КОНТРОЛНО № 6 ПО ДИЗАЙН И АНАЛИЗ НА АЛГОРИТМИ (9. VI. 2022 Г.)
ЗА СПЕЦИАЛНОСТ "КОМПЮТЪРНИ НАУКИ" — СУ, ФМИ, II КУРС, I ПОТОК

Задача 1 (анализ на алгоритъм).

```
ALG(n: цяло положително число)
k ← 3
j ← 3
while k ≤ n do
   while j < k do
      print "a"
      j ← j × j
   k ← k × k
```

Намерете асимптотично точна оценка за времевата сложност на алгоритъма.

(20 точки)

Задача 3 (динамично програмиране). Даден е масив $A[1 \ldots n]$ от положителни реални числа и метален прът с дължина n. Искаме да нарежем пръта на части (може и само една част, т.е. можем да не режем) по такъв начин, че сборът от цените на всички парчета да е възможно най-голям. Дължините на парчетата са цели числа и $A[k]$ е цената на парче с дължина k. Съставете алгоритъм, намиращ отговора за време $O(n^2)$ при всякакви входни данни. Опишете алгоритъма на псевдокод или Си и го демонстрирайте при $A = (1; 7; 12; 3)$.

(10 точки)

Още 10 т. за намиране на едно оптимално разрязване на пръта. **(10 точки)**

Отново е нужна демонстрация и код на Си или на псевдокод.

Задача 4 (алгоритми върху графи). Ползвайки известна алгоритмична схема, опишете словесно как за време $O(m + n)$ се намира ред за изтриване на върховете на свързан граф с n върха и m ребра така, че след всяко изтриване на връх остава свързан граф. **(30 точки)**

Задача 2 (търсене). С помощта на текстообработваща програма набирате документ с двустранно подравняване на текста и с размер на шрифта n. Последната написана дума е твърде дълга и автоматично минава на следващ ред, при което се раздалечават (пак автоматично) всички думи от текущия ред. Ако смалите интервала пред последната дума до размер на шрифта 1, тогава тя се побира на предишния ред, но изглежда залепена за думата преди нея. С цел добър външен вид на документа искате да намерите най-големия размер k на шрифта, при който последната дума няма да се прехвърли на следващия ред. Можете единствено да опитвате размери на шрифта — цели числа от 1 до n вкл. Опишете на езика Си или на псевдокод бърз алгоритъм за търсене на k, като алгоритъмът притежава следната спецификация: findMaxFontSize (n: integer): integer (тоест приема и връща цяло число, по-голямо от нула). Алгоритъмът може да се обръща единствено към функцията test (x: integer): boolean, която връща "лъжа" (false), когато при интервал с размер на шрифта x последната дума отива на нов ред, и "истина" (true), когато думата се побира на текущия ред. Приема се само възможно най-бърз алгоритъм за търсене на максималния размер на шрифта. **(10 точки)**

Задача 5 (долна граница $n \log n$). Докажете, че разпознаването дали в даден масив с n реални числа има четирикратно повторение, изисква време $\Omega(n \log n)$. **(30 точки)**

РЕШЕНИЯ

Задача 1. Целочислените променливи j и k се изменят по един и същ начин: растат, започвайки от 3, и на всяка стъпка се повдигат на квадрат. Освен това при всяко изпълнение на тялото на външния цикъл k изпреварва j с една стъпка, а при всяко изпълнение на тялото на вътрешния цикъл j настига k, ето защо за всяка стойност на k след първата (тоест за всяка стойност, по-голяма от 3) тялото на вътрешния цикъл се изпълнява веднъж (то не се изпълнява при $k = 3$). Затова времевата сложност на целия алгоритъм е равна по порядък на броя на различните стойности на k.

Не е трудно да се докаже с математическа индукция, че след r изпълнения на тялото на външния цикъл променливата k притежава стойност 3^{2^r} (при $r = 0$ от тази формула се получава началната стойност на k). Следователно r се мени от нула до някаква максимална стойност, която се определя от неравенството $k \le n \Leftrightarrow 3^{2^r} \le n \Leftrightarrow 2^r \le \log_3 n. \Leftrightarrow r \le \log_2 \log_3 n$. Понеже r е цяло число, то приема $\lfloor \log_2 \log_3 n \rfloor + 1$ стойности, следователно времевата сложност на алгоритъма от задача 1 е от порядък $\Theta(\log \log n)$.

