Projeto e Análise de Algoritmos

Programação Dinâmica

Programação Dinâmica

- Paradigma de programação poderoso.

Programação Dinâmica

- Paradigma de programação poderoso.
- Ótima ideia; difícil, mas simples ao mesmo tempo.

Programação Dinâmica

- Paradigma de programação poderoso.
- Ótima ideia; difícil, mas simples ao mesmo tempo.
- Usa soluções ótimas de subproblemas para chegar no resultado final.
  - Optimality principle.

Programação Dinâmica

- Paradigma de programação poderoso.
- Ótima ideia; difícil, mas simples ao mesmo tempo.
- Usa soluções ótimas de subproblemas para chegar no resultado final.
  - Optimality principle.
- PD ≈ força bruta controlada.

Programação Dinâmica

- Para usar PD, é preciso conhecer:
  - Os estados do problema;
  - As transições entre eles.

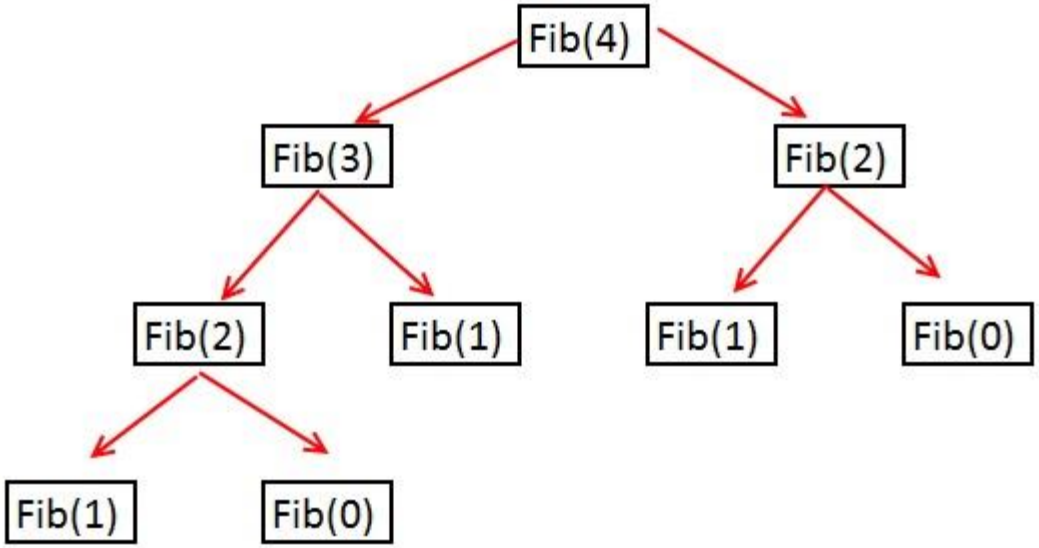
Programação Dinâmica

- Para usar PD, é preciso conhecer:
  - Os estados do problema;
  - As transições entre eles.
- A ideia é ir resolvendo problemas menores iterativamente.
  - E salvar seus resultados - memo(r)ization.

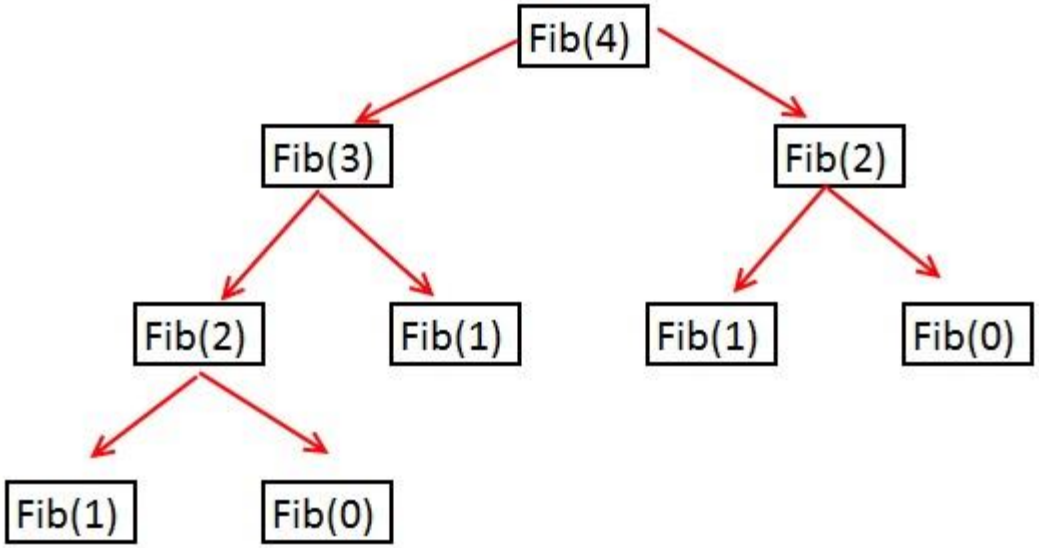
Programação Dinâmica

- Para usar PD, é preciso conhecer:
  - Os estados do problema;
  - As transições entre eles.
- A ideia é ir resolvendo problemas menores iterativamente.
  - E salvar seus resultados - memo(r)ization.
- Ter os resultados salvos evita calcular mais de uma vez um mesmo subproblema.

