Динамично програмиране

Зад. 1

Дадени са $n \in \mathbb{N}^+$ различни вида монети - coins[1 .. n] $\in (\mathbb{N}^+)^n$. Да се намери минималния брой монети, необходими да се получи дадена сума $S \in \mathbb{N}_0$ при условие, че:

***a*)** имаме по 1 монета от вид

***b*)** имаме неограничен брой монети

Пример:

$\begin{cases} \text{Instance : coins[1 ..4] = [1, 5, 6, 8], } S = 11 \\ \text{Solution : 2} \end{cases}$

***a*)**

Ясно е, че задачата можем да я решим **тъпо**, като просто пробваме всички възможности и взимаме най-добрата от които. Сега ще разгледаме как да построим решение на задачата по схемата **Динамично Програмиране** и ще покажем защо привидно полиномиалното решение, всъщност е експоненциално.

Нека да разгледаме директно попълнена табличката (всяка задача решена по схемата ДП си има такава) и ще коментираме по нея:

	0	1	2	3	4	5	6	7	8	9	10	11
{}	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
{1}	0	1	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
{1, 5}	0	1	∞	∞	∞	1	2	∞	∞	∞	∞	∞
{1, 5, 6}	0	1	∞	∞	∞	1	1	2	∞	∞	∞	2
{1, 5, 6, 8}	0	1	∞	∞	∞	1	1	2	1	2	∞	**2**

На всяка клетка имаме: $\begin{cases} \infty & \text{, ако не можем да го получим сумата от текущия стълб с монетите за текущия ред} \\ \text{минималния брой монети} & \text{, ако можем да го получим сумата от текущия стълб с монетите за текущия ред} \end{cases}$.

Примерно в клетката DP[{1, 5, 6}][7] = 2 имаме, че 7 се получава минимално чрез 2 монети от {1, 5, 6}.

Разбира се търсеното от ***a*)** се намира в клетката DP[{1, 5, 6, 8}][11] = 2.

Как се образува таблицата:

· Първия ред и първата колона са ясни - нули за колоната и ∞ за реда

· Образуваме редовете последователно от горе надолу и всеки ред от ляво надясно по следната ф-ла:
dp[r][c] ← min(dp[r − 1][c], 1 + dp[r − 1][c − coins[c]]), като внимаваме индексите да не излязат извън матрицата.

```
1. task1a(coins[1 .. n], S) : // coins ∈ (ℕ⁺)ⁿ, n ∈ ℕ⁺, S ∈ ℕ₀
2.       dp[0 .. n][0 .. S] ← [[∞, …, ∞], …, [∞, …, ∞]]
3.       for r ← 0 to n
4.                dp[r][0] ← 0
5.       for r ← 1 to n
6.                for c ← 1 to S
7.                         dp[r][c] ← min(dp[r − 1][c], 1 + dp[r − 1][c − coins[r]])
8.       return dp[n][S]
```

Забележка Ако излезем извън матрицата, то приемаме че стойността там е +∞

Важно Този алгоритъм очевидно е с времева сложност $\theta(n.S)$, което е полином на n и S.. как така сложността се смъкна до привидно полиномиална? Нека да разгледаме следните две прости функции за илюстрация:

```
1. f(n) : // n ∈ ℕ0
2.     for i ← 1 to n
3.         print 'a'
```

```
1. g(A[1 .. n]) : // A ∈ ℤ^n, n ∈ ℕ0
2.     for i ← 1 to n
3.         print 'a'
```

Очевидно и двете функции са със сложност $\theta(n)$. Уловката е в това какво дефинираме като големина на входа.

· Когато сме във втроя случай (с който сме свикнали), то големината на входа е броя елементи на масива - тоест големината на входа е n. Оттук сложността спрямо големината на входа е $\theta(n)$. **[1]**

· Когато сме в първия случай, то големината на входа е броя символи, от които е съставено числото n или по-конкретно $\log_{10}(n)$, като основата е без значение. Ще използваме основа 10 за по-добра илюстрация. Нека положим $m = \log_{10}(n)$ е големината на входа. Тогава сложността спрямо големината на входа m е $\theta(n) = \theta(10^m)$ или с други думи - експоненциална спрямо големината на входа.

