UNIVERSIDADE FEDERAL DE SERGIPE

DEPARTAMENTO DE COMPUTAÇÃO

Programação dinâmica

Projeto e Análise de Algoritmos

Bruno Prado

Departamento de Computação / UFS

Introdução

- O que é programação dinâmica?
  - É uma estratégia de resolução de problemas similar a divisão e conquista, onde a solução é obtida através da resolução e armazenamento das soluções parciais

Introdução

- O que é programação dinâmica?
  - É uma estratégia de resolução de problemas similar a divisão e conquista, onde a solução é obtida através da resolução e armazenamento das soluções parciais
  - Esta estratégia é geralmente aplicada em problemas de otimização que possuem várias soluções possíveis

Introdução

- O que é programação dinâmica?
  - É uma estratégia de resolução de problemas similar a divisão e conquista, onde a solução é obtida através da resolução e armazenamento das soluções parciais
  - Esta estratégia é geralmente aplicada em problemas de otimização que possuem várias soluções possíveis

$$\begin{aligned} \textit{Programação} &\longrightarrow \textit{Organização} \\ \textit{Dinâmica} &\longrightarrow \textit{Tempo de execução} \end{aligned}$$

Introdução

- Quando utilizar programação dinâmica?
  - Sobreposição de problemas

Introdução

- Quando utilizar programação dinâmica?
  - Sobreposição de problemas
    - Os subproblemas são repetidamente resolvidos

Introdução

- Quando utilizar programação dinâmica?
  - Sobreposição de problemas
    - Os subproblemas são repetidamente resolvidos
    - É comum em implementações recursivas

Introdução

- Quando utilizar programação dinâmica?
  - Sobreposição de problemas
    - Os subproblemas são repetidamente resolvidos
    - É comum em implementações recursivas
  - Subestrutura ótima

Introdução

- Quando utilizar programação dinâmica?
  - Sobreposição de problemas
    - Os subproblemas são repetidamente resolvidos
    - É comum em implementações recursivas
  - Subestrutura ótima
    - A partir da combinação das soluções parciais ótimas é obtida a solução completa ótima do problema

Programação dinâmica

- Implementação recursiva do algoritmo de Fibonacci
  - Caso base ($n \leq 1$)

```c
// Padrão de tipos por tamanho
#include <stdint.h>
// Fibonacci recursivo
uint64_t fibonacci(uint32_t n) {
    if(n <= 1)
        return n;
    else
        return fibonacci(n - 1) + fibonacci(n - 2);
}
```

$$\updownarrow$$

$$F(n) = \begin{cases} 0 & n = 0 \\ 1 & n = 1 \\ F(n-1) + F(n-2) & n > 1 \end{cases}$$

Programação dinâmica

- Implementação recursiva do algoritmo de Fibonacci
  - Recorrência ($n > 1$)

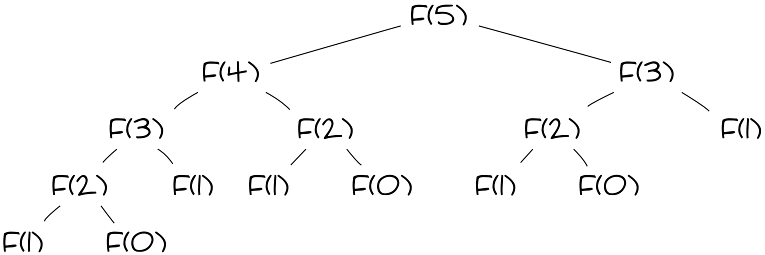
```c
// Padrão de tipos por tamanho
#include <stdint.h>
// Fibonacci recursivo
uint64_t fibonacci(uint32_t n) {
    if(n <= 1)
        return n;
    else
        return fibonacci(n - 1) + fibonacci(n - 2);
}
```

$$\updownarrow$$

$$F(n) = \begin{cases} 0 & n = 0 \\ 1 & n = 1 \\ F(n-1) + F(n-2) & n > 1 \end{cases}$$

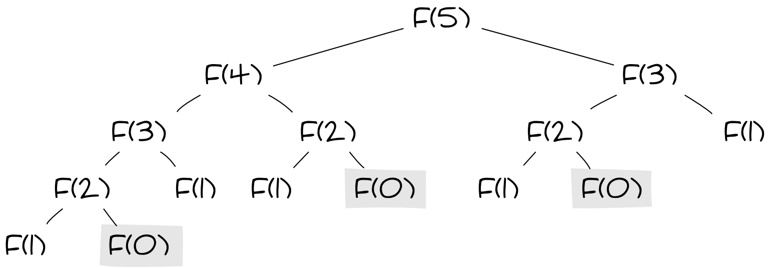
Programação dinâmica

- Árvore de execução do algoritmo de Fibonacci



Programação dinâmica

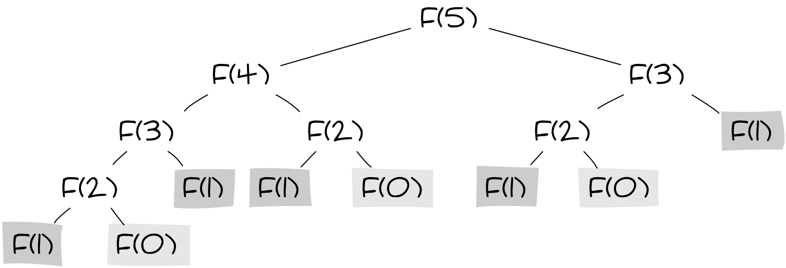
- Árvore de execução do algoritmo de Fibonacci



