Второ контролно за курса по ДАА (избираем), СУ, ФМИ, 08. 02. 2016 г.

Име: _____ ФН: _____ Спец.: ___ Курс: ___

Задача	1	2	3	4	5	6	Общо
получени точки							
максимум точки	30	20	30	20	20	20	140

Забележка: За отлична оценка са достатъчни 100 точки!

Задача 1. Тази година Дядо Коледа май не е чел внимателно писмата: поръчахте си камина, а той Ви донесе картина. Разпитахте колеги и приятели за възможни размени на подаръци и получихте k предложения: "Заменям гоблен за картина." "Заменям камина за климатик." "Заменям чисто нов компютър за пълен комплект записки по ДАА." И тъй нататък. Понеже никой не поиска да замени камина за картина, налага се да проявите изобретателност, като извършите серия от размени. Тези k човека, които Ви направиха предложения, не се познават, затова Вие трябва да участвате във всяка размяна.

а) Съставете алгоритъм, който за време $\Theta(k)$ да намери най-къса редица от замени, така че да се сдобиете с желаната камина (ако такава редица съществува). (20 точки)

б) Допълнително да предположим, че всяка оферта има краен срок, а Вие можете да правите само по една размяна на ден. Променете алгоритъма така, че за време $O\left(k^2\right)$ да намери най-къса редица от замени, която изпълнява желанието Ви и спазва сроковете. (10 точки)

Опишете алгоритмите словесно, илюстрирайте ги с примери и анализирайте времевите сложности като функции на k.

Задача 2. Някои от компютрите в една мрежа са свързани с комуникационни канали. Всеки от каналите се характеризира с определена надеждност — число от интервала $(0\,;\,1)$, което показва вероятността за безпроблемно предаване на съобщение по канала. Предложете бърз алгоритъм, който намира най-надеждния път между два дадени компютъра от мрежата. (Надеждността на пътя е произведението от надеждностите на съставящите го канали.)

Задача 3. Съставете алгоритъм, който по зададено цяло положително число n намира най-малкия брой точни квадрати със сбор n. (20 точки)

Разширете алгоритъма така, че да отпечатва представянето на n като сбор на най-малък брой точни квадрати. (10 точки)

Демонстрирайте алгоритъма при $n = 10$. Анализирайте сложността по време и по памет за $\forall n$.

Задача 4. Има две купчинки с камъни. Двама играчи се редуват: този, който е на ход, взима от някоя купчинка произволен брой камъни (поне един). Който вземе последния камък, печели. Кой от двамата има печеливша стратегия, ако:

а) във всяка купчинка има по 7 камъка; (10 точки)

б) в едната купчинка има 20 камъка, а в другата има 30 камъка? (10 точки)

Във всеки от двата случая опишете стратегията и докажете подробно, че тя е печеливша.

Задача 5. Долната граница $\Omega(n \log n)$ за времевата сложност на задачата за сортиране на числов масив остава ли в сила за частния случай, когато числата в масива са положителни? Ако да — предложете подходяща редукция. Ако не — съставете бърз алгоритъм.

Задача 6. Задачата Клика остава ли NP-пълна за двуделни графи?
Ако да — предложете подходяща редукция. Ако не — съставете бърз алгоритъм.

