ВТОРО КОНТРОЛНО ПО ДИЗАЙН И АНАЛИЗ НА АЛГОРИТМИ ЗА СПЕЦИАЛНОСТ "КОМПЮТЪРНИ НАУКИ" — 2. КУРС, 1. ПОТОК (СУ, ФМИ, 28 ЮНИ 2020 УЧ. Г.)

Указания:

1) Имената на всички алгоритми и структури от данни да са на български език!
2) Всички изучени алгоритми върху графи могат да се използват наготово.
3) Съставените от Вас алгоритми трябва да бъдат верни и достатъчно бързи.

Задача 1. Даден е масив $A[1...n]$ от цели положителни числа. Дадено е също едно цяло число $S \geq 0$. Съставете алгоритъм, който намира по колко начина S може да се представи като сбор на елементи от A с различни индекси. С други думи, търсим $\left| \left\{ M \subseteq \{1; 2; 3; \ldots; n\} : \sum_{k \in M} A[k] = S \right\} \right|$.

Алгоритъмът трябва да има времева сложност $O(nS)$ в най-лошия случай. Опишете алгоритъма на псевдокод или на Си или го сведете до позната задача върху ориентирани ациклични графи (опишете с думи графа и познатата задача и докажете, че графът е ацикличен). Анализирайте сложността по време.

(20 точки)

Забележка: Дават се още **10 точки**, ако при най-лоши входни данни алгоритъмът има сложност по време $O(nS)$, сложност по памет $O(S)$ и е описан на псевдокод или на Си. Анализирайте сложността по време и по памет.

Задача 2. Очевидци на автомобилна катастрофа разказват впечатленията си. Разказът на всеки очевидец описва някаква редица от събития. Обикновено този разказ е непълен, тъй като свидетелят може да не е забелязал всичко.

Пример:

Очевидец 1: "Пешеходецът получи червен сигнал и тогава тръгна да пресича."
Очевидец 2: "Пешеходецът тръгна да пресича и колата го блъсна."
Очевидец 3: "Колата блъсна пешеходеца и неговият светофар светна червено."

а) Представете входните данни чрез граф. Какво са върховете и ребрата му, на какво съответстват посоките и теглата на ребрата (ако има посоки и тегла)? Начертайте графа, съответстващ на трите показания от примера.

Съставете алгоритми с линейна времева сложност в най-лошия случай, които получават графа като вход и решават следните подзадачи:

б) Откриване на противоречие между очевидците относно реда на събитията.
в) Подреждане на събитията в хронологичен ред. Ако има повече от един ред, да се открие само една възможна последователност от събития.
г) Откриване на втора последователност на събитията, ако има такава.

Уточнявайте вида на всяко обхождане на граф: в ширина или в дълбочина. Всяко подусловие се оценява с **5 точки;** цялата задача носи **20 точки.**

РЕШЕНИЯ

Задача 1 прилича на SubsetSum и се решава чрез ***динамично програмиране.***

```
CountSubsetSum(A[1...n],S)
 1) dyn[0...n][0...S]: масив от цели неотрицателни числа;
 2) // dyn[k][t] = броят на начините, по които числото t
 3) // се представя като сбор на елементи на A[1...k]
 4) // с различни индекси.
 5) for k ← 0 to n do
 6)    dyn[k][0] ← 1
 7) for t ← 1 to S do
 8)    dyn[0][t] ← 0
 9) for k ← 1 to n do
10)    for t ← 1 to S do
11)       if t ≥ A[k]
12)          dyn[k][t] ← dyn[k-1][t] + dyn[k-1][t-A[k]]
13)       else // t < A[k]
14)          dyn[k][t] ← dyn[k-1][t]
15) return dyn[n][S]
```

Коректност на алгоритъма:
— Ред № 6 казва, че сбор от положителни събираеми може да има стойност 0 по единствен начин: когато е празен, тоест не съдържа нито едно събираемо.
— Ред № 8 казва, че е невъзможно да се получи сбор с положителна стойност без никакви събираеми.
— Ред № 12 е комбинаторното правило за събиране. Разглеждат се два случая: сборове, които съдържат събираемото $A[k]$, и сборове, които не го съдържат.
— Ред № 14 е нужен, когато първият случай е невъзможен; иначе на ред № 12 ще се получи невалиден (отрицателен) индекс.

Сложност на алгоритъма:
— Инициализацията отнема време $\Theta(n + S)$, вложените цикли — време $\Theta(nS)$. Определящо влияние има последното събираемо: то е най-голямо по порядък. Затова времевата сложност на алгоритъма е $\Theta(nS)$ при всякакви входни данни.
— Заради динамичната таблица сложността по памет на алгоритъма е $\Theta(nS)$ при всякакви входни данни.

