ИЗПИТ ПО ДИЗАЙН И АНАЛИЗ НА АЛГОРИТМИ
ЗА СПЕЦИАЛНОСТ "КОМПЮТЪРНИ НАУКИ" — 2. КУРС, 1. ПОТОК
(СУ, ФМИ, 17 АВГУСТ 2020 Г.)

Указания:

1) Имената на всички алгоритми и структури от данни да са на български език.
2) Ако сортирате някакви данни, уточнявайте вида на използваната сортировка.
3) Решенията да бъдат описани подробно.
4) Можете да използвате наготово всички алгоритми, изучени на лекции.

Задача 1. Дадени са два масива $A[1 \ldots m]$ и $B[1 \ldots n]$ от реални числа, $m \leq n$. Търси се броят на наредените двойки $(a\,;\,b)$, за които $a \in A$, $b \in B$ и $a < b$. Съставете алгоритъм, който пресмята търсения брой за време $O((m + n) \log m)$ при всякакви входни данни. Опишете алгоритъма с думи възможно най-ясно. Направете пълна демонстрация за масивите $A = (10\,;\,5\,;\,8)$ и $B = (3\,;\,20\,;\,6\,;\,7)$.

Задача 2. Имате данни за различията между всеки два от n щама на вирус. Различията са представени като реални неотрицателни числа — брой различия в генотиповете или в процента на смъртните случаи и т.н. Не е важно как точно са измерени различията. Може да не са получени чрез изваждане на числа, т.е. може да не са разлики в аритметичния смисъл. Знаете само, че два щама си приличат толкова повече, колкото по-малка е разликата между тях. Предполагате, че всички щамове са произлезли чрез мутации от един щам, но не ви е известно кой е той. Искате да възстановите историята на вируса — кой щам от кой друг щам произхожда пряко. Опростявате тази трудна задача: отказвате се да търсите първия щам и посоката на родство (стига ви да знаете, че щамът A е произлязъл пряко от $Б$ или обратно, но не е важно кое от двете). Нужно е предположение, почиващо на факта, че мутации настъпват рядко.

Представете входните данни чрез граф: какво са върховете и ребрата му? Формулирайте търсеното в термините на графи: уточнете предположението за малкия общ брой мутации. Сведете решението до позната задача за графи. Изберете най-бързия алгоритъм и уточнете реализацията му. Обосновете се, като сравните времевите сложности (зависещи само от n) при най-лоши данни.

Задача 3. Съставете алгоритъм със сложност по време $O(n)$ и по памет $O(1)$, който пресмята по колко начина можем да разделим n приятели на групи, ако всяка група трябва да се състои от двама души или от сам човек. Опишете алгоритъма на Си или на псевдокод като функция `int F(int n)`.

Задача 4. Докажете, че алгоритмичната задача Problem4 е **NP**-пълна.

Вход: 1) неориентиран граф G с n върха и m ребра;
2) две цели неотрицателни числа K и L, като $K \leq L$.

Въпрос: G съдържа ли клика с брой върхове между K и L включително?

Всички алгоритми (вкл. редукцията) да се опишат на псевдокод или на Си.

РЕШЕНИЯ

Задача 1 се решава чрез ***сортиране,*** последвано от ***двоично търсене:***

1) Сортираме масива $A[1 \ldots m]$ за време $\Theta(m \log m)$ с някоя бърза сортировка, например с пирамидално сортиране или сортиране чрез сливане.
2) Инициализираме с нула един брояч d.
3) За всяко цяло число k от 1 до n вкл. намираме с двоично търсене мястото p на елемента $b = B[k]$ в сортирания масив $A[1 \ldots m]$ и добавяме p към d.
4) Връщаме последната стойност на брояча d.

Тъй като между елементите на масивите може да има равни, нека уточним: двоичното търсене от стъпка № 3 е организирано по такъв начин, че да намира най-големия индекс p, за който $A[p] < b$. Ако няма такъв индекс, тогава $p = 0$. С други думи, p е броят на двойките $(a ; b)$, за които $a < b$, където a пробягва A, а $b = B[k]$ е фиксирано. Но щом индексът k се мени от 1 до n включително, то в крайна сметка b пробягва целия масив B, т.е. преброяват се всички двойки, за които $a \in A$, $b \in B$ и $a < b$. Общият им брой е сборът от стойностите на p, а той се натрупва постепенно в променливата d, така че последната ѝ стойност е тъкмо сборът от стойностите на p, тоест броят на наредените двойки $(a ; b)$, за които $a \in A$, $b \in B$ и $a < b$.