Деф (псевдо-полиномиален алгоритъм)

Алгоритми, които са с полиномиална сложност спрямо числовата стойност на вх. параметър, наричаме псевдо-полиномиални

Забележка task1a(n, S) е псевдо-полиномиален алгоритъм със сложност $\theta(n.S)$, а е с експоненциална сложност спрямо входа

***b*)**

Не е трудно да съобразим, че схемата за изчисление е абсолютно аналогична с една единствена разлика: $\text{dp}[r][c] \leftarrow \min\left(\text{dp}[r-1][c],\ 1 + \text{dp}[\underbrace{r}_{\text{беше } r-1}][c - \text{coins}[c]]\right)$. Идеята е, че може оптималното решение за $c - \text{coins}[c]$ може да използва текущата монета, а на предходната подточка искаме да си гарантираме, че не не сме я използвали до момента. Табличката ѝ:

	0	1	2	3	4	5	6	7	8	9	10	11
{}	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
{1}	0	1	2	3	4	5	6	7	8	9	10	11
{1, 5}	0	1	2	3	4	1	2	3	4	5	2	3
{1, 5, 6}	0	1	2	3	4	1	1	2	3	4	2	2
{1, 5, 6, 8}	0	1	2	3	4	1	1	2	1	2	2	**2**

Съответно и псевдокода, като отново внимаваме за индексите:

```
1. task1b(coins[1 .. n], S) : // coins ∈ (ℕ+)^n, n ∈ ℕ+, S ∈ ℕ0
2.     dp[0 .. n][0 .. S] ← [[∞, …, ∞], …, [∞, …, ∞]]
3.     for r ← 0 to n
4.         dp[r][0] ← 0
5.     for r ← 1 to n
6.         for c ← 1 to S
7.             dp[r][c] ← min(dp[r − 1][c], 1 + dp[r][c − coins[r]])
8.     return dp[n][S]
```

Забележка Ако излезем извън матрицата, то приемаме че стойността там е $+\infty$

Зад. 2

Дадени са $n \in \mathbb{N}^+$ различни вида монети - $coins[1 \,..\, n] \in (\mathbb{N}^+)^n$. При условие, че имаме неограничен брой монети от всеки вид, да се намери броя различни начини да се получи дадена сума $S \in \mathbb{N}_0$.

Пример:

$\begin{cases} \text{Instance : } coins[1 \,..3] = [1,\ 2,\ 5],\ S = 5 \\ \text{Solution : } 4 \text{ // } 1 + 1 + 1 + 1 + 1 = 1 + 1 + 1 + 2 = 1 + 2 + 2 = 5 \end{cases}$

Ясно е, че схемата за изчисление ще е много подобна на тази от **Зад. 1**. Даже може да направим аналогични случаи за *a*) и *b*). В случая ни изискват само аналога на *b*). Схемата за изчисление е следната: $dp[r][c] \leftarrow dp[r-1][c] + dp[r][c - coins[r]]$. Съответно таблицата с историята на изчисления е:

	0	1	2	3	4	5
{}	1	0	0	0	0	0
{1}	1	1	1	1	1	1
{1, 2}	1	1	2	2	3	3
{1, 2, 5}	1	1	2	2	3	**4**

Остана да напишем и псевдокода:

```
1. task2(coins[1 .. n], S) : // coins ∈ (ℕ⁺)^n, n ∈ ℕ⁺, S ∈ ℕ₀
2.     dp[0 .. n][0 .. S] ← [[0, …, 0], …, [0, …, 0]]
3.     for r ← 0 to n
4.         dp[r][0] ← 1
5.         for c ← 1 to S
6.             dp[r][c] ← dp[r − 1][c] + dp[r][c − coins[r]]
7.     return dp[n][S]
```

Заблежка Ако излезем извън матрицата, то приемаме че стойността там е 0

Зад. 3

Даден е регулярен израз $regex[1 \,..\, n] \in \{a, b, \ldots, z, .,*\}^n$ и дума $w[1 \,..\, m] \in \{a, b, \ldots, z\}^m$. Да се провери дали думата се *чете* от регулярния израз. Нека за улеснение регулярните изрази се състоят само от малки латински букви и метасимволите '.' и '*'.

Примери (компактно описани):

regex : *a.b*
w : acb, aab, axb ⟼ TRUE
w : ab, *b*, cb, axyb ⟼ FALSE

regex : *a*b*
w : *b*, ab, aab, aaab ⟼ TRUE
w : *a*, acb ⟼ FALSE

regex : *a*b.*c*
w : bc, bxc, bxxc, bxyc, abxc, abxxc, abxyc ⟼ TRUE
w : ay, ab ⟼ FALSE

Схемата за изчисление тук не е толкова тривиална, даже напротив - изглежда като магия:

$$d[i][j] \leftarrow \begin{cases} d[i-1][j-1] & , w[i] = regex[j] \vee regex[j] = '.' \\ d[i][j-2] & , regex[j] = '*' \,\&\, d[i][j-2] = \text{TRUE} \\ d[i-1][j] & , regex[j] = '*' \,\&\, d[i][j-2] = \text{FALSE} \,\&\, (w[i] = regex[j-1] \vee regex[j-1] = '.') \\ \text{FALSE} & , \text{else} \end{cases}$$

Накратко какво прави всеки случай:

· Ако текущата буква на регекса съвпада с текущата буква на думата, то вземи стойността в табличката, отговаряща на същия регекс и дума, но без последните им символи. Аналогично ако регекса е метасимвола '.'.

