ПОПРАВИТЕЛЕН ИЗПИТ ПО ДИЗАЙН И АНАЛИЗ НА АЛГОРИТМИ (СУ, ФМИ, СПЕЦИАЛНОСТ "КОМПЮТЪРНИ НАУКИ", I ПОТОК)

27 АВГУСТ 2021 Г.

Указания:

1) Всички алгоритми, изучени на лекциите, могат да се използват наготово.
2) Решенията, включително термините, трябва да бъдат на български език.
3) Една задача се отчита като решена само ако е решена изцяло.

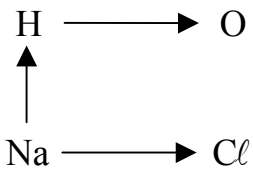
Задача 1. Има ли алгоритъм, който може да сортира за време $o(n \log n)$ произволен масив $A[1 \ldots n]$ от n цели числа, при условие че разликата на всеки два съседни елемента на масива не надхвърля 10 ?

Ако да — предложете бърз алгоритъм и анализирайте времето му.

Ако не — докажете долна граница $\Omega(n \log n)$ за времевата сложност.

Задача 2. Даден е списък с m молекулни формули на химични съединения на общо n химични елемента; всяко съединение е от два химични елемента, например H_2O, NaH, NaCℓ и т.н. Всяка молекулна формула може да се тълкува като правило знакът на кой от два химични елемента се пише преди знака на другия химичен елемент. Например H_2O означава, че H се пише преди O.

Списъкът е конвертиран към граф, чиито върхове съответстват на дадените n химични елемента, а ребрата му представят дадените m молекулни формули. Всяко ребро сочи от първия към втория химичен елемент във формулата. Например списъкът по-горе се представя чрез следния граф ($n = 4$, $m = 3$):



Приемаме, че предхождането на химичните елементи е строга наредба. Всяка такава релация е транзитивна. Щом например Na се пише преди H (NaH) и H предхожда O (H_2O), то Na предхожда O, макар че това съединение (Na_2O) липсва в списъка.

Като използвате наготово алгоритмите, изучени на лекциите, съставете и опишете словесно алгоритъм, който получава като вход граф от описания вид и за време $O(m + n)$ при всякакви входни данни намира двойка върхове, за чиито химични елементи не може да се определи кой преди кой се пише, или установява, че няма такава двойка върхове, или намира противоречие във входните данни. Изтълкувайте израза "противоречие във входните данни". Опишете как алгоритъмът разпознава в кой от трите случая е попаднал. Ако алгоритъмът използва някое от стандартните видове обхождане на граф (в ширина или в дълбочина), уточнете вида на обхождането. Направете анализ на времевата сложност на алгоритъма при най-лоши входни данни.

Задача 3. Съставете възможно най-бърз алгоритъм, изразходващ възможно най-малко допълнителна памет, който по даден масив $A[1 \ldots n]$ от цели числа отпечатва най-дългата му подредица, в която се редуват четни и нечетни числа (няма изискване какво число да бъде първият член на подредицата).

Опишете алгоритъма словесно и на псевдокод (или на Си).

Демонстрирайте алгоритъма с примерни входни данни.

Обосновете коректността на алгоритъма.

Дайте асимптотична оценка на сложността на алгоритъма по време и памет при най-лоши входни данни.

Задача 4. Дадена е двоична матрица $A[1 \ldots n][1 \ldots n]$. Един индекс k (цяло число от 1 до n вкл.) наричаме приятен, ако ред № k съдържа само нули, а стълб № k — само единици (тези изисквания не важат за елемента $A[k][k]$: той може да е произволен). По-формално, индексът k е приятен, ако и само ако $A[k][j] = 0$ за $\forall j \neq k$ и $A[i][k] = 1$ за $\forall i \neq k$. Търсим приятен индекс (ако има). Докажете, че тази алгоритмична задача притежава времева сложност $\Theta(n)$ при най-лоши входни данни, като:

а) съставите алгоритъм с времева сложност $O(n)$ в най-лошия случай;

б) докажете долна граница на времевата сложност $\Omega(n)$ в най-лошия случай.

