\left\lfloor \frac{n}{2} \right\rfloor + 2$, … , n — никого, то за всеки кандидат от първата половина ще има точно едно извикване на предиката `know` (понеже още първото извикване ще установи, че текущият кандидат познава някого; предполагаме непълно булево изчисление).

За всеки кандидат i от втората половина проверките с $j = 1, 2, 3, \ldots, \left\lfloor \frac{n}{2} \right\rfloor$ успяват и чак проверката с $j = \left\lfloor \frac{n}{2} \right\rfloor + 1$ показва, че кандидат № i не е идеален. Това са $2\left\lfloor \frac{n}{2} \right\rfloor + 2$ извиквания на `know` за всяко $i = \left\lfloor \frac{n}{2} \right\rfloor + 1, \left\lfloor \frac{n}{2} \right\rfloor + 2, \ldots, n$, което прави общо $\left(2\left\lfloor \frac{n}{2} \right\rfloor + 2\right) \cdot \left(n - \left\lfloor \frac{n}{2} \right\rfloor\right) \sim n \cdot \frac{n}{2} = \frac{n^2}{2}$ извиквания на предиката `know`. В най-лошия случай броят на извикванията е също толкова голям или по-голям, така че по този алгоритъм предикатът `know` се вика $\Omega\left(n^2\right)$ пъти, т.е. времевата сложност на този алгоритъм е поне квадратична.

Можем да се справим по-добре, ако съобразим, че при всяко извикване на предиката `know` отпада един кандидат. Наистина, след извикването `know(i,j)` има само две възможности:

1. Ако № i познава № j, то № i не е идеален (тъй като познава някого, а именно № j); затова кандидат № i отпада.
2. Ако № i не познава № j, то № j не е идеален (тъй като не всички го познават); затова кандидат № j отпада.

Така с $n - 1$ извиквания на предиката `know` можем да ограничим броя на кандидатите до точно един. Този един е предполагаем идеален кандидат, Можем да проверим дали е идеален чрез още $2(n - 1)$ извиквания на `know`. Общият брой извиквания става линеен: $O(n)$, което е много по-добро от предишното решение.

```
findPerfect(n: int): int // -1 <=> няма идеален кандидат
i ← 1
for j ← 2 to n
  if know(i,j)
    i ← j
if isPerfect(i,n)
  return i
return -1
```

Първият цикъл (отсяването на потенциалния идеален кандидат) съдържа $n - 1$ извиквания на предиката `know`. Проверката на i-тия кандидат съдържа $2(n - 1)$ извиквания на `know` в най-лошия случай (когато i-тият кандидат наистина е идеален). Така общият брой извиквания на предиката `know` при този алгоритъм е равен на $3(n - 1) = \Theta(n)$ в най-лошия случай.

Може ли да бъде ускорен и този алгоритъм? По порядък — не може. Ще докажем долна граница $\Omega(n)$ на максималната сложност на задачата по време (като мерим времето с броя на извикванията на предиката `know`). Заедно с доказаната горна граница $O(n)$, която следва от наличието на линеен алгоритъм, това води до извода, че времевата сложност на задачата е $\Theta(n)$.

По-точно, ще докажем, че в най-лошия случай задачата не може да бъде решена с по-малко от $2(n-1)$ извиквания на предиката `know`. Ще разделим доказателството на два етапа.

1-ви етап: Ще докажем, че не съществува реализация на функцията `isPerfect(i,n)` с по-малко от $2(n-1)$ извиквания на `know` в най-лошия случай. Действително, ако кандидат № i е идеален, то функцията, както и да е реализирана, трябва да провери, че всички го познават и че той не познава никого, а това не може да стане с по-малко от $2(n-1)$ извиквания на `know` (а именно: всички извиквания от вида `know(i,j)` и `know(j,i)` за $\forall j \neq i$), защото наличието или отсъствието на познанство между двама души е независимо от наличието или отсъствието на познанство между други двама, следователно тези $2(n-1)$ стойности не могат да бъдат получени като следствия от други извиквания на `know`, по-малко на брой. Разбира се, дотук не сме доказали, че това е най-лошият случай, но най-лошият случай е поне толкова лош, затова доказаната долна граница важи и за него.

2-ри етап: Ще докажем, че не съществува реализация на функцията `findPerfect(n)` с по-малко от $2(n-1)$ извиквания на `know` в най-лошия случай. Да допуснем противното: че има реализация на `findPerfect(n)` с по-малко от $2(n-1)$ извиквания на `know` в най-лошия случай. Тогава следната реализация на `isPerfect(i,n)` ще работи във всички случаи (вкл. в най-лошия) с по-малко от $2(n-1)$ извиквания на предиката `know`, а това противоречи на доказаното в първия етап.

```
isPerfect(i,n: int): bool
return findPerfect(n) = i
```

Полученото противоречие показва, че направеното допускане не е вярно. Следователно е вярно първоначалното твърдение: не съществува реализация на функцията `findPerfect(n)` с по-малко от $2(n-1)$ извиквания на `know` в най-лошия случай. С това долната граница е доказана.

Заб. Получените граници — горната $3(n-1)$ и долната $2(n-1)$ — са с еднакъв порядък (и двете са линейни функции на n), но не съвпадат напълно. Така въпросът за точната стойност на времевата сложност на задачата остава отворен. Тук няма да се задълбочаваме в този проблем, а ще се задоволим, както обикновено се прави в теорията на алгоритмите, с това, че полученият от нас алгоритъм (вторият, подобреният алгоритъм) е оптимален по порядък.

2) Сливащи се свързани списъци.

Дадени са указатели към първите елементи на два ациклични едносвързани списъка. Може ли за линейно време и с константен брой допълнителни променливи от примитивен тип да се определи дали двата списъка се сливат (тоест дали имат общ елемент)? А да се намери първият им общ елемент?

Решение: Очевидно за линейно време и с константен брой допълнителни променливи от примитивен тип (например указатели) можем да обходим списъците, докато стигнем до последните им елементи. Списъците се сливат тогава и само тогава, когато последните им елементи съвпадат.

Ако двата списъка се сливат, то мястото на сливането, т.е. първият им общ елемент, отстои на едно и също разстояние от краищата на списъците (но не непременно и от началата им). Обаче достатъчно е да намерим дължините на списъците (това става с помощта на два брояча в рамките на гореспоменатото обхождане), тогава лесно ще можем да кажем кои двойки елементи евентуално съвпадат. Ако например списъкът A има дължина 9, а списъкът B има дължина 13, то първите $13 - 9 = 4$ елемента на B със сигурност не са в списъка A, така че започваме сравнението от $B[5]$. Тоест сравняваме $B[5]$ и $A[1]$, после $B[6]$ и $A[2]$, $B[7]$ и $A[3]$, ... , $B[13]$ и $A[9]$; по-точно: до първата двойка съвпадащи елементи. Тази процедура очевидно изисква линейно време и константен брой допълнителни променливи от примитивен тип (един брояч отначало и два указателя през цялото време).

3) Последователни числа с максимален сбор.

Даден е масив с n числа. Постройте алгоритъм, който намира множество от последователни елементи с максимален сбор.

Решение: Изходът на кой да е алгоритъм, решаващ задачата, е наредена двойка от вида `(First, Last)`, където `First` е индексът на първия, а `Last` — на последния от последователните елементи с максимален сбор, `First ≤ Last`. (Алгоритъмът може да върне и максималния сбор.)

Едно решение е да проверим всички наредени двойки от описания вид. Това решение има времева сложност $\Theta(n^2)$, която не е оптимална.

Съществува алгоритъм, който решава задачата за линейно време: $\Theta(n)$. Той се основава на следния факт: множество от последователни елементи не може да е оптимално, ако сборът на някоя негова представка или наставка е отрицателен. Например множеството {1, 2, –5, 6, 3, –1, 4, –2, 3, 3, –2, 3} има представка с отрицателна сума, а именно {1, 2, –5}. В този случай е по-добре да вземем множеството {6, 3, –1, 4, –2, 3, 3, –2, 3}, т.е. да махнем представката. Същото важи и за наставки с отрицателен сбор.

Забелязаният факт води до следния алгоритъм:

```
typedef Answer = struct {First, Last, MaxSum: integer}

maxSumConseq(A[1...n]: array of integers): Answer
First  ← 1
Last   ← 0 // индикатор за празно множество;
MaxSum ← 0 // сборът на празното множество е нула
Start  ← 1
Finish ← 0
Sum    ← 0
while Finish < n
  while (Sum ≥ 0) and (Finish < n)
    Finish ← Finish + 1
    Sum    ← Sum + A[Finish]
    if Sum > MaxSum
      First  ← Start
      Last   ← Finish
      MaxSum ← Sum
  if Sum < 0
    Start  ← Finish + 1
    Sum    ← 0
return (First, Last, MaxSum)
```

П р и м е р : Нека като вход на алгоритъма да подадем следния масив: (4, –1, 5, –10, 9, 3, –20, 7). Алгоритъмът започва от първото число и натрупва сбор: 4, 4 + (–1) = 3, 4 + (–1) + 5 = 8, 4 + (–1) + 5 + (–10) = –2, докато получи отрицателен сбор (–2). След това алгоритъмът изтрива получената представка (4, –1, 5, –10) с отрицателна сума и продължава аналогично с остатъка от масива, т.е. (9, 3, –20, 7). Отново се натрупват сборове: 9, 9 + 3 = 12, 9 + 3 + (–20) = –8, при което се получава нова представка (9, 3, –20) с отрицателен сбор. След нейното премахване от масива остава само последният елемент — числото 7. От него се образува само един сбор, а именно 7. Максималният от всички сборове е 9 + 3 = 12. Той се получава като стойност на променливата `MaxSum`, а променливите `First` и `Last` дават индекса на първия и последния елемент на този сбор (т.е. 5 и 6 съответно).