· Ако имаме, че текущия символ на регекса е '*', то може да не го гледаме нито него, нито предходния символ - затова взимаме стойността в табличката, отговаряща на същата дума и на същия регекс, но с дължина две по-малко. Да обърнем внимание, че го взимаме само ако стойността е истина, иначе разглеждаме третия случай.

· Ако видим, че текущия символ на регекса е '*' и предходната буква на регекса съвпада с последната буква на думата, то тогава регекса ще *чете* текущата дума ↔ регекса *чете* текущата дума без последния символ.

· Иначе връщаме лъжа - примерно се различават символите.

Пример:

{ Instance : regex[1 ..6] = [x, a, $*$, b, ., c], w = [x, a, a, b, y, c]
{ Solution : TRUE

Таблица с история на изчисленията:

	eps	x	a	*	b	.	c
eps	T	F	F	F	F	F	F
x	F	T	F	T	F	F	F
a	F	F	T	T	F	F	F
a	F	F	F	T	F	F	F
b	F	F	F	F	T	F	F
y	F	F	F	F	F	T	F
c	F	F	F	F	F	F	**T**

Остана да напишем и псевдокода:

```
1. task3(regex[1 .. n], w[1 .. m]) : // regex ∈ {a, b, …, z, .,*}^n, w ∈ {a, b, …, z}^m, n, m ∈ ℕ0
2.     d[0 .. m][0 .. n] ← [[FALSE, …, FALSE], …, [FALSE, …, FALSE]]
3.     d[0][0] ← TRUE
4.     for i ← 2 to n with step 2
5.         d[0][i] ← (d[0][i − 2] and regex[i] = '*')
6.     for i ← 1 to m
7.         for j ← 1 to n
8.             if regex[j] = w[i] then
9.                 d[i][j] ← d[i − 1][j − 1]
10.            else if j > 2 and regex[j] = '*' and [i][j − 2] then
11.                d[i][j] ← TRUE
12.            else if regex[j] = '*' and (w[i] = regex[j] or regex[j] = '.') and d[i − 1][j] then
13.                d[i][j] ← TRUE
14.    return d[m][n]
```

Зад. 4

Дадена е матрица $A[1 .. m][1 .. n] \in (\{0, 1\}^m)^n$. Да се намери най-голям квадрат от единици и да се изведе от колко единици е съставен.

Пример:

{ Instance : A[1 ..5][1 ..6] = [[0,1,0,1,1,0],[1,1,1,1,1,1],[1,0,0,1,1,1],[1,1,0,1,1,1],[1,1,1,1,1,1]]
{ Solution : 9

Обяснение на пример:

С удебелен шрифт е обозначен примерен максимален квадрат от единици. Има още точно един максимален квадрат от единици (същият, но изместен с един ред надолу).

$$\begin{pmatrix} 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & \mathbf{1} & \mathbf{1} & \mathbf{1} \\ 1 & 0 & 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} \\ 1 & 1 & 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} \\ 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Идея:

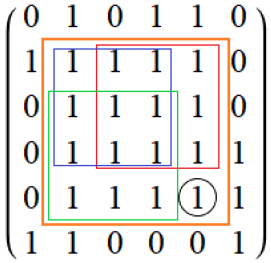
Инициализираме си табличка $m \times n$ като първия ред и първата колона копират тези на входната матрица:

	1	2	3	4	5	6
1	0	1	0	1	1	0
2	1					
3	1					
4	1					
5	1					

След това започваме да попълваме ред по ред (разбира се може и колона по колона) по следния начин:

$$d[i][j] \leftarrow \begin{cases} 1 + \min(d[i-1][j],\ d[i-1][j-1],\ d[i][j-1]) & , A[i][j] = 1 \\ 0 & , A[i][j] = 0 \end{cases}$$

Идеята на тази табличка е да ни казва за всяка клетка $d[i][j]$ колко е максималната страна на квадрат от единици с долен десен ъгъл $\langle i, j \rangle$. Ясно е, че ако горната, горе лявата и лявата клетка могат да бъдат долен десен ъгъл на квадрат с дължина на страната k, то текущата клетка може да бъде долен десен ъгъл на квадрат с дължина на страната $k + 1$. Картинката изглежда нещо от сорта:



Разбира се може някое от червеното/синьото/зеленото квадратче да са дори с по-голяма дължина.. но взимаме минималната страна от трите. Сега прилагаме схемата на изчисление за остатъка от табличката и получаваме:

	1	2	3	4	5	6
1	0	1	0	1	1	0
2	1	1	1	1	2	1
3	1	0	0	1	2	2
4	1	1	0	1	2	3
5	1	2	1	1	2	3

Отговора е квадрата на максималната стойност в табличката. Да обърнем внимание, че **НЕ** е задължително отговора да ни се намира в клетката най-долу и най-дясно! В случая излезе на късмет!

