КОНТРОЛНО № 3 ПО ДИЗАЙН И АНАЛИЗ НА АЛГОРИТМИ (СУ, ФМИ, ЗИМЕН СЕМЕСТЪР 2020 / 2021 Г.)

Указания:

1) Всички решения да бъдат обосновани подробно.
2) Имената на алгоритмите и структурите от данни да са на български език.

Задача 1. Разглеждаме аритметични изрази — сборове от положителни цели числа, записани в двоична бройна система, например “1001+1+10+101”. Всеки такъв израз се състои само от цифрите нула и единица и знака плюс. Знакът плюс не е задължителен: сбор може да съдържа единствено събираемо, например “1010111”. Сборът не може да бъде празен (“ ”), тоест той трябва да има поне едно събираемо. Двоично число не може да започва с цифрата 0 (само числото 0 започва така, но то е изключено по условие, защото не променя стойността на сбора). Двоично число не може да бъде празно; с други думи, низове като следните не са допустими: “101++10” (липсва второто събираемо), “+100” (липсва първото събираемо), “11+10+” (липсва последното събираемо). Кавичките не са част от аритметичния израз, а само го отделят от другия текст. Не слагаме интервали около знака плюс.

а) Съставете два алгоритъма, които по дадено цяло положително число n намират броя на сборовете от описания вид, съставени от n знака. **2 точки**
Единият алгоритъм да използва степенуване на матрицата на подходящ граф, а другият алгоритъм да използва схемата динамично програмиране. Намерете порядъците на времевите сложности на двата алгоритъма като функции на n и сравнете двете времена, тоест кажете кой от двата алгоритъма е по-бърз при големи n. Демонстрирайте алгоритмите при $n = 5$; довършете сметките до отговор, който представлява едно число — броят на аритметичните изрази, съставени от пет знака по описаните правила.

б) Предложете алгоритъм, който по даден низ, съставен само от знаци плюс и цифрите 0 и 1, разпознава дали даденият низ представлява аритметичен израз от описания вид (с произволна дължина). Приемат се всякакви алгоритми, чиято времева сложност е $O(n^3)$ при най-лоши входни данни, където n означава дължината на входа (броя на знаците в низа). **1 точка**

Задача 2. Даден е масивът $P[1\ldots n]$ от n точки в I квадрант на равнината, тоест всичките им координати са положителни числа (цели или дробни). Искаме да подредим точките в масива по такъв начин, че при преминаване от индекса k към индекса $k+1$ радиус-векторът на точката да се завърта само в посока обратна на посоката на движение на часовниковата стрелка, тоест най-краткото завъртане от вектора $\overrightarrow{OP_k}$ към вектора $\overrightarrow{OP_{k+1}}$ да е обратно на движението на часовниковата стрелка, където долният индекс показва мястото на точката в масива след подреждането му, а пък точката $O(0\,;\,0)$ е началото на координатната система.

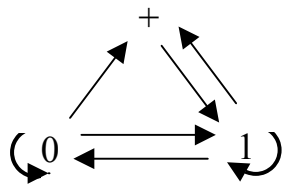
Докажете, че тази алгоритмична задача има времева сложност $\Omega(n \log n)$ при най-лоши входни данни, ако се решава чрез сравнения. **1 точка**

Оценката = 2 + броя на точките.

РЕШЕНИЯ

Задача 1.

а) *Първи начин:* Разглеждаме граф, чиито върхове съответстват на знаците, допустими в аритметичните изрази — цифрите нула и единица и знака плюс. Графът е ориентиран: ребро от u към v означава, че след знака u можем да поставим знака v.



Върховете 0 и 1 имат примки, защото всеки от тях може да се повтаря поред. На знака плюс няма примка, защото той не може да се повтаря последователно.

Матрицата на съседство на този граф изглежда така:

$$A = \begin{matrix} \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{pmatrix} & \begin{matrix} 0 \\ 1 \\ + \end{matrix} \\ \begin{matrix} 0 & 1 & + \end{matrix} & \end{matrix}$$

Елементите на матрицата на съседство притежават обичайните си значения: единицата означава, че има ребро, а нулата означава, че в графа няма ребро от върха, съответстващ на реда, към върха, съответстващ на стълба на елемента.

Аритметичните изрази, съставени от n знака по правилата на задачата, се представят в графа като пътища с n върха, следователно с $n-1$ ребра; но само такива пътища, които започват с единица и не завършват със знак плюс. Преброяваме пътищата с дължина $n-1$, повдигайки матрицата на степен $n-1$. В степенуваната матрица събираме първите две числа от втория ред, защото броим изразите, които започват с единица и завършват на нула или единица.

Анализ на алгоритъма: Като използваме бързо степенуване, правим само $\Theta(\log(n-1)) = \Theta(\log n)$ умножения на матрици. Матриците са от тип 3×3, значи всяко умножение изисква константно време и общото време е $\Theta(\log n)$.

Пример: При $n = 5$ търсим $A^{n-1} = A^4$:

$$A^2 = \begin{matrix} \begin{pmatrix} 2 & 3 & 2 \\ 2 & 3 & 2 \\ 1 & 1 & 1 \end{pmatrix} & \begin{matrix} 0 \\ 1 \\ + \end{matrix} \\ \begin{matrix} 0 & 1 & + \end{matrix} & \end{matrix} \qquad A^4 = \begin{matrix} \begin{pmatrix} 12 & 17 & 12 \\ 12 & 17 & 12 \\ 5 & 7 & 5 \end{pmatrix} & \begin{matrix} 0 \\ 1 \\ + \end{matrix} \\ \begin{matrix} 0 & 1 & + \end{matrix} & \end{matrix}$$

Събираме първите две числа във втория ред на матрицата A^4: $12 + 17 = 29$. Следователно има 29 аритметични израза, съставени от пет знака.

