NP-ПЪЛНИ ЗАДАЧИ

КОНТРОЛНО № 5 ПО "ДИЗАЙН И АНАЛИЗ НА АЛГОРИТМИ" — СУ, ФМИ
(ЗА СПЕЦИАЛНОСТ "КОМПЮТЪРНИ НАУКИ", 1. ПОТОК; 23 МАЙ 2018 Г.)

Задача 1. Разглеждаме задачата за разпознаване `NonEmptyRows`:
— Вход: цяло положително число K
и двоична матрица $A[1...n][1...m]$ с m стълба и n реда.
— Въпрос: Съществуват ли K стълба на матрицата A, такива че подматрицата, образувана от тези K стълба и всичките n реда на A, съдържа единица във всеки свой ред?

Докажете, че `NonEmptyRows` е NP-пълна задача.

Задача 2. Разглеждаме задачата за разпознаване `SumMatrix`:
— Вход: цяло число $S \geq 0$ и матрица $B[1...n][1...m]$ с m стълба и n реда, съставена от цели неотрицателни числа.
— Въпрос: Може ли да се изберат n числа, по едно от всеки ред на матрицата B, така че сборът на избраните n числа да е равен на S?

Докажете, че `SumMatrix` е NP-пълна задача.

Точкуване: Всяка задача носи 2 точки:
1 точка за доказване, че съответната алгоритмична задача е NP-трудна,
и 1 точка за доказване, че тя принадлежи на класа NP.

Двете задачи носят общо 4 точки.

Оценката = 2 + броя на получените точки.

РЕШЕНИЯ

Задача 1. Ще докажем, че `NonEmptyRows` е NP-трудна задача, с помощта на полиномиална редукция: `DominatingSet` $\propto$ `NonEmptyRows`, където `DominatingSet` е задачата за разпознаване дали неориентиран нетегловен граф G има доминиращо множество с не повече от K върха (множество D, такова че всеки връх, който не е от D, е свързан чрез ребро с някой връх от D).

Описание на редукцията:

```
DominatingSet(G: неориентиран нетегловен граф с n върха;
              K: цяло положително число)
1) A[1...n][1...n] ← матрицата на съседствата на G
2) for i ← 1 to n do
3)    A[i][i] ← 1
4) return NonEmptyRows(A[1...n][1...n], K)
```

Доказателството за коректност на редукцията се състои от две части:

— Параметрите A и K, с които извикваме функцията `NonEmptyRows`, имат допустими стойности: K е цяло число, $K > 0$, а матрицата A е двоична, защото между всеки два върха на граф или има ребро, или няма.

— Функцията `DominatingSet` връща правилен резултат. Наистина,
функцията `DominatingSet(G, K)` връща стойност "истина"

$\Updownarrow$ (от ред № 4 на алгоритъма)

функцията `NonEmptyRows(A, K)` връща стойност "истина"

$\Updownarrow$ (от определението на задачата `NonEmptyRows`)

съществуват K стълба на матрицата A, такива че подматрицата, образувана от тези K стълба и всичките n реда на A, съдържа единица във всеки свой ред

$\Updownarrow$ (от редове № 2 и № 3 на алгоритъма: $A[i][i] = 1$ за всяко i)

съществуват K стълба на матрицата A, такива че подматрицата, образувана от тези K стълба и другите $n - K$ реда на A, съдържа единица във всеки свой ред ("другите $n - K$ реда" са редовете с различни номера от избраните K стълба)

$\Updownarrow$ (от ред № 1 на алгоритъма)

съществуват K върха на графа G, такива че всеки от другите $n - K$ върха е свързан чрез ребро с някой от тези K върха

$\Updownarrow$ (от определението на доминиращо множество)

G съдържа доминиращо множество с не повече от K върха.

И тъй, функцията `DominatingSet(G, K)` връща стойност "истина" $\Leftrightarrow$ графът G съдържа доминиращо множество с не повече от K върха. Това съвпада с въпроса на задачата `DominatingSet`, т.е. редукцията е коректна.

Бързина на редукцията: Редукцията се състои от редове № 1, № 2 и № 3 на функцията `DominatingSet`. Ред № 1 изисква квадратично време: $\Theta\left(n^2\right)$, а цикълът на редове № 2 и № 3 — линейно време $\Theta(n)$. Поради това общото време на редукцията е $\Theta\left(n^2\right)$. То е квадратично, значи полиномиално, тоест редукцията е достатъчно бърза.

Дотук доказахме, че алгоритмичната задача `NonEmptyRows` е NP-трудна. Остава да докажем, че тя принадлежи на **NP**, тоест предложено решение може да бъде проверено за полиномиално време. Решение може да бъде предложено само при отговор “да” — когато A съдържа подходящи K стълба. Сертификат може да бъде множеството от индексите им (списък или логически масив C).

