CI1238 - Otimização

Aula 10 - Programação Dinâmica

Professor Murilo V. G. da Silva

Departamento de Informática

Universidade Federal do Paraná

21/03/2023

Programação Dinâmica (PD)

- PD é uma técnica para projeto de algorimos

Programação Dinâmica (PD)

- PD é uma técnica para projeto de algorimos
- Baseada na ideia de resolver subproblemas menores e combinar as repostas na resolução do problema maior

Programação Dinâmica (PD)

- PD é uma técnica para projeto de algorimos
- Baseada na ideia de resolver subproblemas menores e combinar as repostas na resolução do problema maior
  - Entretanto, distinta da técnica de dividir para conquistar

Programação Dinâmica (PD)

- PD é uma técnica para projeto de algorimos
- Baseada na ideia de resolver subproblemas menores e combinar as repostas na resolução do problema maior
  - Entretanto, distinta da técnica de dividir para conquistar
  - No caso de PD, subproblemas **podem ter intersecção**

Programação Dinâmica (PD)

- PD é uma técnica para projeto de algorimos
- Baseada na ideia de resolver subproblemas menores e combinar as repostas na resolução do problema maior
  - Entretanto, distinta da técnica de dividir para conquistar
  - No caso de PD, subproblemas **podem ter intersecção**
  - Idéia central: Identificar quando é possível não resolver subproblemas mais de uma vez.

Programação Dinâmica (PD)

- PD é uma técnica para projeto de algorimos
- Baseada na ideia de resolver subproblemas menores e combinar as repostas na resolução do problema maior
  - Entretanto, distinta da técnica de dividir para conquistar
  - No caso de PD, subproblemas **podem ter intersecção**
  - Idéia central: Identificar quando é possível não resolver subproblemas mais de uma vez.

Vamos ilustrar PD com um exemplo: o Problema do Corte de Haste.

Programação Dinâmica (PD)

- PD é uma técnica para projeto de algorimos
- Baseada na ideia de resolver subproblemas menores e combinar as repostas na resolução do problema maior
  - Entretanto, distinta da técnica de dividir para conquistar
  - No caso de PD, subproblemas **podem ter intersecção**
  - Idéia central: Identificar quando é possível não resolver subproblemas mais de uma vez.

Vamos ilustrar PD com um exemplo: o Problema do Corte de Haste.

- Para este problema vamos ver a técnica em bastante detalhe

Programação Dinâmica (PD)

- PD é uma técnica para projeto de algorimos
- Baseada na ideia de resolver subproblemas menores e combinar as repostas na resolução do problema maior
  - Entretanto, distinta da técnica de dividir para conquistar
  - No caso de PD, subproblemas **podem ter intersecção**
  - Idéia central: Identificar quando é possível não resolver subproblemas mais de uma vez.

Vamos ilustrar PD com um exemplo: o Problema do Corte de Haste.

- Para este problema vamos ver a técnica em bastante detalhe
- Fonte: Capítulo 15 do livro *Introduction to Algorithms*

Programação Dinâmica (PD)

- PD é uma técnica para projeto de algorimos
- Baseada na ideia de resolver subproblemas menores e combinar as repostas na resolução do problema maior
  - Entretanto, distinta da técnica de dividir para conquistar
  - No caso de PD, subproblemas **podem ter intersecção**
  - Idéia central: Identificar quando é possível não resolver subproblemas mais de uma vez.

Vamos ilustrar PD com um exemplo: o Problema do Corte de Haste.

- Para este problema vamos ver a técnica em bastante detalhe
- Fonte: Capítulo 15 do livro *Introduction to Algorithms*

- Em seguida veremos a mesma técnica em mais alto nível para vários problemas

Programação Dinâmica (PD)

- PD é uma técnica para projeto de algorimos
- Baseada na ideia de resolver subproblemas menores e combinar as repostas na resolução do problema maior
  - Entretanto, distinta da técnica de dividir para conquistar
  - No caso de PD, subproblemas **podem ter intersecção**
  - Idéia central: Identificar quando é possível não resolver subproblemas mais de uma vez.

Vamos ilustrar PD com um exemplo: o Problema do Corte de Haste.