Коректност на алгоритъма: Тъй като при всяка итерация на вътрешния цикъл променливата `Finish` се увеличава с 1, то тя непременно ще достигне стойност n, което ще доведе до излизане от двата цикъла. Следователно алгоритъмът не може да се зацикли, т.е. винаги завършва изпълнението си.

Дотук показахме, че алгоритъмът винаги дава резултат. Остава да докажем, че резултатът винаги е правилен. Това става с помощта на инварианти на двата цикъла — външния и вътрешния.

Инварианта на външния цикъл: Всеки път, когато се проверява условието за край на външния цикъл, $\texttt{MaxSum} = \sum_{k=\texttt{First}}^{\texttt{Last}} A[k]$ е най-големият от сборовете от последователни елементи на подмасива $A[1...\texttt{Finish}]$ и текущите стойности на елементите на масива $A[1...n]$ съвпадат с входните стойности.

Доказателство: Че текущите елементи на масива съвпадат с оригиналните, е ясно: алгоритъмът не променя елементите на масива. Останалото се доказва чрез индукция по номера на итерацията на външния цикъл.

База: При влизане във външния цикъл равенството е в сила: двете страни имат стойност 0 съгласно с инициализацията. Сумата и подмасивът са празни и твърдението е тривиално вярно: сбор на празно множество е нула.

Индуктивна стъпка: Нека твърдението е изпълнено при някоя проверка за край на външния цикъл. Ще докажем, че то е изпълнено и при следващата проверка (стига да има такава).

Щом започва изпълнение на тялото на външния цикъл, то $\texttt{Finish} < n$. Освен това, $\texttt{Sum} \geq 0$ (това следва от инициализацията на Sum и от факта, че винаги, когато Sum стане отрицателна, веднага след това Sum става нула). Ето защо започва изпълнение на тялото на вътрешния цикъл.

Инварианта на вътрешния цикъл: Всеки път, когато се проверява условието за край на вътрешния цикъл, е изпълнено двойното равенство

$$\texttt{Sum} = \sum_{k=\texttt{Start}}^{\texttt{Finish}} A[k] = \max \left\{ \sum_{k=j}^{\texttt{Finish}} A[k] \;\middle|\; j = 1,\ 2,\ 3,\ \ldots,\ \texttt{Start} \right\};$$

също така, $\texttt{MaxSum} = \sum_{k=\texttt{First}}^{\texttt{Last}} A[k]$ е най-големият сбор от последователни елементи на подмасива $A[1...\texttt{Finish}]$; освен това, текущите стойности на елементите на масива $A[1...n]$ съвпадат с входните стойности.

Прекъсваме временно доказателството на инвариантата на външния цикъл.

Доказателство на инвариантата на вътрешния цикъл: Съвпадението на текущите с входните стойности беше вече доказано. Останалото се доказва чрез индукция по номера на итерацията.

База: При влизане във вътрешния цикъл твърдението за MaxSum е вярно съгласно с индуктивното предположение за инвариантата на външния цикъл; променливата Start току-що е получила стойност Finish + 1, т.е. сумата и множеството са празни, а променливата Sum току-що е получила стойност 0 (вж. инициализацията и тялото на условния оператор "if Sum < 0"), т.е. двойното равенство има вида 0 = 0 = 0.

Индуктивна стъпка: При всяко увеличение на Finish с единица, към Sum се добавя поредният елемент A[Finish] на масива. Така лявата половина от двойното равенство остава в сила.

Дясната половина е равносилна на следното твърдение:

$$\sum_{k=j}^{\texttt{Start}-1} A[k] \le 0, \ \forall j = 1, 2, 3, \ldots, \texttt{Start}.$$

При $j = \texttt{Start}$ неравенството е тривиално: $0 \le 0$. За останалите стойности, т.е. за $j = 1, 2, 3, \ldots, \texttt{Start} - 1$, твърдението се доказва с индукция по L, където $\texttt{Start}_1, \texttt{Start}_2, \ldots, \texttt{Start}_L$ са поредните стойности на Start ($\texttt{Start}_1 = 1$, $\texttt{Start}_L$ е текущата стойност).

База: При $L = 1$ твърдението е тривиално: $\texttt{Start} = \texttt{Start}_L = \texttt{Start}_1 = 1$, множеството $\{1, 2, 3, \ldots, \texttt{Start} - 1\} = \varnothing$ от допустимите стойности на j е празно, затова твърдението, че неравенството е вярно за $\forall j$, е тривиално.

Индуктивна стъпка: Нека неравенството е вярно за предишната стойност на L, т.е. за $L - 1$, т.е. $\sum_{k=j}^{\texttt{Start}_{L-1}-1} A[k] \le 0, \ \forall j = 1, 2, 3, \ldots, \texttt{Start}_{L-1} - 1$.

Ще докажем, че неравенството е в сила и за текущата стойност на L, тоест $\sum_{k=j}^{\texttt{Start}_L - 1} A[k] \le 0, \ \forall j = 1, 2, 3, \ldots, \texttt{Start}_L - 1$.

Най-напред, ясно е, че по време на работата на алгоритъма променливата Finish само расте (увеличава се с единица, започвайки от нулата). Тъй като на променливата Start се присвоява отначало стойност единица, а после само стойност Finish + 1, то ясно е, че стойностите на променливата Start образуват строго растяща редица: $\texttt{Start}_1 < \texttt{Start}_2 < \ldots < \texttt{Start}_L$.

Следователно за $\forall j = 1, 2, 3, \ldots, \texttt{Start}_{L-1} - 1$:

$\sum_{k=j}^{\texttt{Start}_L - 1} A[k] = \sum_{k=j}^{\texttt{Start}_{L-1} - 1} A[k] + \sum_{k=\texttt{Start}_{L-1}}^{\texttt{Start}_L - 1} A[k] \le 0$ като сбор от две неположителни събираеми. Това, че първата сума е неположителна, съвпада с индуктивното предположение. Това, че втората сума е неположителна (и дори отрицателна), следва от начина на работа на алгоритъма: като имаме предвид кога променливата Start получава нова стойност, става ясно, че $\sum_{k=\texttt{Start}_{L-1}}^{\texttt{Start}_L - 1} A[k]$ е такава стойност на Sum, при която алгоритъмът присвоява нова стойност на Start, а това става само когато $\texttt{Sum} < 0$.

Остана да разгледаме случая $j = \texttt{Start}_{L-1}, \ldots, \texttt{Start}_L - 1$.

Тогава $\sum_{k=j}^{\texttt{Start}_L - 1} A[k] = \sum_{k=\texttt{Start}_{L-1}}^{\texttt{Start}_L - 1} A[k] - \sum_{k=\texttt{Start}_{L-1}}^{j-1} A[k] \le 0$, защото умаляемото е отрицателно (вж. разсъжденията от края на предишния абзац),

а умалителят е неотрицателен (това пък следва от факта, че $\texttt{Start}_{L-1}$ и $\texttt{Start}_L$ са *поредни* стойности на променливата Start, т.е. междувременно стойностите на Sum — а това са тъкмо стойностите на умалителя — са само неотрицателни).

Остана да установим верността на индуктивното заключение за втората част от инвариантата на вътрешния цикъл (където се говори за MaxSum).

След всяка промяна на стойностите на трите променливи

First ← Start

Last ← Finish

MaxSum ← Sum

равенството $\texttt{MaxSum} = \sum_{k=\texttt{First}}^{\texttt{Last}} A[k]$ остава в сила, защото непосредствено след тези присвоявания се преминава към проверка за край на вътрешния цикъл, а тогава (съгласно с вече доказаното) имаме

$$\texttt{Sum} = \sum_{k=\texttt{Start}}^{\texttt{Finish}} A[k], \text{ т.е. } \texttt{MaxSum} = \sum_{k=\texttt{First}}^{\texttt{Last}} A[k].$$

Това, че $\sum_{k=\texttt{First}}^{\texttt{Last}} A[k]$ е най-големият от сборовете от последователни елементи на подмасива A[1...Finish], се доказва така: При предишната проверка то е било в сила (според индуктивното предположение). Нека допуснем, че при текущата (новата) проверка (след увеличението на Finish с единица) това вече не е в сила, т.е. има неравенство. Обаче не е възможно $\sum_{k=\texttt{First}}^{\texttt{Last}} A[k] >$ най-големия от сборовете от последователни елементи на подмасива A[1...Finish], защото $\sum_{k=\texttt{First}}^{\texttt{Last}} A[k]$ винаги е някой от тези сборове (според присвояванията на First и Last). Остава да опровергаем възможността $\sum_{k=\texttt{First}}^{\texttt{Last}} A[k] <$ най-големия от сборовете от последователни елементи на подмасива A[1...Finish]. Ако допуснем, че това се е сбъднало, то тъй като $\sum_{k=\texttt{First}}^{\texttt{Last}} A[k]$ не може да намалява (следва пак оттам), то значи алгоритъмът е пропуснал нов сбор, по-голям от досегашния максимум. "Нов" означава, че в този сбор участва новодобавеният елемент A[Finish] (след увеличаването на Finish с единица). Ние обаче разглеждаме само сборове от последователни елементи, затова пропуснатият сбор би бил от вида $\sum_{k=j}^{\texttt{Finish}} A[k]$ за някое j.

