DAA Упражнение 12 (Динамично програмиране)

зад.1/ Knapsack 0/1. Дадени са n предмета, като всеки от тях има тегло и цена. Дадено е число K: капацитет е овена на раницата. Всеки предмет може или да го вземем целия, или да не го взимаме (не можем да вземем 1/2 предмет). Да се намери максималната сума, която можем да поберем в раницата, така че сумарното тегло от предметите да е $\leq K$.

Решение: Ще разглеждаме всеки един предмет и дали неговото слагане ще надвиши капацитета на раницата. Ако не надвишава капацитета на раницата взимаме дадения предмет и свеждаме задачата, като намаляваме капацитета на раницата с теглото на дадения предмет. (+премахваме предмета) За този ход ще конструираме решението, като разглеждаме капацитета на раницата да е 0, 1, .., K-1, K и също ще разглеждаме постепенно до кой предмет можем да взимаме.

Заб: Не е непременно задължително да напълним капацитета на раницата напълно. В някои случаи оптималното решение ще е такова, че сумарното тегло на предметите ще е $< K$.

Пример: $P = \{1, 2, 5, 6\}$ $K = 8$

$W = \{2, 3, 4, 5\}$

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	1	1	1	1	1	1	1
2	0	0	1	2	2	3	3	3	3
3	0	0	1	2	5	5	6	7	7
4	0	0	1	2	5	6	6	7	8

- капацитетът на раницата

- Решението от първия до кой предмет разглеждаме

Рекурсивната декомпозиция е $M[i][w] = \max(M[i-1][w], M[i-1][w - W[i]] + P[i])$ чиято семантика е или взимаме предмета и свеждаме капацитета на раницата до $K - W[i]$ (теглото на предмета) и премахваме дадения предмет (за да не можем да го вземем повторно), или не взимаме предмета и свеждаме задачата до същата, но все едно този предмет го нямаме.

```
SOLUTION(A[1..n]: масив от предмети, K - капацитет на раницата)
1. M[0...n][0...k] - заделяме масив
2. M[0][0...k] ← 0 // Правим първия ред и стълб да са 0-ли
3. M[1...n][0] ← 0 //
4. for i ← 1 to n
5.     for j ← 1 to k
6.         if i.weight ≤ j //проверяваме да не надвишим капацитета на раницата
7.             M[i][j] ← max(M[i-1][j], M[i-1][j - i.weight] + i.price)
8. return M[n][k]
```

$T(n) = O(k.n)$

$M(n) = O(k.n)$

Може да забележите, че единствено използваме текущият ред и предишния, така че можем да намалим сложността по памет до $O(n)$. Напишете това решение за домашно.

зад.2/ Дадени са n на брой възли, които са в редица едни до други. Всеки възел трябва да бъде боядисан в един от трите цвята зелен/червен/син. За всеки възел ни е дадено колко ще струва да я боядисаме в даден цвят. Предложете алгоритъм, който намира цената (най-малката) за боядисване на всички възли, като не може да има две съседни възли в един и същ цвят.

Решение: Ще разглеждаме по-малки редици от възли, като разглеждаме възможностите само я боядисаме в червено + оптималното решение за подредицата с дължина $n-1$, която завършва в синьо/зелено. Аналогично и за другите цветове. (Тук под подредица се смята непрекъснат подмасив от 1 до i).

Пример: цена за боядисване в

#възел	1	2	3	4	5	6
червено	1	8	3	6	4	2
синьо	2	6	2	9	1	15
зелено	4	4	2	1	2	5

оптимално решение завършващо на

червено	0	1	8+2=10	3+5=8	6+7=13	4+8=12
синьо	0	2	6+1=7	2+5=7	9+8=17	1+8=9
зелено	0	4	4+1=5	2+7=9	1+7=8	2+13=15

Решението тук е $\min(11, 27, 14) = 11$.