Programação Dinâmica



Programação Dinâmica



- Salvar resultados anteriores é uma boa ideia?

Programação Dinâmica

- Há uma relação intrínseca entre PD e um grafo acíclico não-direcionado (DAG).

Programação Dinâmica

- Há uma relação intrínseca entre PD e um grafo acíclico não-direcionado (DAG).
- Vamos ver um exemplo pra facilitar...

Programação Dinâmica - LIS

- Longest Increasing Subsequence (LIS).

Programação Dinâmica - LIS

- Longest Increasing Subsequence (LIS).

- Dada uma sequência de números $a_1 \ldots a_n$, uma *subsequência* é qualquer subconjunto desses números, na ordem que aparecem, na forma $a_{i1}, a_{i2}, \ldots a_{ik}$ tal que $1 \leq i_1 < i_2 < \cdots < i_k \leq n$ e uma *subsequência crescente* é uma tal que os números são estritamente maiores que seus antecessores.

Programação Dinâmica - LIS

- Por exemplo, considere a seguinte sequência:
  - 5, 2, 8, 6, 3, 6, 9, 7

Programação Dinâmica - LIS

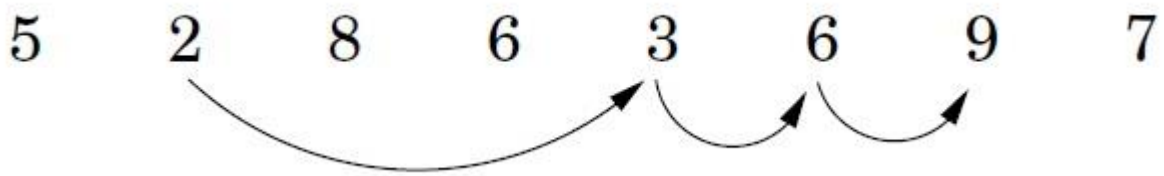
- Por exemplo, considere a seguinte sequência:
  - 5, 2, 8, 6, 3, 6, 9, 7

- Nesse caso, a LIS é 2, 3, 6, 9.

Programação Dinâmica - LIS

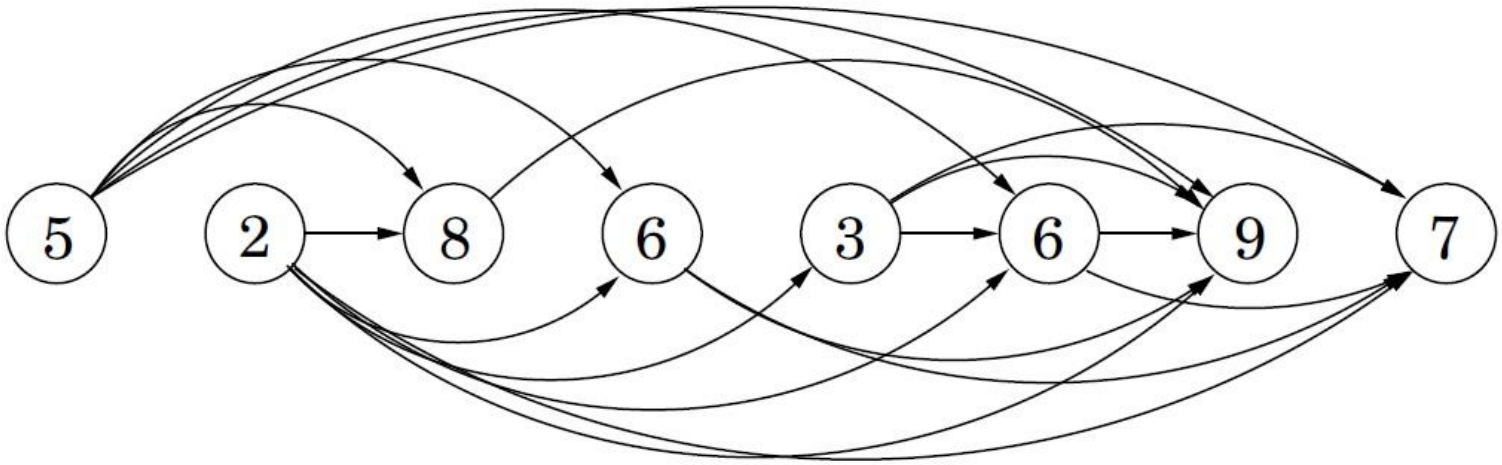
- Por exemplo, considere a seguinte sequência:
  - 5, 2, 8, 6, 3, 6, 9, 7

- Nesse caso, a LIS é 2, 3, 6, 9.