- Para este problema vamos ver a técnica em bastante detalhe
- Fonte: Capítulo 15 do livro *Introduction to Algorithms*

- Em seguida veremos a mesma técnica em mais alto nível para vários problemas
- Fonte: Capítulo 6 do livro *Algorithms*

O Problema do Corte de Hastes

O Problema do Corte de Hastes

- Ideia: Você tem uma haste de comprimento n e quer cortar em pedaços menores para vender os pedaços

O Problema do Corte de Hastes

- Ideia: Você tem uma haste de comprimento n e quer cortar em pedaços menores para vender os pedaços
- Tanto n quanto o tamanho dos pedaços são números inteiros

O Problema do Corte de Hastes

- Ideia: Você tem uma haste de comprimento n e quer cortar em pedaços menores para vender os pedaços
- Tanto n quanto o tamanho dos pedaços são números inteiros
- Pedaços de tamanhos diferentes tem valores (número real) diferentes

O Problema do Corte de Hastes

- Ideia: Você tem uma haste de comprimento n e quer cortar em pedaços menores para vender os pedaços
- Tanto n quanto o tamanho dos pedaços são números inteiros
- Pedaços de tamanhos diferentes tem valores (número real) diferentes

Instância: (p, n), onde p é um vetor de n valores reais

O Problema do Corte de Hastes

- Ideia: Você tem uma haste de comprimento n e quer cortar em pedaços menores para vender os pedaços
- Tanto n quanto o tamanho dos pedaços são números inteiros
- Pedaços de tamanhos diferentes tem valores (número real) diferentes

Instância: (p, n), onde p é um vetor de n valores reais

Resposta: lista inteiros $[i_1, ..., i_k]$ respeitando $n = i_1 + i_2 + ... + i_k$ tal que $r_n = p_{i_1} + p_{i_2} + ... + p_{i_k}$ seja máximo.

O Problema do Corte de Hastes

- Ideia: Você tem uma haste de comprimento n e quer cortar em pedaços menores para vender os pedaços
- Tanto n quanto o tamanho dos pedaços são números inteiros
- Pedaços de tamanhos diferentes tem valores (número real) diferentes

Instância: (p, n), onde p é um vetor de n valores reais

Resposta: lista inteiros $[i_1, ..., i_k]$ respeitando $n = i_1 + i_2 + ... + i_k$ tal que $r_n = p_{i_1} + p_{i_2} + ... + p_{i_k}$ seja máximo.

Na instância, cada v_i é o valor de venda de uma haste de comprimento i, $\quad i = 1, ..., n$.