Анализ на времевата сложност: Първата стъпка (сортирането на $A[1 \ldots m]$) изисква време $\Theta(m \log m)$ в най-лошия случай, при условие че се използва някоя бърза сортировка, например пирамидалното сортиране. Втората стъпка (инициализирането на брояча с нула) отнема време $\Theta(1)$. Двоичното търсене на всеки отделен елемент b в сортирания масив A изразходва време $\Theta(\log m)$, а общо за всичките n елемента на масива B — време $\Theta(n \log m)$. Следователно времето на целия алгоритъм е $\Theta((m + n) \log m)$.

Задача 2. Представяме входните данни с пълен неориентиран тегловен граф: върховете на графа съответстват на различните щамове на вируса (n на брой); има ребро между всеки два върха; теглото на всяко ребро $\{ X ; Y \}$ е дадено реално неотрицателно число — мярка за разликата между щамовете X и Y. Историята на вируса е кореново дърво: коренът съответства на родоначалника, ребрата на дървото са ориентирани в посока от корена към листата и показват кой щам от кой произхожда пряко. Този подграф е именно дърво, защото всеки щам с изключение на родоначалника произхожда пряко от един друг. Тъй като всички щамове са произлезли (пряко или непряко) от родоначалника, то дървото трябва да съдържа всички върхове на графа, тоест трябва да бъде негово покриващо дърво. Предположението за най-малък общ брой мутации означава, че търсим покриващо дърво с най-малък сбор от теглата на ребрата, тоест ***минимално покриващо дърво.*** Опростяването на задачата се състои в отказ от търсене на корена на дървото и ориентацията на ребрата му; не можем да ги намерим без допълнителни предположения.

Минимално покриващо дърво се търси с някой от следните алгоритми:

— ***алгоритъма на Крускал***, реализиран с помощта на структурата ***Union-Find*;** времевата сложност е $\Theta(m \log m) = \Theta(n^2 \log n)$ при най-лоши входни данни;

— ***алгоритъма на Прим—Ярник***, програмиран чрез ***приоритетна опашка***, реализирана с ***пирамида на Фибоначи***; времевата сложност на тази версия е $\Theta(m + n \log n) = \Theta(n^2)$ при най-лоши входни данни.

При преобразуването на изразите за сложността се възползвахме от равенството $m = \Theta(n^2)$. То важи за пълни графи, какъвто е графът от настоящата задача.

От $n^2 = o(n^2 \log n)$ следва, че по-бърз е ***алгоритъмът на Прим—Ярник***, реализиран с ***пирамида на Фибоначи.***

Задача 3 можем да решим с ***динамично програмиране:*** отначало я решаваме за по-малки стойности на n, после от тях получаваме решенията за по-големи n с помощта на рекурентното уравнение $a_n = a_{n-1} + (n-1)\,a_{n-2}$ за $\forall n \geq 3$. Това уравнение е съставено чрез разглеждане на два случая за n-тия човек:

— Ако остане сам, другите $n-1$ приятели се разпределят по a_{n-1} начина.

— Ако се групира с някого от другите $n-1$ човека, то останалите $n-2$ могат да се разпределят по a_{n-2} начина.

Начални условия: $a_1 = 1$, защото сам човек може да бъде "разпределен" по единствен начин — като остане сам; а пък $a_2 = 2$, защото за двама души има две възможности: да се обединят в една група или всеки да остане сам.

Забележка: Може да се добави един член в началото на редицата: $a_0 = 1$.

Започвайки от началните условия, намираме a_n от рекурентното уравнение. От него следва, че за всяка стойност в редицата са нужни само две предишни.

Описание на алгоритъма на псевдокод:

```
F(n: цяло положително число)
 1) if n = 1
 2)    return 1
 3) a ← 1  // Брой разпределения при n = 1.
 4) b ← 2  // Брой разпределения при n = 2.
 5) c ← 2  // Брой разпределения при n = 2.
 6) for k ← 3 to n do
 7)    c ← b + a × (n - 1) // Бр. разпр. на k приятели.
 8)    a ← b               // Бр. разпр. на k-1 приятели.
 9)    b ← c               // Бр. разпр. на k приятели.
10) return c               // Бр. разпр. на n приятели.
```

Алгоритъмът има сложност по време $\Theta(n)$ заради цикъла с начало на ред № 6. Сложността по памет е $\Theta(1)$: локалните променливи a, b и c са цели числа.