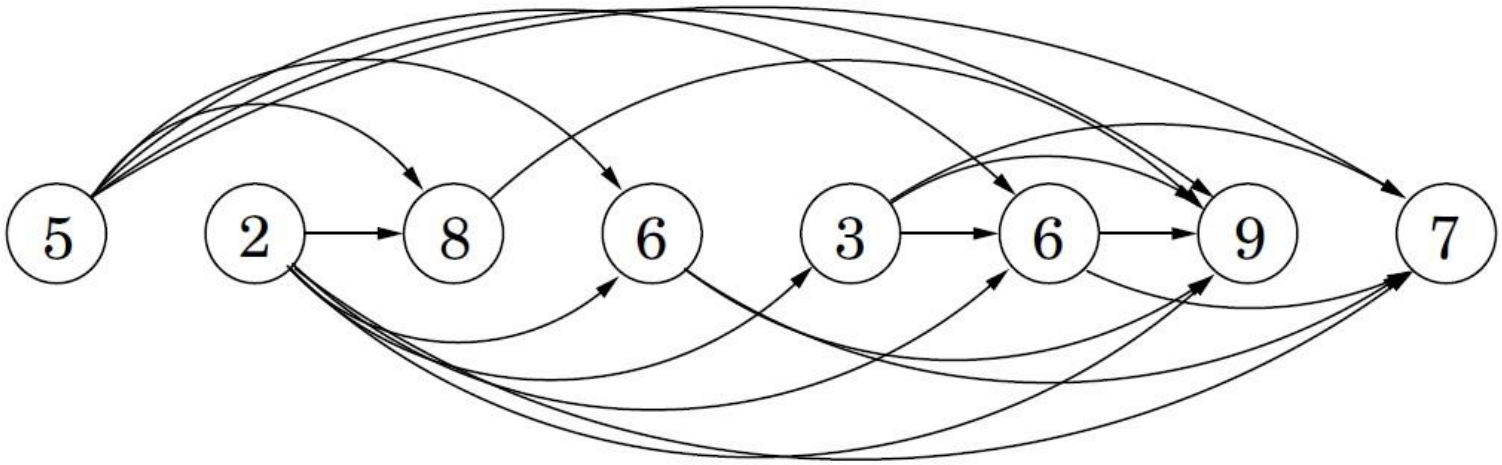
Programação Dinâmica - LIS

- Assim, o DAG é



Programação Dinâmica - LIS

- Assim, o DAG é



- Para chegar na solução, basta achar o **maior** caminho presente no grafo!

Programação Dinâmica - LIS

- Assim, seja (i, j) uma aresta que sai do nó i e chega no nó j;

Programação Dinâmica - LIS

- Assim, seja (i, j) uma aresta que sai do nó i e chega no nó j;
- Seja E o conjunto de todas as arestas;

Programação Dinâmica - LIS

- Assim, seja (i, j) uma aresta que sai do nó i e chega no nó j;
- Seja E o conjunto de todas as arestas;
- Seja $L(j)$ o maior caminho terminando em j.

Programação Dinâmica - LIS

- Assim, seja (i, j) uma aresta que sai do nó i e chega no nó j;
- Seja E o conjunto de todas as arestas;
- Seja $L(j)$ o maior caminho terminando em j.

```
for j = 1 to n
     L[j] = 1 + max(L[i] : (i, j) ϵ E)
return max(L)
```

Programação Dinâmica - LIS

- Assim, seja (i, j) uma aresta que sai do nó i e chega no nó j;
- Seja E o conjunto de todas as arestas;
- Seja $L(j)$ o maior caminho terminando em j.

```
for j = 1 to n
     L[j] = 1 + max(L[i] : (i, j) ϵ E)
return max(L)
```

Programação Dinâmica – Uso

- Usamos PD quando há grande sobreposição de subproblemas.
  - Maximizações e minimizações caem nessa categoria.

Programação Dinâmica – Uso

- Usamos PD quando há grande sobreposição de subproblemas.
  - Maximizações e minimizações caem nessa categoria.
- Também usamos quando o cálculo de um subproblema é recorrente.

Programação Dinâmica – Uso

- Usamos PD quando há grande sobreposição de subproblemas.
  - Maximizações e minimizações caem nessa categoria.
- Também usamos quando o cálculo de um subproblema é recorrente.
- Muitos problemas “exponenciais” podem ser resolvidos em tempo polinomial usando PD!

Programação Dinâmica – Uso

- De forma geral:
  - Defina os subproblemas.
  - Ache parte da solução.
  - Relacione subproblemas já resolvidos.
  - Guarde os resultados já calculados (bottom-up).
  - Resolva o problema original.

Exemplos

Largest Sum Contiguous Subarray (LSCS)

Programação Dinâmica – LSCS

- Dada uma sequência de números inteiros, retorne a maior soma possível de ser obtida usando apenas elementos de posições contíguas.