Ако j > Start, то сборът $\sum_{k=\text{Start}}^{\text{Finish}} A[k]$ ще е не по-малък: иначе масивът A[Start...j–1] би имал отрицателен сбор, което е невъзможно (при намиране на такъв масив Start винаги минава надясно от масива). Но щом сборът $\sum_{k=\text{Start}}^{\text{Finish}} A[k]$ е още по-голям, то и той е пропуснат, значи можем да считаме, че j = Start (т.е. случаят j > Start е излишен).

И така, без ограничение ще считаме, че $j \le$ Start. Вече доказахме, че

$$\text{Sum} = \sum_{k=\text{Start}}^{\text{Finish}} A[k] = \max \left\{ \sum_{k=j}^{\text{Finish}} A[k] \;\middle|\; j = 1, 2, 3, \ldots, \text{Start} \right\}.$$

Затова пропуснатият сбор (или още по-голям) непременно се пази в Sum. "Пропуснат" означава, че алгоритъмът не е актуализирал стойностите на променливите MaxSum, First и Last. Тогава Sum > MaxSum, а тъй като алгоритъмът прави такава проверка на всяка итерация на вътрешния цикъл, то пропускът е невъзможен.

С това е доказана инвариантата на вътрешния цикъл.

Продължаваме с доказателството на инвариантата на външния цикъл: Когато вътрешният цикъл завърши, неговата инвариантата ще бъде в сила, а именно: при последната проверка на условието за край на вътрешния цикъл. В този миг е в сила и инвариантата на външния цикъл (защото тя е част от инвариантата на вътрешния). След излизането от вътрешния цикъл се прави проверката "if Sum < 0" и евентуално се присвояват стойности на променливите Start и Sum, но не и на First, Last и MaxSum, т.е. инвариантата на външния цикъл остава в сила. Сега се преминава към проверка на условието за край на външния цикъл. Току-що видяхме, че неговата инвариантата е в сила, така че нейното доказателство е завършено.

Външният цикъл завършва тогава, когато Finish стане равно на n. От доказаната инвариантата на външния цикъл следва, че при последната проверка на условието за край на външния цикъл е изпълнено следното:

MaxSum = $\sum_{k=\text{First}}^{\text{Last}} A[k]$ е най-големият от сборовете от последователни елементи на масива A[1...Finish], т.е. A[1...n], и текущите стойности на елементите на масива A[1...n] съвпадат с входните стойности. Което значи, че променливите First, Last и MaxSum имат тъкмо необходимите стойности. Следователно резултатът на алгоритъма е верен, т.е. алгоритъмът работи правилно.

С това е установена коректността на алгоритъма.

Анализ на времевата сложност на алгоритъма:

В най-вътрешното ниво на вложеност променливата `Finish` расте с 1. Когато `Finish` стане равно на n, алгоритъмът ще излезе от двата цикъла. Следователно най-вътрешните оператори се изпълняват не повече от n пъти, значи времевата сложност на алгоритъма е $O(n)$. От друга страна, външният цикъл завършва едва когато `Finish` стане равно на n (не по-рано). За това са нужни поне $n-1$ увеличения на `Finish` с единица, следователно времето на алгоритъма е $\Omega(n)$. Окончателно, сложността на алгоритъма е $\Theta(n)$.

Анализ на времевата сложност на задачата:

От съществуването на линеен алгоритъм следва, че времевата сложност на задачата е $O(n)$. От друга страна, ако допуснем, че някой алгоритъм си спестява прочитането на поне един от елементите на масива, то може да се окаже, че непрочетеният елемент е много голямо цяло число (по-голямо от максималния сбор на последователни елементи от останалата част от масива). Тогава непрочетеният елемент сам би образувал сбор, по-голям от максималния сбор, намерен от алгоритъма, което значи, че отговорът на алгоритъма е некоректен. Следователно всеки коректен алгоритъм прочита всички елементи на масива, за което са нужни поне n операции, затова сложността на задачата е $\Omega(n)$. Окончателно, времевата сложност на задачата е равна на $\Theta(n)$.

Това значи, че предложеният алгоритъм е оптимален (поне по порядък на бързодействието).

4) Случайно разбъркване.

Съставете алгоритъм, който разбърква по случаен начин елементите на масив с дължина n, тоест генерира случайна пермутация на елементите на масива. Всички пермутации трябва да бъдат равновероятни. Можете да използвате само функцията `rand()`, която връща случайна стойност, равномерно разпределена в интервала $[0;1)$.

Решение:

```
randomShuffle(A[1...n]: array)
for k ← n downto 1
  swap(A[k], A[floor(rand() × k + 1)])
```

Заб. Операцията `swap` разменя два елемента на масива. Ако индексите им съвпадат (т.е. те са един и същ елемент), то `swap` не прави нищо.

Заб. Функцията `floor` закръгля до най-близкото цяло число, не по-голямо от дадения аргумент.

Всеки от n-те елемента може с еднаква вероятност $(1/n)$ да отиде на последно място. Всеки от останалите $n-1$ елемента може с еднаква вероятност да отиде на предпоследно място и т.н. Значи всички пермутации са равновероятни.

5) Цикличен ли е даден едносвързан списък?

а) Даден е указател към първия елемент на едносвързан списък, чийто брой елементи не е известен. Да се състави алгоритъм, който за линейно време и с фиксиран брой допълнителни променливи от примитивен тип проверява дали списъкът се зацикля.

Решение: Не можем просто да обходим списъка, защото, ако има цикъл, никога няма да стигнем до края на списъка, но няма и да сме сигурни, че има цикъл, тъй като може броят на елементите да е по-голям от изминатите досега. (Не можем да помним елементите, през които сме минали: за това е нужна много памет. Не можем и да проверяваме всеки нов елемент дали не е сред вече изминатите: това изисква два вложени цикъла — един за самото обхождане и един за проверката, — т.е. квадратично време.)

Правилното решение е следното: Движим се по списъка с три указателя (към текущия, предходния и следващия елемент) и обръщаме списъка, т.е. указателят за наследника на текущия елемент се пренасочва от досегашния наследник (“следващия елемент”) към предходния елемент.

Ако няма цикъл в списъка, то алгоритъмът ще стигне до края, отбелязан чрез нулев указател към наследник (липса на следващ елемент).

Ако списъкът съдържа цикъл, алгоритъмът ще стигне повторно до мястото на зациклянето (без да го разпознае) и ще продължи по списъка, но той е вече обърнат, тоест движението ще продължи в посока към началото на списъка (и тази част от списъка ще бъде обърната още веднъж, т.е. указателите ще възстановят първоначалната си посока). В крайна сметка алгоритъмът ще стигне до първия елемент на списъка (вместо до последен елемент) и ще разбере това, тъй като указателят към този елемент съвпада с указателя, даден по условие. Така алгоритъмът ще разпознае, че е имало цикъл.

Недостатък на това решение, е че променя списъка.

б) А ако поставим допълнителното условие да не се променя списъкът?

Решение: Обхождаме списъка с два указателя: на всяка стъпка от алгоритъма единият указател се придвижва напред с един елемент, а другият указател — с два елемента. Значи разстоянието между указателите расте с единица на всяка стъпка.

Ако списъкът не се зацикля, то по-бързият указател ще стигне първи до края на списъка.

Ако списъкът съдържа цикъл, то по някое време двата указателя ще навлязат в цикъла и ще започнат да обикалят по него. В момента, в който разстоянието между тях стане кратно на дължината на цикъла (това ще стане непременно, тъй като разстоянието между тях нараства с единица всеки път), по-бързият указател ще настигне по-бавния. В този миг двата указателя ще сочат към един и същи елемент (т.е. ще бъдат равни), което ще бъде признак за наличието на цикъл в списъка.

6) Тройка с нулева сума.

Даден е масив с n цели числа. Съставете алгоритъм, който разпознава дали съществуват три елемента със сбор нула. Съставете алгоритъм, който да брои колко такива тройки има.

Решение: Най-простият алгоритъм се състои от три вложени цикъла:

```
sum3zero(A[1...n]: array of integers): bool
for i ← 1 to n-2
  for j ← i+1 to n-1
    for k ← j+1 to n
      if A[i]+A[j]+A[k] = 0
        print i,j,k
        return true
return false
```

Времевата сложност на този алгоритъм очевидно е $\Theta(n^3)$ в най-лошия случай: когато масивът не съдържа тройка числа с нулева сума.

Ако фиксираме две от числата, например A и B, то ще има само два вложени цикъла. Съществува ли обаче бърз начин да проверим дали има трето число със стойност $-(A+B)$? Такъв начин съществува: предварително сортираме масива (за време $\Theta(n \cdot \log n)$), след което с двоично търсене установяваме дали масивът съдържа число $-(A+B)$. Този алгоритъм има времева сложност $\Theta(n \cdot \log n) + \Theta(n^2 \cdot \log n) = \Theta(n^2 \cdot \log n)$. Първото събираемо е сложността на сортирането, второто събираемо е сложността на двоичното търсене ($\Theta(\log n)$ за всяка от общо $\Theta(n^2)$ двойки от елементи на масива). По порядък надделява втората сложност. Вторият алгоритъм в псевдокод изглежда така:

```
sum3zero(A[1...n]: array of integers): bool
sort(A)
for i ← 1 to n-2
  for j ← i+1 to n-1
    k ← BinarySearch(A[j+1...n], -A[i]-A[j])
    if k > 0
      print i,j,k
      return true
return false
```

Този алгоритъм е по-бърз от предишния, но не е най-бързият възможен. Известен е по-бърз алгоритъм с времева сложност $\Theta(n^2)$. Той също сортира масива, след което в цикъл по един елемент, напр. $A[i]$, променя позициите j и k на другите два така, че общо изразходват време $\Theta(n)$: на всяка стъпка по-малкият индекс расте или по-големият намалява с единица.