Псевдокод:

```
1. maxSqArea(A[1 .. m][1 .. n]) : // A ∈ ({0, 1}^m)^n
2.     maxSide ← 0
3.     d[1 .. m][1 .. n] ← [[0, …, 0], …, [0, …, 0]]
4.     for j ← 1 to n
5.         d[1][j] ← A[1][j]
6.         maxSide ← max(maxSide, d[1][j])
7.     for i ← 2 to m
8.         d[i][1] ← A[i][1]
9.         maxSide ← max(maxSide, d[i][1])
10.        for j ← 2 to n
11.            d[i][j] ← A[i][j]*(1 + min(d[i − 1][j], d[i − 1][j − 1], d[i][j − 1]))
12.            maxSide ← max(maxSide, d[i][j])
13.    return (maxSide)^2 // area = (side)^2
```

Сложност

- $\text{maxSqArea}(A[1 .. m][1 .. n]) = \theta(mn)$

Зад. 5

Дадена е матрица $A[1 .. m][1 .. n] \in (\{0, 1\}^m)^n$. Да се намери най-голям правоъгълник от единици и да се изведе от колко единици е съставен.

Пример:

$$\begin{cases} \text{Instance : } A[1 ..5][1 ..6] = [[0,1,0,1,1,0],[1,1,1,1,1,1],[1,0,0,1,1,1],[1,1,0,1,1,1],[1,1,1,1,1,1]] \\ \text{Solution : } 12 \end{cases}$$

Обяснение на пример:

С удебелен шрифт е обозначен примерен максимален правоъгълник от единици. Този път е единствен, но не е задължително винаги да има точно един макимален правоъгълник.

$$\begin{pmatrix} 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & \mathbf{1} & \mathbf{1} & \mathbf{1} \\ 1 & 0 & 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} \\ 1 & 1 & 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} \\ 1 & 1 & 1 & \mathbf{1} & \mathbf{1} & \mathbf{1} \end{pmatrix}$$

Идея:

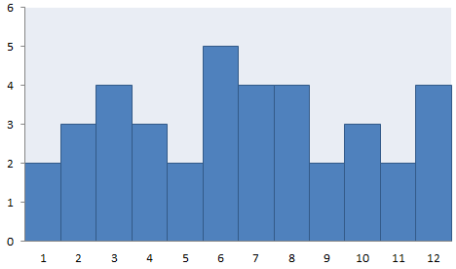
За разлика от предходната задача, тази е доста по-сложна. Ще я решим чрез редукция до по-лесната задача:

- Намиране на максимално лице на правоъгълник от хистограма: $\begin{cases} \text{Вход : } A[1 .. n] \in (\mathbb{N}_0)^n \\ \text{Изход : } \max_{1 \le i \le j \le n}[(j - i + 1)\min_{i \le k \le j}(A[k])] \end{cases}$.

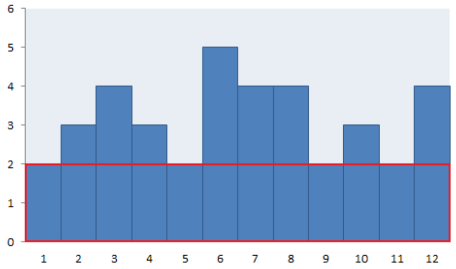
Пример:

$$\begin{cases} \text{Instance : } A[1 ..12] = [2, 3, 4, 3, 2, 5, 4, 4, 2, 3, 2, 4] \\ \text{Solution : } 24 \end{cases}$$

Обяснение на пример:



Питаме се колко е максималното лице на правоъгълник от тази хистограма (при условие, че всеки стълб е с широчина единица). В случая максималното лице е 12.2 = 24:



Решението на тази задача отново не е тривиално, но е по-лесно от това на оригиналната задача. За целта ще ни е необходим един стек и едно линейно минаване през масива $A[1 \,..\, n]$.