2+9=11
15+12=27
5+9=14

SOLUTION(R[1...n], G[1...n], B[1...n] : масиви с цена за боядисване)

1. OPT_R[1...n], OPT_R[1] ← R[1]
2. OPT_B[1...n], OPT_B[1] ← B[1]
3. OPT_G[1...n], OPT_G[1] ← G[1]
4. for i ← 2 to n
5. OPT_R[i] ← R[i] + min(OPT_B[i-1], OPT_G[i-1])
6. OPT_B[i] ← B[i] + min(OPT_R[i-1], OPT_G[i-1])
7. OPT_G[i] ← G[i] + min(OPT_R[i-1], OPT_B[i-1])
8. return min(OPT_G[n], OPT_B[n], OPT_R[n])

Помислете как бихте решили задачата, ако освен оптимална цена се искаме да се върне и начин, по който е боядисана всяка къща.

зад. 3/ Даден е масив A[1...m][1...n]. Започваме в клетка A[1][1] и искаме да стигнем до клетка A[m][n], като имаме право да се движим с една клетка надолу и една клетка надясно. Да се намери максималната тежест на такъв път.

Решение: За определена клетка A[i][j] можем да стигнем или от A[i-1][j], или A[i][j-1]. Следователно максималната тежест на път от A[1][1] до A[i][j] е теглото в A[i][j] + max(M[i][j-1], M[i-1][j], където M[i][j] е максимално тежък път от A[1][1] до A[i][j].

Пример:

1	7	12
3	11	5
12	8	10

=: A[m][n] входният масив

M[m][n] := (изходният масив)

1 →	8	20
	↓	
4	19	25
	↓	
16	27 →	37

SOLUTION(A[1...m][1...n] : масив от +числа) // приемаме, че където излизаме от // масива, то е 0

1. M[1...m][1...n] : заделяне масив
2. for i ← 1 to m
3. for j ← 1 to n
4. M[i][j] ← A[i][j] + max(M[i-1][j], M[i][j-1])
5. return M[m][n]

Може да използваме константна допълнителна памет, като презаписваме върху масива A[1...m][1...n]. Помислете как може да се възстанови пътя през който сме минали.

зад. 4/ Дадена е азбука $\Sigma = \{a, b, c, .., z\}$ и дума Σ^n с дължина n. Да се намери дължината на най-дългата поддума, която е палиндром.

Решение: Празната дума и еднобуквените думи са палиндроми. Палиндроми са също и думи, чиито първа и последна буква е еднаква и поддумата без първата и последната буква е палиндром. За решението ще направим точно това, като разглеждаме всяка поддума. Първо с дължина 1, после 2 и така до цялата дума.

Пример: word[1...6] = a c b c c b
1 2 3 4 5 6

	1	2	3	4	5	6
1	T	F	F	F	F	F
2	T	T	F	T	F	F
3		T	T	F	F	T
4			T	T	T	F
5				T	T	F
6					T	T

от кой символ гледаме

:= до кой символ гледаме

```
SOLUTION(word[1...n])
1. palindrome[1...n][1...n] ← {{F...F},...}
2. maxLen ← 1
3. for i ← 1 to n // едноуквена дума
4.     palindrome[i][i] ← T
5. for i ← 1 to n-1 // празната дума
6.     palindrome[i+1][i] ← T
7. for k ← 1 to n-1
8.     for i ← 1 to n-k
9.         palindrome[i][i+k] ← (word[i] = word[i+k]
           and palindrome[i+1][i+k-1])
10.        if palindrome[i][i+k]
11.            maxLen ← k+1
12. return maxLen
```

зад. 5/ Даден е регулярен израз с дължина n над азбуката $\{a, .., z, -, *\}$ и низова дума с дължина m над $\{a, .., z\}$. Да се провери дали дадената дума може да се генерира от регулярния израз. Семантиката на _ е някоя буква от $\{a...z\}$, семантиката на a^* е ε (празната дума), a, aa, aaa, ... Семантиката на $\_^*$ е произволен стринг (т.е. не се фиксира една буква).

Решение: Ще сведем задачата до това дали подизраз от 1 до i успява да генерира думата от 1 до j. Това е възможно във следните случаи:

1) Ако на i-та позиция в рег израз стои същата буква като на j-та позиция в думата, то проверяваме дали рег. израз от 1 до i-1 генерира думата от 1 до j-1.