Programação Dinâmica – LSCS

- Dada uma sequência de números inteiros, retorne a maior soma possível de ser obtida usando apenas elementos de posições contíguas.

- Esse problema já foi visto em sala.
  - Uma solução de divisão e conquista já foi apresentada.
  - Qual era a complexidade?

Programação Dinâmica – LSCS

- Esse problema pode ser resolvido usando PD!

Programação Dinâmica – LSCS

- Esse problema pode ser resolvido usando PD!
- Quais são os subproblemas?

Programação Dinâmica – LSCS

- Esse problema pode ser resolvido usando PD!
- Quais são os subproblemas?
- Como relacioná-los pra resolver o problema original?

Programação Dinâmica – LSCS

- Esse problema pode ser resolvido usando PD!
- Quais são os subproblemas?
- Como relacioná-los pra resolver o problema original?
- Vamos trabalhar com essa sequência:

Programação Dinâmica – LSCS

- Esse problema pode ser resolvido usando PD!
- Quais são os subproblemas?
- Como relacioná-los pra resolver o problema original?
- Vamos trabalhar com essa sequência:

-3	-7	10	-3	-2	1	3	4	-6

Programação Dinâmica – LSCS

-3	-7	10	-3	-2	1	3	4	-6

- Sabemos que a maior soma é 13.

Programação Dinâmica – LSCS

-3	-7	10	-3	-2	1	3	4	-6

- Sabemos que a maior soma é 13.
- PD exige um pouco de experimentação.

Programação Dinâmica – LSCS

-3	-7	10	-3	-2	1	3	4	-6

- Sabemos que a maior soma é 13.
- PD exige um pouco de experimentação.
- Uma boa estratégia: qual é o caso mais simples?

Programação Dinâmica – LSCS

- Quando a sequência tem um só número:
  - O resultado é o próprio número.
  - Parece um bom caso-base!

Programação Dinâmica – LSCS

-3	-7	10	-3	-2	1	3	4	-6

- Quando a sequência tem um só número:
  - O resultado é o próprio número.
  - Parece um bom caso-base!

- Quando colocamos mais números na sequência...
  - Como decidir se ele faz parte da LSCS?

Programação Dinâmica – LSCS

-3	-7	10	-3	-2	1	3	4	-6

- Se o elemento atual **somado** ao valor da sequência até o momento for **maior** que o próprio elemento, então ele entra na sequência também.

Programação Dinâmica – LSCS

-3	-7	10	-3	-2	1	3	4	-6

- Se o elemento atual **somado** ao valor da sequência até o momento for **maior** que o próprio elemento, então ele entra na sequência também.

- Se não for, descartamos a sequência que tínhamos e começamos ela partindo do elemento atual.

Programação Dinâmica – LSCS

- Mais formalmente, seja **`array`** um vetor que armazena a sequência e **`lscs`** um vetor que armazena os resultados dos subproblemas.

Programação Dinâmica – LSCS

-3	-7	10	-3	-2	1	3	4	-6

- Mais formalmente, seja **`array`** um vetor que armazena a sequência e **`lscs`** um vetor que armazena os resultados dos subproblemas.

- O algoritmo fica:

Programação Dinâmica – LSCS

-3	-7	10	-3	-2	1	3	4	-6

- Mais formalmente, seja **`array`** um vetor que armazena a sequência e **`lscs`** um vetor que armazena os resultados dos subproblemas.

- O algoritmo fica:

```
lscs[0] = array[0]
for j = 1, …, n - 1:
   lscs[j] = max(array[j], array[j] + lscs[j – 1])
return max(lscs)
```

Programação Dinâmica – LSCS

-3	-7	10	-3	-2	1	3	4	-6

- Problema resolvido!

Programação Dinâmica – LSCS

-3	-7	10	-3	-2	1	3	4	-6

- Problema resolvido!
- Qual é a complexidade da solução?

Programação Dinâmica – LSCS

-3	-7	10	-3	-2	1	3	4	-6

- Problema resolvido!
- Qual é a complexidade da solução?
- Como fica o DAG desse problema?

Troco

Programação Dinâmica – Troco

- O problema do troco é clássico.

Programação Dinâmica – Troco

- O problema do troco é clássico.

- Dados alguns valores de moedas, qual o menor número de moedas que podemos usar para formar um valor de troco?

Programação Dinâmica – Troco

- Considere que os valores são

Programação Dinâmica – Troco

- Considere que os valores são

1	2	5	10	20	50	100

Programação Dinâmica – Troco

- Considere que os valores são

1	2	5	10	20	50	100

- Considere ainda que o valor do troco é 37.

Programação Dinâmica – Troco

- Considere que os valores são

1	2	5	10	20	50	100

- Considere ainda que o valor do troco é 37.
- Como resolver?

Programação Dinâmica – Troco

- E se os valores das moedas mudarem?

Programação Dinâmica – Troco

- E se os valores das moedas mudarem?