Espaço e tempo exponencial $O(2^n)$
com sobreposição de problemas

Programação dinâmica

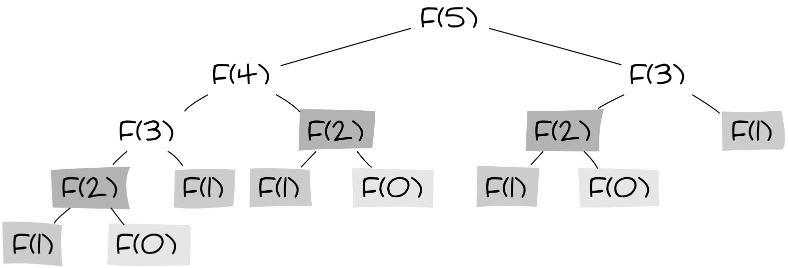
- Árvore de execução do algoritmo de Fibonacci



Espaço e tempo exponencial $O(2^n)$
com sobreposição de problemas

Programação dinâmica

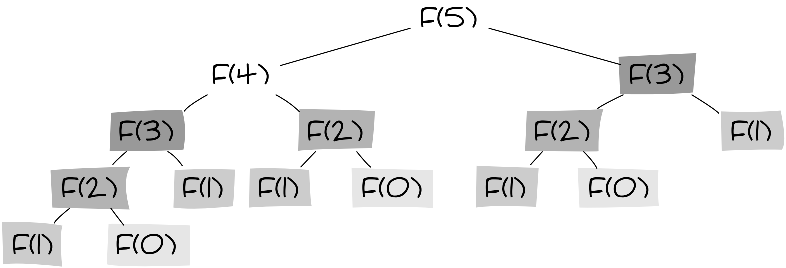
- Árvore de execução do algoritmo de Fibonacci



Espaço e tempo exponencial $O(2^n)$
com sobreposição de problemas

Programação dinâmica

- Árvore de execução do algoritmo de Fibonacci



Espaço e tempo exponencial $O(2^n)$
com sobreposição de problemas

Programação dinâmica

- Implementação iterativa do algoritmo de Fibonacci

```c
// Padrão de tipos por tamanho
#include <stdint.h>
// Fibonacci com programação dinâmica
uint64_t fibonacci_pd(uint32_t n) {
    // Alocação estática de vetor de respostas
    static uint64_t* V = NULL;
    // Checagem de índice para cálculo de valores
    for(uint32_t i = indice(V, n); i <= n; i++) {
        // F(n) = F(n - 1) + F(n - 2)
        V[i] = V[i - 1] + V[i - 2];
    }
    // Retorno do resultado armazenado no vetor
    return V[n];
}
```

Armazenamento dos resultados já calculados

Programação dinâmica

- Implementação iterativa do algoritmo de Fibonacci

```c
// Padrão de tipos por tamanho
#include <stdint.h>
// Fibonacci com programação dinâmica
uint64_t fibonacci_pd(uint32_t n) {
    // Alocação estática de vetor de respostas
    static uint64_t* V = NULL;
    // Checagem de índice para cálculo de valores
    for(uint32_t i = indice(V, n); i <= n; i++) {
        // F(n) = F(n - 1) + F(n - 2)
        V[i] = V[i - 1] + V[i - 2];
    }
    // Retorno do resultado armazenado no vetor
    return V[n];
}
```

Checagem de novos subproblemas não resolvidos

Programação dinâmica

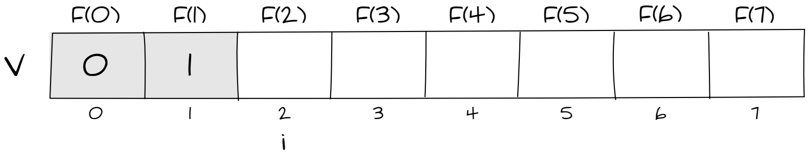
- Implementação iterativa do algoritmo de Fibonacci

```c
// Padrão de tipos por tamanho
#include <stdint.h>
// Fibonacci com programação dinâmica
uint64_t fibonacci_pd(uint32_t n) {
    // Alocação estática de vetor de respostas
    static uint64_t* V = NULL;
    // Checagem de índice para cálculo de valores
    for(uint32_t i = indice(V, n); i <= n; i++) {
        // F(n) = F(n - 1) + F(n - 2)
        V[i] = V[i - 1] + V[i - 2];
    }
    // Retorno do resultado armazenado no vetor
    return V[n];
}
```

