РЕШЕНИЯ НА ЗАДАЧИТЕ ОТ ИЗПИТА
ПО "ДИЗАЙН И АНАЛИЗ НА АЛГОРИТМИ"
(ПОПРАВИТЕЛНА СЕСИЯ — СУ, ФМИ, 27 ЮНИ 2017 Г.)

Задача 1. В клетките на електронна таблица са въведени формули. Например, ако в клетката B8 е написана формулата "= A4 + C7 – 2 ∗ D9", то трябва първо да се пресметнат стойностите на клетките A4, C7 и D9, а след това — на B8. Предложете възможно най-бърз алгоритъм, който да определя в какъв ред да се изчисляват стойностите на клетките. Опишете алгоритъма словесно.

Решение: Моделираме задачата чрез ориентиран граф: върхове са клетките, ребра са зависимостите между клетките. По-точно, ако формулата в клетката X се позовава на стойността на клетката Y, то графът съдържа ребро от Y към X. В задачата се търси такова подреждане на върховете (клетките), че всяка клетка да предхожда всички клетки, зависещи от нея. Затова задачата се решава с помощта на известния алгоритъм за ***топологично сортиране***, който извършва ***обхождане в дълбочина.*** Времевата сложност е $\Theta(m+n)$, където n е броят на върховете, а m е броят на ребрата на графа.

Задача 2. Оценете по порядък времевата сложност на алгоритъма:

```
P(A[1...n]: array of integers)
s ← 15
for k ← n downto 1 do
  s ← s + A[k] - P(A[1...k-1])
return s
```

Решение: Времевата сложност $T(n)$ удовлетворява рекурентното уравнение $T(n) = a + \sum_{k=1}^{n} \big(T(k-1) + b\big)$, където a е времето за присвояването на ред № 1, b е времето за събирането, изваждането и присвояването в тялото на цикъла.

Заместваме n със $n-1$: $T(n-1) = a + \sum_{k=1}^{n-1} \big(T(k-1) + b\big)$.

Изваждаме новото уравнение от първоначалното: $T(n) - T(n-1) = T(n-1) + b$, тоест $T(n) = 2T(n-1) + b$, което е линейно-рекурентно уравнение. Решаваме го с помощта на характеристично уравнение и намираме характеристични корени 2 от хомогенната част и 1 от нехомогенната. Затова $T(n) = C_1 2^n + C_2 = \Theta\big(2^n\big)$.

Задача 4. Професор има n научни публикации. Масивът $A[1\ldots n]$ съдържа данни за това, колко пъти всяка от тях е била цитирана от други учени; тоест $A[k]$ е броят на цитиранията на k-тата публикация. Предложете алгоритъм за пресмятане на т. нар. h-индекс — най-голямото цяло h, за което е вярно, че професорът има поне h публикации, всяка от които е цитирана поне h пъти.

Алгоритъм с времева сложност $T(n) = \mathrm{O}(n \log n)$ се оценява с 20 точки; за сложност $T(n) = \mathrm{O}(n)$ се дават 40 точки. За по-бавни алгоритми: 0 точки.

Опишете алгоритъма с думи и го демонстрирайте. Анализирайте сложността.

Решение: *Първи алгоритъм:* За време $\Theta(n \log n)$ ***сортираме*** масива A в намаляващ ред с някой бърз алгоритъм, например пирамидално сортиране. После за време $\mathrm{O}(n)$ намираме (чрез последователно или двоично търсене) най-голямото h, за което $A[h] \geq h$. Времето на целия алгоритъм се определя от по-бавния етап — първия; т.е. времевата сложност е $\Theta(n \log n)$.

Вторият алгоритъм е модификация на първия. Ускоряваме първия етап до време $\Theta(n)$ с помощта на ***сортиране чрез броене***. То не бива да се прилага направо върху оригиналните данни (те може да са много големи цели числа). Но ако заменим с n всички числа, по-големи от n, това няма да повлияе на h. Сега вече числата са малки в сравнение с n, затова можем да приложим това сортиране. На практика няма нужда да променяме оригиналните данни.

```
hIndex(A[1...n]: array of non-negative integers): integer
C[0...n]: array of non-negative integers  // честоти
for k ← 0 to n do
   C[k] ← 0
for k ← 1 to n do
   if A[k] > n
      C[n] ← C[n] + 1
   else
      C[A[k]] ← C[A[k]] + 1
S ← 0
k ← n + 1
while k > S do
   k ← k - 1
   S ← S + C[k]
return k
```

Времевата сложност на първите два цикъла е $\Theta(n)$, а на третия цикъл е $\mathrm{O}(n)$, защото броячът `k` намалява с единица на всяка стъпка, а цикълът завършва най-късно при `k` = 0. Затова сложността по време на целия алгоритъм е $\Theta(n)$.

