Динамично програмиране

- Когато имаме общи подзадачи (общи сметки)
- оптимално решение чрез оптимални подрешения
- при комбинаторни задачи

- Memoization → top-down

  DP → bottom-up

Пример за ДП - общи сметки:

Числа на Фибоначи:

Рекурсивно:

```
fib(n)
1  if n=0
2    return 0
3  if n=1
4    return 1
5  return fib(n-1) + fib(n-2)
```

Итеративно - чрез DP:

```
fib(n):
1   res[1..n] ← празен масив
2   res[1] ← 1
3   res[2] ← 1
4   for i ← 3 to n
5     res[i] ← res[i-1] + res[i-2]
6   return res[n]
```

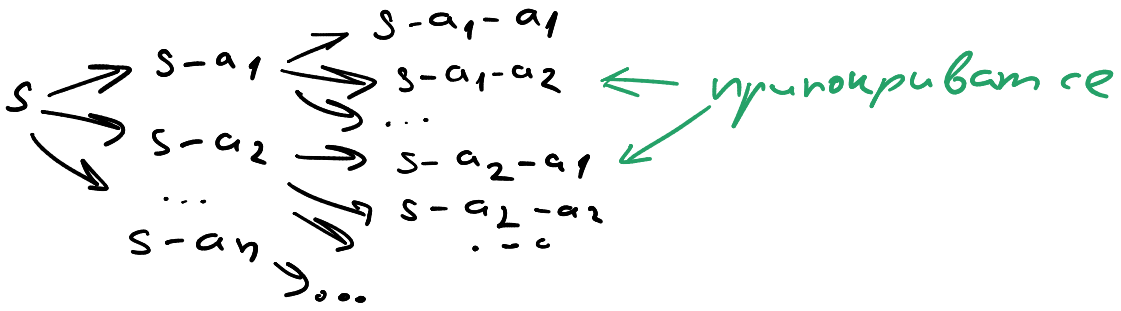
Зад Разполагаме с неограничен брой монети с номинали съответно записани в $A[1..n]$. Да се намери с колко най-малко монети можем да съберем сума S.

Пр $S=20 \quad A=[16, 1, 10]$

алчен подход $\Rightarrow$ $16+1+1+1+1$

оптимално $\Rightarrow$ $10+10$

Решение:



Може направо да пишем код, но рекурсивната декомпозиция е важна за доказване на коректността!

$$f(x) = \min \begin{cases} 1+f(x-A[1]) \\ 1+f(x-A[2]) \\ \dots \\ 1+f(x-A[n]) \end{cases} = 1+\min_{i=1..n} f(x-A[i])$$

$f(x) = +\infty$ за $x<0$

1. $M[0...S]$ // всяка получена стойност на индекс x съдържа $f(x)$, т.е. min брой монети за сума x
2. $M[0] \leftarrow 0$
3. for $x \leftarrow 1$ to S
4. $\quad M[x] \leftarrow +\infty$
5. $\quad$ for $i \leftarrow 1$ to n
6. $\quad\quad$ if $A[i] \le x$
7. $\quad\quad\quad M[x] \leftarrow \min(M[x], M[x-A[i]])$
8. return $M[S]$

$T(n,S) = \Theta(n.S)$

$\rightarrow$ псевдополиномиален

Зад Дадени са два символни низа $S_1[1..n]$ и $S_2[1..m]$. Да се намери дължината на най-дългата обща подредица на S_1 и S_2.

// Тази задача е пример за припокриващи се подструктури

Решение: Нека $f(i,j)$ е дължината на на най-дългата търсена подредица за $S_1[1...i]$ и $S_2[1..j]$

$$f(i,j) = \begin{cases} 0 & \text{, ако } i=j=0 \\ f(i-1,j-1)+1 & \text{, ако } S_1[i]=S_2[j] \text{ и } i,j>0 \\ \max(f(i,j-1), f(i-1,j)) & \text{, ако } i,j>0 \text{ и } S_1[i] \neq S_2[j] \end{cases}$$