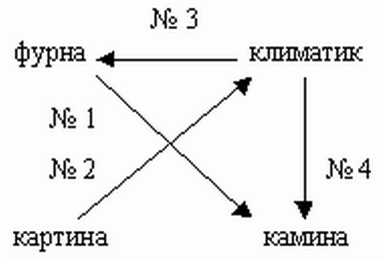
РЕШЕНИЯ

Задача 1. а) Нека p е броят на различните предмети в списъка с оферти. Тогава $p \leq 2k$ (неравенството е възможно, ако някои предмети се повтарят). Ако предметите са номерирани с целите числа от 1 до p, то всяка оферта се описва във входните данни от наредена двойка $(\, i \,,\, j \,)$ от различни цели числа: съответният Ваш познат иска да замени предмет № i за предмет № j. За време $\Theta(k)$ обхождате списъка с офертите, намирате p (максималния номер на предмет) и построявате насочен мултиграф $G(V, E)$, където $V = \Big\{1,\ 2,\ 3,\ \ldots\ ,\ p\Big\}$, т.е. на всеки предмет съответства връх и обратно. Всяко ребро съответства на една оферта: ако поредният познат иска да замени предмет № i за предмет № j, то Вие можете да замените предмет № j (ако го имате) за предмет № i, затова прекарвате ребро от връх № j към връх № i. Нека $n = |V|$ и $m = |E|$ са съответно броят на върховете и броят на ребрата на мултиграфа. Тогава $m = k,\ n = p \leq 2k$. Търси се най-къс път от един даден връх (картината) до друг даден връх (камината). Понеже ребрата нямат тегла, то дължината на пътя се мери с броя на ребрата. Най-бързият алгоритъм в този случай е търсенето в ширина. Неговата времева сложност е $\Theta(m + n) = \Theta(k)$, защото от $m = k$ и $0 \leq n \leq 2k$ следва, че $k \leq m + n \leq 3k$. Двата етапа — построяването на мултиграфа и търсенето на най-къс път — изразходват общо време $\Theta(k) + \Theta(k) = \Theta(k)$. Окончателно, времевата сложност на алгоритъма е линейна: $\Theta(k)$.

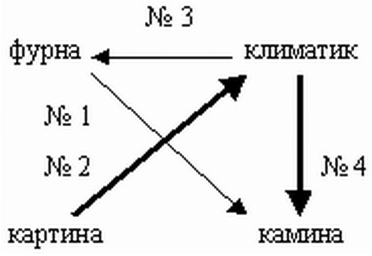
Забележка 1: Всяко ребро трябва да пази поредния номер на офертата, за да знаете с кого правите размяна. Поредният номер не е дължина на реброто, т.е. G остава нетегловен.

Забележка 2: Ако искате G да не съдържа излишни ребра (т.е. да бъде граф, а не мултиграф), трябва да откриете кои оферти се повтарят. Алгоритъм, основан на сравнения, ще изразходва време $\Omega(k \log k)$. Модификация на CountingSort би помогнала при предметите (номерата им са малки числа — до $2k$ максимум), но не и при офертите: броят на възможните оферти е $p^2 - p = 4k^2 - 2k$ (при $p = 2k$), т.е. квадратична сложност. Решението с мултиграф е по-бързо.

Пример: Получили сте предложения от $k = 4$ познати. Първият има камина, а иска фурна. Вторият има климатик, а иска картина. Третият от Вашите познати има фурна, а иска климатик. Четвъртият има камина, а иска климатик. Мултиграфът G (в случая той е граф) изглежда така:



Вие имате картина, а искате камина. С помощта на търсене в ширина намирате най-къс път от върха "картина" до върха "камина" :



Получи се пътят "картина" – "климатик" – "камина", който минава по ребрата № 2 и № 4. Тоест първо давате картината на втория от Вашите познати и взимате от него климатика; после давате климатика на четвъртия човек и взимате от него камината, която желаете. В този случай две замени се оказват достатъчни и това е минималният брой.

б) Нека входните данни съдържат t_1 , t_2 , ... , t_k — цели положителни числа, които показват колко дена остават до изтичането на всяка оферта. Без ограничение можете да предполагате, че тези числа не надхвърлят k (тъй като k дена със сигурност са достатъчни, то ако има стойности, по-големи от k, те могат да бъдат намалени до k). За време $\Theta(k)$ обхождате списъка с оферти и намирате най-дългия срок T, т.е. $T = \max\left\{t_1\ ,\ t_2\ ,\ \ldots\ ,\ t_k\right\}$. Следователно числото T е цяло положително и $T \le k$.