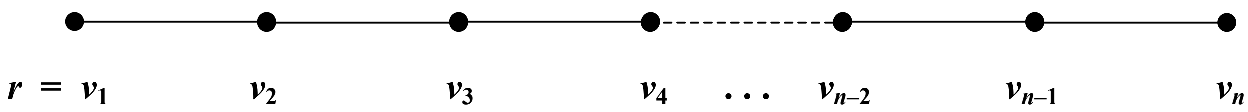
Трети алгоритъм — с време $\Theta(n)$**:** Намираме медианата с ***алгоритъма PICK***. После разбиваме масива**:** поставяме големите елементи наляво от медианата, а малките — надясно. Нека k е новият индекс на медианата. Следва рекурсия**:** ако $A[k] \geq k$, то $h \geq k$ и търсенето продължава в дясната половина на масива; ако $A[k] < k$, то $h < k$ и търсенето продължава в лявата половина на масива; т.е. извършваме ***двоично търсене***. Дъно на рекурсията: масив с един елемент. При достигане на дъното: ако $A\,[k] \geq k$, то $h = k$; ако $A\,[k] < k$, то $h = k–1$.

Анализ: $T(n) = T\left(\dfrac{n}{2}\right) + n$; събираемото n е времето на алгоритъма PICK, другото събираемо е времето за рекурсията. От мастър-теоремата $T(n) = \Theta(n)$.

Задача 3. За $\forall n$ намерете граф G с n върха и връх $r \in G$, та ако пуснем BFS и DFS от върха r, стекът на DFS да има $n-1$ върха в някой миг, а опашката на BFS винаги да има най-много един връх.

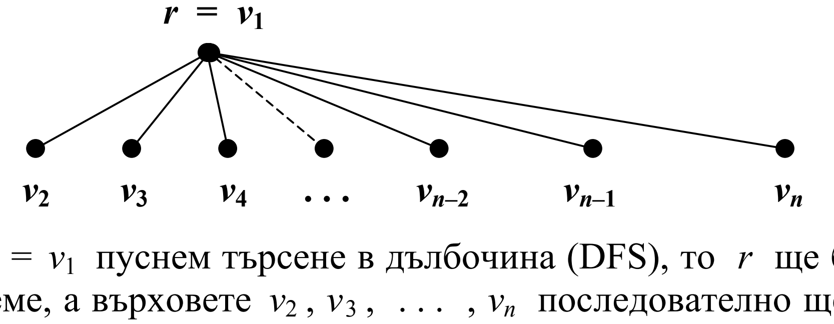
Намерете друг граф G и връх $r \in G$, но сега стекът винаги да е с максимум два върха, а опашката да има $n-1$ върха в някой миг.

Решение: Нека G е следният граф:



Ако пуснем търсенето в ширина (BFS) от върха $r = v_1$, то в опашката ще има най-много един връх във всеки миг от изпълнението на алгоритъма: ако в даден момент в опашката се намира върхът v_i, той ще бъде изваден, а в опашката ще бъде добавен единственият му необработен досега наследник — върхът v_{i+1}. Ако от r пуснем търсене в дълбочина (DFS), то при достигане до върха v_n в стека ще се намират $n-1$ върха: $v_1, v_2, \ldots, v_{n-1}$.

Сега да разгледаме друг граф:



Ако от върха $r = v_1$ пуснем търсене в дълбочина (DFS), то r ще бъде в стека през цялото време, а върховете $v_2, v_3, \ldots, v_n$ последователно ще се добавят и изваждат от стека, затова във всеки миг в стека ще има най-много два върха. Ако от r пуснем търсене в ширина (BFS), то когато алгоритъмът добави v_n, в опашката ще има $n-1$ върха: $v_2, v_3, \ldots, v_n$.

Задача 5. По дадени цели положителни числа A_1 , A_2 , ... , A_n и B намерете броя на решенията на уравнението

$$A_1 x_1 + A_2 x_2 + \ldots + A_n x_n = B$$

в цели неотрицателни числа, тоест броя на наредените n-орки $(x_1, x_2, \ldots, x_n)$ от цели неотрицателни числа, които удовлетворяват уравнението.