Execução com espaço $\Theta(n)$ e tempo entre $\Omega(1)$ e $O(n)$

Programação dinâmica

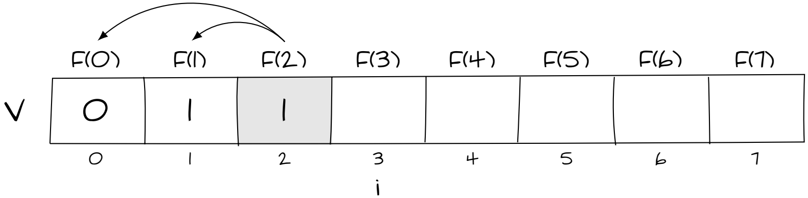
- Implementação iterativa do algoritmo de Fibonacci
  - Execução de *fibonacci_pd*(7)



Inicialização do vetor com 8 posições,
valores casos base e índice *i* na posição 2

Programação dinâmica

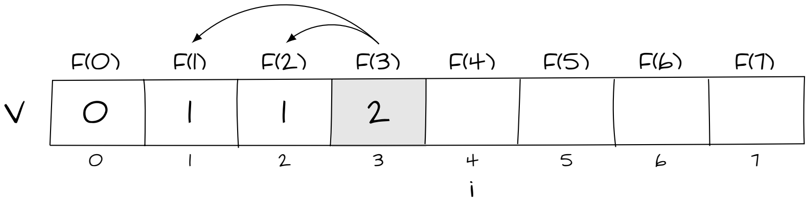
- Implementação iterativa do algoritmo de Fibonacci
  - Execução de *fibonacci_pd*(7)



Cálculo iterativo da sequência de Fibonacci

Programação dinâmica

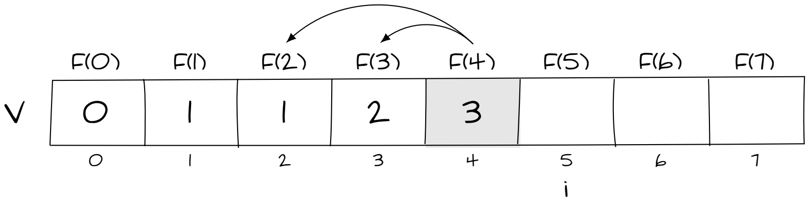
- Implementação iterativa do algoritmo de Fibonacci
  - Execução de *fibonacci_pd*(7)



Cálculo iterativo da sequência de Fibonacci

Programação dinâmica

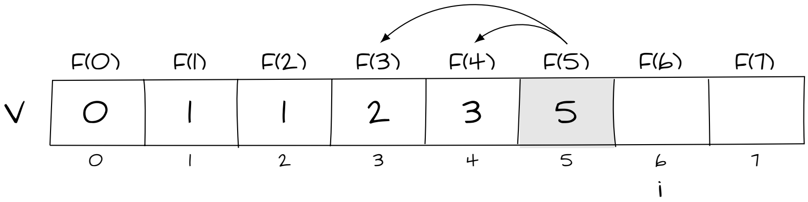
- Implementação iterativa do algoritmo de Fibonacci
  - Execução de *fibonacci_pd*(7)



Cálculo iterativo da sequência de Fibonacci

Programação dinâmica

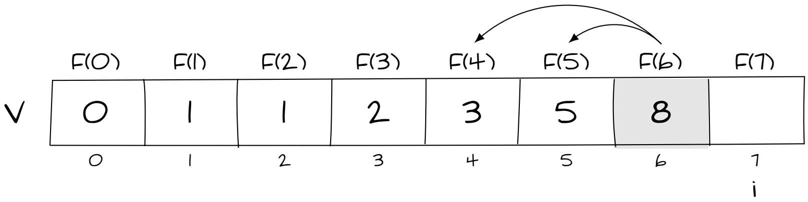
- Implementação iterativa do algoritmo de Fibonacci
  - Execução de *fibonacci_pd*(7)



Cálculo iterativo da sequência de Fibonacci

Programação dinâmica

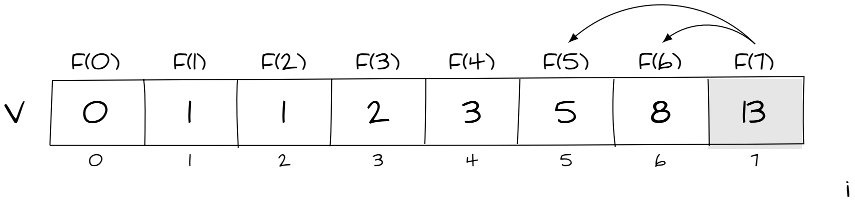
- Implementação iterativa do algoritmo de Fibonacci
  - Execução de *fibonacci_pd*(7)



Cálculo iterativo da sequência de Fibonacci

Programação dinâmica

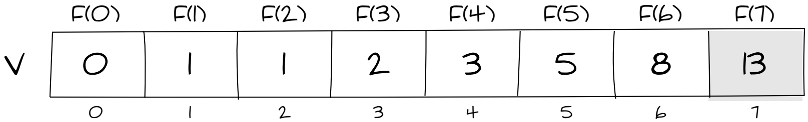
- Implementação iterativa do algoritmo de Fibonacci
  - Execução de *fibonacci_pd*(7)