Идея (за изменената задача):

Започваме със стек s, който съдържа точно един елемент - нула. Стека ще съдържа индекси от масива, като нулата е фиктивна стойност, служеща за улеснение. Имаме също така и променлива maxArea, която е инициализирана с нула. Започваме да обхождаме от ляво надясно. Нека се намираме на индекс $i \in \{1, \ldots, n+1\}$, като си дефинираме $A[n+1] = 0$ отново за улеснение. Имаме 2 възможности:

$$\begin{cases} s.\text{push}(i) & , A[i] \geq A[s.\text{top}()] \\ \textit{докато } A[i] < A[s.\text{top}()] \textit{ прави } \text{temp} \leftarrow s.\text{pop}(); \text{maxArea} = \max\{\text{maxArea}, A[\text{temp}]*(i - \text{stack.top}() - 1)\} & , A[i] < A[s.\text{top}()] \end{cases}$$

След като завърши и последната итерация, то отговора ни се намира в maxArea. Тъй като е твърде абстрактно, нека да разгледаме стъпка по стъпка как работи алгоритъма:

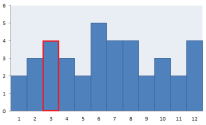
$$\begin{cases} i = 0 \\ s = \underline{\lfloor 0} \\ \text{maxArea} = 0 \end{cases}$$

$$\begin{cases} i = 1 \\ s = \underline{\lfloor 0\ 1} \\ \text{maxArea} = 0 \end{cases}$$

$$\begin{cases} i = 2 \\ s = \underline{\lfloor 0\ 1\ 2} \\ \text{maxArea} = 0 \end{cases}$$

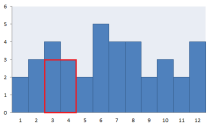
$$\begin{cases} i = 3 \\ s = \underline{\lfloor 0\ 1\ 2\ 3} \\ \text{maxArea} = 0 \end{cases}$$

$$\begin{cases} i = 4 \\ \text{temp} = 3 \text{ // } s.\text{pop}() \\ s = \underline{\lfloor 0\ 1\ 2} \\ \text{maxArea} = \max(0,\ 4.\ (4 - 2 - 1)) = 4 \end{cases}$$

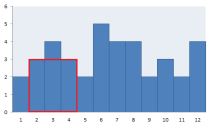
// 

$i = 4$
$s = \lfloor 0\ 1\ 2\ 4$
maxArea = 4

$i = 5$
temp = 4 // s.pop()
$s = \lfloor 0\ 1\ 2$
maxArea = max(4, 3. (5 − 2 − 1)) = 6

// 

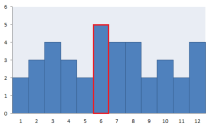
$i = 5$
temp = 2 // s.pop()
$s = \lfloor 0\ 1$
maxArea = max(6, 3. (5 − 1 − 1)) = 9

// 

$i = 5$
$s = \lfloor 0\ 1\ 5$
maxArea = 9

$i = 6$
$s = \lfloor 0\ 1\ 5\ 6$
maxArea = 9

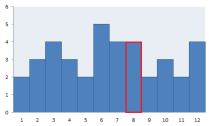
$i = 7$
temp = 6 // s.pop()
$s = \lfloor 0\ 1\ 5$
maxArea = max(9, 5. (7 − 5 − 1)) = 9

// 

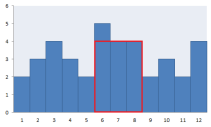
$i = 7$
$s = \lfloor 0\ 1\ 5\ 7$
maxArea = 9

$i = 8$
$s = \lfloor 0\ 1\ 5\ 7\ 8$
maxArea = 9

$i = 9$
temp = 8 // s.pop()
maxArea = max(9, 4. (9 − 7 − 1)) = 9

// 

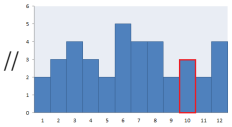
$i = 9$
temp = 7 // s.pop()
maxArea = max(9, 4. (9 − 5 − 1)) = 12

// 

$i = 9$
$s = \lfloor 0\ 1\ 5\ 9$
maxArea = 12

$i = 10$
$s = \lfloor 0\ 1\ 5\ 9\ 10$
maxArea = 12

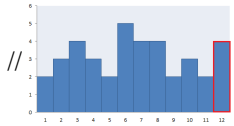
$i = 11$
temp = 10
$s = \lfloor 0\ 1\ 5\ 9$
maxArea = max(12, 3. (11 − 9 − 1)) = 12

// 

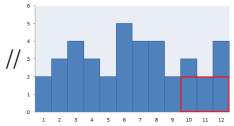
$i = 11$
$s = \lfloor 0\ 1\ 5\ 9\ 11$
maxArea = 12

$i = 12$
$s = \lfloor 0\ 1\ 5\ 9\ 11\ 12$
maxArea = 12

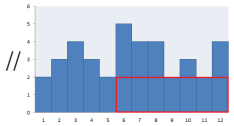
$i = 13$
temp = 12 // s.pop()
$s = \overline{|\,0\ 1\ 5\ 9\ 11}$
maxArea = max(12, 4. (13 − 11 − 1)) = 12



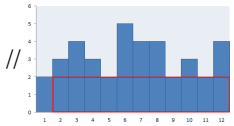
$i = 13$
temp = 11 // s.pop()
$s = \overline{|\,0\ 1\ 5\ 9}$
maxArea = max(12, 2. (13 − 9 − 1)) = 12