В псевдокод третият алгоритъм изглежда така:

```
sum3zero(A[1...n]: array of integers): bool
sort(A)
for i ← 1 to n-2
  j ← i+1
  k ← n
  while k > j
    if A[i]+A[j]+A[k] = 0
      print i,j,k
      return true
    else if A[i]+A[j]+A[k] > 0
      k ← k-1
    else // A[i]+A[j]+A[k] < 0
      j ← j+1
return false
```

Коректност на алгоритъма: При фиксирано i, ако $A[i] + A[j] + A[k] > 0$, то сборът трябва да бъде намален, а тъй като масивът е сортиран, то някой от индексите трябва да бъде намален. Но индексът i не може да бъде намален, защото е фиксиран; индексът j също не може да бъде намален, защото при текущата стойност на i е установено, че никое по-малко j не върши работа. Така остава да бъде намален само индексът k.

Обратно: ако $A[i] + A[j] + A[k] < 0$, то трябва да бъде увеличен някой от индексите, а това може да бъде само индексът j.

Анализ на алгоритъма: В най-лошия случай (т.е. когато масивът не съдържа тройка елементи с нулева сума) тялото на вътрешния цикъл се изпълнява $n-i-1$ пъти, защото при влизане в цикъла изразът $k-j$ има стойност $n-i-1$, а след всяка итерация стойността на този израз намалява с единица. Като сумираме по $i = 1, 2, 3, \ldots, n-2$, получаваме, че броят на всички изпълнения на най-вътрешните оператори е равен на

$$\sum_{i=1}^{n-2}(n-i-1) = (n-2)+(n-3)+(n-4)+\ldots+3+2+1 = \frac{(n-2)(n-1)}{2} = \Theta(n^2).$$

Това е времевата сложност на алгоритъма (в най-лошия случай).

П р и м е р : Сортираният масив се състои от числата $-6, -4, 0, 1, 3, 7$.
$(-6) + (-4) + 7 < 0$, затова увеличаваме второто число;
$(-6) + 0 + 7 > 0$, затова намаляваме третото число;
$(-6) + 0 + 3 < 0$, затова увеличаваме второто число;
$(-6) + 1 + 3 < 0$, затова увеличаваме второто число;
край на опитите с първото число (т.е. -6);
$(-4) + 0 + 7 > 0$, затова намаляваме третото число;
$(-4) + 0 + 3 < 0$, затова увеличаваме второто число;
$(-4) + 1 + 3 = 0$, намерена е тройка числа с нулева сума.

Ако искаме да преброим тройките с нулева сума, това може да стане с малка промяна на последния алгоритъм: като намерим една такава тройка, увеличаваме някакъв брояч с единица и продължаваме търсенето на още тройки. Дали да увеличим индекса j, или да намалим индекса k? Ясно е, че ако увеличим j, ще получим положителна сума, така че на следващата стъпка ще трябва да намалим k. Обратно, ако намалим индекса k, ще получим отрицателна сума, така че на следващата стъпка ще трябва да увеличим j. Затова най-добре е едновременно да увеличим j и да намалим k.

```
cnt3zero(A[1...n]: array of integers): integer
sort(A)
cnt ← 0
for i ← 1 to n-2
  j ← i+1
  k ← n
  while k > j
    if A[i]+A[j]+A[k] = 0
      print i,j,k
      cnt ← cnt + 1
      j ← j+1
      k ← k-1
    else if A[i]+A[j]+A[k] > 0
      k ← k-1
    else // A[i]+A[j]+A[k] < 0
      j ← j+1
return cnt
```

Този алгоритъм има времева сложност също $\Theta(n^2)$, но важи само ако масивът не съдържа еднакви елементи. В противен случай може да пропусне някои тройки с нулева сума (а именно: тези, които се получават от преди това намерена тройка с изменение само на единия индекс j или k). Такива тройки може да има много. Например, ако една трета от числата са равни на -6, една трета са равни на +1 и една трета са равни на +5, то броят на всички тройки с нулева сума е равен на $(n/3)^3$, т.е. всеки алгоритъм, който печата тройките, ще има сложност $\Omega(n^3)$, т.е. няма как да е квадратичен. Ако обаче се откажем от печатането на тройките, а оставим само изискването за тяхното преброяване, то последният алгоритъм може да се преработи така, че да се запази квадратичното време. За целта е достатъчно да бъдат изтрити всички повторения на елементи, а броят на повторенията да се запази в друг масив; тогава при всяко срещане на тройка с нулева сума броячът `cnt` ще расте не с единица, а с произведението на броя на повторенията.

```
cnt3zero(X[1...n]: array of integers): integer
sort(X)
A[1...n]: array of integers // X без повторения
B[1...n]: array of integers // брой на повторенията
m ← 1
A[1] ← X[1]
B[1] ← 1
for p ← 2 to n
  if X[p] = X[p-1]
    B[m] ← B[m]+1
  else
    m ← m+1
    A[m] ← X[p]
    B[m] ← 1
cnt ← 0
for i ← 1 to m-2
  j ← i+1
  k ← m
  while k > j
    if A[i]+A[j]+A[k] = 0
      cnt ← cnt+B[i]×B[j]×B[k]
      j ← j+1
      k ← k-1
    else if A[i]+A[j]+A[k] > 0
      k ← k-1
    else // A[i]+A[j]+A[k] < 0
      j ← j+1
return cnt
```

Този алгоритъм, както казахме, също има квадратична времева сложност.

Като задача за разпознаване (т.е. има или няма тройка с нулева сума) проблемът е интересен с това, че все още няма пълно решение. Известен под името 3SUM, проблемът привлича вниманието на изследователите от години, още повече че редица други проблеми (в т.ч. геометрични) зависят от него. За задачата 3SUM има алгоритъм с времева сложност $O\left(n^2.\frac{(\log\log n)^2}{\log n}\right)$, т.е. малко по-бърз от квадратичен. Но не е доказано, че няма още по-бърз алгоритъм.

7) Повтарящ се елемент в масив.

Даден е масив с n цели числа, всяко от които е между 1 и n включително. Съставете алгоритъм, който проверява дали има повтарящо се число. Имате право да променяте елементите на масива. Каква най-добра сложност по време и по памет можете да постигнете?

Решение: Едно очевидно решение е да сортираме числата и да обходим масива, проверявайки дали има равни съседни елементи.

```
hasDuplicates(A[1...n]: array of integers): bool
sort(A)
for k ← 2 to n
  if A[k] = A[k-1]
    return true
return false
```

Това решение е със сложност $\Theta(n \cdot \log n)$, а като допълнителна памет ползва константен брой променливи от примитивен тип.

Друго решение е да заделим един нов масив с n елемента и да обходим дадените числа, като отчитаме кои стойности от 1 до n сме срещнали. Ако сме срещнали всяка стойност, то всяка се среща по веднъж (защото масивът A има само n елемента). Ако пък някоя стойност от 1 до n липсва, то остават $n - 1$ различни стойности, следователно някоя стойност се повтаря.

```
hasDuplicates(A[1...n]: array of integers): bool
C[1...n]: array of bool
for k ← 1 to n
  C[k] ← false
for k ← 1 to n
  C[A[k]] ← true
for k ← 1 to n
  if C[k] = false
    return true
return false
```

Този подход е модификация на алгоритъма “сортиране чрез броене” (CountingSort). Времевата сложност е $\Theta(n)$, което е по-бързо от предишното решение; но сега използваме и допълнителна памет с количество $\Theta(n)$. За сметка на това не променяме оригиналния масив.

Най-доброто решение работи в линейно време $\Theta(n)$ и използва константен брой допълнителни променливи от примитивен тип (например броячи), като съществено се възползва от възможността да променя входния масив. Той може да побере информацията от масива C; един допълнителен бит може да се пази в знака на всяко от числата. Оригиналните данни са положителни цели числа и ще съответстват на стойност false, а ако някоя от тях стане отрицателна, ще съответства на стойност true.

```
hasDuplicates(A[1...n]: array of integers): bool
for k ← 1 to n
  A[abs(A[k])] ← -abs(A[abs(A[k])])
for k ← 1 to n
  if A[k] > 0
    return true
return false
```

Заб. Сложността по памет обхваща само допълнителната памет, но не и паметта за входните данни. Тоест масивът A не се брои при пресмятане на сложността по памет.

8) Тек сред чифтове.

Даден е масив с n цели числа, където n е нечетно. Всяко число се среща по два пъти с изключение на едно, което е уникално. Намерете това число.

Решение: Тривиалното решение е с помощта на два вложени цикъла да проверим кои числа се повтарят. Времевата сложност е $\Theta(n^2)$.

По-добро решение е да сортираме масива и после да сравняваме само съседните елементи. Времевата сложност е $\Theta(n \cdot \log n) + \Theta(n) = \Theta(n \cdot \log n)$.