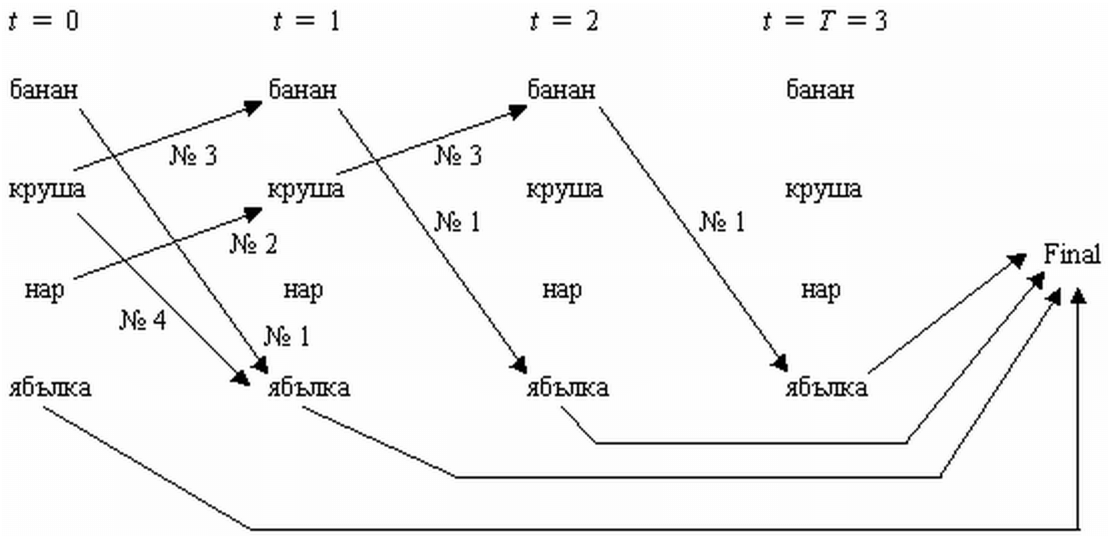
Построението на ориентирания нетегловен мултиграф $G(V,E)$ се променя така: Правят се $T+1$ копия на върховете; всяко копие съответства на границата между два поредни дена. Всеки връх показва едно възможно състояние: ако се намирате във върха (i,t), това значи, че са изминали t дена (т.е. направили сте t замени) и в този момент притежавате предмет № i. Създават се и различен брой копия на ребрата; всяко ребро сочи от един ден към следващия и съответства на възможна замяна през определен ден. Както по-горе, във всяко ребро се пази поредният номер на офертата.

По-формално: $V = \left\{1,\ 2,\ 3,\ \ldots\ ,\ p\right\} \times \left\{0,\ 1,\ 2,\ \ldots\ ,\ T\right\}$ е множеството на върховете, p е най-големият от номерата на предметите, $p \le 2k$. Броят на върховете на мултиграфа е $n = |V| = p\,(T+1) \le 2k(k+1)$, тоест $n \le 2k(k+1)$. Ребрата на графа се строят така: за всяка оферта $(i\,,\,j)$ със срок t дена се добавят ребра от върха $(\,j\,,\,0)$ към върха $(i\,,\,1)$, от върха $(\,j\,,\,1)$ към върха $(i\,,\,2)$ и т.н., накрая — от върха $(\,j\,,\,t-1)$ към върха $(i\,,\,t)$. Нека S е номерът на предмета, който имате, а F е номерът на предмета, който искате; $S \ne F$. (В примера от условието S е поредният номер на картината, а F е номерът на камината.)