Задача 4. Това, че алгоритмичната задача Problem4 е **NP**-трудна, се доказва чрез полиномиална редукция: задачата "Клика" $\propto_P$ Problem4:

Описание на редукцията на псевдокод:

```
Клика(G: граф с n върха и m ребра; K: цяло число) // K ≥ 0
// Функцията връща "истина" ⇔ G има клика с поне K върха.
1) L ← n
2) return Problem4(G, K, L)
```

Редукцията е полиномиална, защото първият ред се изпълнява за време $\Theta(1)$, което е полином на n от нулева степен.

Коректност на редукцията:

Функцията "Клика", извикана с аргументи G и K, връща "истина".

⇳ (От ред № 2.)

Problem4(G, K, L) връща "истина".

⇳ (От постановката на задачата Problem4.)

Графът G съдържа клика с брой върхове между K и L включително.

⇳ (От ред № 1 следва, че $L = n$.)

Графът G съдържа клика с брой върхове между K и n включително.

⇳ (Броят на върховете на кликата по принцип не надхвърля n.)

Графът G съдържа клика с поне K върха.

Тоест функцията "Клика", извикана с аргументи G и K, връща "истина" ⇔ графът G съдържа клика с поне K върха.

Тази еквивалентност точно съвпада с постановката на задачата "Клика", следователно редукцията е коректна.

Можем да използваме другия вариант на задачата "Клика" — този, в който се търси клика с точно определен брой върхове.

Описание на редукцията на псевдокод:

```
Клика(G: граф с n върха и m ребра; K: цяло число) // K ≥ 0
// Функцията връща "истина" ⇔ G има клика с точно K върха.
1) L ← K
2) return Problem4(G, K, L)
```

Редукцията е полиномиална, защото първият ред се изпълнява за време $\Theta(1)$.

Коректността на редукцията се доказва подобно на предишната, само че сега броят на върховете на кликата е между K и K включително, тоест точно K.

За да докажем, че Problem4 $\in$ **NP**, трябва да съставим бърз алгоритъм за проверка на предложено решение. Сертификат ще бъде подмножеството от върхове, което образува търсената клика. Ще го представим с помощта на логически масив $P[1 \ldots n]$.

```
CheckProblem4(G: граф с n върха и m ребра, K, L, P[1...n])
 1) cnt ← 0
 2) for k ← 1 to n do
 3)    if P[k]
 4)       cnt ← cnt + 1
 5) if cnt < K or cnt > L
 6)    return false // Броят на върховете не е между K и L.
 7) for k ← 1 to n-1 do
 8)    if P[k]
 9)       for s ← k+1 to n do
10)          if P[s]
11)             if s ∉ Adj(k)    // Липсва ребро.
12)                return false // P не описва клика.
13) return true
```

Тук Adj(k) е списъкът на ребрата, излизащи от връх № k на графа G. За удобство представяме всяко ребро чрез номера на върха на другия му край.

Сертификатът P е къс, защото дължината му n е полином (от първа степен) на дължината $\Theta(n+m)$ на входа на задачата Problem4 — тройката $(G\,;K\,;L)$. Цикълът с начало на ред № 2 изразходва време $\Theta(n)$. Двата вложени цикъла, започващи на редове № 7 и № 9, са причина за $O(n^2)$ изпълнения на ред № 11, а всяко отделно изпълнение на ред № 11 съдържа скрит цикъл, който обхожда списъка Adj(k), чиято дължина е не повече от m, т.е. броя на всички ребра на G. Следователно алгоритъмът за проверка на предложено решение се изпълнява за време $O(n^2m)$ при всякакви входни данни, тоест времето на алгоритъма е ограничено отгоре от полином (от трета степен) на дължината на входа на задачата Problem4.

Щом има къс сертификат и бърз алгоритъм за проверка, то Problem4 $\in$ **NP**.

Тъй като Problem4 е **NP**-трудна задача от **NP**, то тя е **NP**-пълна.

СХЕМА ЗА ТОЧКУВАНЕ

Всяка задача носи по 20 точки с изключение на задача 4. Тя носи 40 точки:
— 20 точки за установяване, че Problem4 $\in$ **NP**;
— 20 точки за доказателство, че Problem4 е **NP**-трудна задача.