Задача 2 се решава с помощта на ***двоично търсене:***

```
findMaxFontSize(n)
1)   K ← 1
2)   L ← 2
3)   H ← n-1
4)   while L ≤ H
5)      M ← ⌊(L+H) / 2⌋
6)      if test(M)
7)         L ← M+1
8)         K ← M
9)      else
10)        H ← M-1
11)  return K
```

Този алгоритъм има времева сложност $\Theta(\log n)$ при всякакви входни данни, тъй като завършва чак когато изчерпи неизследвания интервал (от L до H вкл.), чиято дължина, започвайки от $n - 2$, намалява около два пъти на всяка стъпка. Алгоритъмът е най-бърз по порядък; това се доказва с игра срещу противник: за всяко извикване `test(M)` противникът (тоест функцията `test`) връща "истина" точно когато M е по-близо до L, отколкото до H. Така противникът гарантира, че дължината на интервала намалява най-много двойно (плюс малка константа), затова са нужни $\Omega(\log n)$ извиквания на функцията `test`.

Задача 3 се решава с динамично програмиране:

```
optimalCut(A[1...n]: реални положителни числа)
1)   dyn[1...n]: реални положителни числа
2)   last[1...n]: цели положителни числа от 1 до n вкл.
3)   for k ← 1 to n do
4)      dyn[k] ← A[k]
5)      last[k] ← k
6)      for i ← 1 to k-1 do
7)         current ← A[i]+dyn[k-i]
8)         if current > dyn[k]
9)            dyn[k] ← current
10)           last[k] ← i
11)  k ← n
12)  while k > 0 do
13)     print last[k]
14)     k ← k - last[k]
15)  return dyn[n]
```

Времевата сложност е $\Theta(n^2)$ при всякакви данни заради вложените цикли; dyn[k] = най-голямата печалба, която може да се получи от прът с дължина k; last[k] = дължината на последното парче, което се отрязва от прът с дължина k за най-голяма печалба (понеже редът на парчетата няма значение, то може да е и първото парче). Ред № 13 отпечатва оптималните дължини на парчетата.

П р и м е р : При A = (1; 7; 12; 3) помощните масиви изглеждат така:

k	1	2	3	4
dyn[k]	1	7	12	14
last[k]	1	2	3	2

Най-голямата възможна печалба от разрязване на прът с дължина 4 е 14; тя се постига, като го разрежем на две парчета с дължина 2; от тях получаваме 7 + 7 = 14 единици печалба.

Задача 4. Обхождаме графа в дълбочина и подреждаме върховете му в реда на затварянето. В този ред трябва да ги изтрием. При такова обхождане всеки връх се затваря преди родителя си в дървото на обхождането T, тоест върховете ще се изтриват от листата към корена на T и остатъкът T' на T винаги ще бъде дърво. В остатъка G' на G ще има път от всеки до всеки връх, например пътя от ребрата на T'. Затова остатъкът G' винаги е свързан граф.

Задача 5. Използваме редукция от задачата “Липса на повторения”, която има времева сложност $\Omega(n \log n)$, както е доказано в теорията.

```
Липса_на_повторения(A[1...n])
1)   B[1...2n]
2)   for k ← 1 to n do
3)      B[2k-1] ← A[k]
4)      B[2k]   ← A[k]
5)   return Липса_на_четирикратни_повторения(B[1...2n])
```

Коректност на редукцията: Редове № 3 и № 4 удвояват абсолютните честоти на всички елементи. Ето защо масивът A съдържа (обикновено) повторение тогава и само тогава, когато масивът B съдържа четирикратно повторение. Затова ред № 5 основателно обявява върнатата стойност на едната задача за върната стойност и на другата задача. Тоест описаната редукция е коректна.

Бързина на редукцията: Цикълът на редове № 2, № 3 и № 4 обхожда веднъж масивите, поради което времевата сложност на редукцията е $\Theta(n) = o(n \log n)$, тоест редукцията е достатъчно бърза за целите на доказателството.

ТОЧКУВАНЕ

Задача 1 носи общо 20 точки — по 4 точки за всяка от следните стъпки:
— установяване, че тялото на вътрешния цикъл се изпълнява веднъж при всяко изпълнение на тялото на външния цикъл;
— извод, че времевата сложност на алгоритъма е равна по порядък на броя на изпълненията на тялото на външния цикъл;
— намиране на формула за общия член на редицата от стойностите на k;
— съставяне и решаване на подходящо неравенство относно k;
— определяне на времевата сложност на алгоритъма.

Задача 2 носи 10 точки само ако е решена изцяло.

Задача 3 носи 20 точки — по 10 точки на подусловие, които на свой ред се разпределят така: 5 т. за съставяне на правилен алгоритъм (с програмен код) и 5 т. за демонстрацията на алгоритъма (проиграването му върху примерните входни данни от условието на задачата).

Задача 4 носи 30 точки само ако е решена изцяло.

Задача 5 носи общо 30 точки — по 10 точки за всяка от следните стъпки:
— описание на редукцията на псевдокод;
— доказателство за коректност на редукцията;
— анализ на времевата сложност на редукцията.