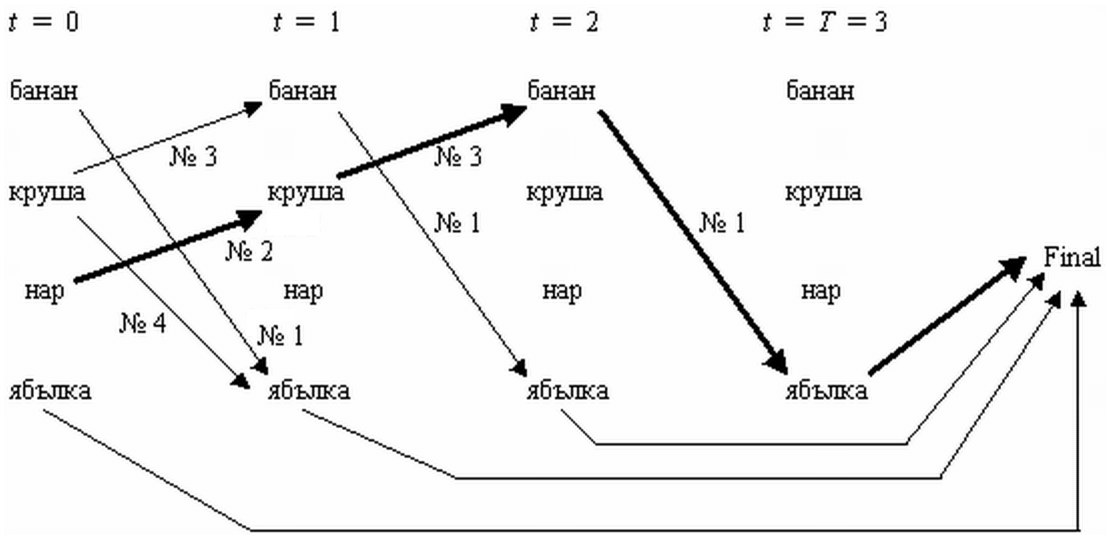
Задачата се свежда до търсене на най-къс път в мултиграфа G от върха $(S\,,\,0)$ до някой от върховете $(F\,,\,0)$, $(F\,,\,1)$, ... , $(F\,,\,T\,)$. Отново търсенето в ширина е най-бързият метод; изразходваното време е $\Theta(m+n)$. За да получим оценка на времето като функция на k, взимаме предвид неравенството за броя на върховете на G, доказано по-горе: $n \le 2k(k+1)$. От друга страна, броят на ребрата е $m = |E| = t_1 + t_2 + \ldots + t_k \le kT \le k^2$, защото $T \le k$. Следователно $m+n \le 2k(k+1)+k^2 = 3k^2+2k$, т.е. времевата сложност е $\mathrm{O}\left(k^2\right)$.

Ако държим да сведем нашата задача до задачата за търсене на най-къс път от един връх до друг връх (а не до множество върхове), можем да постигнем това, като добавим един изкуствен връх Final, т.е. $V = \left\{1,\ 2,\ 3,\ \ldots\ ,\ p\right\} \times \left\{0,\ 1,\ 2,\ \ldots\ ,\ T\right\} \cup \left\{\mathrm{Final}\right\}$. Новият връх съответства на състоянието "постигната цел" и не съдържа информация за деня. За да може върхът Final да играе тази роля, трябва да добавим ребра към него от върховете $(F\,,\,0)$, $(F\,,\,1)$, ... , $(F\,,\,T\,)$.

Пример: Получили сте предложения от $k=4$ познати. Първият има ябълка, а иска банан най-късно на третия ден. Вторият има круша, а иска нар най-късно на първия ден. Третият има банан, а иска круша най-късно на втория ден. Четвъртият има ябълка, а иска круша най-късно на първия ден. Вие имате нар, а искате ябълка. Сега мултиграфът изглежда така:



Чрез търсене в ширина намирате най-къс път от върха “нар” в първата колона ($t = 0$) до Final:



Сега най-късата редица от замени (в конкретния пример тя е единствената възможна редица) се състои от три замени (последното ребро не се брои за замяна):
Давате нара на втория човек и взимате от него крушата;
после давате крушата на третия човек и взимате от него банана;
накрая давате банана на първия човек и взимате от него ябълката.

Добавянето на върха Final забавя малко алгоритъма, но забавянето е незначително по порядък. По-точно, добавянето на $T + 1$ ребра към Final изразходва време $\Theta(T) = \mathrm{O}(k) = \mathrm{o}\left(k^2\right)$. Върхът Final не е задължителен, както видяхме по-горе: търсенето в ширина е приложимо и когато се търси най-къс път от един връх до множество от върхове. Времевата сложност остава $\Theta(m + n) = \mathrm{O}\left(k^2\right)$, при условие че алгоритъмът бъде реализиран внимателно: така, че проверката за достигане на целта да отнема константно време!