length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

O Problema do Corte de Hastes

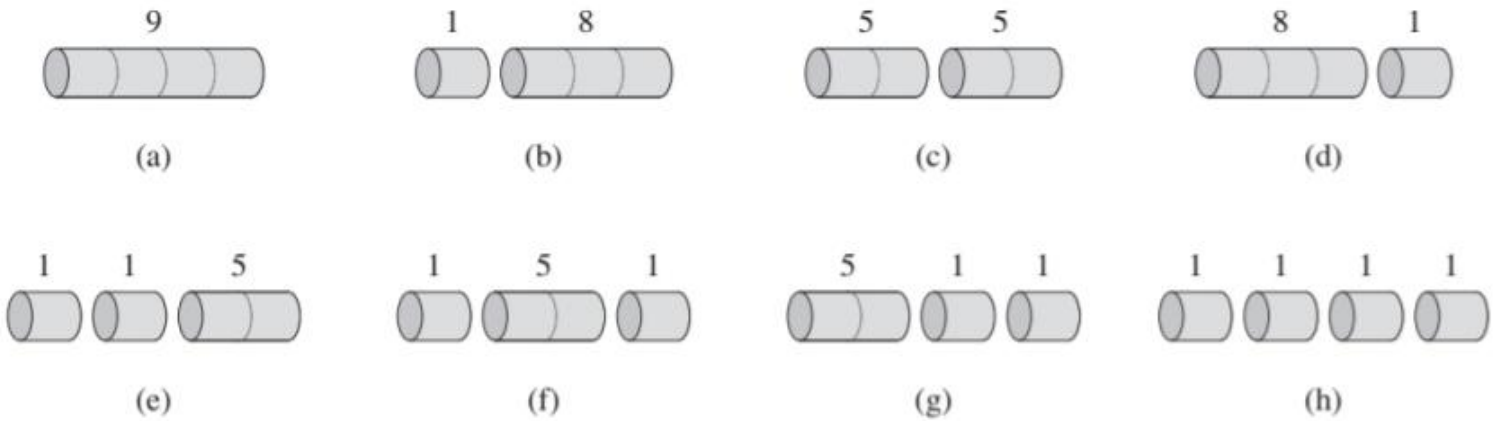
length i	1	2	3	4
price p_i	1	5	8	9

Considere o caso $n = 4$

O Problema do Corte de Hastes

length i	1	2	3	4
price p_i	1	5	8	9

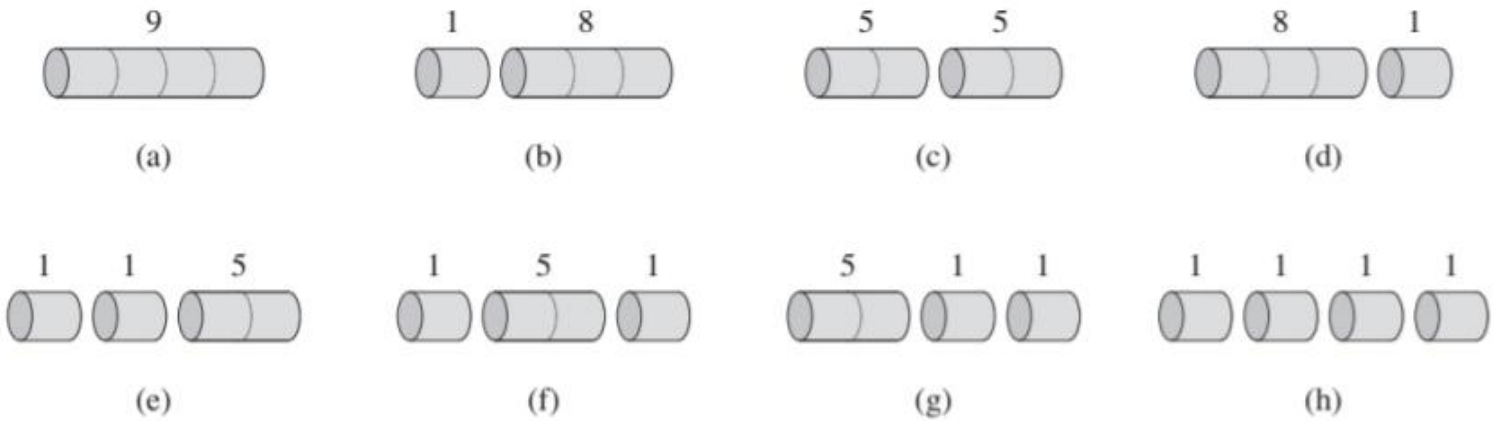
Considere o caso $n = 4$



O Problema do Corte de Hastes

length i	1	2	3	4
price p_i	1	5	8	9

Considere o caso $n = 4$

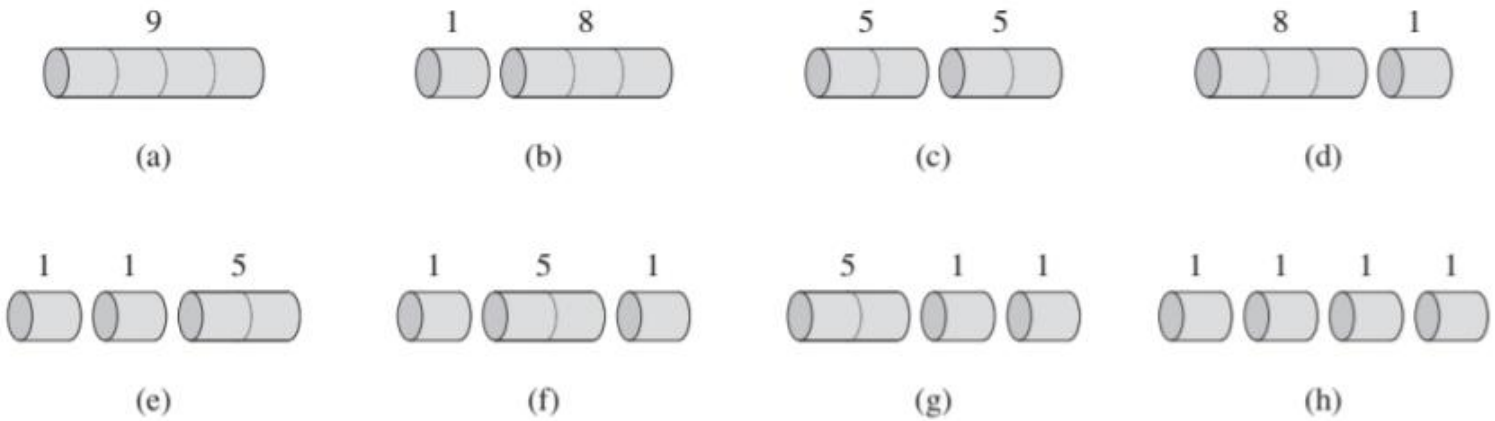


- Note que uma haste de comprimento n pode ser cortada em $n - 1$ posições

O Problema do Corte de Hastes

length i	1	2	3	4
price p_i	1	5	8	9

Considere o caso $n = 4$

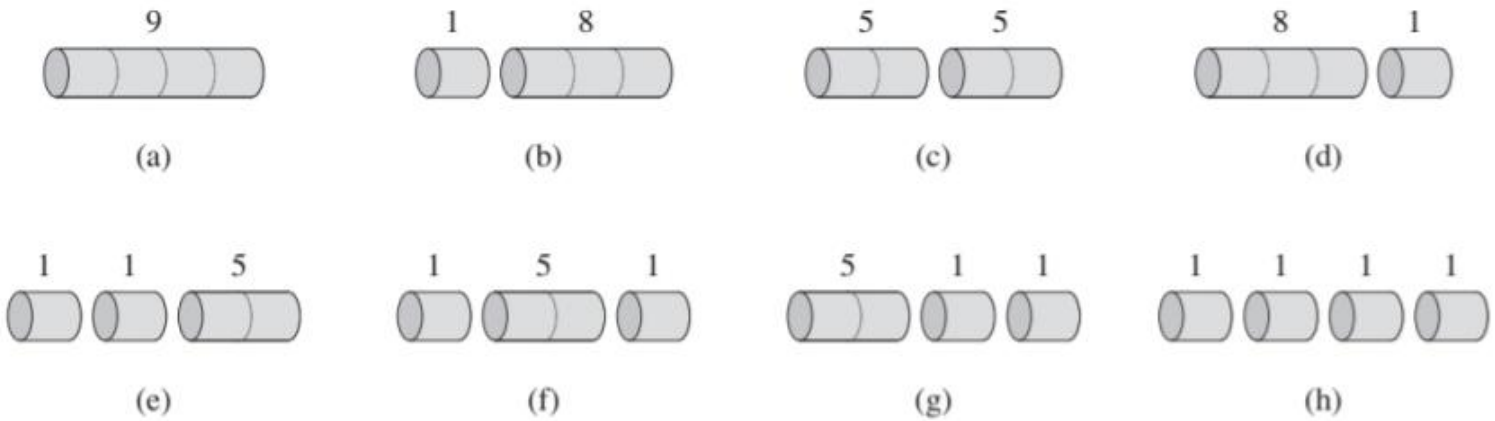


- Note que uma haste de comprimento n pode ser cortada em $n - 1$ posições
- Portanto há 2^{n-1} maneiras de se cortar a barra

O Problema do Corte de Hastes

length i	1	2	3	4
price p_i	1	5	8	9

Considere o caso $n = 4$



- Note que uma haste de comprimento n pode ser cortada em $n - 1$ posições
- Portanto há 2^{n-1} maneiras de se cortar a barra
- Mesmo que consideremos (e), (f) e (g) como o "mesmo corte", o número de cortes ainda cresce exponencialmente com n

O Problema do Corte de Hastes

length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

O Problema do Corte de Hastes

length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

Considere os casos $n = 1, 2, \ldots, 10$

O Problema do Corte de Hastes

length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

Considere os casos $n = 1, 2, \ldots, 10$

- $r_1 = 1$ (nenhum corte)

O Problema do Corte de Hastes

length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

Considere os casos $n = 1, 2, \ldots, 10$

- $r_1 = 1$ (nenhum corte)
- $r_2 = 5$ (nenhum corte)

