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. Пешеходец иска да стигне от едно място до друго, но се пази от силното лятно слънце. Затова той се стреми да ходи в сенките на дърветата. Съставете алгоритъм, който за време $O(n^2)$ намира маршрут с най-малка обща дължина на слънчевите участъци. Вход на алгоритъма:
X [1...n], Y [1...n], R [1...n] — центрове и радиуси на сенките на дърветата (смятаме ги за кръгове);
X_start, Y_start — абсцисата и ординатата на началната точка на маршрута;
X_end, Y_end — абсцисата и ординатата на крайната точка на маршрута.
а) Опишете алгоритъма с думи. **(5 точки)**
б) Анализирайте времевата сложност на алгоритъма. **(4 точки)**

Задача 2. Дадени са дължините X [1...n] и ширините Y [1...n] на n пощенски плика.
Ще смятаме, че плик № i може да бъде поставен в плик № j тогава и само тогава, когато
X [i] < X [j] и Y [i] < Y [j] или X [i] < Y [j] и Y [i] < X [j].
Да се състави алгоритъм, който за време $O(n^2)$ избира възможно най-много пликове, които могат да се вложат последователно, т.е. първият избран плик може да се сложи във втория избран плик; вторият — в третия; третият — в четвъртия; и тъй нататък.
а) Опишете алгоритъма с думи. **(5 точки)**
б) Анализирайте времевата сложност на алгоритъма. **(4 точки)**

Задача 3. Числовият масив $A[1...n]$ съдържа дължините на n филма (в секунди). Имаме M секунди и искаме да изгледаме възможно най-голям брой филми за това време. Предложете алгоритъм за подбор на филмите с времева сложност $O(n)$ в най-лошия случай.
а) Опишете алгоритъма с думи. **(5 точки)**
б) Анализирайте времевата сложност на алгоритъма. **(4 точки)**

Задача 4. Масивът от дробни положителни числа $A[1...n]$ съдържа радиусите на n звезди. Астрономът, извършил наблюденията и пресмятанията, предполага, че някои от тези обекти може да са двойници — образи на една и съща звезда, създадени от гравитационна леща. Помогнете на астронома да намери предполагаемите двойници, като съставите алгоритъм, който проверява за време $O(n \log n)$ в най-лошия случай дали измежду дадените радиуси има два, чиято разлика не надхвърля 3% от по-малкия радиус. (Ако има няколко подходящи двойки звезди, достатъчно е алгоритъмът да изведе една от тези двойки.)
а) Опишете алгоритъма на псевдокод. **(5 точки)**
б) Анализирайте времевата сложност на алгоритъма. **(4 точки)**

Задача 5. Предложете алгоритъм, който за време $o(n^2)$ намира най-малкия брой точни кубове със сбор n. Сборът и събираемите са естествени числа.
а) Опишете алгоритъма на псевдокод. **(5 точки)**
б) Направете подробна демонстрация на алгоритъма за $n = 38$. **(3 точки)**
в) По време на демонстрацията за $n = 38$ обяснете с думи и покажете как можем да намерим и самите събираеми, а не само техния брой. **(2 точки)**
г) Анализирайте времевата сложност на алгоритъма. **(4 точки)**

Р Е Ш Е Н И Я

Задача 1. Изчисляваме разстоянията между всеки два кръга, вкл. началната и крайната точка (които тълкуваме като кръгове с нулеви радиуси). Този етап изисква време $\Theta(n^2)$. След това търсим най-къс път по ***алгоритъма на Дейкстра.*** Ако не използваме приоритетна опашка, то времето на втория етап също е $\Theta(n^2)$, следователно времето на целия алгоритъм е $\Theta(n^2)$. Ако използваме приоритетна опашка, реализирана чрез пирамида на Фибоначи, тогава времето на втория етап е $\Theta(m + n \log n)$, следователно времето на целия алгоритъм отново е $\Theta(n^2)$. Не бива да използваме реализацията с двоична пирамида, защото тогава времето на втория етап става $\Theta((m + n) \log n)$, което при плътни графи, тоест при $m = \Theta(n^2)$, какъвто е нашият граф, надхвърля зададеното ограничение на времето: $(m + n) \log n = \Theta(n^2 \log n) = \omega(n^2)$.

Задача 2. За време $\Theta(n^2)$ проверяваме всеки два плика: дали единият може да се сложи в другия. Същевременно построяваме граф с n върха и m ребра: върхове ма графа са пощенските пликове, а ребра са откритите възможности за влагане. По-точно, добавяме ребро от връх № i към връх № j тогава и само тогава, когато плик № i може да бъде сложен в плик № j. Със сигурност се получава ***ориентиран ацикличен граф,*** защото всеки път в графа съответства на строго растяща редица от лица на пощенски пликове, а никоя строго растяща редица не може да съдържа повторения. Върху този граф извършваме ***топологично сортиране*** по схемата ***обхождане в дълбочина.*** Най-дълга редица от вложени пощенски пликове съответства на най-дълъг път в получения граф. Търсим ***най-дълъг път*** в ориентирания ацикличен граф с помощта на ***динамично програмиране.*** Топологичното сортиране и динамичното програмиране изразходват време $\Theta(m + n) = O(n^2)$. Така общото време на алгоритъма е $\Theta(n^2)$.