Пример за правилна реализация е използването на структурата от данни $E\big[1 \ldots p\big]\big[0 \ldots T\big]$: двумерен масив от списъци с ребра. Всеки връх представлява наредена двойка от индекси $(i\,,\,t)$; всеки елемент $E\big[i\big]\big[t\big]$ на масива е списък от ребра — ребрата, излизащи от върха $(i\,,\,t)$. Проверката, дали даден връх е целеви, се свежда до сравнението $i \stackrel{?}{=} F$, а то отнема константно време.

Пример за неправилна реализация е представянето на върховете като обекти. Проверката, дали текущият връх е целеви, се свежда до поредица от сравнения на указателя към него с указателите към върховете $(F\,,\,0)$, $(F\,,\,1)$, $\ldots$, $(F\,,\,T\,)$ чрез цикъл със сложност $\Theta(T)$, който увеличава сложността на търсенето в ширина до $\Theta\big((m + n)\,T\,\big) = \mathrm{O}\left(k^2 k\right) = \mathrm{O}\left(k^3\right)$. Грешката може да се поправи, като в обекта на всеки връх освен списъка с изходящи ребра се пази допълнително поле — поредния номер i на съответния предмет. Така отново се стига до сравнението $i \stackrel{?}{=} F$, което работи в константно време.

Обратно, възможни са дребни оптимизации, които ускоряват алгоритъма, но ускорението е незначително (т.е. не намалява порядъка на сложността). Например върховете $(i\,,\,0)$ са излишни при $\forall\, i \neq S$. Излишни са и върховете $(i\,,\,T)$ при $\forall\, i \neq F$.

Мултиграфът от подусловие “б” е ацикличен, тъй като преминаването по всяко ребро увеличава времето t с единица. Затова задачата може да се реши с динамично програмиране, чиято времева сложност е същата по порядък като на търсенето в ширина: $\Theta(m + n) = \mathrm{O}\left(k^2\right)$. По принцип този подход би бил около два пъти по-бавен заради двукратното обхождане на G: при топологичното сортиране и при динамичното програмиране. В тази задача няма забавяне, защото топологичното сортиране отпада: G е сортиран от самото начало (върховете с малки t предхождат върховете с големи t; върховете с еднакви t могат да се подредят в произволен ред).

Задача 3 може да се реши чрез динамично програмиране. Таблицата представлява едномерен масив $\mathrm{dyn}\big[0\ldots n\big]$ от цели числа, като $\mathrm{dyn}\big[k\big]$ е най-малкият брой точни квадрати със сбор k. Масивът се попълва с помощта на началното условие $\mathrm{dyn}\big[0\big] = 0$ и рекурентната формула $\mathrm{dyn}\big[k\big] = 1 + \min\limits_{i} \Big\{\mathrm{dyn}\,\big[k - i^{\,2}\,\big]\Big\}$, $k > 0$, където минимумът е по всички $i = 1,\ 2,\ 3,\ \ldots\ ,\ \big\lfloor\sqrt{k}\,\big\rfloor$.

SQUARES(n: positive integer)

```
 1  dyn[0...n]: array of positive integers;
 2  // dyn[k] = най-малкият брой точни квадрати със сбор k.
 3  addend[1...n]: array of positive integers;
 4  // addend[k] = най-малкото събираемо в представянето на k
 5  //                 като сбор на минимален брой точни квадрати.
 6  dyn[0] ← 0
 7  for k ← 1 to n
 8      dyn[k] ← k + 1
 9      i ← 1
10      j ← 1
11      while j ≤ k do
12          if dyn[k − j] < dyn[k]
13              dyn[k] ← dyn[k − j]
14              addend[k] ← j
15          i ← i + 1
16          j ← i × i
17      dyn[k] ← dyn[k] + 1
18  k ← n
19  while k > 0 do
20      j ← addend[k]
21      print j
22      k ← k − j
23  return dyn[n]
```

