Изпит по “Дизайн и анализ на алгоритми” — СУ, ФМИ, 5 юли 2018 г.

Задача 1. Намерете асимптотичния порядък на времевата сложност на алгоритъма:

```
ALG(n: positive integer)
s ← 0
i ← n
while i > 0 do
   for j = 1 to i do
      s ← s + 1
   i ← ⌊i/2⌋
return s
```

Задача 2. Даден е числов масив $A[1 \ldots n]$ с различни елементи и две цели числа k и p, $1 \le k < p \le n$. Съставете алгоритъм с времева сложност $O(n)$ в най-лошия случай, пресмятащ сбора на числата от масива A, които по големина са от № k до № p вкл.

Задача 3. Дадени са k сортирани масива с общо n елемента. Предложете алгоритъм с времева сложност $O(n \log k)$, който слива дадените масиви в един сортиран масив.

Задача 4. Дадени са n обекта и разстоянията между всеки два от тях. Дадено е също цяло число k от 2 до n вкл. Постройте алгоритъм с максимално време $O(n^2 \log n)$, разделящ обектите на k групи тъй, че разстоянията вътре във всяка група да са “малки”, а между групите да са “големи” в някакъв разумен смисъл. Анализирайте времевата сложност на алгоритъма и определете в какъв смисъл разстоянията са големи и малки.

Примери: групиране на селища в общини, групиране на звезди в съзвездия и т.н.

Упътване: Моделирайте входните данни чрез граф и използвайте променена версия на някой от известните алгоритми върху графи.

Задача 5. Внучка наследява от баба си винарска изба. В избата има n бутилки вино, наредени в редица и номерирани от 1 до n. Цените им са цели неотрицателни числа, дадени в масив $P[1 \ldots n]$. Колкото повече отлежават бутилките, толкова по-скъпи стават: цената на бутилка № k след x години ще е равна на $x \cdot P[k]$. В завещанието си бабата е поискала внучката да продава по една бутилка всяка година, 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? Смятаме, че бутилките са отлежали една година, когато се продава първата от тях.

Пример: Най-добрият начин да продадем бутилки с цени 10, 40, 20 и 30 долара:
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.

Приемливи са алгоритми с времева сложност $O(n^2)$ в най-лошия случай.

Задача 6. Разглеждаме следната задача за разпознаване, да я наречем `Problem6`:
Вход: неориентиран нетегловен граф G с n върха и m ребра.
Въпрос: Може ли върховете на графа да се номерират с целите числа от 1 до n така, че номерата на двата края на всяко ребро да не са последователни цели числа?
Докажете, че задачата `Problem6` е NP-трудна.

РЕШЕНИЯ

Задача 1. За всяко `i` присвояването `s ← s + 1` се изпълнява точно `i` пъти. Затова сложността на алгоритъма е сборът от всички стойности на `i`:

$$n + \left\lfloor \frac{n}{2} \right\rfloor + \left\lfloor \frac{n}{4} \right\rfloor + \left\lfloor \frac{n}{8} \right\rfloor + \cdots$$

Тази сума е крайна, защото от известно място нататък всички събираеми стават нули. За да получим оценка отгоре, махаме закръгленията и взимаме безкрайната сума:

$$n + \frac{n}{2} + \frac{n}{4} + \frac{n}{8} + \cdots = 2n \text{ (сума на безкрайна геометрична прогресия).}$$

Оценка отдолу получаваме от първото събираемо: щом събираемите са неотрицателни, то сумата им е по-голяма от всяко от тях, вкл. първото. Следователно търсената сума се намира между числата n и $2n$, поради което тя има порядък n, тоест $T(n) = \Theta(n)$.

Задача 2 се решава за линейно време $\Theta(n)$ чрез алгоритъма `PICK`.

```
SUM(A[1...n])
f ← PICK(A[1...n],k)
g ← PICK(A[1...n],p)
s ← 0
for i ← 1 to n do
   if (A[i] ≥ f) and (A[i] ≤ g)
      s ← s + A[i]
return s
```

Цикълът и двете извиквания на `PICK` изразходват линейно време, а инициализацията на `s` — константно време. Затова общото време на `SUM` е линейно: $T(n) = \Theta(n)$.

Задача 3. Използваме двоична пирамида с k елемента, на чийто връх стои най-малкият елемент на пирамидата. Отначало слагаме в пирамидата първия елемент на всеки масив. По-нататък повтаряме многократно следната стъпка: извличаме най-малкия елемент на пирамидата, печатаме го на изхода на алгоритъма и добавяме в пирамидата следващия елемент от масива, на който е принадлежал отпечатаният елемент.

Коректността на построения алгоритъм следва от свойството на двоичната пирамида да пази на върха своя най-малък елемент. Затова на всяка стъпка се печата най-малкият от оставащите елементи. Ето защо алгоритъмът печата елементите в нарастващ ред.

