СОФИЙСКИ УНИВЕРСИТЕТ "СВ. КЛИМЕНТ ОХРИДСКИ", ФМИ — ИЗПИТ ПО УЧЕБНАТА ДИСЦИПЛИНА "ДИЗАЙН И АНАЛИЗ НА АЛГОРИТМИ" ЗА СПЕЦИАЛНОСТ "КОМПЮТЪРНИ НАУКИ", 1. ПОТОК, И ЗА ИЗБИРАЕМИЯ КУРС (05 СЕПТЕМВРИ 2019 ГОД.)

Изисквания към решенията на задачите:
1) При сортиране и търсене уточнявайте кой алгоритъм за сортиране или търсене ползвате.
2) Ако моделирате някоя задача с граф, уточнете какво представляват върховете и ребрата и какво значат посоките и теглата им. Ако обхождате граф, уточнете вида на обхождането.
3) Ако някой алгоритъм има няколко възможни реализации, уточнете коя от тях използвате.
4) Имената на известните алгоритми и структури от данни да бъдат на български език.

Задача 1. Следната алгоритмична задача за разпознаване да наречем `Problem_1`.
Дадени са две цели неотрицателни числа S и L и масив $A[1...n]$ от цели положителни числа.
Въпрос: Числото S дали е сбор на не повече от L елемента на масива $A[1...n]$?
Всеки елемент на масива може да участва в сбора най-много веднъж.
Формално въпросът се поставя така: Съществува ли подмножество M на $\{ 1 ; 2 ; 3 ; \ldots ; n \}$, такова че $|M| \leq L$ и $\sum_{k \in M} A[k] = S$?

Да се докаже, че алгоритмичната задача `Problem_1` е NP-пълна.

Задача 2. Предложете алгоритъм с времева сложност $O(n \log n)$ в най-лошия случай, който по даден числов масив $A[1...n]$, $n \geq 3$, открива индекси i, j, k, два по два различни, за които сборът $|A[i] - A[j]| + |A[j] - A[k]| + |A[k] - A[i]|$ има възможно най-малка стойност. Опишете алгоритъма на псевдокод.

Задача 3. Опишете с думи алгоритъм с линейна времева сложност, подреждащ върховете на даден неориентиран свързан граф така, че ако изтриваме върховете един по един в реда, предложен от алгоритъма, след всяко изтриване на връх да остава свързан граф.

Задача 4. Едно село с n къщи се нуждае от водоснабдяване. Всяко домакинство има избор: да изкопае кладенец в двора на къщата или да прекара водопровод от някоя друга къща, която вече има вода. Масивите $A[1...n]$ и $B[1...n][1...n]$ съдържат цените на двете операции: $A[k]$ е цената за изкопаване на кладенец в двора на k-тата къща, а пък $B[i][j]$ е цената за прекарване на водопровод от i-тата до j-тата къща. Матрицата $B[1...n][1...n]$ е симетрична, тоест $B[i][j] = B[j][i]$.

Опишете с думи алгоритъм с времева сложност в най-лошия случай $O(n^2)$, който съставя план за водоснабдяването на селото: за всяка къща определя дали да се копае кладенец в двора на къщата, или да се прекара водопровод от някоя друга къща (и от коя точно). Алгоритъмът трябва да минимизира общата цена.

Задача 5. В българския език има думи със следното свойство: ако изтриваме буквите им една по една в правилния ред, то на всяка стъпка се получава смислена българска дума. Изразът "на всяка стъпка" означава, че редицата завършва със смислена еднобуквена дума. Пример: откровенията → откровеният → откровения → откровени → откровен → отровен → тровен → троен → трон → тон → то → о.

Речникът на българския език е зададен като списък от думи. Постройте подходящ граф G и опишете словесно бърз алгоритъм с вход G за търсене на най-дълга дума с това свойство.

РЕШЕНИЯ

Задача 1. Идеята е, че задачата SubsetSum е частен случай на Problem_1 при $L = n$. Тоест задачата Problem_1 е обобщение на NP-трудна задача, затова също е NP-трудна.