Cálculo iterativo da sequência de Fibonacci

Programação dinâmica

- Implementação iterativa do algoritmo de Fibonacci
  - Execução de *fibonacci_pd*(7)



$V[7]$ é o resultado de *fibonacci_pd*(7)

Programação dinâmica

- Algoritmo de Fibonacci com programação dinâmica
  - ✓ Eficiência de tempo entre $\Omega(1)$ e $O(n)$

Programação dinâmica

- Algoritmo de Fibonacci com programação dinâmica
  - ✓ Eficiência de tempo entre $\Omega(1)$ e $O(n)$
  - ✓ Não calcula repetidamente valores já processados

Programação dinâmica

- Algoritmo de Fibonacci com programação dinâmica
  - ✓ Eficiência de tempo entre $\Omega(1)$ e $O(n)$
  - ✓ Não calcula repetidamente valores já processados
  - ✗ Maior complexidade de implementação

Programação dinâmica

- Algoritmo de Fibonacci com programação dinâmica
  - ✓ Eficiência de tempo entre $\Omega(1)$ e $O(n)$
  - ✓ Não calcula repetidamente valores já processados
  - ✗ Maior complexidade de implementação
  - ✗ Espaço linear $\Theta(n)$ para armazenamento dos resultados

Programação dinâmica

- Algoritmo de Fibonacci com programação dinâmica
  - ✓ Eficiência de tempo entre $\Omega(1)$ e $O(n)$
  - ✓ Não calcula repetidamente valores já processados
  - ✗ Maior complexidade de implementação
  - ✗ Espaço linear $\Theta(n)$ para armazenamento dos resultados

E a subestrutura ótima?

Programação dinâmica

- Subestrutura ótima do algoritmo de Fibonacci
  - Para obter o valor da n-ésima posição da sequência, todos os $n - 1$ valores precisam ser calculados

Programação dinâmica

- Subestrutura ótima do algoritmo de Fibonacci
  - Para obter o valor da n-ésima posição da sequência, todos os $n - 1$ valores precisam ser calculados
  - Por sua vez, cada um destes $n - 1$ valores representa um subproblema com sua respectiva solução parcial

Programação dinâmica

- Subestrutura ótima do algoritmo de Fibonacci
  - Para obter o valor da n-ésima posição da sequência, todos os $n-1$ valores precisam ser calculados
  - Por sua vez, cada um destes $n-1$ valores representa um subproblema com sua respectiva solução parcial
  - A composição das $n-1$ soluções parciais calculadas contribui para obtenção da solução completa

Programação dinâmica

- Problema da Mochila
  - Existem n itens distintos e únicos com valores v_i e pesos não negativos w_i para serem guardados em uma mochila com capacidade máxima de peso W, onde $1 \leq i \leq n$ e $1 \leq w \leq W$

Programação dinâmica

- Problema da Mochila
  - Existem n itens distintos e únicos com valores v_i e pesos não negativos w_i para serem guardados em uma mochila com capacidade máxima de peso W, onde $1 \leq i \leq n$ e $1 \leq w \leq W$
  - Função objetivo: maximizar o valor dos itens na mochila, sem exceder a capacidade total W

$$max \left(\sum_{i=1}^{n} v_i \times x_i \right) \leftrightarrow \sum_{i=1}^{n} w_i \times x_i \leq W, \quad x_i \in \{0, 1\}$$

Programação dinâmica

- Problema da Mochila com programação dinâmica
  - Sobreposição de problemas
    - Uma mesma coleção de itens com o mesmo valor pode ser organizado de maneiras distintas na mochila

Programação dinâmica

- Problema da Mochila com programação dinâmica
  - Sobreposição de problemas
    - Uma mesma coleção de itens com o mesmo valor pode ser organizado de maneiras distintas na mochila
    - Estas soluções repetidas não contribuem para encontrar o maior valor dos itens na mochila

Programação dinâmica

- Problema da Mochila com programação dinâmica
  - Sobreposição de problemas
    - Uma mesma coleção de itens com o mesmo valor pode ser organizado de maneiras distintas na mochila
    - Estas soluções repetidas não contribuem para encontrar o maior valor dos itens na mochila
  - Subestrutura ótima
    - Aplicando a programação dinâmica, para cada iteração é armazenada a solução parcial ótima

Programação dinâmica

- Problema da Mochila com programação dinâmica
  - Sobreposição de problemas
    - Uma mesma coleção de itens com o mesmo valor pode ser organizado de maneiras distintas na mochila
    - Estas soluções repetidas não contribuem para encontrar o maior valor dos itens na mochila
  - Subestrutura ótima
    - Aplicando a programação dinâmica, para cada iteração é armazenada a solução parcial ótima
    - A combinação de soluções parciais ótimas dos itens permitem obter o valor total máximo na mochila

Programação dinâmica

- Problema da Mochila
  - Cenário com $n = 4$ itens, com valores v_i, pesos w_i e mochila capacidade máxima de peso $W = 5$