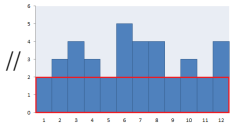
$i = 13$
temp = 9 // s.pop()
$s = \overline{|\,0\ 1\ 5}$
maxArea = max(12, 2. (13 − 5 − 1)) = 14



$i = 13$
temp = 5 // s.pop()
$s = \overline{|\,0\ 1}$
maxArea = max(14, 2. (13 − 1 − 1)) = 22



$i = 13$
temp = 1 // s.pop()
$s = \overline{|\,0}$
maxArea = max(22, 2. (13 − 1 − 1)) = 24



Когато се намираме на $i = n + 1$ и стека се състои от индекси само на нули, то приключваме и връщаме maxArea. Разбира се отново приемаме, че $A[0] = 0$. Проверете, че алгоритъма работи и за хистограми, съдържащи *дупки* (т.е. стълбове с височина 0).

Псевдокод:

```
1. maxRectAreaHist(A[1 .. n]) : // A ∈ (ℕ0)^n, n ∈ ℕ0
2.       maxArea ← 0
3.       s ← Stack.Init()
4.       s.push(0)
5.       extend A[1 .. n] to A[0 .. n + 1],  where A[0] = A[n + 1] = 0
5.       for i ← 1 to n + 1
6.             while A[i] < A[s.top()] do
7.                   temp ← s.pop()
8.                   area ← A[temp]*(i − s.top() − 1)
9.                   maxArea ← max(maxArea, area)
10.            s.push(i)
11.      return maxArea
```

Нека сега се върнем на оригиналната задача. Идеята е да имаме една хистограма, която да се актуализира за всеки ред i и такава, че за индекс j имаме броя последователни единици от долу нагоре за колоната j, започваща от ред i. Нека разгледаме с пример за най-лесно:

$$A = \begin{pmatrix} 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

- За ред 1: hist[1 .. n] = [0, 1, 0, 1, 1, 0]
- За ред 2: hist[1 .. n] = [1, 2, 1, 2, 2, 1]
- За ред 3: hist[1 .. n] = [2, 0, 0, 3, 3, 2]
- За ред 4: hist[1 .. n] = [3, 1, 0, 4, 4, 3]
- За ред 5: hist[1 .. n] = [4, 2, 1, 5, 5, 4]

За всяка такава хистограма пускаме помощния алгоритъм и взимаме максимума от всичките резултати. Убедете се, че е коректно.

```
1. maxRectArea(A[1 .. m][1 .. n]) : // A ∈ ∈ ({0, 1}^m)^n
2.     maxArea ← 0
3.     hist[1 .. n] ← [0, …, 0]
4.     for i ← 1 to m
5.         for j ← 1 to n
6.             if A[i][j] = 1 then
7.                 hist[j] ← hist[j] + 1
8.             else  // A[i][j] = 0
9.                 hist[j] ← 0
10.        maxArea ← max(maxArea, maxRectAreaHist(hist))
11.    return maxArea
```

Сложност

- maxAreaRectHist($A[1 .. n]$) = $\theta(n)$, тъй като всеки индекс $i \in \{0, \ldots, n+1\}$ бива добавян/премахван най − много един път в стека
- maxRectArea($A[1 .. m][1 .. n]$) = $\theta(mn)$

Зад. 6

Даден е низ $s[1 .. n] \in \{a, \ldots, z\}^n$. Да се намери дължината на най-дълъг палиндром на

***a*)** подмасив на $s[1 .. n]$

***b*)** подредица на $s[1 .. n]$

***a*)**

Пример:

$\begin{cases} \text{Instance : } s[1 ..7] = [a, a, a, b, c, b, a] \\ \text{Solution : } 5 \text{ // } abcba \end{cases}$

Идея:

Строим таблица $d[1 .. n][1 .. n]$, от която ще използваме главния диагонал и отгоре му със следната семантика: $d[i][j] = \begin{cases} \text{TRUE} & , s[i .. j] \text{ е палиндром} \\ \text{FALSE} & , s[i .. j] \text{ не е палиндром} \end{cases}$.

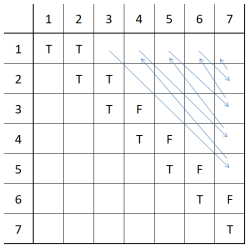
Започваме с главния диагонал на таблицата - по него всичко е истина, защото всеки подмасив с дължина единица е палиндром.

	1	2	3	4	5	6	7
1	T						
2		T					
3			T				
4				T			
5					T		
6						T	
7							T

След това проверяваме една клетка над главния диагонал - това е базов случай с който проверяваме за палиндроми с дължина 2.