Този алгоритъм е коректен и достатъчно бърз, но изразходва много памет, поради което се оценява с **20 точки.**

Можем да намалим сложността по памет до порядък $\Theta(S)$, като съобразим, че във всеки миг алгоритъмът се нуждае от един ред на таблицата.

```
CountSubsetSum(A[1...n],S)
1) dyn[0...S]: масив от цели неотрицателни числа
2) dyn[0] ← 1
3) for t ← 1 to S do
4)    dyn[t] ← 0
5) for k ← 1 to n do
6)    for t ← S downto 1 do
7)       if t ≥ A[k]
8)          dyn[t] ← dyn[t] + dyn[t-A[k]]
9) return dyn[S]
```

Алгоритъмът е коректен, има сложност по време $\Theta(nS)$ заради редове $5-8$, сложност по памет $\Theta(S)$ заради динамичната таблица. Асимптотичните оценки важат при всякакви входни данни. Алгоритъмът се оценява с **30 точки.**

Можем да спестим проверката на ред № 7, като променим долната граница на цикъла с начало на ред № 6:

```
CountSubsetSum(A[1...n],S)
1) dyn[0...S]: масив от цели неотрицателни числа
2) dyn[0] ← 1
3) for t ← 1 to S do
4)    dyn[t] ← 0
5) for k ← 1 to n do
6)    for t ← S downto A[k] do
7)       dyn[t] ← dyn[t] + dyn[t-A[k]]
8) return dyn[S]
```

Този алгоритъм е коректен, има сложност по памет $\Theta(S)$ във всички случаи и времева сложност $\Theta(nS)$ в най-лошия случай — когато числата в масива A са малки спрямо S (например всички числа в A са единици). Но ако числата в A са големи (например по-големи от S), то ред № 7 не се изпълнява изобщо, затова вложените цикли изразходват време $\Theta(n)$, цикълът с начало на ред № 3 отнема време $\Theta(S)$, а времето на целия алгоритъм е $\Theta(n+S)$. Това представлява времевата сложност на алгоритъма при най-добри входни данни.

Последният алгоритъм се оценява с **40 точки.**

Задачата може да бъде решена и посредством рекурсивния вариант на динамичното програмиране — рекурсия със запомняща функция:

```
CountSubsetSum(A[1...n],S)
1) dyn[0...n][0...S]: масив от цели числа;
2) // dyn[k][t] = броят на начините, по които числото t
3) // се представя като сбор на елементи на A[1...k]
4) // с различни индекси.
5) for k ← 0 to n do
6)    for t ← 0 to S do
7)       dyn[k][t] ← -1  // Невалидна стойност.
8) return F(dyn,A,n,S)
```

Запомнящата функция F се дефинира така:

```
F(dyn[0...n][0...S], A[1...n], k, t)
 1) if dyn[k][t] = -1
 2)    if t = 0
 3)       dyn[k][t] ← 1
 4)    else if k = 0
 5)       dyn[k][t] ← 0
 6)    else if t ≥ A[k]
 7)       dyn[k][t] ← F(dyn,A,k-1,t) + F(dyn,A,k-1,t-A[k])
 8)    else // t < A[k]
 9)       dyn[k][t] ← F(dyn,A,k-1,t)
10) return dyn[k][t]
```

Този алгоритъм е коректен, има сложност по време и по памет $\Theta(nS)$ при всякакви входни данни. Той е малко по-бавен от итеративния вариант, тъй като обхожда динамичната таблица два пъти: първия път я инициализира с невалидни стойности, втория път смята верните бройки. Второто обхождане дори при най-лоши входни данни (когато масивът A съдържа все единици) попълва само част от таблицата. Ако числата в масива A са по-големи от S, тогава ред № 7 от запомнящата функция F не се изпълнява нито един път, поради което ред № 8 от алгоритъма CountSubsetSum изразходва време $\Theta(n)$, но общото време на CountSubsetSum остава $\Theta(nS)$ заради инициализирането на динамичната таблица с невалидни стойности.