1	7	11

Programação Dinâmica – Troco

- E se os valores das moedas mudarem?

1	7	11

- Considere ainda que o valor do troco é 14.

Programação Dinâmica – Troco

- E se os valores das moedas mudarem?

1	7	11

- Considere ainda que o valor do troco é 14.
- A solução pensada anteriormente ainda funciona?

Programação Dinâmica – Troco

1	7	11

Programação Dinâmica – Troco

1	7	11

- Quais são os subproblemas?

Programação Dinâmica – Troco

1	7	11

- Quais são os subproblemas?
- Como eles se relacionam?

Programação Dinâmica – Troco

1	7	11

- Quais são os subproblemas?
- Como eles se relacionam?
- E o caso-base?

Programação Dinâmica – Troco

1	7	11

Programação Dinâmica – Troco

1	7	11

- Podemos ir resolvendo o problema para valores menores que 14.

Programação Dinâmica – Troco

1	7	11

- Podemos ir resolvendo o problema para valores menores que 14.
- E depois usarmos essas soluções para resolver o problema original.

Programação Dinâmica – Troco

1	7	11

Programação Dinâmica – Troco

1	7	11

V	0
M(V)	0

Programação Dinâmica – Troco

1	7	11

V	0
M(V)	0

V	1	2	3	4	5	6	7
M(V)							

Programação Dinâmica – Troco

1	7	11

V	0
M(V)	0

V	1	2	3	4	5	6	7
M(V)							

V	8	9	10	11	12	13	14
M(V)							

Programação Dinâmica – Troco

1	7	11

V	0
M(V)	0

V	1	2	3	4	5	6	7
M(V)							

V	8	9	10	11	12	13	14
M(V)							