Рекурсивно пресмятане → експоненциална сложност

Итеративно чрез ДП:

```
Alg LCS(S1[1..n], S2[1..m])
1    dp[0..n][0...m] ← празна матрица
2    for i←0 to n
3        dp[i][0] = 0
4    for j←0 to m
5        dp[0][j] = 0
6    for i←1 to n
7        for j←1 to m
8            if S1[i] = S2[j]
9                dp[i][j] = dp[i-1][j-1] + 1
10           else
11               dp[i][j] = max(dp[i-1][j], dp[i][j-1])
12   return dp[n][m]
```

// Ако има интересна наредба: низове, масиви от числа, коренови дървета и пр. → ДП

- Още един алгоритъм за най-къс път в граф:

Ако търсим най-къс път от u до v за $\forall u \forall v \in V$, то може да използваме алг. на

Floyd - Warshall с $T(n) = \Theta(n^3)$

// вместо $\Theta(n^4)$ за n-кратно пускане на Bellman-Ford

Зад. Minimum Coin Count Limited

Дадено е мултимножество от монети $\{c_1, c_2 \dots c_n\}_M$, с които разполагаме. Искаме да отдерем сума S с възможно най-малко монети.

Решение:

Ще разглеждаме подзадачите за всяка сума $x \leq S$ за „всяко" подмножество на $\{c_1 \dots c_n\}$.

// Ако разглеждаме наистина всички подмножества, то ще имаме таблица с 2^n реда (или колони)

За намиране на подмножествата на $\{a_1 \dots a_m\}$ имаме 2 възможности:
- a_1 + подмножества $\{a_2 \dots a_m\}$ // взимаме a_1 или не
- подмножества $\{a_2 \dots a_m\}$

Нека $f(x, i)$ е минималният брой монети за сума x използвайки някои от първите i на брой монети.

$$\begin{cases} f(0, i) = 0 & \text{за } i = 1 \dots n \\ f(x, 0) = +\infty & \text{за } x \in [0; S] \\ f(0, 0) = 0 \end{cases}$$

$$f(x,i) = \min \begin{cases} 1 + f(x - c_i, i-1), & \text{ако } x \geq c_i \\ f(x, i-1) \end{cases}$$

// взимаме i-тата монета

// не я взимаме

⇒ продължаваме за същата сума с по-малкото монети

Зад По колко начина може да покрием стена с размер $2 \times N$ с правоъгълни плочки 1×2?

Решение:

$f(x)$ - броят начини за покриване на дъска с дължина x

$$\begin{cases} f(0) = 0 \\ f(1) = 1 \\ f(x) = f(x-1) + f(x-2) \end{cases}$$

Зад Предложете ефикасен алгоритъм за намиране на броя на n-цифрените числа (в десетична бройна сис.), чиято сума на цифрите им е S.

Решение: n цифри, сума S →
- $n-1$ ц., сума $S-1$
- $n-1$ ц., сума $S-2$
- ...
- $n-1$, $S-8$
- $n-1$, S

За по-лесно да мислим гледаме n-цифрени стрингове.

$f(i, x)$ - брой i-цифрени стрингове със сума на цифрите x

$f(i, 0) = 1$ за $i = 1..n$

$$f(1, k) = \begin{cases} 1, & \text{за } k = 0..9 \\ 0, & \text{за } k > 9 \end{cases}$$

$$f(i, x) = \sum_{k=0}^{\min(9;x)} f(i-1, x-k)$$

Псевдокод:

```
1  M[1..n][0..S] // (съдържа f(i,x), ако е попълнена
2. M[1][0..9] ← 1; M[1][10..S] ← 0
3. for i ← 2 to n
4.       M[i][0] ← 1
5.       for x ← 1 to S
6.           M[i][x] ← 0
7.           for k ← 0 to min(9;x)
8.               M[i][x] += M[i-1, x-k]
9. return M[n][s] - M[n-1][s]
```

Отговорът на задачата не е просто $M[n][S]$, защото не всички стрингове са валидни числа: Тези, които започват с 0 са $M[n-1][S]$ на брой.

$T(n, S) = S(n, S) = \Theta(n.S)$

// Може да направим $S(n,S) = \Theta(S)$ ако пазим само 2 реда - предишен и текущ, но губим възможност за възстановяване на решението.