Задача 3. За да изгледаме възможно най-много филми за даденото време M, избираме най-късите. Чрез сортиране на филмите не можем да постигнем линейна сложност, затова се отказваме от сортирането. Вместо това използваме ***алгоритъма PICK в съчетание с двоично търсене.*** По-конкретно:

1) Намираме медианата на дължините на филмите посредством алгоритъма PICK.
2) Разделяме филмите на къси и дълги спрямо медианата: пренареждаме масива, като слагаме късите филми вляво от медианата, а дългите — вдясно. Ще смятаме и медианата за къс филм. Ако има няколко филма с дължини, равни на медианата, то слагаме някои от тях сред късите, а други — сред дългите филми, така че двата подмасива (на късите и на дългите филми) да имат равен или почти равен брой елементи (дължините на двата подмасива да се различават най-много с единица).
3) Пресмятаме сбора S от дължините на късите филми. Разглеждаме три случая:
   а) Ако $S = M$, то избираме да гледаме всички къси филми и само тях; край на алгоритъма.
   б) Ако $S > M$, то времето, с което разполагаме, е недостатъчно дори за късите филми. Затова се отказваме от дългите филми и изпълняваме алгоритъма рекурсивно върху масива от късите филми, като запазваме стойността на M.
   в) Ако $S < M$, то времето, с което разполагаме, е достатъчно за всички къси филми, а вероятно ще остане време и за някои от дългите филми. Затова решаваме, че ще гледаме всички къси филми, и изпълняваме алгоритъма рекурсивно върху масива от дългите филми, като подаваме $M - S$ вместо M. Тоест времето, което ни остава, след като сме изгледали всички къси филми, служи като ограничение при избора на дълги филми.

Дъното на рекурсията е при $n \le 2$. Ако $n = 0$, алгоритъмът ще връща празен списък При $n = 1$ ще гледаме единствения филм само ако дължината му не надвишава M. При $n = 2$ гледаме по-късия филм, ако дължината му не надвишава M; а после — и по-дългия, стига дължината му да не надхвърля оставащото време (M минус дължината на по-късия филм).

Търсим времевата сложност $T(n)$ на този алгоритъм в най-лошия случай: когато алгоритъмът не попада в подточка "а" на точка 3. Функцията $T(n)$ удовлетворява рекурентното уравнение

$$T(n) = T\left(\frac{n}{2}\right) + \Theta(n).$$

Делението на две идва от това, че при рекурсията дължината на масива намалява два пъти. Това е така, защото разделянето на филмите на къси и дълги се извършва относно медианата. Коефициентът пред неизвестната функция в дясната страна на уравнението е единица, понеже всеки път се изпълнява само едно от двете рекурсивни извиквания в подточки "б" и "в" на т.3: не е възможно сборът S да бъде едновременно и по-голям, и по-малък от стойността на M. Свободният член на рекурентното уравнение е общото време, изразходвано от алгоритъма PICK, разделянето на филмите на къси и дълги, събирането на дължините, маркирането на филми дали ще бъдат гледани. Всяка стъпка изисква линейно време, затова и общото време е линейно: $\Theta(n)$.

Полученото рекурентно уравнение решаваме с мастър-теоремата и намираме $T(n) = \Theta(n)$.

Задача 4. Сортираме дадения масив с някой бърз алгоритъм, например ***пирамидално сортиране.*** Ако масивът съдържа близки стойности, то след сортирането те ще се окажат една до друга. Затова след сортирането е достатъчно да проверим двойките от съседни елементи. Псевдокод:

```
STARS(A[1...n])
HeapSort(A[1...n])
for k ← 2 to n do
   if A[k] < 1,03 × A[k-1]
      return k // подходяща двойка звезди са № k и № k–1
return 0 // няма подходяща двойка звезди
```

Разбиран буквално, този код не работи правилно: алгоритъмът връща индексите след сортирането, а те нямат нищо общо с първоначалните индекси. Подразбира се, че всеки елемент помни първоначалното си местоположение и че именно то е върнатата стойност на алгоритъма. В истински код това може да се постигне така: попълваме един допълнителен масив $B[1...n]$ с числата от 1 до n. После, докато сортираме A, разместваме и съответните елементи на B. Накрая връщаме $B[k]$ и $B[k–1]$ вместо k и $k–1$.

Анализ на времевата сложност на алгоритъма: Най-лошият случай е, когато масивът A не съдържа подходяща двойка елементи или такава двойка образуват двете най-големи числа в масива. Времето за инициализиране на помощния масив B е $\Theta(n)$, пирамидалното сортиране изразходва време $\Theta(n \log n)$, а времевата сложност на цикъла е $\Theta(n)$. Затова общото време е $\Theta(n \log n)$. Очевидно помощният масив B не увеличава порядъка на времевата сложност, поради което не е споменат изрично в псевдокода.

Задача 5 е подобна на задачата за представяне на естествено число като сбор от най-малък брой точни квадрати, която е решена подробно в публикуваните учебни материали. Решава се чрез ***динамично програмиране.*** За числото 38 са нужни поне пет точни куба: $38 = 3^3 + 2^3 + 1^3 + 1^3 + 1^3$. Попълването на клетката за числото k от динамичната таблица изисква $\sqrt[3]{k}$ стъпки — разглеждане на всички възможности за последното събираемо. Общото време, когато k се мени от 1 до n, е $\sqrt[3]{1} + \sqrt[3]{2} + \sqrt[3]{3} + \cdots + \sqrt[3]{n} = \Theta\left(n \,.\sqrt[3]{n}\right)$. На пръв поглед изглежда, че има по-бързо решение — с алчен алгоритъм: на всяка стъпка взимаме най-големия възможен куб. Този алгоритъм е бърз, но не е коректен! Например числото $n = 2000$ алчният алгоритъм представя с шест събираеми: $2000 = 12^3 + 6^3 + 3^3 + 3^3 + 1^3 + 1^3$. Обаче съществува представяне с две събираеми: $2000 = 10^3 + 10^3$.