Време за работа: 120 минути.

СХЕМА ЗА ОЦЕНЯВАНЕ

Всяка задача носи една точка само ако е решена изцяло. Оценката = 2 + k, където k е броят на точките, събрани от всички задачи.

РЕШЕНИЯ

Задача 1. Има алгоритъм с времева сложност $o(n \log n)$ в най-лошия случай. Щом съседните елементи се различават най-много с 10 единици, то следва, че разликата между най-големия и най-малкия елемент не надхвърля $10(n - 1)$. Тоест диапазонът на изменение е малък (в сравнение с n), поради което можем да използваме ***сортиране чрез броене.*** Времевата сложност е $\Theta(n) = o(n \log n)$.

Задача 2. Входните данни съдържат противоречие тогава и само тогава, когато не можем да подредим дадените n химични елемента в желаната редица (указваща кой химичен елемент преди кой се пише в молекулните формули). А това става точно когато графът съдържа ***ориентиран цикъл.*** Един цикъл, ако има такъв, можем да намерим чрез ***обхождане в дълбочина.***

Ако няма противоречие във входните данни, то дадените химични елементи (тоест върховете на графа) могат да се подредят в поне една редица така, че подредбата в редицата да отразява предхождането в молекулните формули (т.е. всички ребра на графа да сочат надясно). Една такава редица намираме чрез ***топологично сортиране,*** което също използва ***обхождане в дълбочина.***

Не са нужни две обхождания в дълбочина. Търсенето на ориентиран цикъл и топологичното сортиране можем да извършим с едно обхождане в дълбочина (ако открием ориентиран цикъл, прекъсваме обхождането; в противен случай обхождаме графа докрай и нареждаме върховете в ред, обратен на затварянето).

Сега знаем, че има подредба на върховете и сме получили една такава. Въпросът е дали това подреждане е еднозначно. Ако не е — това означава, че можем да разменим местата на поне една двойка върхове, т.е. съществуват два химични елемента, за които не е ясно кой преди кой се пише.

За да проверим дали подредбата е еднозначно определена, постъпваме така: обхождаме върховете на графа $v_1, v_2, v_3, \ldots, v_n$, както са подредени в резултат на топологичното сортиране. За всяка двойка последователни върхове v_k и v_{k+1} проверяваме има ли ребро от v_k към v_{k+1}. Ако няма ребро, спираме обхождането и връщаме намерената двойка върхове от този вид: те могат да се разместят, при което се получава втора валидна подредба, следователно редът им не може да се определи еднозначно. Обратно, ако не открием такава двойка, това значи, че графът съдържа хамилтонов път $v_1 \to v_2 \to v_3 \to \ldots \to v_n$, поради което подредбата е определена еднозначно. Тоест за всеки два химични елемента знаем със сигурност кой преди кой се пише; ето защо не съществува двойка, за която това да не е сигурно.

Решихме задачата с две обхождания на графа. Всяко от тях минава веднъж през всеки връх и всяко ребро, следователно времевата сложност е $\Theta(m + n)$ при най-лоши входни данни — когато и двете обхождания се изпълнят докрай, тоест когато графът е ацикличен и съдържа хамилтонов път; с други думи, когато съществува еднозначно определена подредба на върховете.

Задача 3. Взимаме първото число на масива $A[1...n]$; после взимаме първото число след взетото с различна четност и т.н. На всяка стъпка взимаме първото следващо число с различна четност от последното взето.