Programação Dinâmica – Troco

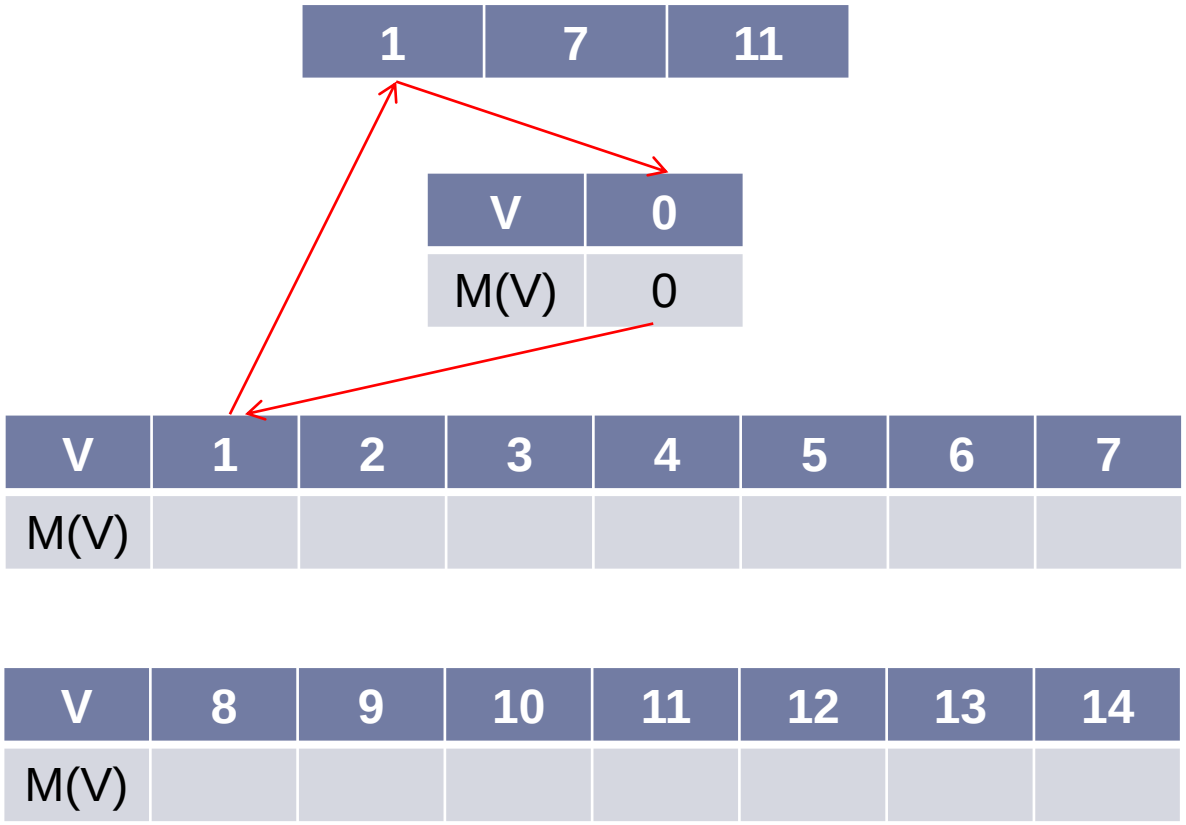
1	7	11

V	0
M(V)	0

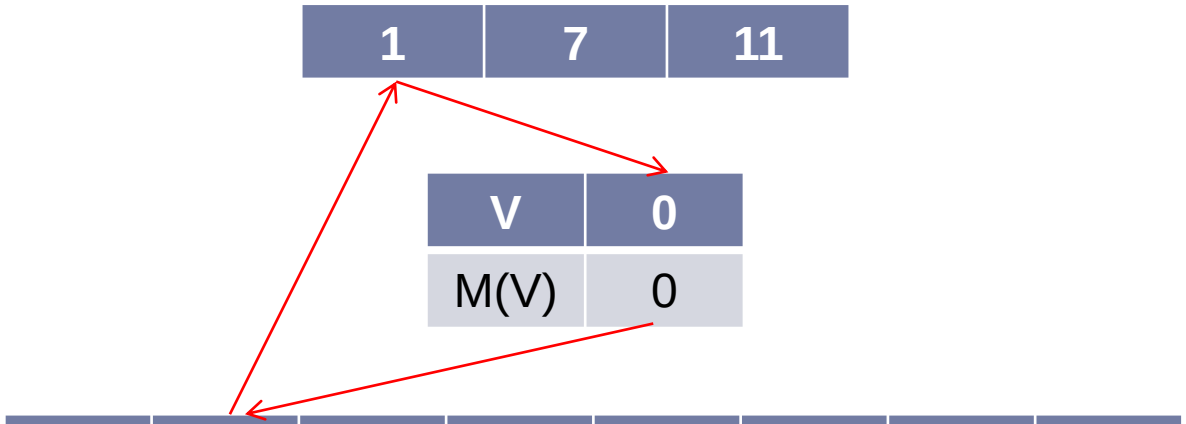
V	1	2	3	4	5	6	7
M(V)							

V	8	9	10	11	12	13	14
M(V)							

Programação Dinâmica – Troco



Programação Dinâmica – Troco



V	1	2	3	4	5	6	7
M(V)	1						

V	8	9	10	11	12	13	14
M(V)							

Programação Dinâmica – Troco

1	7	11

V	0
M(V)	0

V	1	2	3	4	5	6	7
M(V)	1	2	3	4	5	6	?

V	8	9	10	11	12	13	14
M(V)							

Programação Dinâmica – Troco

1	7	11

V	0
M(V)	0

V	1	2	3	4	5	6	7
M(V)	1	2	3	4	5	6	

V	8	9	10	11	12	13	14
M(V)							

Programação Dinâmica – Troco

1	7	11

V	0
M(V)	0

V	1	2	3	4	5	6	7
M(V)	1	2	3	4	5	6	

V	8	9	10	11	12	13	14
M(V)							