O Problema do Corte de Hastes

length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

Considere os casos $n = 1, 2, \ldots, 10$

- $r_1 = 1$ (nenhum corte)
- $r_2 = 5$ (nenhum corte)
- $r_3 = 3$ (nenhum corte)

O Problema do Corte de Hastes

length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

Considere os casos $n = 1, 2, \ldots, 10$

- $r_1 = 1$ (nenhum corte)
- $r_2 = 5$ (nenhum corte)
- $r_3 = 3$ (nenhum corte)
- $r_4 = 10$ (solução $4 = 2 + 2$)

O Problema do Corte de Hastes

length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

Considere os casos $n = 1, 2, \ldots, 10$

- $r_1 = 1$ (nenhum corte)
- $r_2 = 5$ (nenhum corte)
- $r_3 = 3$ (nenhum corte)
- $r_4 = 10$ (solução $4 = 2 + 2$)
- $r_5 = 13$ (solução $5 = 2 + 3$)

O Problema do Corte de Hastes

length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

Considere os casos $n = 1, 2, \ldots, 10$

- $r_1 = 1$ (nenhum corte)
- $r_2 = 5$ (nenhum corte)
- $r_3 = 3$ (nenhum corte)
- $r_4 = 10$ (solução $4 = 2 + 2$)
- $r_5 = 13$ (solução $5 = 2 + 3$)
- $r_6 = 17$ (nenhum corte)

O Problema do Corte de Hastes

length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

Considere os casos $n = 1, 2, \ldots, 10$

- $r_1 = 1$ (nenhum corte)
- $r_2 = 5$ (nenhum corte)
- $r_3 = 3$ (nenhum corte)
- $r_4 = 10$ (solução $4 = 2 + 2$)
- $r_5 = 13$ (solução $5 = 2 + 3$)
- $r_6 = 17$ (nenhum corte)
- $r_7 = 18$ (solução $7 = 1 + 6$ ou $7 = 2 + 2 + 3$)

O Problema do Corte de Hastes

length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

Considere os casos $n = 1, 2, \ldots, 10$

- $r_1 = 1$ (nenhum corte)
- $r_2 = 5$ (nenhum corte)
- $r_3 = 3$ (nenhum corte)
- $r_4 = 10$ (solução $4 = 2 + 2$)
- $r_5 = 13$ (solução $5 = 2 + 3$)
- $r_6 = 17$ (nenhum corte)
- $r_7 = 18$ (solução $7 = 1 + 6$ ou $7 = 2 + 2 + 3$)
- $r_8 = 22$ (solução $8 = 2 + 6$)

O Problema do Corte de Hastes

length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

Considere os casos $n = 1, 2, \ldots, 10$

- $r_1 = 1$ (nenhum corte)
- $r_2 = 5$ (nenhum corte)
- $r_3 = 3$ (nenhum corte)
- $r_4 = 10$ (solução $4 = 2 + 2$)
- $r_5 = 13$ (solução $5 = 2 + 3$)
- $r_6 = 17$ (nenhum corte)
- $r_7 = 18$ (solução $7 = 1 + 6$ ou $7 = 2 + 2 + 3$)
- $r_8 = 22$ (solução $8 = 2 + 6$)
- $r_9 = 25$ (solução $9 = 3 + 6$)

O Problema do Corte de Hastes

length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

Considere os casos $n = 1, 2, \ldots, 10$

- $r_1 = 1$ (nenhum corte)
- $r_2 = 5$ (nenhum corte)
- $r_3 = 3$ (nenhum corte)
- $r_4 = 10$ (solução $4 = 2 + 2$)
- $r_5 = 13$ (solução $5 = 2 + 3$)
- $r_6 = 17$ (nenhum corte)
- $r_7 = 18$ (solução $7 = 1 + 6$ ou $7 = 2 + 2 + 3$)
- $r_8 = 22$ (solução $8 = 2 + 6$)
- $r_9 = 25$ (solução $9 = 3 + 6$)
- $r_{10} = 30$ (nenhum corte)