	1	2	3	4	5	6	7
1	T	T					
2		T	T				
3			T	F			
4				T	F		
5					T	F	
6						T	F
7							T

Продължаваме да попълваме табличката по описания долу начин:



Ще използваме следната схема за изчисление: $d[i][j] = \begin{cases} \text{TRUE} & , s[i] = s[j] \;\&\; d[i+1][j-1] = \text{TRUE} \\ \text{FALSE} & , \text{else} \end{cases}$.

Семантиката е, че ако първата буква на подмасива $s[i \,..\, j]$ съвпада с последната буква и от предходни изчисления знаем, че $s[i+1 \,..\, j-1]$ е палиндром, то тогава и $s[i \,..\, j]$ е палиндром. В крайна сметка получаваме следната таблица:

	1	2	3	4	5	6	7
1	T	T	T	F	F	F	F
2		T	T	F	F	F	F
3			T	F	F	F	T
4				T	F	T	F
5					T	F	F
6						T	F
7							T

Дължината на най-дългия палиндром, който е подмасив на $s[1 \,..\, n]$ е (column − row + 1) при най-последното срещане на TRUE при попълването. В случая е TRUE-то намиращо се в клетка $\langle 3, 7 \rangle$. Ясно е, че $s[3 \,..7] = [a, b, c, b, a]$ е палиндром и при това най-дългия измежду подмасивите на $s[1 \,..\, n]$. Неговата дължина е $7 - 3 + 1 = 5$. По време на попълването на таблицата можем да следим за последната истина и да актуализираме отговора и след като попълним цялата таблица да го върнем директно.

Псевдокод:

```
1. PalindromeSubArr(s[1 .. n]) : // s ∈ {a, …, z}^n, n ∈ ℕ+
2.     d[1 .. n][1 .. n] ← [[FALSE, …, FALSE], …, [FALSE, …, FALSE]]
3.     maxLen ← 0
4.     for i ← 1 to n
5.         d[i][i] ← TRUE
6.         maxLen ← 1
7.     for i ← 1 to n − 1
8.         d[i][i + 1] ← (s[i] = s[i + 1])
9.         maxLen ← 2
10.    for k ← 2 to n − 1
11.        for i ← 1 to n − k
12.            d[i][i + k] ← (s[i] = s[i + k] and d[i + 1][i + k − 1])
13.            if d[i][i + k] then
14.                maxLen ← k + 1
15.    return maxLen
```

Сложност

· PalindromeSubArr($s[1 .. n]$) = $\theta(n^2)$

***b*)**

Пример:

$\begin{cases} \text{Instance : } s[1 ..6] = [a, b, a, c, a, a] \\ \text{Solution : } 4 \text{ // } aaaa \end{cases}$

Идея:

Абсолютно аналогично на *a*), само че в клетките $\langle i, j \rangle$ на таблицата не записваме TRUE/FALSE, ами директно най-дългия палиндром от подредица на $s[i .. j]$. Тогава отговора ще ни се намира в клетка $\langle 1, n \rangle$ (за разлика от *a*)). Отново започваме с попълване на главния диагонал и една клетка над него:

	1	2	3	4	5	6
1	1	1				
2		1	1			
3			1	1		
4				1	1	
5					1	2
6						1

Сега попълваме остатъка от таблицата аналогично на *a*), използвайки следната схема: $d[i][j] = \begin{cases} 2 + d[i+1][j-1] & , s[i] = s[j] \\ \max(d[i+1][j], d[i][j-1]) & , s[i] \neq s[j] \end{cases}$.

	1	2	3	4	5	6
1	1	1	3	3	3	**5**
2		1	1	1	3	3
3			1	1	3	3
4				1	1	2
5					1	2
6						1

Псевдокод:

```
1. PalindromeSubSeq(s[1 .. n]) : // s ∈ {a, …, z}^n, n ∈ ℕ+
2.     d[1 .. n][1 .. n] ← [[1, …, 1], …, [1, …, 1]] // инициализираме с 1 за да не инициализираме експлиситно главния диагонал
3.     for i ← 1 to n − 1
4.         if s[i] = s[i + 1] then
5.             d[i][i + 1] ← 2
6.     for k ← 2 to n − 1
7.         for i ← 1 to n − k
8.             if s[i] = s[i + k] then
9.                 d[i][i + k] ← 2 + d[i + 1][i + k − 1]
10.            else
11.                d[i][i + k] ← max(s[i + 1][i + k], s[i][i + k − 1])
12.    return d[1][n]
```

Зад. 7

Даден е масив $A[1\,..\,n] \in (\mathbb{N}_0)^n$. Двама играчи играят последователни ходове, като на всеки ход взимат или най-левия или най-десния елемент на масвиа, като елемента се премахва от масива и хода им приключва. Играта приключва, когато няма повече елементи в масива. Печели този играч, чиято сума от взети елементи (числа) е най-голяма. Да се определи максималната сума за двамата играча, ако и двамата играят оптимално.

