С Ъ С Т А В Я Н Е Н А А Л Г О Р И Т М И

КОНТРОЛНО № 2 ПО "ДИЗАЙН И АНАЛИЗ НА АЛГОРИТМИ" — СУ, ФМИ, 19. 12. 2018 Г.

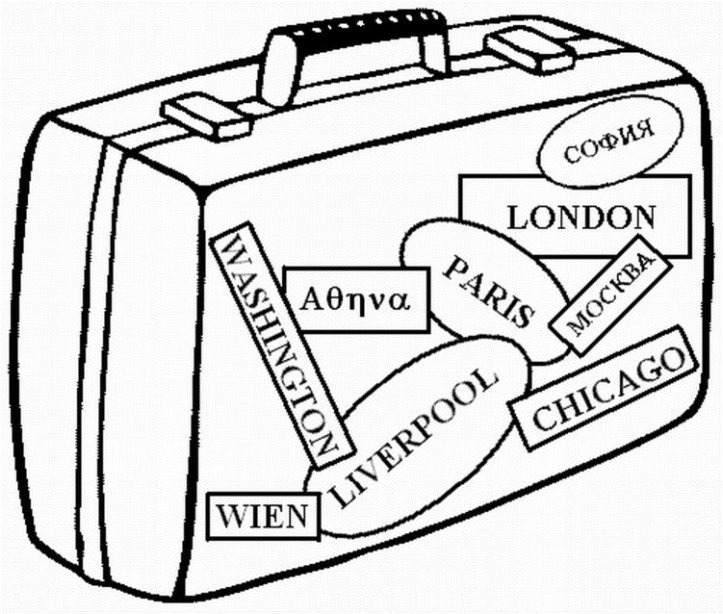
Имате право да използвате наготово всички алгоритми, изучени на лекции. Всеки алгоритъм, използван наготово, трябва да бъде посочен чрез името си на български език. Ако такъв алгоритъм има няколко реализации, трябва да се уточни използваната реализация. Когато използвате графи, описвайте подробно процеса на моделиране: какво представляват върховете и ребрата на графа. За алгоритми върху графи указвайте използваната алгоритмична схема (ако има такава) — обхождане в ширина или обхождане в дълбочина. Уточнявайте името на използваните сортировки.

Задача 1. В ориентиран граф с неотрицателни тегла на ребрата да се намери най-къс цикъл (ако има такъв) за полиномиално време.

а) Опишете алгоритъма с думи. **(5 точки)**

б) Анализирайте времевата сложност на алгоритъма. **(4 точки)**

Задача 2. Върху куфар са сложени лепенки от разни градове. Коя лепенка е сложена първа? (Ако има няколко възможни отговора, достатъчно е да бъде посочен един от тях.)



а) Решете задачата в частния случай, показан на картинката. Отговорът да се обоснове! **(3 точки)**

б) Опишете с думи алгоритъм за общия случай с линейна времева сложност. Илюстрирайте алгоритъма с частния случай, показан на картинката. **(3 точки)**

в) Допълнете алгоритъма така, че да разпознава грешни данни, тоест такива, при които задачата няма решение. Времевата сложност да остане линейна. **(3 точки)**

Задача 3. Нека $A[1...n]$ е логически масив, $A[1]$ е истина, $A[n]$ е лъжа. Предложете най-бърз алгоритъм, намиращ два последователни елемента, първият от които е истина, а вторият — лъжа. Алгоритъмът трябва да връща по-малкия от двата индекса. Само най-бърз алгоритъм носи точки!

а) Опишете алгоритъма на псевдокод. **(5 точки)**

б) Анализирайте времевата сложност на алгоритъма. **(4 точки)**

Задача 4. Числовият масив $A[1...n]$ съдържа цели числа — възрастите на няколкостотин деца в години. Съставете алгоритъм, който за време $O(n)$ избира пет деца така, че разликата във възрастите на най-голямото и най-малкото от избраните деца да бъде минимална.

а) Опишете алгоритъма на псевдокод. **(5 точки)**

б) Анализирайте времевата сложност на алгоритъма. **(4 точки)**

Задача 5. За време $o(n^2)$ намерете по колко начина положителното цяло число n се представя като сбор от кубове на цели положителни числа. Сборове, различаващи се по реда на събираемите, да се броят като различни представяния.