Демонстрация на работата на алгоритъма при $n = 10$:

k	0	1	2	3	4	5	6	7	8	9	10
dyn	0	1	2	3	1	2	3	4	2	1	2
addend		1	1	1	4	1	1	1	4	9	1

Таблицата се попълва отляво надясно. Например последната колонка е получена тъй:
$\mathrm{dyn}\big[10\big] = 1 + \min\Big\{\mathrm{dyn}\,\big[10 - 1^{\,2}\,\big]\ ;\ \mathrm{dyn}\,\big[10 - 2^{\,2}\,\big]\ ;\ \mathrm{dyn}\,\big[10 - 3^{\,2}\,\big]\Big\} =$
$= 1 + \min\Big\{\mathrm{dyn}\big[9\big]\ ;\ \mathrm{dyn}\big[6\big]\ ;\ \mathrm{dyn}\big[1\big]\Big\} = 1 + \min\Big\{1\ ;\ 3\ ;\ 1\Big\} = 1 + 1 = 2,$
което означава, че за числото 10 са нужни два квадрата.

Най-малкият елемент на мултимножеството $\Big\{1\ ;\ 3\ ;\ 1\Big\}$ е числото 1, което се среща два пъти и съответства на събираемите $1^{\,2}$ и $3^{\,2}$. Без значение е кое от събираемите ще вземем. Така, както алгоритъмът е оформен по-горе (със строго неравенство на ред № 12), той взима по-малкото събираемо, затова $\mathrm{addend}\big[10\big] = 1^{\,2} = 1$. (Ако на ред № 12 има нестрого неравенство, тогава в addend ще се пази по-голямото събираемо. В такъв случай алгоритъмът ще работи вярно, но по-бавно: редове № 13 и № 14 ще се изпълняват излишно.)

Самото представяне на 10 като сбор от точни квадрати се получава с помощта на масива addend. Тръгваме от колонката $k = 10$ и намираме addend = 1, което е най-малкото събираемо в сбора. Изваждаме това събираемо от сумата и остава $10 - 1 = 9$. От колонката $k = 9$ намираме следващото събираемо: addend = 9. Изваждаме това събираемо от оставащата сума: $9 - 9 = 0$. Не остава нищо, така че алгоритъмът приключва работа. Намерено е представянето $10 = 1 + 9$.

Анализ на сложността на алгоритъма:

Най-голямо количество допълнителна памет изразходват масивите dyn и addend, затова те определят сложността по памет: $\Theta(n)$.

Сложността по време зависи от броя итерации на циклите. Цикълът на редове № 11 – № 16 се изпълнява, докато $j = i^2 \le k$, т.е. за $i = 1, 2, 3, \ldots, \lfloor\sqrt{k}\,\rfloor$, което прави $\lfloor\sqrt{k}\,\rfloor$ итерации, а това по порядък е равно на $\sqrt{k}$. Следователно сложността на цикъла на редове № 7 – № 17 е равна по порядък на $\sum_{k=1}^{n} \sqrt{k} = \sum_{k=1}^{n} k^{0,5} = \Theta\left(n^{1,5}\right) = \Theta\left(n\sqrt{n}\right)$.

Тялото на цикъла на редове № 19 – № 22 се изпълнява $\mathrm{O}(n)$ пъти, защото на всяка итерация стойността на k намалява поне с една единица. Останалите команди (ред № 6 и ред № 23) изразходват константно време, затова не влияят на порядъка на сложността.

Времето на алгоритъма е сбор от времената на частите му: $\Theta\left(n\sqrt{n}\right) + \mathrm{O}(n) = \Theta\left(n\sqrt{n}\right)$. Окончателно, времевата сложност на алгоритъма е $\Theta\left(n\sqrt{n}\right)$.

Задача 2. Моделираме компютърната мрежа чрез тегловен граф:
— върховете на графа съответстват на компютрите;
— ребрата съответстват на комуникационните канали;
— теглото на всяко ребро го полагаме да бъде равно на $w = -\ln p$, където p е надеждността на съответния канал, а логаритъмът е натурален (всъщност основата може да бъде всяко число, по-голямо от 1).