Втори начин: Използваме динамично програмиране. Дефинираме следните три функции:

$f_k(m)$ = броя на аритметичните изрази, съставени от m знака, първият от които е $k \in \{ 0 ; 1 ; + \}$, а последният не е плюс.

Имаме нужда от начални условия и рекурентни уравнения, за да пресметнем трите функции.

Начални условия: $f_0(1) = 1$; $f_1(1) = 1$; $f_+(1) = 0$. Причината е, че когато дължината на изразите е единица, има само два израза, незавършващи с плюс: аритметичните изрази "0" и "1".

Рекурентните уравнения въплъщават правилата за следване на знаците:

$$f_0(m + 1) = f_0(m) + f_1(m) + f_+(m);$$
$$f_1(m + 1) = f_0(m) + f_1(m) + f_+(m);$$
$$f_+(m + 1) = f_1(m).$$

Тези уравнения важат за всяко цяло число $m \geq 1$. С тяхна помощ пресмятаме стойностите на трите функции за всяка целочислена стойност на аргумента от 2 до n включително. За всяка конкретна стойност на аргумента извършваме четири събирания и три присвоявания, следователно изразходваме време $\Theta(1)$. Можем да минем само с две събирания, ако забележим, че $f_0(m + 1) = f_1(m + 1)$. Това намалява ненаписаната константа, но не и порядъка. Понеже аргументът се мени от 2 до n включително, прилагаме рекурентните уравнения $n - 1$ пъти, затова времевата сложност на целия алгоритъм е $\Theta(n)$ при всякакви данни. Накрая връщаме $f_1(n)$, защото правилно построените аритметични изрази започват с единица.

П р и м е р : При $n = 5$ динамичната таблица изглежда така:

m	1	2	3	4	5
$f_0(m)$	1	2	5	12	29
$f_1(m)$	1	2	5	12	29
$f_+(m)$	0	1	2	5	12

Отговорът е $f_1(5) = 29$, тоест съществуват двайсет и девет аритметични израза, съставени от пет знака по описаните правила.

Понеже $\log n = o(n)$, по-бърз е първият алгоритъм (който използва граф).

б) Можем да разпознаваме аритметичните изрази от тоя вид за време $O(n^3)$ посредством алгоритъма CYK, като му подадем тази безконтекстна граматика:

<израз> → <събираемо> | <събираемо> + <израз>
<събираемо> → 1 | 1 <двоичен низ>
<двоичен низ> → 1 | 0 | 1 <двоичен низ> | 0 <двоичен низ>

Тази граматика е безконтекстна, но не е в нормална форма на Чомски. Тя може да бъде нормализирана по известния начин.

Разпознаването става много по-бързо — с линейна времева сложност $\Theta(n)$, ако се използва следният алгоритъм:

```
Разпознаване(T[1...n]: масив от знаци)
1) // Връща стойност истина ⇔ низът T се разпознава.
2) waiting1 ← true
3) for k ← 1 to n do
4)     if T[k] ≠ '1' and T[k] ≠ '0' and T[k] ≠ '+'
5)         return false
6)     if waiting1 and T[k] ≠ '1'
7)         return false
8)     waiting1 ← (T[k] = '+')
9) return not waiting1
```

Този алгоритъм е верен и бърз, затова носи допълнителна точка, тоест той се оценява с 2 точки.

Задача 2. Желаната долна граница $\Omega(n \log n)$ се доказва чрез редукция от задачата за сортиране на положителни реални числа (цели или дробни), която при най-лоши входни данни изразходва време $\Omega(n \log n)$, ако се решава чрез сравнения. Описваме редукцията на псевдокод:

```
Сортиране(A[1...n]: масив от реални положителни числа)
1) P[1...n]: масив от точки в I квадрант
2) for k ← 1 to n do
3)     P[k].x ← 1
4)     P[k].y ← A[k]
5) Нареждане на точки в I квадрант(P[1...n])
6) for k ← 1 to n do
7)     A[k] ← P[k].y
```

Коректност на редукцията: За точки от I квадрант с равни абсциси ($x = 1$ според ред № 3) нареждането на радиус-векторите им обратно на часовника е равносилно на подреждането на точките по нарастване на ординатите, тоест равносилно е на сортиране на масива от ординатите на точките.

Бързина на редукцията: Редукцията на входа (редове № 2, № 3 и № 4) е цикъл по брояч, чието тяло се изпълнява точно n пъти. Същото наблюдение важи за редукцията на изхода (редове № 6 и № 7). Следователно редукцията изразходва общо време от порядък $\Theta(n) = o(n \log n)$, т.е. тя е достатъчно бърза за целите на доказателството.

СХЕМА ЗА ТОЧКУВАНЕ

Задача 1:

а) Първото подусловие носи 2 точки — по 0,95 точки за всеки алгоритъм, ако е подробно описан (0,45 т.), анализиран (0,05 т.) и демонстриран (0,45 т.); и 0,10 точки за сравняването на двата алгоритъма по бързодействие — от които 0,05 т. за асимптотичното сравняване на сложностите и 0,05 т. за тълкуването на получения резултат (кой от двата алгоритъма е по-бърз).

б) Второто подусловие носи 1 точка, ако е решено вярно, тоест ако е избран алгоритъмът CYK и е съставена безконтекстна граматика (тя може да не е в нормална форма на Чомски). Дава се допълнителна точка (т.е. общо 2 точки) за верен алгоритъм с линейна времева сложност.

Задача 2 носи 1 точка само ако е решена изцяло — описание на редукцията, доказателство за коректност и анализ на бързодействието. Частични решения не носят точки.

Оценката = 2 + броя на точките.