Формално описание на редукцията:

```
SubsetSum(A[1...n], S)
1) L ← n
2) return Problem_1(A[1...n], S, L)
```

Коректност на редукцията:

Функцията SubsetSum($A[1...n]$, S) връща истина

⇕ (от ред № 2 на алгоритъма)

Problem_1($A[1...n]$, S, L) връща истина

⇕ (от постановката на задачата Problem_1)

числото S е сбор на не повече от L елемента на масива $A[1...n]$

⇕ (от ред № 1 на алгоритъма: $L = n$)

числото S е сбор на не повече от n елемента на масива $A[1...n]$

⇕ (масивът A съдържа n елемента, тоест от него няма как да вземем повече елементи)

числото S е сбор на няколко (може и нула) елемента на масива $A[1...n]$.

Доказахме следната еквивалентност:

Функцията SubsetSum($A[1...n]$) връща истина

⇕

числото S е сбор на няколко (може и нула) елемента на масива $A[1...n]$.

Тази еквивалентност съвпада с постановката на алгоритмичната задача SubsetSum. Следователно функцията SubsetSum работи правилно, тоест редукцията е коректна.

Бързина на редукцията: Редукцията се състои от присвояването $L \leftarrow n$, което има константна, следователно полиномиална сложност. Това показва, че построената редукция е достатъчно бърза за целите на доказателството.

Дотук доказахме, че алгоритмичната задача Problem_1 е NP-трудна. За да докажем, че тя е NP-пълна, остава да проверим, че принадлежи на класа NP. За целта имаме нужда от алгоритъм с полиномиална времева сложност, който да проверява предложено решение. Сертификатът, описващ предложеното решение, е множеството M от индексите на избраните елементи на масива A. Задаваме множеството M като логически масив с n елемента: $M[k]$ = истина $\Leftrightarrow$ $k \in M$.. Останалата част от входа се състои от $n + 2 = \Theta(n)$ числа: елементите на масива $A[1...n]$ плюс числата S и L. Сертификатът $M[1...n]$ е къс, защото дължината му n е полином (от първа степен) на дължината на останалата част от входа на алгоритъма за проверка.

Алгоритъм за проверка на предложено решение:

```
Check_Problem_1(A[1...n], S, L, M[1...n])
1) cnt ← 0
2) sum ← 0
3) for k ← 1 to n do
4)    if M[k]
5)       cnt ← cnt + 1
6)       sum ← sum + A[k]
7) return (sum = S) and (cnt ≤ L)
```

Времевата сложност на този алгоритъм е полиномиална: $\Theta(n)$. Следователно задачата `Problem_1` принадлежи на класа NP. По-горе беше доказано, че тази задача е NP-трудна. А щом е NP-трудна и принадлежи на NP, то тя е NP-пълна.

Задача 2. Неформално казано, трябва трите елемента да се различават възможно най-малко. След сортиране на масива елементите с близки стойности се разполагат един до друг. Тъй като изразът

$$|A[i]-A[j]|+|A[j]-A[k]|+|A[k]-A[i]|$$

е симетричен, можем без ограничение да приемем, че индексите са наредени така: $i<j<k$. Следователно $A[i]\le A[j]\le A[k]$ и

$$|A[i]-A[j]|+|A[j]-A[k]|+|A[k]-A[i]| = 2.(A[k]-A[i]).$$

При произволно, но фиксирано j изразът $2.(A[k]-A[i])$ има най-малка стойност, когато $A[k]$ е възможно най-малко, а пък $A[i]$ — възможно най-голямо, т.е. при $k=j+1$ и $i=j-1$. Накратко, след сортировката трябва да сравним само тройките последователни елементи. Трябва да използваме някоя от бързите сортировки, например пирамидалното сортиране.

Псевдокод на алгоритъма:

```
MinDiff3(A[1...n]) // n ≥ 3
Sort(A[1...n]) // например пирамидално сортиране
minDiff ← +∞
for j ← 2 to n-1 do
   diff ← A[j+1] - A[j-1]
   if diff < minDiff
      minDiff ← diff
      bestIndex ← j
print bestIndex - 1, bestIndex, bestIndex + 1
return 2 × minDiff
```

Формално погледнато, този алгоритъм е некоректен, тъй като отпечатва индексите след сортирането. Можем да запазим оригиналните индекси в друг масив и да ги разместваме заедно с елементите на масива `A[1...n]` по време на сортирането. В края на алгоритъма ще отпечатаме оригиналните индекси.