От $p < 1$ следва, че $\ln p < 0$, откъдето $w = -\ln p > 0$, т.е. теглата на ребрата са положителни, следователно могат да се интерпретират като дължини. Логаритмуването свежда умножението на надеждности до събиране на техните логаритми. Така теглото на път е равно на сбора от теглата на неговите ребра. Следователно теглото на път също може да се интерпретира като дължина на пътя (поради цитираното адитивно свойство). Заради знака минус теглото w е намаляваща функция на надеждността p: колкото по-къс е пътят, толкова по-надежден е. Дадената задача се свежда до задачата за намиране на най-къс път в граф с положителни тегла на ребрата. Подходящ за този случай е алгоритъмът на Дейкстра.

Задача 4. а) Тъй като двете купчинки имат еднакъв брой камъни ($7 = 7$), то вторият играч има печеливша стратегия: на всеки ход, колкото камъка вземе първият играч от едната купчинка, толкова камъка трябва да вземе вторият играч от другата купчинка. По индукция се доказва, че след всеки ход на първия играч купчинките съдържат различен брой камъни, а след всеки ход на втория играч купчинките съдържат еднакъв брой камъни. Играта със сигурност ще завърши (защото на всеки ход се взима поне един камък), а когато това стане, купчинките са празни, т.е. съдържат еднакъв брой камъни ($0 = 0$). Следователно последният ход е бил на втория играч и той печели играта.

б) Този път двете купчинки имат различен брой камъни, затова печеливша стратегия има първият играч: на първия си ход взима 10 камъка от по-голямата купчинка (така броят на камъните в двете купчинки се изравнява — по 20 във всяка), после прилага стратегията, описана в предишното подусловие.

Задача 5. Долната граница $\Omega(n \log n)$ за времевата сложност на задачата за сортиране на числов масив остава в сила за частния случай, когато числата в масива са положителни. За да докажем това, ще сведем общия случай до частния чрез достатъчно бърза редукция.

Нека Sort е общият случай — задачата за сортиране на произволен числов масив (който може да съдържа нули, положителни и отрицателни числа).

Нека SortPositive е частният случай — задачата за сортиране на масив, който съдържа само положителни числа.

Идеята на решението е да сортираме два помощни масива от положителни числа: единият помощен масив ще се състои от положителните числа в оригиналния масив; другият помощен масив ще се състои от абсолютните стойности на отрицателните числа в оригиналния масив. (На практика, можем да ползваме един голям масив и да сортираме отделните му части.)

```
Sort( A[1...n]: array of numbers )
 1  B[1...n]: array of numbers          // B: помощен масив;
 2  p ← n+1                             // p = n+1 минус броя на положителните числа;
 3  s ← 0                               // s = броя на отрицателните числа.
 4  for k ← 1 to n
 5      if A[k] > 0
 6          p ← p−1
 7          B[p] ← A[k]
 8      else if A[k] < 0
 9          s ← s+1
10          B[s] ← −A[k]
11  SortPositive(B[1...s])
12  SortPositive(B[p...n])
13  for k ← 1 to s
14      A[k] ← −B[s+1−k]
15  for k ← s+1 to p−1
16      A[k] ← 0
17  for k ← p to n
18      A[k] ← B[k]
```

Пример: Входни данни — масивът A : 20 , 10 , 0 , −2 , 8 , −1 , 0 , −6 , 90 , 20 , −2 , 0; общо $n = 12$ числа: четири отрицателни, пет положителни и три нули. Затова $s = 4$, $p = 8$.
Непосредствено преди да се изпълни ред № 11 от алгоритъма, помощният масив изглежда така:
B : 2 , 1 , 6 , 2 , ? , ? , ? , 20 , 90 , 8 , 10 , 20.
Елементите на B от № 1 до № s (т.е. 2 , 1 , 6 , 2) са абсолютните стойности на отрицателните числа от A в реда, в който се срещат в A. Елементите на B от № p до № n (20 , 90 , 8 , 10 , 20) са положителните числа от A в обратен ред (подмасивът $B[p \ldots n]$ се попълва отзад напред). Елементите на B от № $s+1$ до № $p-1$ (означени с въпросителни) остават неинициализирани; по брой те съответстват на нулите от A.
Непосредствено след изпълнението на ред № 12 от алгоритъма помощният масив изглежда така:
B : 1 , 2 , 2 , 6 , ? , ? , ? , 8 , 10 , 20 , 20 , 90; тоест всяка от двете части е сортирана.
Първите $s = 4$ елемента от B се копират в началото на A в обратен ред и с отрицателни знаци:
A : −6 , −2 , −2 , −1 ...
След това в масива A се дописват три нули:
A : −6 , −2 , −2 , −1 , 0 , 0 , 0 ...
Най-сетне положителните числа от края на B се копират в края на A в същия ред:
A : −6 , −2 , −2 , −1 , 0 , 0 , 0 , 8 , 10 , 20 , 20 , 90.
Масивът A е сортиран.