Най-доброто решение работи в линейно време $\Theta(n)$ и използва константен брой допълнителни променливи от примитивен тип (напр. броячи):

```
uniqueElement(A[1...n]: array of integers): integer
result ← A[1]
for k ← 2 to n
  result ← result xor A[k]
return result
```

Това решение се основава на свойствата на изключващата дизюнкция (xor), която в случая се прилага побитово. А именно:
0 xor 0 = 0, 0 xor 1 = 1, 1 xor 0 = 1, 1 xor 1 = 0.
Операцията е асоциативна и комутативна, поради което може да се приложи и към повече от два аргумента. В този случай резултатът е 1, ако нечетен брой от аргументите са единици, 0 — в противен случай. Затова, ако всички стойности без една се повтарят, резултатът е тъкмо уникалната стойност.

9) Два тека сред чифтове.

Даден е масив с n цели числа, където n е четно. Всяко число се среща по два пъти с изключение на две, които са уникални и различни помежду си. Намерете тези две числа.

Решение: Тази задача е променен вариант на предходната, затова ще се опитаме да намерим подобно решение. Пресмятаме побитовата изключваща дизюнкция (xor) на всички числа в масива. Чифтовете се унищожават, затова резултатът е равен на x xor y, където x и y са двата тека. Понеже $x \neq y$, то x xor $y \neq 0$, т.е. поне една от двоичните цифри на x xor y е различна от 0. Следователно тази цифра има различна стойност в тековете (напр. 0 в x, 1 в y). Числата от масива се разделят на две групи според тази си цифра: подмасив X от числата, при които тази цифра е 0, и подмасив Y от числата, при които тази цифра е 1. Всеки от масивите X и Y съдържа точно един от тековете, а всички други елементи (на кой да е от подмасивите) се групират в чифтове. За всеки от подмасивите X и Y решаваме предходната задача и намираме всеки от двата тека: x — от X, y — от Y.

Заб. Не е нужно да образуваме два нови масива X и Y. Вместо това обхождаме оригиналния масив още два пъти, но в съответната изключваща дизюнкция участват само онези числа, чиято двоична цифра е 0, респ. 1.

Предложеният алгоритъм извършва три обхождания на масива, като използва константен брой допълнителни променливи от примитивен тип (x, y и брояч). Сложността по време е линейна: $\Theta(n)$.

10) Точно едно повтарящо се число.

Дадени са n числа от 1 до n–1 включително. Всички числа са различни освен едно, което се повтаря. Намерете кое е то.

Решение: Събираме дадените числа и от получения сбор вадим сумата на числата от 1 до n–1, т.е. вадим $(n-1) \cdot n / 2$. Разликата е тъкмо числото, което се повтаря. Алгоритъмът е линеен, т.е. работи във време $\Theta(n)$.

11) Точно едно липсващо число.

Дадени са n различни числа в диапазона от 1 до n+1 включително. Намерете липсващото число от диапазона.

Решение: Същата идея води отново до линеен алгоритъм. Събираме дадените числа и вадим получения сбор от сумата на числата от 1 до n+1, тоест от $(n+1) \cdot (n+2) / 2$. Разликата е тъкмо липсващото число.

12) Първо липсващо число.

Даден е масив от n цели положителни числа, които могат да са много големи. Кое е първото цяло положително число, липсващо в масива?

Решение: Очевидно отговорът е някое от числата 1, 2, 3, ... , n+1.

Тривиалният алгоритъм проверява първо дали масивът съдържа 1, после 2 и т.н. до n. Всяка проверка изисква линейно време. В най-лошия случай (когато масивът съдържа всички числа от 1 до n) се правят n проверки. Следователно времевата сложност на този алгоритъм е $\Theta(n^2)$.

Второ възможно решение е да сортираме масива за време $\Theta(n \cdot \log n)$, а после да търсим първото липсващо число за линейно време:

```
firstMissing(A[1...n]: array of integers): integer
sort(A)
if A[1] > 1
  return 1
for k ← 1 to n-1
  if A[k+1] - A[k] > 1
    return A[k] + 1
return A[n] + 1
```

Времевата сложност на този алгоритъм е $\Theta(n \cdot \log n) + \Theta(n) = \Theta(n \cdot \log n)$, което е по-добре от предишния алгоритъм, но все още не е най-бързо.

Тъй като първото липсващо число е от 1 до n+1, можем да приложим *модификация* на CountingSort, работеща в линейно време. (Оригиналният вариант е неприложим, понеже числата в масива могат да са много големи.)

```
firstMissing(A[1...n]: array of integers): integer
C[1...n]: array of bool
for k ← 1 to n
  C[k] ← false
for k ← 1 to n
  if A[k] ≤ n
    C[A[k]] ← true
for k ← 1 to n
  if C[k] = false
    return k
return n+1
```

Това решение очевидно работи в линейно време $\Theta(n)$ и ползва допълнителна памет (за масива C) в размер $\Theta(n)$, при това без да променя оригиналния масив. Ако такава промяна е допустима, то можем да спестим паметта за масива C и да сведем допълнителната памет до константен брой променливи от примитивен тип (напр. броячи).

```
firstMissing(A[1...n]: array of integers): integer
for k ← 1 to n
  if abs(A[k]) ≤ n
    A[abs(A[k])] ← -abs(A[abs(A[k])])
for k ← 1 to n
  if A[k] > 0
    return k
return n+1
```

Този алгоритъм се възползва от възможността да променя оригиналния масив и от факта, че всички оригинални стойности (т.е. входните данни) са цели положителни числа. Знакът на всяко число замества допълнителния бит информация и масивът C става излишен.

13) Липсващо число в сортиран масив.

Даден е сортиран масив с n числа от 1 до n+1 включително, без повторения. Как да намерим липсващото число?

Решение: Тази задача е подобна на зад. 11 и може да бъде решена по същия начин за линейно време. Сега обаче имаме допълнителното условие, че числата в масива са подредени в нарастващ ред. Това позволява да ускорим алгоритъма с помощта на двоично търсене, което работи в логаритмично време $\Theta(\log n)$, не променя оригиналния масив и ползва константен брой допълнителни променлви от примитивен тип.

```
missingInSorted(A[1...n]: array of integers): integer
if A[1] > 1
  return 1
if A[n] < n+1
  return n+1
left ← 1
right ← n
while right > left
  middle ← ⌊(left + right) / 2⌋
  if A[middle] = middle
    left ← middle+1
  else
    right ← middle
return right
```

14) Произведения на останалите числа.

Даден е масив с n числа. Търси се нов масив с n числа, k-тото от които е равно на произведението на всички числа от входния масив без k-тото ($k = 1, 2, 3, \ldots, n$). Съставете алгоритъм, който не използва деление (това може да се наложи например при умножение по даден модул).

Решение: Алгоритъмът, който за всяко число пресмята произведението на другите, е с квадратична времева сложност. Постижима е линейна сложност:

```
 Input: A[1...n]: array of numbers
Output: B[1...n]: array of numbers
B[n] ← 1
for k ← n downto 2
  B[k-1] ← B[k] × A[k]
FromLeft ← A[1]
for k ← 2 to n
  B[k] ← B[k] × FromLeft
  FromLeft ← A[k] × FromLeft
```

Идеята на алгоритъма е следната: в масива B се записват произведенията на числата на масива A отдясно наляво, т.е. $B[k] = A[k+1] \,.\, A[k+2] \ldots A[n]$; после в променливата *FromLeft* се пресмятат последователно произведенията на числата на масива A отляво надясно, т.е. $FromLeft = A[1] \,.\, A[2] \ldots A[k-1]$; всяко произведение от втория вид се умножава със съответното произведение от първия вид; затова $B[k] = A[1] \,.\, A[2] \ldots A[k-1] \,.\, A[k+1] \,.\, A[k+2] \ldots A[n]$.

Сложността на алгоритъма и по време, и по допълнителна памет е $\Theta(n)$. Сложността по време очевидно е оптимална по порядък, тъй като за по-малко време не е възможно дори да се прочетат входните данни.

15) Чупливи ел. крушки.

Имаме електрически крушки, които се чупят, ако паднат от определена височина (една и съща за всички крушки). Имаме и една 100-етажна сграда. Как да намерим с минимален брой опити от кой етаж се чупят крушките?

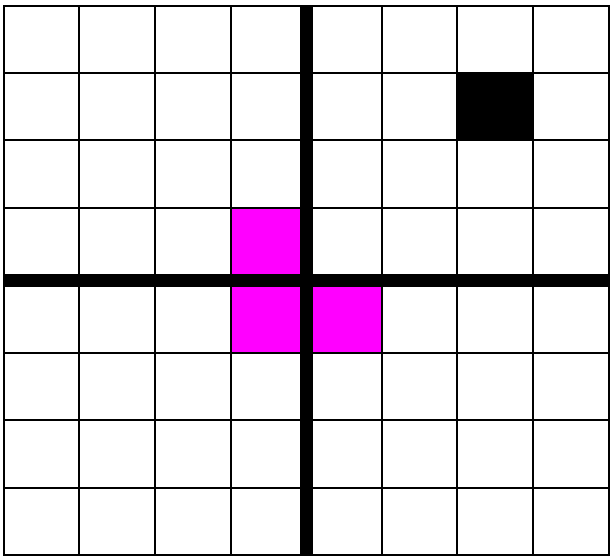
Решение: Тъй като искаме да минимизираме броя опити, а не броя счупени ел. крушки, оптималната стратегия е да приложим *двоично търсене*. Тоест пускаме крушка от 50-ия етаж. Ако тя се счупи, пускаме втората крушка от по-нисък етаж — 25-ия. Ако първата крушка не се счупи, то пускаме втората от по-висок етаж — от 75-ия и т.н. Така са нужни само $\lceil \log_2 100 \rceil = 7$ опита.