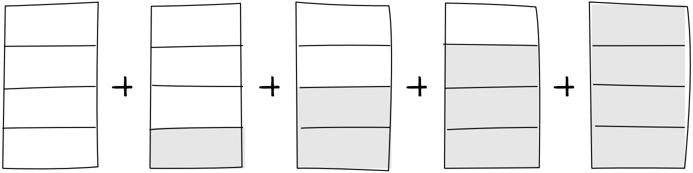
i	1	2	3	4
v_i	12	10	20	15
w_i	2	1	3	2

Programação dinâmica

- Problema da Mochila
  - Cenário com $n = 4$ itens, com valores v_i, pesos w_i e mochila capacidade máxima de peso $W = 5$

i	1	2	3	4
v_i	12	10	20	15
w_i	2	1	3	2

  - Busca exaustiva $\sum_{k=0}^{n}\binom{n}{k} = \binom{n}{0} + \cdots + \binom{n}{n} = \Theta(2^n)$

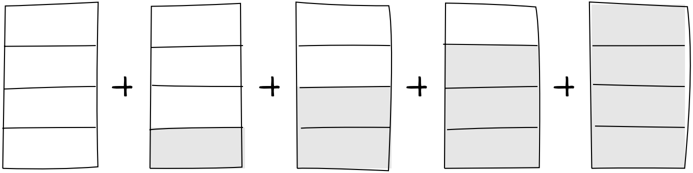


Programação dinâmica

- Problema da Mochila
  - Cenário com $n = 4$ itens, com valores v_i, pesos w_i e mochila capacidade máxima de peso $W = 5$

i	1	2	3	4
v_i	12	10	20	15
w_i	2	1	3	2

  - Busca exaustiva $\sum_{k=0}^{n} \binom{n}{k} = \binom{n}{0} + \cdots + \binom{n}{n} = \Theta(2^n)$



Evitar soluções repetidas e que excedem o peso máximo

Programação dinâmica

- Problema da Mochila
  - A função $V(i, w)$ é usada para calcular o valor dos itens contidos na mochila, com valor 0 para nenhum item $i = 0$ ou com peso nulo $w = 0$

V(i,w)	0	1	2	3	4	5
0						
1						
2						
3						
4						

Programação dinâmica

- Problema da Mochila
  - A função $V(i, w)$ é usada para calcular o valor dos itens contidos na mochila, com valor 0 para nenhum item $i = 0$ ou com peso nulo $w = 0$

V(i,w)	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0					
2	0					
3	0					
4	0					

Programação dinâmica

- Problema da Mochila
  - O item i não cabe na mochila
    - A capacidade disponível w é menor que o peso w_i adicionado pelo novo item

Programação dinâmica

- Problema da Mochila
  - O item i não cabe na mochila
    - A capacidade disponível w é menor que o peso w_i adicionado pelo novo item
    - O valor total atingido pelo último item é repetido

Programação dinâmica

- Problema da Mochila
  - O item i não cabe na mochila
    - A capacidade disponível w é menor que o peso w_i adicionado pelo novo item
    - O valor total atingido pelo último item é repetido
  - O item i cabe na mochila
    - A capacidade disponível w é maior ou igual ao peso w_i do novo item inserido na mochila

Programação dinâmica

- Problema da Mochila
  - O item i não cabe na mochila
    - A capacidade disponível w é menor que o peso w_i adicionado pelo novo item
    - O valor total atingido pelo último item é repetido
  - O item i cabe na mochila
    - A capacidade disponível w é maior ou igual ao peso w_i do novo item inserido na mochila
    - Para o valor do item adicionado ser considerado, é incrementado o valor total contido na mochila

Programação dinâmica

- Problema da Mochila
  - O item i não cabe na mochila
    - A capacidade disponível w é menor que o peso w_i adicionado pelo novo item
    - O valor total atingido pelo último item é repetido
  - O item i cabe na mochila
    - A capacidade disponível w é maior ou igual ao peso w_i do novo item inserido na mochila
    - Para o valor do item adicionado ser considerado, é incrementado o valor total contido na mochila

$$V(i, w) = \begin{cases} V(i-1, w) & w - w_i < 0 \\ max\left(V(i-1, w), V(i-1, w-w_i) + v_i\right) & w - w_i \geq 0 \end{cases}$$

Programação dinâmica

- Problema da Mochila

i	1	2	3	4
v_i	12	10	20	15
w_i	2	1	3	2

$V(i,w)$	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0					
2	0					
3	0					
4	0					

Programação dinâmica

- Problema da Mochila

i	1	2	3	4
v_i	12	10	20	15
w_i	2	1	3	2

$V(i,w)$	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0					
2	0					
3	0					
4	0					

O item 1 não cabe na mochila $w = 1 < w_1$

$$V(1,1) = V(0,1)$$

Programação dinâmica

- Problema da Mochila

i	1	2	3	4
v_i	12	10	20	15
w_i	2	1	3	2

$V(i,w)$	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0				
2	0					
3	0					
4	0					

