ЗАБЕЛЕЖКИ КЪМ РЕШЕНИЯТА НА ЗАДАЧИТЕ ПО ДАА ОТ РЕДОВНАТА СЕСИЯ В СУ, ФМИ (08. 02. 2016 ГОД.)

Задача 1 б може да се реши за линейно вместо за квадратично време чрез алгоритъма от подусловие "а", т.е. търсене в ширина. Отново разглеждаме насочен мултиграф $G(V, E)$, чиито върхове са предметите, а ребра — предложенията за замяна. Всяко ребро съдържа не само номера на човека, направил предложението, но и най-късния възможен момент на замяната. При търсенето в ширина за всеки връх u пазим момента t на първото достигане, а в опашката на търсенето добавяме само онези ребра, излизащи от u, чийто момент $\geq t+1$. Например, ако сме стигнали до някой връх за 3 стъпки, то при обхождане на наследниците му има смисъл да преминаваме само по такива ребра, чиито замени могат да бъдат извършени на стъпка № 4, № 5, № 6 и т.н.

```
BFS( G(V,E), s, f )
if f = s
    print "Няма нужда от замени: желаният предмет е у нас."
    return
for v ∈ V
    v.state ← unexplored
s.state ← open
s.time ← 0   // s е предметът, който имаме отначало
Q ← CreateEmptyQueue
Q.Append(s)
while Q ≠ ∅ do
    u ← Q.ExtractFirst
    t ← u.time + 1
    for ⟨v, offerer, deadline⟩ ∈ Adj(u)
        if v.state = unexplored and deadline ≥ t
            v.state ← open
            v.parent ← u          // предметът, който сме заменили за предмета v
            v.giver ← offerer     // човекът, от когото сме получили предмета v
            v.time ← t            // моментът на замяната
            if v ≠ f       // f е предметът, който искаме
                Q.Append(v)
            else
                // желаният предмет е намерен
                // отпечатваме редицата от замени в обратен ред
                while v ≠ s do
                    u ← v.parent
                    t ← v.time
                    h ← v.giver
                    print "Заменяме", u, "за", v, "в момента", t, "с човека", h, "."
                    v ← u
                return
print "Не можем да се сдобием с желания предмет."
```

Този алгоритъм е конкретна реализация на търсенето в ширина, затова времевата сложност е линейна: $\Theta(m+n) = \Theta(k)$, тъй като $m = k$, $0 \leq n \leq 2k$, следователно $k \leq m+n \leq 3k$. Условието допуска квадратична сложност с цел по-голямо разнообразие от възможни решения.

Задача 2. Оригиналното решение прилага алгоритъма на Дейкстра, като модифицира теглата на ребрата на графа: заменя всяка от надеждностите с отрицателния ѝ логаритъм. Вместо това можем да променим алгоритъма, а да запазим теглата (т.е. да не логаритмуваме надеждностите):
— Инициализираме върховете с минимални стойности, например 0 или -1.
— Инициализираме началния връх с 1.
— При натрупване на надеждностите използваме умножение вместо събиране.
— При избор на текущ връх и при релаксация търсим максимум вместо минимум.
По принцип класическият алгоритъм на Дейкстра *не може* да търси най-дълъг път в граф. Причината е, че дължините на ребрата се натрупват чрез събиране, което води до неограничено нарастване. Заменянето на събирането с умножение не спасява положението. В тази задача обаче сме облагодетелствани от факта, че надеждностите на комуникационните канали са между 0 и 1: умножението на все повече надеждности води до намаляване на произведението, така че *в този конкретен случай* алгоритъмът правилно ще търси най-дълъг път.

Задача 3. В много решения се прави опит за прилагане на *алчен алгоритъм.* За съжаление, тези решения са неправилни.

Пример 1: Намираме най-големия точен квадрат, ненадвишаващ n. Изваждаме го от n и към остатъка прилагаме същата стратегия. Продължаваме, докато получим 0. Например при $n = 10$ намираме 9, за разликата $10 - 9 = 1$ намираме 1, остава $1 - 1 = 0$ и спираме търсенето; получено е представянето $10 = 9 + 1$, което наистина е оптимално.

Контрапример 1: При $n = 18$ намираме 16, за разликата $18 - 16 = 2$ намираме 1, за новата разлика $2 - 1 = 1$ пак намираме 1, остава $1 - 1 = 0$ и търсенето приключва; получено е представянето $18 = 16 + 1 + 1$, което обаче не е оптимално: $18 = 9 + 9$ е по-кратко.

Пример 2: Въвеждаме втори параметър $k \leq n$ и търсим представяне на n като сбор от най-малък брой квадрати, ненадвишаващи k. Решението на оригиналната задача се получава при $k = n$. За да има по-малко събираеми, се стремим те да са възможно най-големи. Ето защо намираме най-големия точен квадрат, ненадхвърлящ k, т.е. $\lfloor \sqrt{k} \rfloor^2$, и го добавяме в сумата колкото може повече пъти, т.е. $\left\lfloor \dfrac{n}{\lfloor \sqrt{k} \rfloor^2} \right\rfloor$ пъти. За остатъка прилагаме същия алгоритъм. Например при $n = 10$ и $k = 10$ намираме $\lfloor \sqrt{k} \rfloor^2 = 9$, което може да се добави в сбора само веднъж; остатъкът $10 - 9 = 1$ е точен квадрат; получава се представянето $10 = 9 + 1$.

Контрапример 2: При $n = 243$ и $k = 100$ числото 100 може да участва в сбора най-много два пъти. За остатъка 43, както и да продължи изпълнението на алгоритъма, ще са нужни поне три точни квадрата, например $43 = 25 + 9 + 9$ (два квадрата не стигат, защото 43 дава остатък 3 при деление на 4). Така за 243 ще са нужни общо поне пет квадрата: $243 = 100 + 100 + 25 + 9 + 9$ според този алгоритъм. В действителност можем да минем с три квадрата: $243 = 81 + 81 + 81$. Следователно алгоритъмът не е правилен.

Задачата може да се реши чрез *динамично програмиране*, както е показано в публикуваните решения. Този алгоритъм има сложност $n\sqrt{n}$. Възможно е да се състави по-бърз алгоритъм, ако се използват някои сведения от теорията на числата.

Задача 5 може да се реши по още един начин:
1) Намираме най-малкото число x в масива.
2) От всички числа в масива изваждаме $x - 1$.
3) Сортираме получения масив от положителни числа.
4) Към всички числа в масива прибавяме $x - 1$.
Тази редукция също има линейна времева сложност и работи коректно.

Пример: Ако даденият масив е $(9\ ;\ 8\ ;\ -5\ ;\ 3\ ;\ -4)$, намираме $x = -5$, изваждаме $x - 1 = -6$, т.е. прибавяме 6, и получаваме нов масив: $(15\ ;\ 14\ ;\ 1\ ;\ 9\ ;\ 2)$. Сортираме го: $(1\ ;\ 2\ ;\ 9\ ;\ 14\ ;\ 15)$; след това вадим 6 от всички числа: $(-5\ ;\ -4\ ;\ 3\ ;\ 8\ ;\ 9)$.