Ако се искаше минимален брой счупени крушки, тогава отговорът е 1 (една счупена крушка) и се постига с помощта на *последователно търсене*: пускаме крушката от първия, втория, третия етаж и т.н., докато се счупи.

16) Тримино.

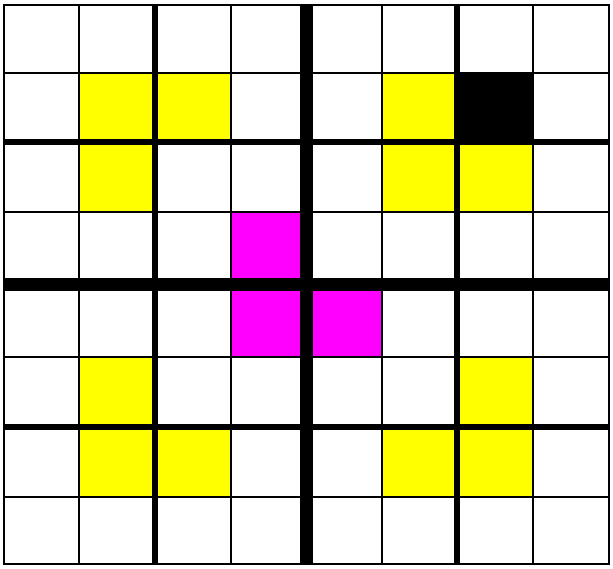
Даден е квадрат със страна 2^n, разделен на малки квадратчета със страна 1. Едно от тях е оцветено в черно, всички останали в бяло. На всеки ход можем да поставим тримино успоредно на страните на квадрата, стига то да попадне само върху бели квадратчета. (Тримино е плочка от три квадратчета под формата на буквата Г, тоест квадрат 2×2, на който липсва една клетка). Съставете алгоритъм за покриване на целия квадрат (без черното квадратче) с тримино, които не се застъпват и не стърчат извън квадрата.

Решение: Задачата се решава чрез *рекурсия*. Като метод за програмиране рекурсията означава една подпрограма да вика сама себе си (*пряка рекурсия*) или няколко подпрограми да се викат взаимно (*косвена рекурсия*). Обикновено при рекурсивното извикване се решава по-малък вариант на същата задача.



първа стъпка

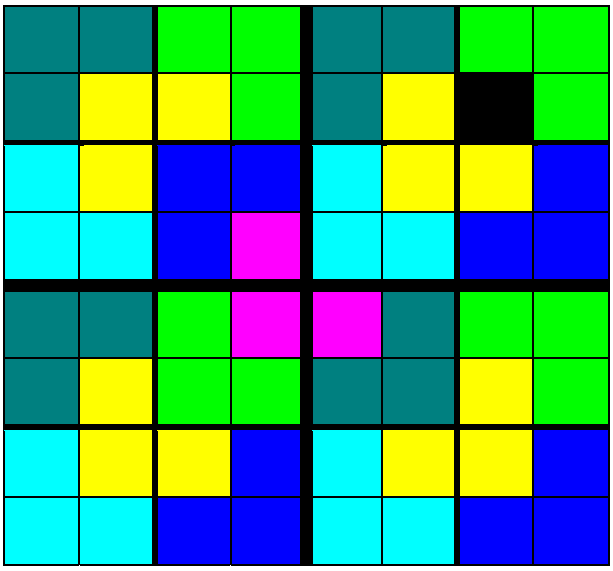
Конкретната задача се решава тъй: разделяме големия квадрат на четири по-малки квадрата със страни 2^{n-1}, слагаме първото тримино в центъра така, че да закрива по едно квадратче от трите квадрата без липсващо поле. Сега всеки от четирите квадрата има по едно липсващо поле (черното или някое от трите розови полета). Остава да покрием с тримино всеки от четирите квадрата, а това е същата задача, но вече за $n - 1$ вместо n.



втора стъпка

На втората стъпка, прилагайки същата стратегия към получените четири квадрата, добавяме по едно тримино в центъра на всеки от тях (това са жълтите клетки). Делим всеки от четирите малки квадрата на четири още по-малки.

Продължаваме по този начин, докато стигнем до квадрати 2×2 (всеки с по едно липсващо поле). Другите три полета се запълват с по едно тримино.



последна (трета) стъпка

В конкретния пример задачата се решава за три стъпки. Фигурата вляво показва крайния отговор. Онези тримино, които са сложени на последната (третата) стъпка, са оцветени в оттенъци на зеления и синия цвят.

17) Застъпващи се мероприятия.

Дадено е множество от мероприятия, всяко от които се характеризира с начален час и краен час. Съставете алгоритъм, който избира възможно най-голям брой мероприятия, така че никои две от тях да не се застъпват (допуска се краят на едното да съвпада с началото на следващото).

Решение: Задачата се решава с помощта на *алчен алгоритъм*, т.е. такъв, който на всяка стъпка избира действието, което е най-добро за момента, без да се интересува от възможна по-голяма изгода на следващите стъпки. Алчните алгоритми обикновено са бързи, но не винаги водят до най-доброто решение. Обаче в тази задача действително се получава желаният максимум от мероприятия. Все пак е нужно да съставим алгоритъма внимателно, иначе той ще се окаже некоректен.

П р и м е р : Нека мероприятията се провеждат в следните времеви интервали: [0 ; 13], [2 ; 5], [4 ; 6], [5 ; 9], [8 ; 14], [11 ; 12], [15 ; 20]. Алчната стратегия се състои в това, да вземем първото възможно мероприятие (за да можем след него да вземем максимален брой). Трябва да се уточни обаче в какъв ред разглеждаме мероприятията, т.е. най-напред трябва да извършим подходящо *сортиране*.

Ако ги взимаме в хронологичен ред по началния час, няма да се получи оптимален резултат. В конкретния пример този алгоритъм води до следното множество: [0 ; 13], [15 ; 20], тоест общо две мероприятия. Това не е най-големият възможен брой.

Ако сортираме мероприятията по дължина (т.е. по разликата на крайния и началния час) и взимаме първо най-късите, ще получим следното множество: [11 ; 12], [4 ; 6], [15 ; 20], т.е. три мероприятия, което пак не е оптимално.

Правилно е да сортираме мероприятията по техния край и после вече да прилагаме алчна стратегия (т.е. всеки път да добавяме към нашия списък първото мероприятие, което не се застъпва с никое от досега избраните).

В конкретния пример резултатът е $[2\ ;\ 5]$, $[5\ ;\ 9]$, $[11\ ;\ 12]$, $[15\ ;\ 20]$, т.е. четири мероприятия, което е максималният възможен брой.

Коректност на алгоритъма: Първо ще докажем, че най-рано завършващото мероприятие непременно е в поне едно от оптималните решения. Избираме произволно оптимално множество. Нека първото (по краен час) мероприятие в това множество не е първото от всички мероприятия. Тогава заменяме първото мероприятие на множеството с най-рано завършващото от всички мероприятия. След тази замяна крайният час на първото мероприятие на множеството ще стане по-ранен, значи това мероприятие ще продължава да не се застъпва с останалите мероприятия в множеството (които също не се застъпват едно с друго). Следователно новото множество пак е оптимално, защото то съдържа същия (максималния) брой две по две незастъпващи се мероприятия. Обаче новото оптимално множество вече съдържа най-рано завършващото от всички мероприятия.

Ето защо няма да сбъркаме, ако, конструирайки оптимално множество стъпка по стъпка, вземем най-рано завършващото мероприятие на първата стъпка от алгоритъма. В такъв случай всички други мероприятия, които ще вземем, трябва или да завършват преди началото, или да завършват след края на най-рано завършващото мероприятие (за да не се застъпват с него). Обаче първият вариант е невъзможен: иначе най-рано завършващото мероприятие не би било такова. Остава другият вариант. Затова отсяваме оставащите мероприятия, като премахваме всички, започващи преди края на най-рано завършващото мероприятие. От останалите трябва да вземем колкото може повече, т.е. решаваме същата задача за останалите мероприятия. По същата логика пак няма да сбъркаме (т.е. ще стигнем до оптимално решение), ако вземем най-рано завършващото от останалите мероприятия и т.н.

Този алчен алгоритъм изисква време $\Theta(n \,.\, \log n)$ заради сортирането, където n е броят на мероприятията. (Вторият етап — построяването на максимално множество от мероприятия, след като масивът е сортиран — се изпълнява за линейно време.)

18) Мажоранта.

Мажоранта на масив (на списък или на мултимножество) се нарича елемент, който се среща повече от половината пъти (такъв елемент може и да няма). Даден е масив с n елемента, за който знаем, че има мажоранта. Съставете бърз алгоритъм, който намира мажорантата на масива.

Решение: Тривиалният алгоритъм (този, който проверява всеки елемент дали е мажоранта) има квадратична сложност по време.

По-бърз алгоритъм се получава чрез следните две твърдения.

Лема 1: Ако едно мултимножество има мажоранта M и бъде разделено на две мултимножества, то M ще бъде мажоранта на поне едно от тях.

Доказателство: Да допуснем противното: че M не е мажоранта на никое от двете мултимножества. Тогава не повече от половината елементи на всяко от тях са равни на M. Следователно не повече от половината елементи на цялото мултимножество са равни на M, т.е. M не е негова мажоранта. Противоречие.

Лема 2: Ако едно мултимножество има мажоранта M и бъде разделено на две мултимножества, едното от които няма мажоранта, то M ще бъде мажоранта на другото мултимножество.