а) Опишете алгоритъма на псевдокод. **(5 точки)**

б) Направете подробна демонстрация на алгоритъма за $n = 31$. **(5 точки)**

в) Анализирайте времевата сложност на алгоритъма. **(4 точки)**

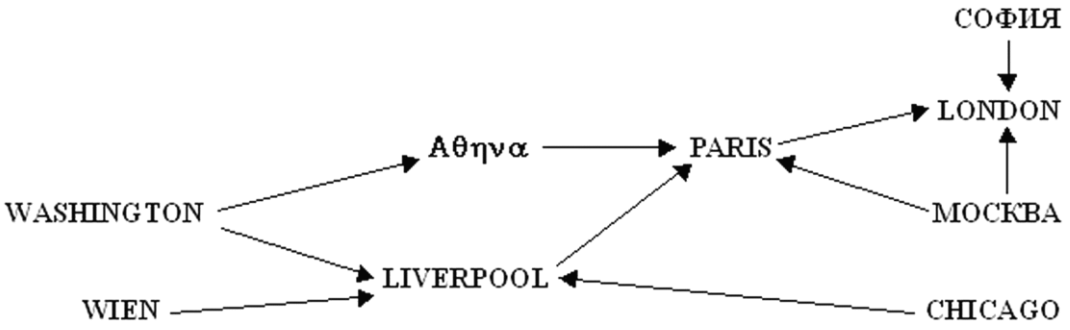
Р Е Ш Е Н И Я

Задача 1. Тъй като не е даден връх, през който да минава цикълът, разглеждаме всички върхове, тоест търсим най-къс път от всеки връх до всеки връх. Това може да стане за време $\Theta(n^3)$ чрез ***алгоритъма на Флойд—Уоршал,*** където n е броят върхове на графа. После за време $\Theta(n)$ обхождаме диагонала на получената матрица $n \times n$ и избираме най-малкия елемент. Тъй като j-тият елемент върху диагонала на матрицата е дължината на най-късия цикъл, който съдържа j-тия връх на графа, то най-малкият от диагоналните елементи е дължината на най-късия цикъл. Самия цикъл можем да намерим с помощта на указателите към предходния връх на най-къс път, пазени в отделна матрица от тип $n \times n$. Времето за проследяване на най-късия цикъл е $O(n)$. Общото време на целия алгоритъм е полиномиално: $\Theta(n^3) + \Theta(n) + O(n) = \Theta(n^3)$, поради което този алгоритъм носи пълен брой точки.

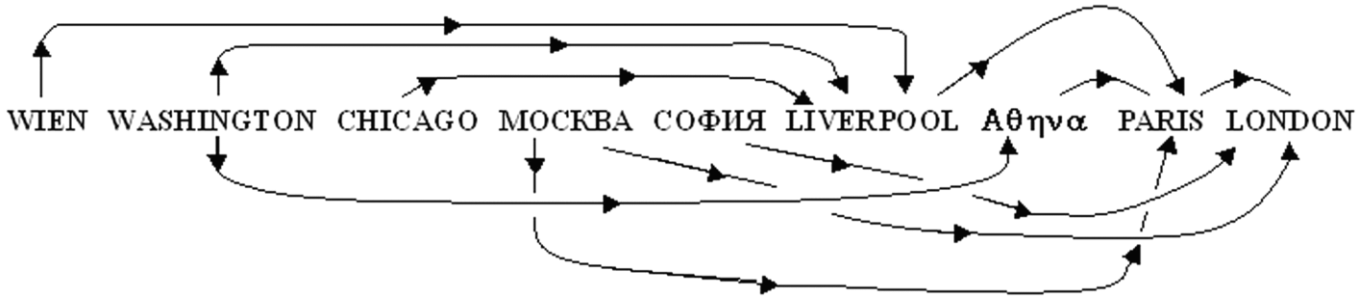
Все пак това не е най-бързият възможен алгоритъм. Ако от най-къс цикъл премахнем някое ребро, останалата част е най-къс път. Понеже теглата на ребрата са неотрицателни, можем да използваме вариант на ***алгоритъма на Дейкстра.*** Ако искаме да намерим най-къс цикъл през даден връх s, то с помощта на алгоритъма на Дейкстра, пуснат от s, построяваме дървото на най-късите пътища от върха s до всички други върхове на графа. Нека $d(v)$ е дължината на най-късия път от s до v. След като числата $d(v)$ са намерени с алгоритъма на Дейкстра, обхождаме върховете v на графа, от които излиза ребро към s, и изчисляваме $d(v) + w(v, s)$, където $w(v, s)$ е теглото на реброто. Този сбор е дължината на най-късия цикъл, съдържащ реброто от v към s. Минимумът по всички допустими върхове v е дължината на най-късия цикъл през върха s. Самия цикъл намираме, като добавим реброто (v, s) към най-късия път от s до v, пазен в дървото на най-късите пътища. Построяването на дървото на най-късите пътища по алгоритъма на Дейкстра става най-бързо, ако алгоритъмът използва ***приоритетна опашка, реализирана чрез пирамида на Фибоначи.*** Тогава се изразходва време $\Theta(m + n \log n)$, където n е броят на върховете, а m е броят на ребрата на графа. Търсенето на минимума и на самия цикъл изисква време $O(n)$. Ето защо общото време за намиране на най-къс цикъл през даден връх е от порядък $\Theta(m + n \log n) + O(n) = \Theta(m + n \log n)$.