На всяка стъпка алгоритъмът извършва действието, изглеждащо най-добро в този миг. Следователно това е ***алчен алгоритъм.***

Описание на псевдокод:

```
Редуване_по_четност(A[1...n]: масив от цели числа)
if n < 1
   return
Last ← A[1]
print Last
for k ← 2 to n do
   if A[k] ≠ Last (mod 2)
      Last ← A[k]
      print Last
```

Анализ на сложността на алгоритъма: Времевата сложност има порядък $\Theta(n)$ при всякакви входни данни заради цикъла. Сложността по памет е $\Theta(1)$, защото алгоритъмът съдържа две локални променливи (k и Last) от примитивен тип — цели числа. Сложността по памет е оптимална, защото $\Theta(1)$ е най-малкият възможен порядък. Сложността по време е оптимална, защото дължината на търсената подредица зависи от всеки елемент на $A[1...n]$, следователно трябва да бъдат прочетени всичките n елемента (тривиална долна граница по размера на входа).

П р и м е р : Нека входният масив е A = (3 ; 5 ; 6 ; 2 ; 8 ; 1). Тогава алгоритъмът отпечатва подредицата от подчертаните числа в масива: 3 ; 6 ; 1. Тя е образувана от първото число на масива (тоест 3), което се случи нечетно; първото следващо четно число (т.е. 6); първото следващо нечетно число (т.е. 1). Тук подредицата завършва, понеже е достигнат краят на масива. Оказва се, че най-дългата подредица, редуваща четни и нечетни числа, има три елемента.

Обосновка на коректността на алгоритъма: Да разделим числата на групи според тяхната четност. Всяка група се състои от последователните числа с еднаква четност. (В примерния масив има три групи, като всяка е оцветена според четността на елементите си.) Нека K е броят на групите, а L е дължината на най-дългата подредица, редуваща четни и нечетни числа.

Да допуснем, че $L > K$. От принципа на Дирихле следва, че някоя група съдържа два или повече члена на подредицата. Те са последователни членове на подредицата; това следва от определението за подредица. Така се нарушава изискването за редуване на четни и нечетни числа.

Полученото противоречие показва, че $L \leq K$. От друга страна, ако вземем по едно число от всяка група (отляво надясно), ще получим подредица, която редува четни и нечетни числа и има дължина K. Ето защо $L = K$.

И така, K е дължината на коя да е най-дълга подредица, в която се редуват четни и нечетни числа. Тези подредици съдържат по едно число от всяка група.

Разгледаният алгоритъм избира първото число от всяка група, затова връща подредица с максимална дължина, тоест K. Ето защо алгоритъмът е коректен.

Задача 4.

а) Съществува най-много един приятен индекс. Той може да бъде намерен за време O(n) така:

```
isPleasant(A[1...n][1...n], k)
for j ← 1 to n do
   if (j ≠ k) and (A[k][j] = 1 or A[j][k] = 0)
      return false
return true
```

```
findPleasant(A[1...n][1...n])
// Връща приятен индекс (-1, ако няма приятен индекс).
k ← 1
for j ← 2 to n do
   if A[k][j] = 1   // Индексът k не е приятен.
      k ← j
if isPleasant(A[1...n][1...n], k)
   return k
return -1
```

б) Долната граница се доказва на два етапа. Първо, не съществува реализация на функцията `isPleasant` с по-малко от $2(n-1)$ прочитания на стойности на елементи на матрицата A: всяка реализация на функцията трябва да прочете ред № k и стълб № k (без пресечната им клетка), за да провери изискването за приятност на k, инак не може да научи стойностите на тези $2(n-1)$ елемента, защото елементите на матрицата са независими. Оттук правим следния извод: алгоритмичната задача `isPleasant` изисква време $\Omega(n)$.

Второ, задачата `findPleasant` също изисква време $\Omega(n)$. Редукция:

```
isPleasant(A[1...n][1...n], k)
return findPleasant(A[1...n][1...n]) = k
```

Редукцията изразходва константно време (сравнява върнатата стойност с k), тоест редукцията е достатъчно бърза. Нейната коректност е пряко следствие от единствеността на приятния индекс (когато съществува такъв).