Рекурсивният алгоритъм със запомняща функция носи **20 точки.**

Задача 1 се свежда до известна задача за ориентирани ациклични графи:

1) Построяваме ориентиран граф, чиито върхове са всички наредени двойки от цели числа $(k\,;\,t)$, където $0 \le k \le n$ и $0 \le t \le S$, и един фиктивен връх X; Във всеки връх $(k\,;\,t)$ при $k \ge 1$ и $t \ge 1$ влизат едно или две ребра:
   — от върха $(k-1;\,t)$ винаги;
   — от върха $(k-1;\,t-\mathrm{A}[k])$, ако $t \ge \mathrm{A}[k]$.
   Във върховете $(k\,;\,0)$ при всяко цяло k от 0 до n вкл. влиза едно ребро от X. Във върховете $(0\,;\,t)$ не влизат ребра при никое цяло число t от 1 до S вкл. Полученият граф е ацикличен, защото във върха X не влизат никакви ребра, а ребрата между останалите върхове сочат от наредена двойка с по-малък към наредена двойка с по-голям първи елемент.
2) В построения ориентиран ацикличен граф търсим броя на всички пътища от върха X до върха $(n\,;\,S)$. За тази цел използваме съответния алгоритъм за динамично програмиране при ориентирани ациклични графи. По принцип е нужно първо да извършим топологично сортиране, но в тази задача можем да пропуснем това действие, защото графът е построен в сортиран вид: създаваме наредените двойки по нарастваща стойност на първия елемент, а върхът X предхожда всички други върхове.

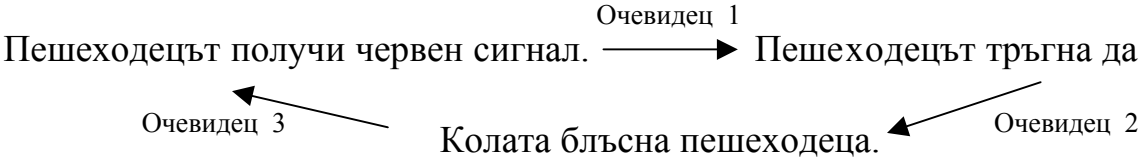
Анализ на алгоритъма: Всяка от двете стъпки изразходва време, което е пропорционално на размера на графа (общия брой на върховете и ребрата му). Същото важи за допълнителното количество памет. Следователно сложността по време и по памет е от един и същи порядък с размера на графа. Порядъкът не зависи от входните данни.

Графът има $(n+1)\,(S+1) + 1 = \Theta(nS)$ върха. Броят на ребрата е $\mathrm{O}(nS)$, защото във всеки връх влизат най-много две ребра. Ето защо размерът на графа има порядък $\Theta(nS)$. Това е също сложността на алгоритъма по време и памет при всякакви входни данни.

Задача 2. Входните данни на задачата се представят като ориентиран граф G: върховете съответстват на събитията, а ребрата показват реда на събитията. По-точно, съществува ребро от върха x към върха y тогава и само тогава, когато някой очевидец е заявил, че събитието x е предхождало събитието y. Върховете и ребрата на графа нямат тегла.

Ако едно показание се дава от неколцина очевидци, съответното ребро може да бъде кратно, тоест G може да се окаже мултиграф. Това не е пречка за алгоритмите, предложени по-нататък в решението на задачата.

Пример: Показанията от условието на задачата се представят така:



Съществува противоречие между очевидците относно реда на събитията, ако и само ако графът съдържа ориентиран цикъл. Търсене на цикъл в граф се извършва с линейна времева сложност чрез ***обхождане в дълбочина.***

Ако няма противоречие, събитията могат да се подредят хронологично поне по един начин. Можем да ги подредим с ***топологично сортиране*** чрез ***обхождане в дълбочина*** на дадения граф.

Първите две подзадачи могат да се решат с едно обхождане на графа. Ако открием противоречие (цикъл), прекратяваме обхождането предсрочно; в противен случай обхождането завършва с един възможен ред на събитията: $v_1, v_2, v_3, \ldots, v_n$.

Този ред е единствен тогава и само тогава, когато има ребро от v_i към v_{i+1} за всяко i. Иначе съществува втора възможна последователност на събитията: тя се получава, като разменим върховете v_i и v_{i+1}. Двата върха се откриват чрез още едно ***обхождане на графа; видът на обхождането няма значение.***

Всяко обхождане има линейна времева сложност в най-лошия случай, затова същото важи за предложените алгоритми.

СХЕМА ЗА ТОЧКУВАНЕ

Задача 1 носи общо 30 точки — както е указано в условието.
Предвидени са допълнителни 10 точки за незадължителна оптимизация.
Ако липсва анализ на сложността, се отнемат 2 точки.

Задача 2 носи общо 20 точки — по пет точки за всяко подусловие.
Предвидени са допълнителни 5 точки, които се дават за уточнението,
че алгоритмите за второто и третото подусловие могат да използват
едно и също обхождане в дълбочина.