Programação Dinâmica – Troco

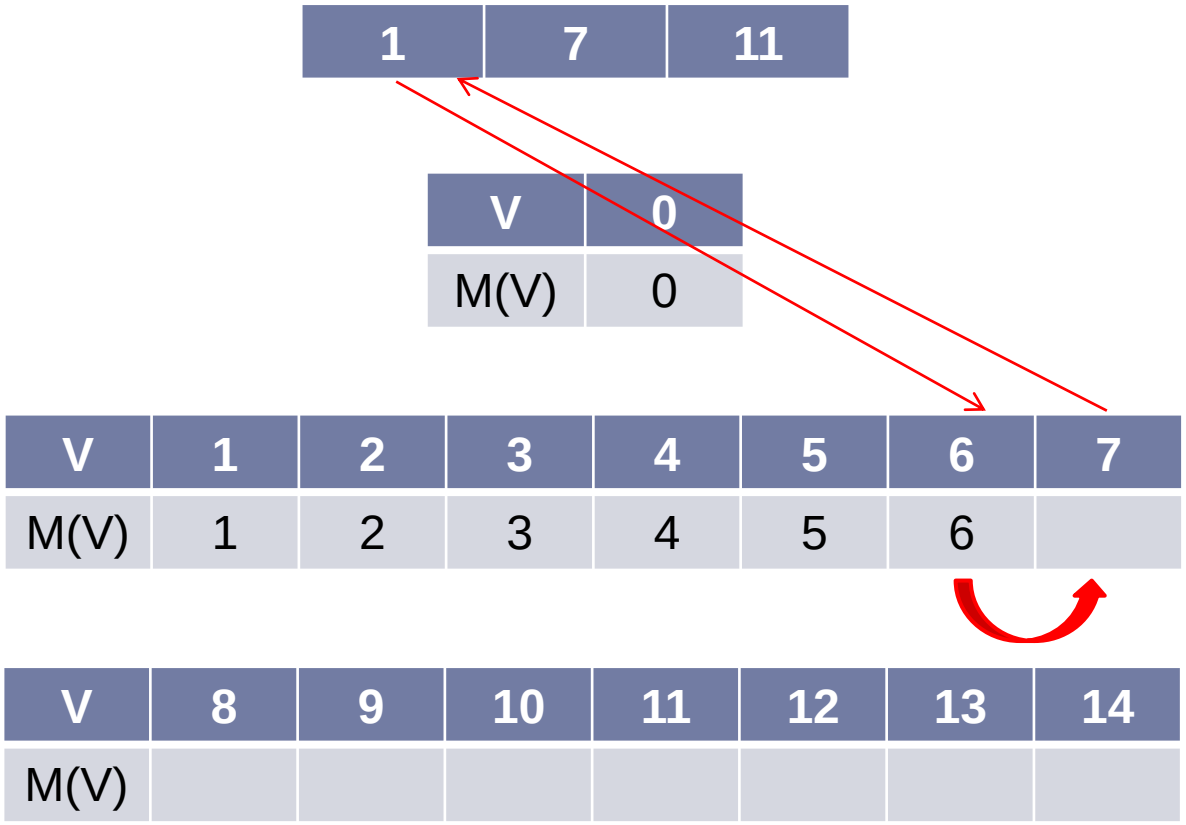
1	7	11

V	0
M(V)	0

V	1	2	3	4	5	6	7
M(V)	1	2	3	4	5	6	

V	8	9	10	11	12	13	14
M(V)							

Programação Dinâmica – Troco



Programação Dinâmica – Troco

1	7	11

V	0
M(V)	0

V	1	2	3	4	5	6	7
M(V)	1	2	3	4	5	6	7

V	8	9	10	11	12	13	14
M(V)							

Programação Dinâmica – Troco

1	7	11

V	0
M(V)	0

V	1	2	3	4	5	6	7
M(V)	1	2	3	4	5	6	7

V	8	9	10	11	12	13	14
M(V)							

Programação Dinâmica – Troco

1	7	11

V	0
M(V)	0

V	1	2	3	4	5	6	7
M(V)	1	2	3	4	5	6	7

V	8	9	10	11	12	13	14
M(V)							

Programação Dinâmica – Troco

1	7	11

V	0
M(V)	0

V	1	2	3	4	5	6	7
M(V)	1	2	3	4	5	6	7

V	8	9	10	11	12	13	14
M(V)							

Programação Dinâmica – Troco

1	7	11

V	0
M(V)	0

V	1	2	3	4	5	6	7
M(V)	1	2	3	4	5	6	7

V	8	9	10	11	12	13	14
M(V)							

Programação Dinâmica – Troco

1	7	11

V	0
M(V)	0

V	1	2	3	4	5	6	7
M(V)	1	2	3	4	5	6	X

V	8	9	10	11	12	13	14
M(V)							

Programação Dinâmica – Troco

1	7	11

V	0
M(V)	0

V	1	2	3	4	5	6	7
M(V)	1	2	3	4	5	6	**1**

V	8	9	10	11	12	13	14
M(V)							

Programação Dinâmica – Troco

1	7	11

V	0
M(V)	0

V	1	2	3	4	5	6	7
M(V)	1	2	3	4	5	6	1

V	8	9	10	11	12	13	14
M(V)							

Programação Dinâmica – Troco

1	7	11

V	0
M(V)	0

V	1	2	3	4	5	6	7
M(V)	1	2	3	4	5	6	1

V	8	9	10	11	12	13	14
M(V)	2	3	4	1	2	3	2

Programação Dinâmica – Troco

1	7	11

V	0
M(V)	0

V	1	2	3	4	5	6	7
M(V)	1	2	3	4	5	6	1

V	8	9	10	11	12	13	14
M(V)	2	3	4	1	2	3	2

Programação Dinâmica – Troco

- Seja **`valores`** um vetor para armazenar o número mínimo de moedas para valores de 0 até o valor do troco;

Programação Dinâmica – Troco

- Seja **`valores`** um vetor para armazenar o número mínimo de moedas para valores de 0 até o valor do troco;
- Seja **`moedas`** um vetor com os valores de cada moeda.

Programação Dinâmica – Troco

- Seja **valores** um vetor para armazenar o número mínimo de moedas para valores de 0 até o valor do troco;
- Seja **moedas** um vetor com os valores de cada moeda.

```
valores[1..n] = +INF
valores[0] = 0
for i = 1, 2, …, n
   j = 0
   while(j < moedas.size() && moedas[j] <= i)
      valores[i] = min(valores[i], valores[i – moedas[j]] + 1)
return valores[n]
```

Programação Dinâmica – Troco

- Qual a complexidade dessa solução?

Programação Dinâmica – Troco

- Qual a complexidade dessa solução?
- Como fica o DAG?

Programação Dinâmica – Troco

- PD também é muito útil para problemas de contagem.

Programação Dinâmica – Troco

- PD também é muito útil para problemas de contagem.

- Dado um inteiro `n` e os valores das moedas, de quantas maneiras diferentes podemos voltar o troco?
  - Fica de exercício.