Обаче върхът s не е даден, поради което пускаме описаната процедура от всеки връх на графа и взимаме най-късия от получените цикли; той е най-къс цикъл в графа. Понеже процедурата се пуска n пъти, то общото време е $\Theta(n\,m + n^2 \log n) = O(n^3)$, т.е. този алгоритъм е не по-бавен от първия алгоритъм. Откриването на втория (по-бързия) алгоритъм се възнаграждава с бонус: удвояват се точките за всяко от подусловията на задачата.

Задача 2. Първа може да бъде само лепенка, която не покрива никоя друга. На дадената картинка само лепенката с надпис "**LONDON**" не покрива никоя друга, така че само тя може да бъде първа. В общия случай задачата се моделира с ориентиран граф: върховете съответстват на лепенките, а ребрата — на застъпванията: графът съдържа ребро от върха x към върха y тогава и само тогава, когато лепенката x покрива лепенката y. За примера от картинката графът изглежда така:



В общия случай задачата може да се реши с ***топологично сортиране*** на графа, а то използва алгоритмичната схема ***обхождане в дълбочина.*** В подредбата на върховете, получена в резултат на топологичното сортиране, от последния връх не излизат ребра, т.е. той съответства на лепенка, която не покрива никоя друга и затова може да бъде първата лепенка, сложена върху куфара. След топологично сортиране графът, изобразен на предишната страница, може да приеме например следния вид (има и други възможни подредби):



Въпреки че върховете на примерния граф могат да бъдат подредени и по други начини, върхът "**LONDON**" винаги е последен, затова в този пример задачата има един възможен отговор: първата лепенка, сложена върху куфара от картинката, е била лепенката с надпис "**LONDON**". В общия случай задачата може да има и повече възможни отговори, но един от тях винаги е лепенката, която се намира на последно място в подредбата на върховете, получена в резултат на топологичното сортиране на графа.

Топологичното сортиране има времева сложност $\Theta(m + n)$. Това е линейна сложност, тъй като $\Theta(m + n)$ е също дължината на входните данни (описанието на графа).

Има възможност за оптимизация: понеже топологичното сортиране подрежда върховете на графа в обратен ред на затварянето им по време на обхождането в дълбочина, то последният връх в наредбата на върховете (тоест отговорът на задачата) е затворен първи по време на обхождането. Затова алгоритъмът може да прекрати обхождането предсрочно — при първото затваряне на връх; този връх съответства на търсената лепенка. Въпреки че тази оптимизация не променя порядъка на времевата сложност, тя все пак ускорява алгоритъма, затова тя носи допълнителни 3 точки.

Входните данни са некоректни, ако и само ако подреждането на лепенките в хронологичен ред е невъзможно, тоест когато графът съдържа ***ориентиран цикъл.*** Проверката за ацикличност се извършва за линейно време $\Theta(m + n)$ чрез алгоритмичната схема ***обхождане в дълбочина.*** Не е нужно второ обхождане: търсенето на цикъл и топологичното сортиране могат да бъдат извършени с едно обхождане на графа. Отбелязването на този факт носи допълнителни 3 точки.