2.) Ако на i-та позиция в рег. израз стои '_', то проверяваме дали рег. израз от 1 до i-1 успява да генерира думата от 1 до j-1.

3.) Ако на i-та позиция ($i \geq 2$) в рег. израз стои '*', то проверяваме дали рег. израз от 1 до i-2 успява да генерира думата от 1 до j (т.е. взимаме празната дума)

4.) Ако на i-та позиция в рег. израз стои '*' и на i-1 позиция в рег. израз стои същата буква като на j-тата позиция в думата или '_', то проверяваме дали рег. израз от 1 до i генерира думата от 1 до j-1 (добавяме една буква)

Заб: Даден ни е валиден регулярен израз, т.е. не започва със *, няма две съседни ** и т.н.

```
Solution(regexpr[1..n], word[1...m])
1. d[0...m][0...n] ← {{F,...F},...}
2. d[0][0] ← T // празният regexpr генерира празната дума
3. for j ← 2 to n with step 2
4.     d[0][j] ← (d[0][j-2] and regexpr[i] = '*') // т.е взимаме ε всеки път
5. for i ← 1 to m
6.     for j ← 1 to n
7.         if regexpr[j] = word[i] or regexpr[j] = '_' // 1) и 2)
8.             d[i][j] ← d[i-1][j-1]
9.         else if j>2 and regexpr[j] = '*' and d[i][j-2] = T // 3.)
10.            d[i][j] ← T
11.        else if regexpr[j] = '*' and (word[i] = regexpr[j] or regexpr[j] = '_')
               and d[i-1][j] = T // 4.)
12.            d[i][j] ← T
13. return d[m][n]
```

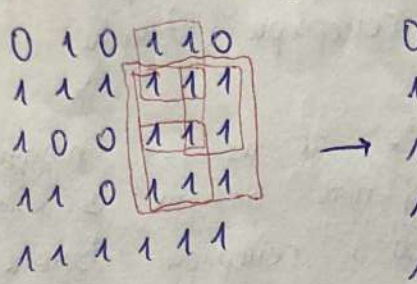
Заб: Клетка d[i][j] има семантиката дали думата от 1 до i-ти символ може да се генерира чрез рег. израз от 1 до j-ти символ.

зад. б/ Дадена е булева матрица A[1...m][1...n]. Да се намери лицето на най-големия квадрат, съдържащ единствено единици.

Решение: Единиците сами по себе си са квадрати с дължина 1. За да образуване квадрат с дължина $n+1$ е нужно на текущата клетка да имаме 1, отгоре и отляво да имаме квадрати с дължина

на страната n. Трябва също така да има 1ца в горния ляв ъгъл на квадрата, което е същото като да има квадрат с дължина на страната n, намиращ се с една позиция наляво и нагоре.

Пример:



$$\begin{matrix}0&1&0&1&1&0\\1&1&1&1&2&1\\1&0&0&1&2&2\\1&1&0&1&2&3\\1&2&1&1&2&3\end{matrix}$$

Рекурсивната декомпозиция е $d[i][j]$ $\begin{cases}1+\min(d[i-1][j-1], d[i-1][j], d[i][j-1]), & \text{ако } A[i][j]=1\\0, & \text{ако } A[i][j]=0\end{cases}$

(В клетка $d[i][j]$ е записана дължината на най-големия квадрат, чието долен десен ъгъл е там)

```
SOLUTION(A[1..m][1..n])
1. maxLen ← 1
2. d[1..m][1..n] ← {{0,..,0},..}
3. for j ← 1 to n
4.     d[1][j] ← A[1][j]
5.     maxLen ← max(maxLen, d[1][j])
6. for i ← 2 to m
7.     d[i][1] ← A[i][1]
8.     maxLen ← max(maxLen, d[i][1])
9.     for j ← 2 to n
10.        if A[i][j] = 1 then
11.            d[i][j] ← 1 + min(d[i-1][j-1], d[i-1][j], d[i][j-1])
12.        maxLen ← max(maxLen, d[i][j])
13. return maxLen * maxLen
```