Или $\Theta(n)$ с 10 стека - аналогично

Зад. Optimal Single Robot Path - В дадена матрица n x m е поставен робот в най-горния ляв ъгъл. Роботът се движи само надолу и надясно като целта е да стигне до долния десен ъгъл.

Предложете алгоритъм, който намира максималната сума на числата по пътя на робота.

Пр

1 → 7 12
3 11 5
12 6 → 10 ⇒ 37

Ид. f(i,j) - максимална печалба, започвайки от клетка (i,j)

$$f(i,j) = A[i][j] + \max\{f(i+1,j); f(i,j+1)\}$$

$$f(i,m) = A[i][m] + f(i+1,m) \quad // \text{Последният ред}$$

$$f(n,j) = A[n][j] + f(n,j+1) \quad // \text{Последен стълб}$$

Ид. $f(i,j) = A[i][j] + \max\{f(i-1,j); f(i,j-1)\}$
максимална печалба по път до (i,j)

Зад. Optimal Two Robot Path - Искаме максимална сума от двата робота заедно, която числото от дадена клетка може да се вземе само веднъж.

Пр

1 → 7 → 12
3 11 5
12 → 0 → 10

→

0 → 0 → 12
3 0 5
12 0 0

I робот - 37 II робот - 17 37+17 = 54

Алчното решение да пуснем единия робот след другия не е оптимално.

Може да съберем $58 = 1+3+12+8+10+7+12+5 = 58 > 54$

Нека (i_1, j_1) е позицията на I робот.
(i_2, j_2) – II робот.

$$f(i_1, j_1, i_2, j_2) = \begin{cases} A[i_1][j_1] + A[i_2][j_2] + \max\{*\}, & \text{ако } i_1 \neq i_2 \vee j_1 \neq j_2 \\ A[i_1][j_1] + \max\{*\}, & \text{иначе} \end{cases}$$

$\max\{*\}$ е най-добрата възможност за ход на роботите

$$\max\{*\} = \max \begin{cases} f(i_1+1, j_1, i_2+1, j_2) & // \text{Down и Down} \\ f(i_1+1, j_1, i_2, j_2+1) & // \text{Down и Right} \\ f(i_1, j_1+1, i_2+1, j_2) & // \text{RD} \\ f(i_1, j_1+1, i_2, j_2+1) & // \text{RR} \end{cases}$$

$T(n,m) = \Theta(n^2 \cdot m^2)$ – квадратично по входа

Можем да спестим един параметър: $i_1 + j_1 = i_2 + j_2$ – номерът на диагонала е винаги един и същ за роботите.

$$\Rightarrow j_2 = i_1 + j_1 - i_2 \quad \text{(например)}$$

$$\Rightarrow \Theta(n^2 m) \text{ или } \Theta(nm^2)$$

Зад Minimum Deletions to Palindrome или
Longest Palindrome Sequence // ще използваме тази

Дадена е дума. Да се намери колко най-малко символа трябва да се премахнат, за да стане палиндром.

Пр a b x c b y z a w
— — — — — палиндроми

Решение: Нека f(i,j) е най-голямата палиндромична поредица в w[i...j]

Търсим f(1,n) =?

$$f(i,j) = \max \begin{cases} f(i+1,j-1)+2, & \text{ако } w[i]=w[j] \\ \text{// Взимаме крайните и гледаме в по-малката дума} \\ f(i+1,j) & \text{// Премахваме първата буква} \\ f(i,j-1) & \text{// Игнорираме последната буква} \\ f(i+1,j+1) & \text{// Игнорираме и двете} \end{cases}$$

$$f(i,i) = 1 \quad \text{// Имаме 2 дена по диагонала}$$

$$f(i,i+1) = \begin{cases} 2, & \text{ако } w[i]=w[i+1] \\ 1, & \text{иначе} \end{cases}$$

Пр abba

i \ j	1	2	3	4
1	1	1		
2		1	2	
3			1	1
4				1

Псевдокод:

```
...
for d ← 3 to n
    for i ← 1 to n-d+1
        for j ← 1 to d-i+1
            dp[i][j] ← ...
return dp[1][n] // или n-dp[1][n] за брой изтривания
```