Предложете итеративен алгоритъм. Опишете го на псевдокод като функция

numSolEq(A[1...n]: array of int, B: int): int

с време $\mathrm{O}(nB)$ и динамична таблица с $\mathrm{O}(nB)$ клетки. **(10 точки)**

Демонстрирайте алгоритъма при $A = (2 ; 3 ; 5)$ и $B = 9$. **(10 точки)**

Оптимизирайте сложността по памет до динамична таблица с $\mathrm{O}(B)$ клетки. Опишете оптимизирания алгоритъм на псевдокод. **(10 точки)**

Решение:

```
numSolEq(A[1...n]: array of int, B: int): int
dyn[1...n][0...B]: array of int
for ñ ← 0 to n do
  dyn[ñ][0] ← 1
for B̃ ← 1 to B do
  dyn[0][B̃] ← 0
for ñ ← 1 to n do
  for B̃ ← 1 to B do
    if A[ñ] > B̃
      dyn[ñ][B̃] ← dyn[ñ-1][B̃]
    else
      dyn[ñ][B̃] ← dyn[ñ-1][B̃] + dyn[ñ][B̃-A[ñ]]
return dyn[n][B]
```

Предпоследният ред се основава на правилото за събиране: всички решения на уравнението $A_1 x_1 + A_2 x_2 + \ldots + A_n x_n = B$ в цели неотрицателни числа са два вида — такива, в които $x_n = 0$ (броят им се дава от първото събираемо), и такива, в които $x_n > 0$ (второто събираемо, т.е. броят на решенията на уравнението $A_1 x_1 + A_2 x_2 + \ldots + A_n y = B - A_n$ в цели неотрицателни числа, където е положено $y = x_n - 1$).

Демонстрация на алгоритъма при $A = (2\,;3\,;5)$ и $B = 9$:

dyn	$\widetilde{B}=0$	$\widetilde{B}=1$	$\widetilde{B}=2$	$\widetilde{B}=3$	$\widetilde{B}=4$	$\widetilde{B}=5$	$\widetilde{B}=6$	$\widetilde{B}=7$	$\widetilde{B}=8$	$\widetilde{B}=9$
$\widetilde{n}=0$	1	0	0	0	0	0	0	0	0	0
$\widetilde{n}=1$	1	0	1	0	1	0	1	0	1	0
$\widetilde{n}=2$	1	0	1	1	1	1	2	1	2	2
$\widetilde{n}=3$	1	0	1	1	1	2	2	2	3	3

В клетката dyn[$\widetilde{n}$][$\widetilde{B}$] се пази броят на решенията на уравнението $A_1 x_1 + A_2 x_2 + \ldots + A_{\widetilde{n}}\, x_{\widetilde{n}} = \widetilde{B}$ в цели неотрицателни числа.

От долния десен ъгъл dyn[n][B], тоест dyn[3][9], се получава отговорът на задачата. Следователно уравнението $2x_1 + 3x_2 + 5x_3 = 9$ има три решения в цели неотрицателни числа.

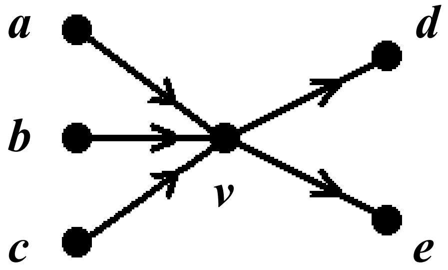
Оптимизация по памет може да се постигне, като се пази само един ред от динамичната таблица.

```
numSolEq(A[1...n]: array of int, B: int): int
dyn[0...B]: array of int
dyn[ 0 ] ← 1
for B̃ ← 1 to B do
  dyn[ B̃ ] ← 0
for ñ ← 1 to n do
  for B̃ ← A[ ñ ] to B do
    dyn[ B̃ ] ← dyn[ B̃ ] + dyn[ B̃ - A[ ñ ]]
return dyn[ B ]
```

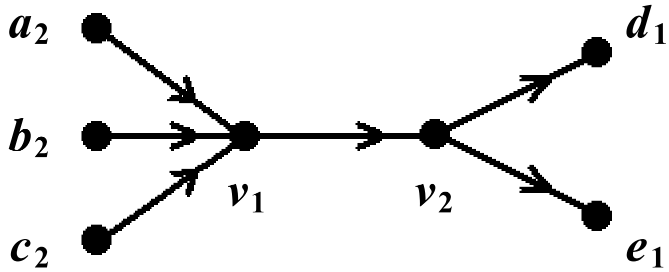
Сложността по време обаче остава $\Theta(nB)$ в най-лошия случай, колкото е при първата версия. Вложеният цикъл прави малка оптимизация на времето: то намалява при големи коефициенти на уравнението, но това е най-добрият, а не най-лошият случай.