Задача 3 се решава за време $\Theta(\log n)$ с помощта на ***двоично търсене:***

```
Problem3(A[1...n]) // n ≥ 2, A[1] = true, A[n] = false
first ← 1
last ← n
while last > first + 1 do
   mid ← ⌊(first + last) / 2⌋
   if A[mid] = true
      first ← mid
   else
      last ← mid
return first
```

Така се проверяват дълги сметки по някоя формула. Ако последният ред (отговорът) е верен, приемаме, че сметките са верни (въпреки че няколко грешки могат да се унищожат). Ако отговорът е грешен, търсим къде е грешката. Проверяваме първия ред: ако е грешен, значи грешката е в самото начало. Ако е верен, намираме грешката с двоично търсене. Така си спестяваме проверката на цялото решение: проверяваме само отделни редове от решението.

Задача 4. Възрастите на децата са малки цели числа — от 0 до 17 включително. Има само 18 различни стойности на възрастта, което е много по-малко от броя на децата (няколкостотин). Затова можем да използваме ***сортиране чрез броене.*** След сортирането децата с близки възрасти идват едно до друго. Затова е достатъчно да проверим само петорките от последователни деца. В условието на задачата не е указан идентификатор на децата (име, ЕГН или нещо друго), ето защо ще посочваме децата с техните индекси в масива $A[1...n]$. Сортирането размества децата, затова в друг масив $B[1...n]$ ще пазим първоначалните индекси.

```
Problem4(A[1...n])
C[0...17]: array of lists // масив от списъци
for p ← 0 to 17 do
   C[p] ← empty list // празен списък
for k ← 1 to n do
   C[A[k]].Append(k) // добавяме индекса k в края на списъка C[A[k]]
B[1...n]: array of integers // масив от първоначалните индекси
j ← 1
for p ← 0 to 17 do
   while C[p] is not empty do
      k ← C[p].ExtractFirst // четем и изтриваме първия елемент от списъка C[p]
      A[j] ← p
      B[j] ← k
      j ← j + 1
minDiff ← +∞
for j ← 1 to n - 4 do    // търсим пет деца с най-малка възрастова разлика
   currentDiff ← A[j + 4] - A[j]
   if currentDiff < minDiff            // намерена е по-добра петорка деца
      minDiff ← currentDiff
      minIndex ← j
for j ← minIndex to minIndex + 4 do   // извеждаме най-добрата петорка
   print B[j]                         // отпечатваме първоначалните индекси
```

Анализ на времевата сложност на алгоритъма:
— Първият цикъл for — този, който инициализира масива $C[0...17]$ с празни списъци — изисква константно време $\Theta(1)$. Теоретичното основание за този извод е, че броят на повторенията (18) на тялото на цикъла не зависи от n. От практическа гледна точка е важно още времето на цикъла да е много по-малко от n. Това изискване също е изпълнено, защото броят на повторенията (18) е много по-малък от дължината n на входния масив (по условие n е няколкостотин).
— Вторият цикъл for — този, който попълва списъците — изразходва време $\Theta(n)$: толкова отива за едно обхождане на масива $A[1...n]$.
— Третият цикъл for — този, който заедно с вложения цикъл while презаписва масива $A[1...n]$, подреждайки децата по възраст — изразходва време $\Theta(n)$, защото най-вътрешните оператори се изпълняват точно n пъти: по веднъж за всяка стойност на променливата j, която се увеличава с единица на всяка стъпка, започвайки от 1 и достигайки до n.
— Тялото на четвъртия цикъл for — който търси най-малката възрастова разлика — се изпълнява точно $n - 4 = \Theta(n)$ пъти, затова има линейна времева сложност.
— Петият цикъл for — този, който отпечатва намерената петорка — има времева сложност $\Theta(1)$: броят (5) на повторенията на тялото му не зависи от n и е малко число (много по-малко от n, което е от порядъка на няколкостотин).

Окончателно, времевата сложност на алгоритъма е $\Theta(1) + \Theta(n) + \Theta(n) + \Theta(n) + \Theta(1) = \Theta(n)$.