Mochila (Knapsack problem)

Programação Dinâmica – Mochila

- O problema da mochila é outro clássico.

Programação Dinâmica – Mochila

- O problema da mochila é outro clássico.

- Dados uma lista de itens, os valores de cada item e a capacidade total da mochila, qual é a melhor escolha de itens a se levar de forma a se maximizar o seu ganho?
  - Sim, o problema assume que você é um ladrão.

Programação Dinâmica – Mochila

- Há duas variações para esse problema.
  - Os itens são ilimitados.
  - Os itens não são ilimitados.

Programação Dinâmica – Mochila

- Há duas variações para esse problema.
  - Os itens são ilimitados.
  - Os itens não são ilimitados.

- O raciocínio pra cada um é um pouco diferente.

Programação Dinâmica – Mochila

- Considere que os itens são esses e a mochila tem capacidade para 10kg.

Item	Peso (kg)	Valor
1	6	R$ 30
2	3	R$ 14
3	4	R$ 16
4	2	R$ 9

Programação Dinâmica – Mochila

- Considere que os itens são esses e a mochila tem capacidade para 10kg.

Item	Peso (kg)	Valor
1	6	R$ 30
2	3	R$ 14
3	4	R$ 16
4	2	R$ 9

- Itens ilimitados: R$ 48 (1 item 1 e 2 itens 4).

Programação Dinâmica – Mochila

- Considere que os itens são esses e a mochila tem capacidade para 10kg.

Item	Peso (kg)	Valor
1	6	R$ 30
2	3	R$ 14
3	4	R$ 16
4	2	R$ 9

- Itens ilimitados: R$ 48 (1 item 1 e 2 itens 4).
- Itens limitados: R$ 46 (item 1 e item 3).

Programação Dinâmica – Mochila

- Quais são os subproblemas?

Programação Dinâmica – Mochila

- Quais são os subproblemas?
- Como eles estão relacionados?

Programação Dinâmica – Mochila

- Quais são os subproblemas?
- Como eles estão relacionados?
- Note que a solução para itens ilimitados é muito parecida com o problema do troco.
  - A de itens limitados fica de exercício.

Programação Dinâmica – Mochila

- A ideia é ir calculado as soluções ótimas para mochilas com capacidades menores.

Programação Dinâmica – Mochila

- A ideia é ir calculado as soluções ótimas para mochilas com capacidades menores.
- E, como de costume, iremos combinar essas soluções ótimas para resolver o problema original.

Programação Dinâmica – Mochila

- A ideia é ir calculado as soluções ótimas para mochilas com capacidades menores.
- E, como de costume, iremos combinar essas soluções ótimas para resolver o problema original.
- Qual é a recorrência?

Programação Dinâmica – Mochila

- Considere que a mochila `K` tem capacidade `W` e cada item `i` tem peso w_i e valor v_i.

Programação Dinâmica – Mochila

- Considere que a mochila **K** tem capacidade **W** e cada item **i** tem peso w_i e valor v_i.

- Se o item **i** faz parte da solução ótima para **K(W)**, então **K(W - w_i)** é solução ótima para uma mochila com capacidade **W - w_i**.

Programação Dinâmica – Mochila

- Considere que a mochila **K** tem capacidade **W** e cada item **i** tem peso w_i e valor v_i.
- Se o item **i** faz parte da solução ótima para **K(W)**, então **K(W - w_i)** é solução ótima para uma mochila com capacidade **W - w_i**.
- Então, **K(W)** pode ser escrito como:
  - **K(W) = K(W - w_i) +** v_i, para algum valor de **i**.

Programação Dinâmica – Mochila

- O algoritmo fica

```
K[0] = 0
for w = 1 to W
     K[w] = max(K[w - w_i] + v_i : w_i ≤ w)
return K[W]
```

Programação Dinâmica – Mochila

- O algoritmo fica

```
K[0] = 0
for w = 1 to W
     K[w] = max(K[w - w_i] + v_i : w_i ≤ w)
return K[W]
```

- Qual é a complexidade?

Programação Dinâmica – Mochila

- O algoritmo fica

```
K[0] = 0
for w = 1 to W
     K[w] = max(K[w - w_i] + v_i : w_i ≤ w)
return K[W]
```

- Qual é a complexidade?

- Como fica o DAG?

Resumo

Programação Dinâmica – Resumo

- PD é uma técnica muito poderosa.
- É preciso identificar se o problema possui a estrutura para uma solução em PD.
  - Tem sobreposição de subproblemas?
  - Os subproblemas se relacionam? (recorrência)
  - Os subproblemas possuem soluções ótimas? (optimality principle).

Programação Dinâmica – Resumo

- Grande aplicabilidade em:
  - Maximização.
  - Minimização.
  - Contagem.
- Tais situações, a princípio, podem parecer ter soluções que irão levar muito tempo.
- Uso de PD pode reduzir **drasticamente** o tempo de execução.

Perguntas?