Пример:

$\begin{cases} \text{Instance} : A[1\,..4] = [3,\ 9,\ 1,\ 2] \\ \text{Solution} : \langle 11,\ 4\rangle \end{cases}$

Обяснение на пример:

На ход 1 започва първия играч и взима най-десния елемент - 2. В масива остават елементите [3, 9, 1]. На ход 2 втроя играч взима най-левия елемент - 3. В масива остават елементите [9, 1]. На ход 3 първия играч взима най-левия елемент - 9. На ход 4 втория играч взима най-левия елемент (той е само един) - 1. Няма повече елементи ⇒ играта приключва. Първия играч е събрал 2 + 9 = 11. Втория играч е събрал 3 + 1 = 4. Алгоритъма връща наредена двойка от сумата на елементите при оптимална игра на двата играча - $\langle 11,\ 4\rangle$.

Забележка Да обърнем внимание, че алчната стратегия тук <u>няма</u> да сработи.

Идея:

Аналогично на предходната задача (първата ѝ подточка) ще разгледаме наредени двойки от оптималните ходове за подмасиви на $A[1\,..\,n]$. Ясно е, че ако имаме масив с големина единица и е на ход играч едно, то той взима този елемент и за втория играч не остава нищо. Тоест инициализираме таблицата и по-конкретно главния ѝ диагонал по следния начин:

	1	2	3	4
1	(3,0)			
2		(9,0)		
3			(1,0)	
4				(2,0)

След това започваме да попълваме талицата аналогично на тази от предходната задача, използвайки следната схема:
$d[i][j] = \begin{cases} \langle A[i] + d[i+1][j].\text{second},\ d[i+1][j].\text{first}\rangle & , A[i] + d[i+1][j].\text{second} \geq A[j] + d[i][j-1].\text{second} \\ \langle A[j] + d[i][j-1].\text{second},\ d[i][j-1].\text{first}\rangle & , A[i] + d[i+1][j].\text{second} < A[j] + d[i][j-1].\text{second} \end{cases}$.

Семантиката е, че след като вземем или $A[i]$ или $A[j]$ от $A[i\,..\,j]$, то тогава ще остане подмасива съответно или $A[i+1\,..\,j]$ или $A[i\,..\,j-1]$, като на тях противника вече е на ход, т.е трябва да вземем оптималното за втория играч оттам. След като попълним, таблицата изглежда по следния начин:

	1	2	3	4
1	(3,0) ←	(9,3) ←	(4,9) ←	**(11,4)**
2		(9,0)	(9,1) ↓	(10,3) ↓
3			(1,0) ←	(2,1)
4				(2,0)

Отговора се намира в клетката с индекс $\langle 1,\ n\rangle$.

Псевдокод:

```
1. task7(A[1 .. n]) : // A ∈ (ℕ0)^n
2.     d[1 .. n][1 .. n] ← [[0, …, 0], …, [0, …, 0]]
3.     for i ← 1 to n
4.         d[i][i] ← ⟨A[i], 0⟩
5.     for k ← 1 to n − 1
6.         for i ← 1 to n − k
7.             if A[i] + d[i + 1][i + k].second ≥ A[i + k] + d[i][i + k − 1] then
8.                 d[i][i + k] ← ⟨A[i] + d[i + 1][i + k].second, d[i + 1][i + k].first⟩
9.             else
10.                d[i][i + k] ← ⟨A[i + k] + d[i][i + k − 1].second, d[i][i + k − 1].first⟩
11.    return d[1][n]
```

[1] Ако имаме вход $A[1..5] = [1, 13, 12\,345, 1\,048\,576, 4\,294\,967\,295]$, то тогава големината на входа е де факто $\log_{10}(1) + \log_{10}(13) + \log_{10}(12\,345) + \log_{10}(1\,048\,576) + \log_{10}(4\,294\,967\,295)$. Практически обаче ни интересува само размера на масива, тъй като числата с които работим (т.е с които процесора може да изпълнява операциите в константно време) са ограничени отгоре - 2^{32} или 2^{64} или други в зависимост от процесора. Тоест всички те са с размер най-много $\log_{10}\left(2^{32}\right) = \text{const}$. Разбира се процесора работи с битове, затова константата е по-скоро $\log_2\left(2^{32}\right) = 32$. Тоест имаме следното неравенство: $\log_{10}(1) + \log_{10}(13) + \log_{10}(12\,345) + \log_{10}(1\,048\,576) + \log_{10}(4\,294\,967\,295) \leq 32 + 32 + 32 + 32 + 32 = 5 * 32$. Тоест големината на входа е $O(32 * \textit{големината на масива})$. Разбира се, ако работим с така наречените “големи числа”, то трябва да работим по-прецизно.