Задача 5 може да се реши с помощта на ***динамично програмиране:***

```
Problem5(n: positive integer)
dyn[0...n]: array of positive integers
// dyn[k] = броя на начините, по които числото k може да се представи
// като сбор от кубове на цели положителни числа
dyn[0] ← 1 // числото 0 има единствено представяне — празния сбор
for k ← 1 to n do
   s ← 0
   p ← 1
   r ← p × p × p
   while r ≤ k do  // всички възможни случаи за последното събираемо (r)
      s ← s + dyn[k - r]
      p ← p + 1
      r ← p × p × p
   dyn[k] ← s
return dyn[n]
```

Демонстрация на алгоритъма за $n = 31$:

k	dyn[k]	k	dyn[k]	k	dyn[k]	k	dyn[k]
0	1	8	2	16	11	24	64
1	1	9	3	17	14	25	78
2	1	10	4	18	18	26	96
3	1	11	5	19	23	27	120
4	1	12	6	20	29	28	150
5	1	13	7	21	36	29	187
6	1	14	8	22	44	30	232
7	1	15	9	23	53	31	286

Отговора прочитаме от последната клетка на таблицата: числото 31 се представя по 286 начина като сбор от кубове на цели положителни числа.

Анализ на времевата сложност на алгоритъма:
Тялото на вътрешния цикъл (while) се изпълнява, докато е в сила неравенството $r = p^3 \le k$, което е равносилно на $p \le \sqrt[3]{k}$, тоест p приема следните стойности: 1, 2, 3, . . . , $\left\lfloor \sqrt[3]{k} \right\rfloor$. Следователно тялото на вътрешния цикъл се изпълнява $\left\lfloor \sqrt[3]{k} \right\rfloor$ пъти за всяко цяло k от 1 до n вкл.; границите на k определяме от външния цикъл (for). Следователно тялото на вътрешния цикъл се изпълнява общо (тоест за целия алгоритъм) $\sqrt[3]{1} + \sqrt[3]{2} + \sqrt[3]{3} + \cdots + \sqrt[3]{n} = \Theta\left(n . \sqrt[3]{n}\right)$ пъти. Това е времевата сложност на алгоритъма: $\Theta\left(n . \sqrt[3]{n}\right)$.

Забележки по решението на задача 4. По тази задача са възможни различни оптимизации. Тук ще ги обсъдим накратко.

Щом възрастите на децата са цели числа от 0 до 17 включително, то по принципа на Дирихле следва, че ако има поне 73 деца (а по условие те са няколкостотин), то между тях ще се намерят поне пет на еднаква възраст. Това опростява втората фаза на алгоритъма (същинското сортиране): не е нужно да променяме масива $A[1...n]$. Вместо това е достатъчно да обходим списъците $C[p]$ един по един, докато открием списък с дължина поне 5 (непременно има такъв). Така спестяваме предпоследния цикъл — обхождането на сортирания масив $A[1...n]$ в търсене на най-малка възрастова разлика. Времевата сложност на алгоритъма остава линейна: $\Theta(n)$, тоест не се променя по порядък, обаче константният множител пред n намалява, тъй като отпадат някои операции от алгоритъма. Затова тази малка оптимизация носи допълнителни 3 точки.

Много по-съществена оптимизация на времевата сложност — по порядък! — ще постигнем, ако се сетим, че е достатъчно да изследваме възрастите само на първите 73 деца: както казахме, между тях със сигурност има поне пет на една и съща възраст. Как точно ще ги изследваме, е подробност. Важното е, че броят им 73 не зависи от n (и е много по-малък от няколкостотин), така че този алгоритъм има константна времева сложност: $\Theta(1)$. Откриването на това решение удвоява точките (за всяко от двете подусловия).

Обратно, следното решение, въпреки че има линейна времева сложност, е непрофесионално, затова носи само 4 точки (по 2 точки за всяко подусловие): 18 пъти обхождаме масива $A[1...n]$ — по веднъж за всяка възможна възраст. Спираме търсенето при първото обхождане, при което успеем да открием пет деца на съответната възраст. В най-лошия случай това обхождане ще бъде последното (осемнадесетото). Сложността е линейна ($18n$), обаче константният множител е голям (правим твърде много обхождания на масива).