Доказателство: От лема 1 следва, че M е мажоранта на поне едно от двете мултимножества. Щом това не е първото от тях, значи е второто.

Въз основа на лема 2 се получава следният алгоритъм:

```
majorant(A[1...n]: array): element
M ← A[1]
cnt ← 1
k ← 2
while k ≤ n
  if A[k] = M
    cnt ← cnt + 1
    k ← k + 1
  else
    cnt ← cnt - 1
    k ← k + 1
    if cnt = 0
      M ← A[k]
      cnt ← 1
      k ← k + 1
return M
```

Идеята на алгоритъма е следната: Допускаме, че първият елемент на масива е негова мажоранта и броим колко пъти се среща този елемент сред първите k елемента, като k се увеличава от 1 до n. Ако първият елемент "издържи", т.е. остане мажоранта, докато k надхвърли n, тогава той е отговорът. Иначе при някое (четно) k се получава, че $A[1]$ не е мажоранта на $A[1...k]$, тъй като се среща в него точно в 50% от случаите. Значи и другите елементи на $A[1...k]$ не са негови мажоранти (те се срещат не повече от половината пъти). Тоест $A[1...k]$ няма мажоранта. От лема 2 следва, че $A[k+1...n]$ има мажоранта и тя е мажоранта и на $A[1...n]$, затова алгоритъмът започва да изследва $A[k+1...n]$.

19) Подниз, съставен от най-много *k* различни символа.

Даден е низ с дължина n. Намерете най-дългия подниз, сред чиито символи различни са най-много k.

П р и м е р : В низа “aababbaacacaacacccaabad” един възможен низ, съдържащ най-много два различни символа, е “aababbaa”, започващ от началото на дадения низ. Най-дългият такъв подниз е “aacacaacacccaa”, започващ от седмата позиция.

Решение: Един алгоритъм е за всяка възможна позиция в низа да намерим най-дългия подниз с не повече от k различни символа, започващ от нея. Този алгоритъм има времева сложност $\Theta\left(n^2\right)$.

По-бърз, линеен алгоритъм може да се получи с помощта на една техника, известна като *метод на плъзгащия се прозорец*. Следва алгоритъмът, описан чрез код на C:

```
// Връща началото на подниза с максимална дължина.
// Индексите започват от 0.
int LongestSubstring(const char* str, int n, int K) {
  int best = 0, startAt = 0;
  int cnt[256] = {0}, in = 0;
  for (int left = 0, right = 0; right < n; right++) {
    // Вкарваме символа в прозореца
    // и ако е нов, увеличаваме броя различни символи.
    if (cnt[str[right]]++ == 0) in++;
    // Ако има поне K+1 различни символа в прозореца,
    // скъсяваме го отляво, докато останат само K.
    while (in > K) {
      if (--cnt[str[left++]] == 0) in--;
    }
    // Ако прозорецът е по-дълъг от най-дългия досега,
    // актуализираме отговора.
    if (best < right - left + 1) {
      best = right - left + 1;
      startAt = left;
    }
  }
  return startAt;
}
```

Този алгоритъм има времева сложност $\Theta(n)$, защото на всяка стъпка една от променливите `left` или `right` се увеличава с 1, така че сборът им расте от 0 до $2n-2$.

20) Кръстословица.

		Я		
	Я	Г	О	Д
Я	Г	О	Д	А
		Д	А	
		А		

По колко начина можем да прочетем думата “ягода”, ако от всяка клетка отиваме или в клетката под нея, или в клетката отдясно?

Решение: В случая думата е достатъчно кратка и таблото е достатъчно малко, така че можем да намерим отговора и с пълно изчерпване. Този метод обаче не е подходящ при голямо игрално табло, защото тогава броят на възможните начини нараства до големи стойности. Вместо това се използва една техника, наречена *динамично програмиране*, чиято основна идея е да се решат всички по-малки задачи от дадената и техните решения да се запомнят (обикновено в таблица, но понякога в друга структура от данни). В конкретния пример можем да използваме дадената таблица. Попълваме я с числа по следния начин:

Във всяка клетка записваме по едно число, което показва по колко начина можем да прочетем края на думата, започвайки от тази клетка, т.е. по колко начина можем да стигнем от тази клетка до някоя буква “А”, като се движим само надолу и надясно. Самата таблица попълваме отзад напред. Първо пишем единици в клетките с буква “А”, защото от тях има само един начин да прочетем края на думата, т.е. буквата “А”. След това попълваме клетките с буква “Д”: във всяка такава клетка пишем сбора от двете числа в клетките отдясно и отдолу. По същия начин попълваме клетките с “О”, “Г” и “Я”. Например числото 4 в една от клетките с буквата “О” означава, че краят на думата, т.е. “ОДА”, може да бъде прочетен по 4 начина, ако започнем от това “О”.

		Я 7		
	Я 11	Г 7	О 3	Д 1
Я 4	Г 4	О 4	Д 2	А 1
		Д 2	А 1	
		А 1		

Накрая събираме числата от клетките с буквата “Я” и получаваме 4 + 11 + 7 = 22, което е търсеният отговор.

21) Винарска изба (втори вариант).

Внучка наследява от баба си винарска изба. В нея има n бутилки вино, наредени в редица и номерирани от 1 до n. Техните начални цени са цели неотрицателни числа, дадени в масива $P[1...n]$. Колкото по-дълго отлежават бутилките, толкова по-скъпи стават: цената на бутилка № k след x години ще бъде $x \,.\, P[k]$.

В завещанието си бабата е поискала всяка година внучката да продава по една бутилка, като избира или най-лявата, или най-дясната останала. Каква е максималната парична сума, която внучката може да спечели и в какъв ред трябва да продава бутилките? Считаме, че бутилките са отлежали една година, когато се продава първата от тях.

П р и м е р : Ако имаме четири бутилки с цени 10, 40, 20 и 30 долара, то оптималният начин да ги продадем е следният:
на първата година продаваме първата бутилка за 1 . 10 = 10 долара;
на втората година продаваме четвъртата бутилка за 2 . 30 = 60 долара;
на третата година продаваме третата бутилка за 3 . 20 = 60 долара;
на четвъртата година продаваме втората бутилка за 4 . 40 = 160 долара;
общо: 10 + 60 + 60 + 160 = 290 долара.

Решение: Най-естественото решение е да продадем скъпите бутилки колкото може по-късно, т.е. от двете крайни бутилки винаги да продаваме първо по-евтината. За съжаление, тази стратегия е погрешна.

К о н т р а п р и м е р : Ако имаме пет бутилки с цени 20, 30, 50, 10 и 40 долара, то описаната стратегия води до следната редица от действия:
на първата година продаваме първата бутилка за 1 . 20 = 20 долара;
на втората година продаваме втората бутилка за 2 . 30 = 60 долара;
на третата година продаваме петата бутилка за 3 . 40 = 120 долара;
на четвъртата година продаваме четвъртата бутилка за 4 . 10 = 40 долара;
на петата година продаваме третата бутилка за 5 . 50 = 250 долара;
общо: 20 + 60 + 120 + 40 + 250 = 490 долара.

В действителност оптималният начин за продажба е следният:
на първата година продаваме първата бутилка за 1 . 20 = 20 долара;
на втората година продаваме петата бутилка за 2 . 40 = 80 долара;
на третата година продаваме четвъртата бутилка за 3 . 10 = 30 долара;
на четвъртата година продаваме втората бутилка за 4 . 30 = 120 долара;
на петата година продаваме третата бутилка за 5 . 50 = 250 долара;
общо: 20 + 80 + 30 + 120 + 250 = 500 долара.

Задачата се решава чрез динамично програмиране. Този метод е приложим в два вида задачи — комбинаторни и оптимизационни. Предишната задача е комбинаторна (в нея се пита “колко”). Настоящата задача е оптимизационна (търси се максимална стойност). Когато се прилага в оптимизационни задачи, динамичното програмиране се нарича още *динамично оптимиране*.

Решаваме всички задачи, по-малки от дадената, т.е. намираме отговорите на всички въпроси от вида “Колко най-много може да спечели внучката, ако разполага само с бутилките от № *left* до № *right*, където $1 \le left \le right \le n$? ”

```
maxWine(P[1...n]: array of integers): integer
dyn[1...n, 1...n]: array of integers; // max печалба
taken[1...n, 1...n]: array of boolean; // true <=> left
for left ← 1 to n
  right ← left
  year ← n
  dyn[left,right] ← year × P[left]
  taken[left,right] ← true
for year ← n-1 downto 1
  for left ← 1 to year
    right ← left + n - year
    winL ← year × P[left]  + dyn[left+1,right]
    winR ← year × P[right] + dyn[left,right-1]
    if winL > winR // взимаме лявата бутилка
      dyn[left,right] ← winL
      taken[left,right] ← true
    else // взимаме дясната бутилка
      dyn[left,right] ← winR
      taken[left,right] ← false
left ← 1
right ← n
while left ≤ right
  if taken[left,right]
    print "left"
    left ← left + 1
  else
    print "right"
    right ← right - 1
return dyn[1,n]
```

Сложността на алгоритъма по време и по памет е $\Theta\left(n^2\right)$, колкото е размерът на таблицата. Това е много по-бързо от пълното изчерпване, чиято сложност по време е $\Theta\left(2^n\right)$: тъй като на всяка стъпка имаме две възможности (можем да продадем или лявата, или дясната бутилка), то всички варианти са 2^{n-1}.

Програмистки трикове

Тези задачи изискват добро познаване на езиците за програмиране, напр. C.

1) Точна степен на двойката.