Пирамидалното сортиране изразходва време $\Theta(n \log n)$ в най-лошия случай, а цикълът след това — време $\Theta(n)$. Времето на целия алгоритъм е сборът от двете времена. Порядъкът на сбора се определя от порядъка на по-голямото събираемо. Окончателно, алгоритъмът има времева сложност $\Theta(n \log n)$ в най-лошия случай.

Задача 3. Обхождаме графа в дълбочина и изтриваме върховете в реда на затваряне, тоест изтриваме връх, когато рекурсията излиза от него.

Времевата сложност на алгоритъма е линейна: обхождането в дълбочина изразходва линейно време, ако всеки връх и всяко ребро се обработват за време $\Theta(1)$, както е в случая.

Коректност: Обхождането на свързан граф в дълбочина поражда дърво на обхождането. Тъй като изтриваме върховете на графа в реда на тяхното затваряне, то всеки връх се изтрива преди своя родител в дървото. Тоест изтриването върви от листата към корена, затова след всяко изтриване от първоначалното дърво остава пак дърво — дърво на обхождане на неизтритата част от графа. Щом тя има покриващо дърво (а не гора), значи е свързана.

Задача 4. Разглеждаме пълен неориентиран граф с $n+1$ върха — къщите на селото плюс един допълнителен връх, съответстващ на земята (по-точно, на водоносния пласт). Изкопаването на кладенец смятаме за прекарване на водопровод от добавения връх (земята) към съответната къща. Ребрата на графа съответстват на водопроводите и кладенците. Теглата на ребрата са цените за изкопаване на кладенци и прекарване на водопроводи. В задачата се търси ***минимално покриващо дърво*** за време $O(n^2)$ в най-лошия случай. За целта трябва да използваме ***алгоритъма на Прим—Ярник без приоритетна опашка или с приоритетна опашка, реализирана с пирамида на Фибоначи,*** но не и версията с приоритетна опашка, реализирана с двоична пирамида: тази версия не е достатъчно бърза. Поради същата причина (недостатъчна бързина) не бива да ползваме алгоритъма на Крускал: времевата му сложност е $\Theta(m \log m)$, тоест $\Theta(n^2 \log n) = \omega(n^2)$ за пълен граф ($m = \Theta(n^2)$).

Задача 5. Даден речник (списък от думите на някакъв език) можем да представим чрез ориентиран граф G: върховете на G съответстват на думите от речника, а ребрата на G съответстват на изтриването на буква. По-точно, ако думата α се получава от думата β чрез изтриване на една буква, то G съдържа ребро от върха α към върха β. Графът G има един допълнителен връх ε, който съответства на празната дума, и ребра от ε към върховете на всички еднобуквени думи.

G е ***ориентиран ацикличен граф:*** всяко ребро сочи от по-къса към по-дълга дума. Преминаване по кое да е ребро увеличава дължината на думата с единица, следователно дължината на всяка дума α е равна на дължината на пътя от ε до α. Търсим най-дълга дума, тоест ***най-дълъг път*** с начало ε. В случая на произволен граф тази задача е NP-пълна. Но графът G е ориентиран ацикличен; затова задачата може да се реши за линейно време с помощта на ***динамично оптимиране.***

Асимптотичната оценка за линейна времева сложност се отнася само за втория етап — търсенето на най-дълъг път в графа G. Скоростта на този етап е оценена спрямо дължината на неговия вход — описанието на графа G. Причината е, че в условието на задачата се иска тъкмо това — бързина на алгоритъма, обработващ графа. Но самото построяване на графа G, тоест първият етап от решението, може да не е с линейна времева сложност. Сложността му (и сложността на целия алгоритъм) се оценява спрямо дължината на неговия вход — списъка от думи (речника). Няма да правим тази оценка, защото не се изисква от условието.

Построяването на графа в явен вид изразходва много памет. Описанието на алгоритъма, дадено по-горе, трябва да се схваща само като кратко изложение на идеята на алгоритъма, но не и на подробностите по реализацията му. В действителност е по-добре да не се строи явен граф, а динамичното оптимиране да се приложи направо върху списъка от думи. Пълното описание на алгоритъма би съдържало множество подробности от реализацията — каква структура от данни се използва за представяне на списък (или масив) от низове; ред на изчисленията; как се търсят думи, получени след изтриване (или добавяне) на буква.

Изложеното решение изпълнява само минималните изисквания от условието на задачата, а те са поставени така с оглед на ограниченото време на изпита.