O item 1 cabe na mochila $w = 2 \geq w_1$

$$V(1,2) = max\,(V(0,2), V(0,0) + 12)$$

Programação dinâmica

- Problema da Mochila

i	1	2	3	4
v_i	12	10	20	15
w_i	2	1	3	2

V(i,w)	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	12			
2	0					
3	0					
4	0					

O item 1 cabe na mochila $w = 3 \geq w_1$

$$V(1,3) = max\,(V(0,3), V(0,1) + 12)$$

Programação dinâmica

- Problema da Mochila

i	1	2	3	4
v_i	12	10	20	15
w_i	2	1	3	2

V(i,w)	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	12	12		
2	0					
3	0					
4	0					

O item 1 cabe na mochila $w = 4 \geq w_1$

$$V(1,4) = max\,(V(0,4), V(0,2) + 12)$$

Programação dinâmica

- Problema da Mochila

i	1	2	3	4
v_i	12	10	20	15
w_i	2	1	3	2

V(i,w)	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	12	12	12	
2	0					
3	0					
4	0					

O item 1 cabe na mochila $w = 5 \geq w_1$

$V(1,5) = max\,(V(0,5), V(0,3) + 12)$

Programação dinâmica

- Problema da Mochila

i	1	2	3	4
v_i	12	10	20	15
w_i	2	1	3	2

V(i,w)	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	12	12	12	12
2	0					
3	0					
4	0					

Programação dinâmica

- Problema da Mochila

i	1	2	3	4
v_i	12	10	20	15
w_i	2	1	3	2

$V(i,w)$	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	12	12	12	12
2	0					
3	0					
4	0					

Programação dinâmica

- Problema da Mochila

i	1	2	3	4
v_i	12	10	20	15
w_i	2	1	3	2

V(i,w)	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	12	12	12	12
2	0					
3	0					
4	0					

Somente o item 2 cabe na mochila $w = 1 \geq w_2$

$$V(2,1) = max\,(V(1,1), V(1,0) + 10)$$

Programação dinâmica

- Problema da Mochila

i	1	2	3	4
v_i	12	10	20	15
w_i	2	1	3	2

$V(i,w)$	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	12	12	12	12
2	0	10				
3	0					
4	0					

Somente o item 1 cabe na mochila $w = 2 \geq w_2$

$$V(2,2) = max\,(V(1,2), V(1,1) + 10)$$

Programação dinâmica

- Problema da Mochila

i	1	2	3	4
v_i	12	10	20	15
w_i	2	1	3	2

$V(i,w)$	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	12	12	12	12
2	0	10	12			
3	0					
4	0					

Os itens 1 e 2 cabem na mochila $w = 3 \geq w_2$

$$V(2,3) = max\,(V(1,3), V(1,2) + 10)$$

Programação dinâmica

- Problema da Mochila

i	1	2	3	4
v_i	12	10	20	15
w_i	2	1	3	2

V(i,w)	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	12	12	12	12
2	0	10	12	22		
3	0					
4	0					

Os itens 1 e 2 cabem na mochila $w = 4 \geq w_2$

$$V(2,4) = max\,(V(1,4), V(1,3) + 10)$$

Programação dinâmica

- Problema da Mochila

i	1	2	3	4
v_i	12	10	20	15
w_i	2	1	3	2

V(i,w)	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	12	12	12	12
2	0	10	12	22	22	
3	0					
4	0					

Os itens 1 e 2 cabem na mochila $w = 5 \geq w_2$

$$V(2,5) = max\,(V(1,5), V(1,4) + 10)$$

Programação dinâmica

- Problema da Mochila

i	1	2	3	4
v_i	12	10	20	15
w_i	2	1	3	2

$V(i,w)$	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	12	12	12	12
2	0	10	12	22	22	22
3	0					
4	0					

Programação dinâmica

- Problema da Mochila

i	1	2	3	4
v_i	12	10	20	15
w_i	2	1	3	2

$V(i,w)$	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	12	12	12	12
2	0	10	12	22	22	22
3	0					
4	0					

Programação dinâmica

- Problema da Mochila

i	1	2	3	4
v_i	12	10	20	15
w_i	2	1	3	2

V(i,w)	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	12	12	12	12
2	0	10	12	22	22	22
3	0					
4	0					

Somente o item 2 cabe na mochila $w = 1 < w_3$

$$V(3,1) = V(2,1)$$

Programação dinâmica

- Problema da Mochila

i	1	2	3	4
v_i	12	10	20	15
w_i	2	1	3	2

V(i,w)	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	12	12	12	12
2	0	10	12	22	22	22
3	0	10				
4	0					

Somente o item 1 cabe na mochila $w = 1 < w_3$

$$V(3,2) = V(2,2)$$

Programação dinâmica

- Problema da Mochila

i	1	2	3	4
v_i	12	10	20	15
w_i	2	1	3	2

$V(i,w)$	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	12	12	12	12
2	0	10	12	22	22	22
3	0	10	12			
4	0					