Дадено е естествено число *X*. Съставете възможно най-прост оператор `if()` (тоест с възможно най-малко операции в него), чрез който да проверите дали числото е точна степен на 2. Не са разрешени цикли. Реализацията

`if ((X==1) || (X==2) || (X==4) || (X==8) || ...)`

е много дълга и зависи от размера на типа int (в истински код многоточието липсва). Търси се по-кратко решение, независещо от размера на типа int.

Отговор: `if (X && ! (X & (X-1)))`

2) Най-младши ненулев бит.

Намерете най-младшия ненулев бит на дадено цяло число; по-точно: числото, което този бит представлява (съответната степен на двойката).

П р и м е р и : от числото 42 (101010) трябва да получите 2 (000010), а от 88 (1011000) трябва да получите 8 (0001000).

Отговор: `X - (X & (X-1))`

3) Нула или едно.

Дадено е цяло число *X*. Съставете израз, съдържащ само *X*, побитови операции и скоби, който има стойност `true` само ако *X* е нула или единица. Побитови операции са `~` , `|` , `&` , `^` , `!` , `<<` и `>>` .

Отговор: `!(X ^ !!X)` или `!(X >> 1)`

4) Умножение по седем.

Покажете начин за целочислено умножение по седем с помощта само на събиране, изваждане и побитови операции.

Отговор: `(X << 3) - X`

5) Препълване.

Напишете логически израз, който проверява дали се получава препълване при събиране на две дадени цели числа без знак.

Отговор: `if (A + B < A)`

6) Максимум.

Съставете израз, чиято стойност е по-голямото от две дадени цели числа, без да използвате разклонение, т.е. без операцията "`?:`".

Отговор: `(A >= B) * A + (A < B) * B`

7) Битова маска.

Битовата маска е цяло число, интерпретирано като множество, а именно — множеството от позициите на битовете със стойност 1, номерирани от най-младшия към най-старшия (номерацията може да започва от 0 или от 1). Съставете функция, която отпечатва всички подмножества на множество, представено с помощта на битова маска.

П р и м е р : Маската 10000011 представя множеството { 0 , 1 , 2 , 7 }, ако номерацията започва от 0, или { 1 , 2 , 3 , 8 }, ако започва от 1. Функцията трябва да отпечата числата 0, 1, 2, 3, 128, 129, 130 и 131, т.е. 00000000, 00000001, 00000010, 00000011, 10000000, 10000001, 10000010 и 10000011, в някакъв ред.

Решение:

```
void iterateThroughSubsetsOfBitMask(unsigned X) {
  for (unsigned Y = X; Y > 0; Y = ((Y - 1) & X))
    printf("%u\n", Y);
  printf("0\n");
}
```

8) Разменяне на стойности.

Разменете стойностите на две числа, без да използвате трета променлива.

Решение:

```
A = A+B;
B = A-B;
A = A-B;
```

9) Линейни програми.

Според концепцията за *структурно програмиране* всяка програма е съвкупност от оператори (команди), организирани с помощта на структурите последователност, разклонение и повторение. В програмния език C разклонение се създава чрез операторите `if` и `switch` и операцията “`?:`”, а повторение — чрез операторите `for` и `while`. Структурното програмиране не предвижда възможност за друг вид преходи, т.е. в него не се използва операторът `goto`.

Една (под)програма се нарича линейна, ако представлява последователност от оператори, т.е. няма разклонения и повторения. Линейните програми са най-простите програми. В зависимост от възможностите на програмния език линейните програми понякога имат доста сложно поведение и дори могат да заместват разклонения и цикли.

Във всяко от подусловията се иска съставянето на линеен програмен код, по резултата си равносилен на нелинейния код между редовете с многоточие.

а) Заместване на разклонение.

```
int a;
int n;
.....................................
switch(n%3) {
  case 0: printf("%d\n",a); break;
  case 1: a += 4; break;
  case 2: n--; break;
}
.....................................
```

Решение: Ролята на разклонението може да бъде изпълнявана от масив, който съдържа *указатели към подпрограми*. Подобен механизъм се използва в *обектноориентираното програмиране*: за всеки клас се пази таблица на *виртуалните методи*, така че методът да може да бъде намерен в зависимост не само от своето име, но и от действителния, а не от декларирания тип на обектния екземпляр.

В конкретния случай тази идея може да бъде осъществена така:

```
int a;
int n;

void PrintA() {
  printf("%d\n",a);
}

void IncA4() {
  a += 4;
}

void DecN() {
  n--;
}
.....................................
typedef void proc();
proc* SubProg[3];
SubProg[0] = PrintA;
SubProg[1] = IncA4;
SubProg[2] = DecN;
SubProg[n%3];
.....................................
```

б) Заместване на цикъл.

```
int n;
.....................................
for (int k = 1; k <= n; k++)
  printf("%d\n",k);
.....................................
```

Решение: *Итерацията* може да бъде заместена от *рекурсия*. Този подход се използва в *логическото и функционалното програмиране*. (И обратно, всяка рекурсивна програма може да бъде заменена с итеративна. За целта програмистът трябва да организира обработката на собствен стек, вместо да разчита на автоматичната обработка на програмния стек.)

```
int n;

typedef void proc(int k);
proc* IterFunc[2];

void ProcessNum(int k) {
  printf("%d\n",k);
  k++;
  proc* Iter = IterFunc[k <= n];
  Iter(k);
}

void DoNothing(int k) {
}

.....................................
IterFunc[0] = DoNothing;
IterFunc[1] = ProcessNum;
int k = 1;
proc* Iter = IterFunc[k <= n];
Iter(k);
.....................................
```

Това решение използва и двете техники:
— рекурсия (за премахване на цикъла);
— масив от функции (за премахване на разклонението, което се появява от проверката за дъно на рекурсията, т.е. края на цикъла).

Неопределени задачи

Това са задачи с непълно условие. При тях не се иска точен отговор, а умение да се разсъждава при недостиг на информация. Тук е важно получаването на приблизителен отговор въз основа на общата култура.

1) Колко топки за голф могат да се поберат в един автобус?

Решение: Отговорът много зависи от размерите на автобуса. Да приемем за определеност, че става дума за дълъг автобус (например марка "Икарус"). Най-вероятно не знаем точните размери, но можем да ги преценим на око. Дължината е около 15–20 метра, ширината е около 2–3 метра, а височината на автобуса е около 2 метра. Следователно обемът му е между $15 \times 2 \times 2 = 60$ и $20 \times 3 \times 2 = 120$ кубични метра. За по-голяма точност да приемем средната стойност, т.е. 90 кубични метра = 90 000 000 кубични сантиметра.

Голфът не е особено популярен в България, но поне по телевизията сме виждали игра на голф. Отново на око преценяме, че диаметърът на топка за голф е около 5 сантиметра, следователно радиусът е 2,5 см, откъдето обемът на топката се получава равен на около 65 кубични сантиметра.

Най-плътната възможна опаковка на сфери заема, както е известно, около 74 % от пространството, което е почти $^3/_4$. Като пресметнем $^3/_4$ от обема на автобуса, тоест $^3/_4$ от 90 000 000, получаваме около 68 млн. куб. см — това е обемът, зает от топките за голф.

Остава да разделим 68 милиона на 65, за да получим отговора: в автобуса могат да се поберат около един милион топки за голф.

2) Колко бензиностанции има в България?

Решение: Броят на бензиностанциите би трябвало да е правопропорционален на населението. Затова взимаме някое населено място, за което знаем както броя на жителите, така и броя на бензиностанциите. Тук трябва да направим компромис: колкото по-голямо е населеното място, толкова по-трудно е да знаем колко бензиностанции има. Добър пример е някой от що-годе големите (но не най-големите) градове в България (50−150 хиляди души) или някой от кварталите на София.

Да кажем, в Благоевград има десетина бензиностанции и 70 000 души, което прави по 7000 души на една бензиностанция. Тъй като цялото население на България е около 7 000 000, т.е. 1000 пъти по 7000, следва, че в страната има около хиляда бензиностанции. Разбира се, трябва да включим и някаква допълнителна бройка за бензиностанциите извън населените места (магистрали и други големи пътища), а те са по-трудни за изчисление. От собствен шофьорски опит преценяме колко често зареждаме извън града (това е рисковано разсъждение, но става за приближение) и добавяме например 20 %. Излиза, че в България има около 1200 бензиностанции.

3) Монета в центрофуга.

Смалили сте се до размерите на монета от 20 стотинки. Попаднали сте в центрофуга, която всеки миг ще започне да се върти. Как ще се спасите?

Решение: Просто ще изскочите. Обемът и теглото на тялото ви намаляват пропорционално на третата степен на размера, докато силата на мускулите — пропорционално на втората степен (защото силата им зависи от напречното сечение). Тоест абсолютната сила намалява, но относителната се увеличава. За дребните животни е много по-лесно да повдигнат тежест, равна на собственото им тегло. Бълхата например скача 600 пъти по-високо от собствения си ръст. Затова, ако се смалите на височина например 100 пъти, ще станете 100 пъти по-силен спрямо собственото си тегло. Така че няма да е проблем да изскочите от центрофугата.

4) Защо уличните шахти са кръгли?

Решение: Кръгът е единствената фигура, която не може да влезе в себе си, както и да се върти. Така капакът на шахтата не може да падне в нея. Обаче квадратът, правоъгълникът и елипсата могат.

Други съображения: цилиндричният отвор лесно се прокопава със свредел; цилиндричните тръби са лесни за монтиране и са устойчиви на вътрешен и на външен натиск.