Проверката на предложеното решение може да се извърши например така:

```
CheckNonEmptyRows(A[1...n][1...n], K, C[1...n])
 1)  cnt ← 0
 2)  for j ← 1 to n do
 3)     if C[j]
 4)        cnt ← cnt + 1
 5)  if cnt ≠ K
 6)     return false
 7)  for i ← 1 to n do
 8)     has1 ← false
 9)     for j ← 1 to n do
10)        if C[j] and A[i][j] = 1
11)           has1 ← true  // A има единица в ред № i
12)     if not has1
13)        return false   // A няма единица в ред № i
14)  return true
```

С редове № 1 – № 6 проверяваме дали предложените стълбове са точно K. С редове № 7 – № 13 проверяваме дали подматрицата, образувана от предложените K стълба, съдържа единица във всеки свой ред. Ако някое от тези две изисквания не е спазено, отхвърляме предложеното решение (редове № 6 и № 13). В противен случай предложеното решение отговаря на всички изисквания, затова го приемаме (ред № 14).

Анализ на бързодействието: Поради двата вложени цикъла проверката на сертификата изисква време, което е най-много квадратично: $\mathrm{O}\left(n^2\right)$, следователно полиномиално. Затова задачата `NonEmptyRows` е от **NP**.

Задача 2. Ще докажем, че `SumMatrix` е NP-трудна задача, с помощта на полиномиална редукция: `SubsetSum` $\propto$ `SumMatrix`, където `SubsetSum` е задачата за разпознаване дали дадено цяло неотрицателно число S може да се образува като сбор от някои елементи на даден масив $A[1...n]$ от цели положителни числа.

Описание на редукцията:

```
SubsetSum(A[1...n], S)
1) B[1...n][1...2]: array of non-negative integers // m = 2
2) for k ← 1 to n do
3)     B[k][1] ← A[k]
4)     B[k][2] ← 0
5) return SumMatrix(B[1...n][1...2], S)
```

Доказателството за коректност на редукцията се състои от две части:

— Параметрите B и S, с които извикваме функцията `SumMatrix`, имат допустими стойности: S е цяло число, $S \geq 0$, а матрицата B се състои от цели неотрицателни числа.

— Функцията `SubsetSum` връща правилен резултат. Наистина,
функцията `SubsetSum(A, S)` връща стойност "истина"

$\Updownarrow$ (от ред № 5 на алгоритъма)

функцията `SumMatrix(B, S)` връща стойност "истина"

$\Updownarrow$ (от определението на задачата `SumMatrix`)

числото S може да се образува като сбор от n числа,
взети по едно от всеки ред на матрицата B

$\Updownarrow$ (от ред № 4 на алгоритъма: нулевите събираеми не влияят на сбора)

числото S може да се образува като сбор от няколко числа,
взети от първия стълб на матрицата B

$\Updownarrow$ (от ред № 3 на алгоритъма)

числото S може да се образува като сбор от няколко елемента на масива A.

И тъй, функцията `SubsetSum(A[1...n], S)` връща стойност "истина" $\Leftrightarrow$ числото S е сбор от няколко елемента на масива A. Това твърдение съвпада с определението на задачата `SubsetSum`, тоест редукцията е коректна.

Бързина на редукцията: Редукцията се състои от редовете № 1 – № 4 на функцията `SubsetSum`. Цикълът изисква време $\Theta(n)$, което е линейно, следователно полиномиално. Това значи, че редукцията е достатъчно бърза за целите на доказателството.

Дотук доказахме, че алгоритмичната задача `SumMatrix` е NP-трудна. Остава да докажем, че тя принадлежи на **NP**, тоест предложено решение може да бъде проверено за полиномиално време. Решение може да бъде предложено само при отговор “да”, т.е. когато матрицата B съдържа подходящи n числа. Най-естествено е сертификатът да бъде масив от индекси на стълбове $C\,[1...n]$: $C\,[\,i\,]$ е номерът на стълба, от който е взето събираемото в i-тия ред на B.

Проверката на предложеното решение може да се извърши например така:

```
CheckSumMatrix(B[1...n][1...m]: non-negative integers
               S: non-negative integer;
               C[1...n]: array of positive integers)
1) sum ← 0
2) for i ← 1 to n do
3)    j ← C[i]
4)    if j > m
5)       return false
6)    sum ← sum + B[i][j]
7) return (sum = S)
```

С ред № 4 проверяваме дали сертификатът съдържа невалиден индекс на стълб. С цикъла събираме предложените елементи на матрицата B. Накрая (ред № 7) проверяваме дали се получава желаният сбор S.

Анализ на бързодействието: Поради цикъла проверката на сертификата изисква време, което е най-много линейно: $\mathrm{O}(n)$, следователно полиномиално. Затова задачата `SumMatrix` е от **NP**.

Щом задачата `SumMatrix` е от **NP** и е NP-трудна, то тя е NP-пълна.