Os itens 1 e 2 cabem na mochila $w = 3 \geq w_3$

$$V(3,3) = max\,(V(2,3), V(2,0) + 20)$$

Programação dinâmica

- Problema da Mochila

i	1	2	3	4
v_i	12	10	20	15
w_i	2	1	3	2

V(i,w)	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	12	12	12	12
2	0	10	12	22	22	22
3	0	10	12	22		
4	0					

Os itens 2 e 3 cabem na mochila $w = 4 \geq w_3$

$$V(3,4) = max\left(V(2,4), V(2,1) + 20\right)$$

Programação dinâmica

- Problema da Mochila

i	1	2	3	4
v_i	12	10	20	15
w_i	2	1	3	2

V(i,w)	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	12	12	12	12
2	0	10	12	22	22	22
3	0	10	12	22	30	
4	0					

Os itens 1 e 3 cabem na mochila $w = 5 \geq w_3$

$$V(3,5) = max\,(V(2,5), V(2,2) + 20)$$

Programação dinâmica

- Problema da Mochila

i	1	2	3	4
v_i	12	10	20	15
w_i	2	1	3	2

V(i,w)	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	12	12	12	12
2	0	10	12	22	22	22
3	0	10	12	22	30	32
4	0					

Programação dinâmica

- Problema da Mochila

i	1	2	3	4
v_i	12	10	20	15
w_i	2	1	3	2

V(i,w)	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	12	12	12	12
2	0	10	12	22	22	22
3	0	10	12	22	30	32
4	0					

Programação dinâmica

- Problema da Mochila

i	1	2	3	4
v_i	12	10	20	15
w_i	2	1	3	2

V(i,w)	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	12	12	12	12
2	0	10	12	22	22	22
3	0	10	12	22	30	32
4	0					

Somente o item 2 cabe na mochila $w = 1 < w_4$

$$V(4,1) = V(3,1)$$

Programação dinâmica

- Problema da Mochila

i	1	2	3	4
v_i	12	10	20	15
w_i	2	1	3	2

V(i,w)	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	12	12	12	12
2	0	10	12	22	22	22
3	0	10	12	22	30	32
4	0	10				

Somente o item 4 cabe na mochila $w = 3 \geq w_4$

$V(4,2) = max\,(V(3,2), V(3,0) + 15)$

Programação dinâmica

- Problema da Mochila

i	1	2	3	4
v_i	12	10	20	15
w_i	2	1	3	2

V(i,w)	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	12	12	12	12
2	0	10	12	22	22	22
3	0	10	12	22	30	32
4	0	10	15			

Os itens 1 e 4 cabem na mochila $w = 3 \geq w_3$

$$V(4,3) = max\,(V(3,3), V(3,1) + 15)$$

Programação dinâmica

- Problema da Mochila

i	1	2	3	4
v_i	12	10	20	15
w_i	2	1	3	2

V(i,w)	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	12	12	12	12
2	0	10	12	22	22	22
3	0	10	12	22	30	32
4	0	10	15	25		

Os itens 2 e 3 cabem na mochila $w = 4 \geq w_3$

$$V(4,4) = max\,(V(3,4), V(3,2) + 15)$$

Programação dinâmica

- Problema da Mochila

i	1	2	3	4
v_i	12	10	20	15
w_i	2	1	3	2

V(i,w)	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	12	12	12	12
2	0	10	12	22	22	22
3	0	10	12	22	30	32
4	0	10	15	25	30	

Os itens 1, 2 e 4 cabem na mochila $w = 5 \geq w_3$

$$V(4,5) = max\,(V(3,5), V(3,3) + 15)$$

Programação dinâmica

- Problema da Mochila

i	1	2	3	4
v_i	12	10	20	15
w_i	2	1	3	2

$V(i,w)$	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	12	12	12	12
2	0	10	12	22	22	22
3	0	10	12	22	30	32
4	0	10	15	25	30	31

Como determinar algoritmicamente quais são os itens que maximizam o valor armazenado pela mochila?

Programação dinâmica

- Problema da Mochila

i	1	2	3	4
v_i	12	10	20	15
w_i	2	1	3	2

V(i,w)	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	12	12	12	12
2	0	10	12	22	22	22
3	0	10	12	22	30	32
4	0	10	15	25	30	31

Como $V(i,j) \neq V(i-1,j)$, então o item $i = 4$ faz parte da solução e $i = i - 1, j = j - w_i$

Programação dinâmica

- Problema da Mochila

i	1	2	3	4
v_i	12	10	20	15
w_i	2	1	3	2

V(i,w)	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	12	12	12	12
2	0	10	12	22	22	22
3	0	10	12	22	30	32
4	0	10	15	25	30	37

Como $V(i,j) = V(i-1,j)$, o item $i = 3$ não faz parte da solução e $i = i - 1$

Programação dinâmica

- Problema da Mochila

i	1	2	3	4
v_i	12	10	20	15
w_i	2	1	3	2

V(i,w)	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	12	12	12	12
2	0	10	12	22	22	22
3	0	10	12	22	30	32
4	0	10	15	25	30	37

Como $V(i,j) \neq V(i-1,j)$, então o item $i = 2$ faz parte da solução e $i = i - 1, j = j - w_i$