Бързодействието на алгоритъма се определя от това, че за двоична пирамида всяка от двете операции добавяне на елемент и изтриване на най-малкия елемент отнема време $\Theta(\log k)$, където k е броят на елементите на пирамидата. Тези операции се изпълняват общо n пъти (по веднъж за всеки от елементите на входните масиви), затова общото време е $\Theta(n \log k)$.

Задача 4. Променяме алгоритъма на Крускал: след сортирането на ребрата по тегла добавяме ребро не $n-1$ пъти, както е в стандартната версия, а само $n-k$ пъти. Получава се минимална покриваща гора от k дървета, които са търсените k групи.

Тъй като алгоритъмът на Крускал добавя ребрата в нарастващ ред на теглата им, то добавените $n-k$ ребра, съставящи дърветата, имат по-малки тегла от пропуснатите $k-1$ ребра, които биха свързали многото дърветата в едно дърво. В този смисъл можем да твърдим, че разстоянията между върховете в една група (дърво) са малки, докато разстоянията между групите (т.е. между дърветата) са големи.

Анализ на времевата сложност на алгоритъма: Входът на алгоритъма е пълен граф с n върха и $m = \Theta\left(n^2\right)$ ребра. Определящ е бавният етап — сортирането на ребрата по дължина. Затова сложността е равна на $\Theta(m \log m) = \Theta(n^2 \log n)$.

Задача 5 се решава с помощта на *динамично програмиране.* За целта намираме отговорите на всички въпроси от вида "Колко най-много може да спечели внучката, ако разполага само с бутилките от № left до № right, където $1 \le \text{left} \le \text{right} \le n$? "

```
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 ⇔ лявата
for left ← 1 to n do
   right ← left
   year ← n
   dyn[left,right] ← year × P[left]
   taken[left,right] ← true
for year ← n-1 downto 1 do
   for left ← 1 to year do
      right ← left + n - year
      winL ← year × P[left]  + dyn[left+1,right]
      winR ← year × P[right] + dyn[left,right-1]
      if winL > winR // внучката продава лявата бутилка
         dyn[left,right] ← winL
         taken[left,right] ← true
      else // внучката продава дясната бутилка
         dyn[left,right] ← winR
         taken[left,right] ← false
left ← 1
right ← n
while left ≤ right do
   if taken[left,right]
      print "left"
      left ← left + 1
   else
      print "right"
      right ← right - 1
return dyn[1,n]
```

Сложността на този алгоритъм по време и памет е $\Theta(n^2)$, колкото е размерът на динамичната таблица.

Естествено решение изглежда *алчният алгоритъм:* продаваме скъпите бутилки възможно най-късно, т.е. от двете крайни бутилки винаги продаваме първо по-евтината. Този алгоритъм е още по-бърз (с линейна сложност), но той, за съжаление, е погрешен.

Контрапример: 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.

Задача 6. Извършваме редукция от известната NP-пълна задача `Хамилтонов_път`:
Вход: неориентиран нетегловен граф G с n върха и m ребра.
Въпрос: Графът G съдържа ли хамилтонов път?

Описание на редукцията:

```
Хамилтонов_път (G: неориентиран нетегловен граф)
1) Образуваме допълнението на множеството от ребрата на графа G,
   тоест изтриваме наличните ребра и добавяме липсващите.
2) return Problem6 (G)
```

Коректност на редукцията:
Функцията `Хамилтонов_път` (от първоначалния граф `G`) връща стойност “истина”.

⇳ (от ред № 2 на алгоритъма)

Функцията `Problem6` (от променения граф `G`) връща стойност “истина”.

⇳ (от определението на задачата `Problem6`)

Върховете на променения граф `G` могат да се номерират с целите числа от 1 до n така, че всеки два върха, номерирани с последователни цели числа, да не са свързани с ребро.

⇳ (от ред № 1 на алгоритъма)

Върховете на първоначалния граф `G` могат да се номерират с целите числа от 1 до n така, че всеки два върха, номерирани с последователни цели числа, да са свързани с ребро.

⇳ (от определението за хамилтонов път)

Първоначалният граф `G` съдържа хамилтонов път.

С това е доказано, че функцията `Хамилтонов_път (G)` връща стойност “истина” ⇔ неориентираният нетегловен граф `G` съдържа хамилтонов път. Това точно съответства на постановката на задачата `Хамилтонов_път`. Следователно редукцията е коректна.

Бързина на редукцията: Редукцията се състои само от образуването на допълнението на множеството от ребрата на графа. Така за всеки връх се обработват $n - 1$ ребра, а общо за целия граф броят на операциите добавяне и изтриване на ребро е равен на $n\,(n-1) \;=\; n^2 - n \;=\; \theta(n^2)$. Тоест времевата сложност на редукцията е квадратична, следователно полиномиална.