Задача 6. Докажете, че задачата за разпознаване, дали даден ориентиран граф съдържа хамилтонов цикъл, остава NP-пълна, когато графът е двуделен.

Решение: Общият случай (произволен граф) чрез полиномиална редукция се свежда до разглеждания частен случай (двуделен граф). Редукцията се състои в разцепването на всеки връх v на два нови върха — v_1 и v_2. Ребрата, които преди разцепването влизат във v, след разцепването влизат във v_1. Ребрата, които преди разцепването излизат от v, след разцепването излизат от v_2. Освен това се добавя ребро от v_1 към v_2.



оригинален граф



модифициран граф

Модифицирането става с едно обхождане на графа, т.е. нужно е линейно (значи, полиномиално) време. При по-внимателно реализиране — копиране на указателите към списъците Adj, а не на самите списъци — даже няма нужда от обхождане на ребрата, затова времевата сложност е $\Theta(n)$, а не $\Theta(m+n)$.

Коректността на редукцията следва от два факта:

1) Модифицираният граф е двуделен. Доказателството се състои в оцветяването на върховете му в два цвята — цвят 1 и цвят 2 (индексите на върховете). Графът е двуделен, тъй като по построение всяко ребро свързва върхове с различни цветове.

2) Оригиналният граф съдържа хамилтонов цикъл тогава и само тогава, когато модифицираният граф съдържа хамилтонов цикъл. Доказателството следва чрез биекция между множествата от хамилтоновите цикли в двата графа. На всеки хамилтонов цикъл в оригиналния граф съответства хамилтонов цикъл в модифицирания граф, получен чрез удвояване на всеки връх и добавяне на индекси 1 и 2 пред първия и втория екземпляр съответно. (Само последният връх не се удвоява.)

   Пример: Ако $pavdxp$ е хамилтонов цикъл в оригиналния граф, то $p_1 p_2 a_1 a_2\ v_1 v_2 d_1 d_2\ x_1 x_2\ p_1$ е хамилтонов цикъл в модифицирания граф.

   Изображението е инекция, защото всяка промяна на върховете в първия цикъл води до съответна промяна на върховете във втория цикъл, тоест на различни хамилтонови цикли в оригиналния граф съответстват различни хамилтонови цикли в модифицирания граф.

Хамилтоновите цикли в модифицирания граф по построение се състоят само от двойки върхове, именувани с еднакви букви, но с различни индекси, като индексите 1 и 2 се редуват (защото модифицираният граф е двуделен). От всеки хамилтонов цикъл в модифицирания граф можем да получим хамилтонов цикъл в оригиналния граф, изтривайки индексите, а после — и втория екземпляр от всяка буква.

Пример: Ако $p_1 p_2 a_1 a_2 v_1 v_2 d_1 d_2\ x_1 x_2\ p_1$ е хамилтонов цикъл в модифицирания граф, то $pavdxp$ е хамилтонов цикъл в оригиналния граф.

Затова описаното изображение е сюрекция. Щом то е инекция и сюрекция, следва, че е биекция. Значи, в оригиналния граф има хамилтонов цикъл тогава и само тогава, когато в модифицирания граф има хамилтонов цикъл. С това е доказана коректността на описаната полиномиална редукция.

Дотук видяхме, че задачата, дали двуделен граф е хамилтонов, е NP-трудна. Остава да докажем, че тя е от класа NP, тоест че предложено решение (пермутация на върховете, описваща предполагаем хамилтонов цикъл) може да бъде проверено (дали наистина е такъв цикъл) за полиномиално време.

```
Check(G(V,E): bipartite graph;  // n = |V|,  m = |E|
      Certificate: array[1...n] of vertices)
Visited: array[1...n] of bool
for k ← 1 to n do
   Visited[k] ← false
for k ← 1 to n do
   Visited[Certificate[k]] ← true
for k ← 1 to n do
   if Visited[k] = false
      return false  // цикълът пропуска връх
for k ← 1 to n-1 do
   if Certificate[k+1] ∉ Adj(Certificate[k])
      return false  // цикълът минава по липсващо ребро
if Certificate[1] ∉ Adj(Certificate[n])
   return false   // цикълът минава по липсващо ребро
return true
```

Проверката за принадлежност на ребро към списък Adj изисква обхождане на списъка. Последният цикъл (с проверката след него) обхожда списъците на всички върхове, затова времето му е $\Theta(m+n)$. Останалите три цикъла изразходват време $\Theta(n)$. Времето за цялата проверка е линейно: $\Theta(m+n)$, следователно полиномиално. Затова разглежданата задача е от класа NP.