Programação dinâmica

- Problema da Mochila

i	1	2	3	4
v_i	12	10	20	15
w_i	2	1	3	2

V(i,w)	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	12	12	12	12
2	0	10	12	22	22	22
3	0	10	12	22	30	32
4	0	10	15	25	30	37

Como $V(i,j) \neq V(i-1,j)$, então o item $i = 1$ faz parte da solução e $i = i - 1, j = j - w_i$

Programação dinâmica

- Problema da Mochila

i	1	2	3	4
v_i	12	10	20	15
w_i	2	1	3	2

V(i,w)	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	12	12	12	12
2	0	10	12	22	22	22
3	0	10	12	22	30	32
4	0	10	15	25	30	37

A solução ótima possui valor 37,
sendo composta pelos itens 1, 2 e 4

Programação dinâmica

- Análise de complexidade
  - Pseudo-polinomial $O(n \times W)$
    - É polinomial para o tamanho da entrada n que é o número de itens que podem ser colocados na mochila

Programação dinâmica

- Análise de complexidade
  - Pseudo-polinomial $O(n \times W)$
    - É polinomial para o tamanho da entrada n que é o número de itens que podem ser colocados na mochila
    - A capacidade máxima da mochila W possui um crescimento exponencial para o valor numérico que pode ser descrito por sua largura em bits

$$
\begin{aligned}
\log_2 W &= m \\
W &= 2^m
\end{aligned}
$$

Programação dinâmica

- Aplicações do problema da Mochila
  - Criação cópia de arquivos: armazenar o máximo de arquivos do disco, sem exceder a capacidade

Programação dinâmica

- Aplicações do problema da Mochila
  - Criação cópia de arquivos: armazenar o máximo de arquivos do disco, sem exceder a capacidade
  - Logística e transporte: otimizar a carga de veículos, maximizando a prioridade ou valor dos itens

Programação dinâmica

- Aplicações do problema da Mochila
  - Criação cópia de arquivos: armazenar o máximo de arquivos do disco, sem exceder a capacidade
  - Logística e transporte: otimizar a carga de veículos, maximizando a prioridade ou valor dos itens
  - Alocação de recursos em um sistema: escalonar os processos para execução e utilização dos dispositivos

Programação dinâmica

- Aplicações do problema da Mochila
  - Criação cópia de arquivos: armazenar o máximo de arquivos do disco, sem exceder a capacidade
  - Logística e transporte: otimizar a carga de veículos, maximizando a prioridade ou valor dos itens
  - Alocação de recursos em um sistema: escalonar os processos para execução e utilização dos dispositivos
  - ...

Exemplo

- Considerando o problema da Mochila e aplicando as técnicas de programação dinâmica vistas, encontre o conjunto de itens que maximiza o valor da mochila que possui capacidade $W = 10$

i	1	2	3	4	5
v_i	35	44	33	10	55
w_i	3	2	4	1	5

Exercício

- A empresa de transportes Poxim Tech está tornando as entregas de encomendas de última milha mais eficientes e rápidas, através da maximização do valor total do frete e levando em consideração as características de cada tipo de veículo
  - Para codificação das placas de trânsito dos veículos e dos códigos de rastreamento dos pacotes são utilizados os símbolos *L* e *N*, que representam letras maiúsculas e números, respectivamente
  - As unidades utilizadas para peso é quilo (*kg*), valor é reais (*R*$) e volume é litro (*l*)
  - Cada veículo é identificado pela sua placa de trânsito, nos formatos *LLLNNNN* ou *LLLNLNN*, além da sua capacidade máxima de carga (peso e volume)
  - Os pacotes possuem um código de rastreamento descrito no padrão *LLNNNNNNNNNLL*, juntamente com informações sobre o valor, peso e volume

Exercício

- Formato do arquivo de entrada
  - [$\#n$]
  - [$Placa_0$] [$Peso_0$] [$Volume_0$]
  - $\vdots$
  - [$Placa_{n-1}$] [$Peso_{n-1}$] [$Volume_{n-1}$]
  - [$\#m$]
  - [$Código_0$] [$Valor_0$] [$Peso_0$] [$Volume_0$]
  - $\vdots$
  - [$Código_{m-1}$] [$Valor_{m-1}$] [$Peso_{m-1}$] [$Volume_{m-1}$]

```
2
AAA1234␣50␣100
BBB5C67␣2000␣12000
5
AB111222333CD␣49.99␣2␣1
EF444555666GH␣5000.01␣1234␣7000
IJ777888999KL␣100␣49␣10
MN000111222OP␣65.01␣3␣125
QR333444555ST␣200.01␣13␣4875
```

Exercício

- Formato do arquivo de saída
  - Para cada veículo é gerada uma sequência de carregamento dos pacotes que maximizam o valor transportado sem exceder a capacidade de carga

```
[AAA1234]R$100.00,49KG(98%),10L(10%)
IJ777888999KL
[BBB5C67]R$5265.03,1250KG(63%),12000L(100%)
EF444555666GH
MN000111222OP
QR333444555ST
[PENDENTE]R$49.99,2KG,1L
AB111222333CD
```