Доказателство за коректност на редукцията: Операторите до ред № 10 (вкл.) гарантират, че непосредствено преди да бъде изпълнен ред № 11 от алгоритъма, са в сила следните твърдения: $s =$ броя на отрицателните числа в A; $p = n + 1$ минус броя на положителните числа в A; подмасивът $B\big[\,1 \ldots s\,\big]$ съдържа абсолютните стойности на отрицателните числа от масива A; подмасивът $B\big[\,p \ldots n\,\big]$ съдържа положителните числа от A; в частност, и двата подмасива съдържат само положителни стойности. След изпълнението на редове № 11 и № 12 подмасивите $B\big[\,1 \ldots s\,\big]$ и $B\big[\,p \ldots n\,\big]$ ще бъдат сортирани, тъй като SortPositive е коректна процедура за сортиране на масиви от положителни числа. Цикълът от редове № 13 и № 14 прехвърля абсолютните стойности на отрицателните числа от $B\big[\,1 \ldots s\,\big]$ в началото на масива A, като добавя отрицателни знаци и обръща реда (когато броячът k расте, индексът $s+1-k$ намалява) в съответствие с аритметичното правило, че по-големите отрицателни числа имат по-малки абсолютни стойности. Цикълът от редове № 15 и № 16 дописва необходимия брой нули в масива A. Цикълът от редове № 17 и № 18 копира положителните числа от $B\big[\,p \ldots n\,\big]$ в края на масива A, без да променя техния ред. Индексирането на елементите на в редове № 13 – № 18 гарантира, че отрицателните числа са в началото на A, положителните — в края, а нулите — между тях (според известното свойство на нулата, че тя е по-голяма от всяко отрицателно число и по-малка от всяко положително). Щом всички аритметични правила за наредбата на числата са спазени, то следва, че Sort е коректна процедура за сортиране на числови масиви.

Времевата сложност на редукцията (всичко без редове № 11 и № 12) очевидно е линейна: $\Theta(n) = \mathrm{o}(n \log n)$, следователно редукцията е достатъчно бърза за целите на доказателството.

Задача 6. Алгоритмичната задача Клика приема като входни данни един граф $G(V, E)$ и едно цяло положително число k и отговаря на въпроса дали графът G съдържа клика от ред k. От теорията е известно, че когато графът G е произволен, тази алгоритмична задача е NP-пълна.

Но ако G е двуделен граф, задачата Клика вече не е NP-пълна (при условие че P $\neq$ NP), тъй като за нея в този случай съществува алгоритъм с полиномиална времева сложност:

1) Ако $k \geq 3$, то няма клика от ред k, защото всяка такава клика съдържа триъгълник, а в двуделен граф не може да има цикъл с нечетна дължина.
2) Ако $k = 1$, то има клика от ред k — кой да е връх на графа G (смятаме, че $V \neq \varnothing$).
3) Остава случаят $k = 2$; в този случай има клика от ред k точно тогава, когато графът G съдържа поне едно ребро, т.е. когато $E \neq \varnothing$.

Сравненията на k с 1 и с 3 работят в константно време. Проверката, дали $E \neq \varnothing$, изисква обхождане на графа, т.е. работи в линейно време. Затова времевата сложност на алгоритъма е линейна, следователно полиномиална, което трябваше да се докаже.