O Problema do Corte de Hastes

length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

Considere os casos $n = 1, 2, \ldots, 10$

- $r_1 = 1$ (nenhum corte)
- $r_2 = 5$ (nenhum corte)
- $r_3 = 3$ (nenhum corte)
- $r_4 = 10$ (solução $4 = 2 + 2$)
- $r_5 = 13$ (solução $5 = 2 + 3$)
- $r_6 = 17$ (nenhum corte)
- $r_7 = 18$ (solução $7 = 1 + 6$ ou $7 = 2 + 2 + 3$)
- $r_8 = 22$ (solução $8 = 2 + 6$)
- $r_9 = 25$ (solução $9 = 3 + 6$)
- $r_{10} = 30$ (nenhum corte)

Importante:

O Problema do Corte de Hastes

length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

Considere os casos $n = 1, 2, \ldots, 10$

- $r_1 = 1$ (nenhum corte)
- $r_2 = 5$ (nenhum corte)
- $r_3 = 3$ (nenhum corte)
- $r_4 = 10$ (solução $4 = 2 + 2$)
- $r_5 = 13$ (solução $5 = 2 + 3$)
- $r_6 = 17$ (nenhum corte)
- $r_7 = 18$ (solução $7 = 1 + 6$ ou $7 = 2 + 2 + 3$)
- $r_8 = 22$ (solução $8 = 2 + 6$)
- $r_9 = 25$ (solução $9 = 3 + 6$)
- $r_{10} = 30$ (nenhum corte)

Importante: Note que $r_n = \max(p_n,\ r_1 + r_{n-1},\ r_2 + r_{n-2},\ \ldots,\ r_{n-1} + r_1)$

O Problema do Corte de Hastes - Solução Recursiva

O Problema do Corte de Hastes - Solução Recursiva

Instância: p, n, onde p é um vetor de n valores reais

Resposta: lista inteiros $[i_1, ..., i_k]$ respeitando $n = i_1 + i_2 + ... + i_k$ tal que $r_n = p_{i_1} + p_{i_2} + ... + p_{i_k}$ seja máximo.

O Problema do Corte de Hastes - Solução Recursiva

Instância: p, n, onde p é um vetor de n valores reais

Resposta: lista inteiros $[i_1, ..., i_k]$ respeitando $n = i_1 + i_2 + ... + i_k$ tal que $r_n = p_{i_1} + p_{i_2} + ... + p_{i_k}$ seja máximo.

Reescrevendo r_n:

O Problema do Corte de Hastes - Solução Recursiva

Instância: p, n, onde p é um vetor de n valores reais

Resposta: lista inteiros $[i_1, ..., i_k]$ respeitando $n = i_1 + i_2 + ... + i_k$ tal que $r_n = p_{i_1} + p_{i_2} + ... + p_{i_k}$ seja máximo.

Reescrevendo r_n:

$$r_n = \max_{1 \leq i \leq n} (p_i + r_{n-i})$$

O Problema do Corte de Hastes - Solução Recursiva

Instância: p, n, onde p é um vetor de n valores reais

Resposta: lista inteiros $[i_1, ..., i_k]$ respeitando $n = i_1 + i_2 + ... + i_k$ tal que $r_n = p_{i_1} + p_{i_2} + ... + p_{i_k}$ seja máximo.

Reescrevendo r_n:

$$r_n = \max_{1 \leq i \leq n} (p_i + r_{n-i}) \qquad \text{onde } r_0 = 0$$

O Problema do Corte de Hastes - Solução Recursiva

Instância: p, n, onde p é um vetor de n valores reais

Resposta: lista inteiros $[i_1, ..., i_k]$ respeitando $n = i_1 + i_2 + ... + i_k$ tal que $r_n = p_{i_1} + p_{i_2} + ... + p_{i_k}$ seja máximo.

Reescrevendo r_n:

$$r_n = \max_{1 \leq i \leq n} (p_i + r_{n-i}) \qquad \text{onde } r_0 = 0$$

```
CUT-ROD(p, n)
1  if n == 0
2       return 0
3  q = -∞
4  for i = 1 to n
5       q = max(q, p[i] + CUT-ROD(p, n - i))
6  return q
```

O Problema do Corte de Hastes - Solução Recursiva

Instância: p, n, onde p é um vetor de n valores reais

Resposta: lista inteiros $[i_1, ..., i_k]$ respeitando $n = i_1 + i_2 + ... + i_k$ tal que $r_n = p_{i_1} + p_{i_2} + ... + p_{i_k}$ seja máximo.

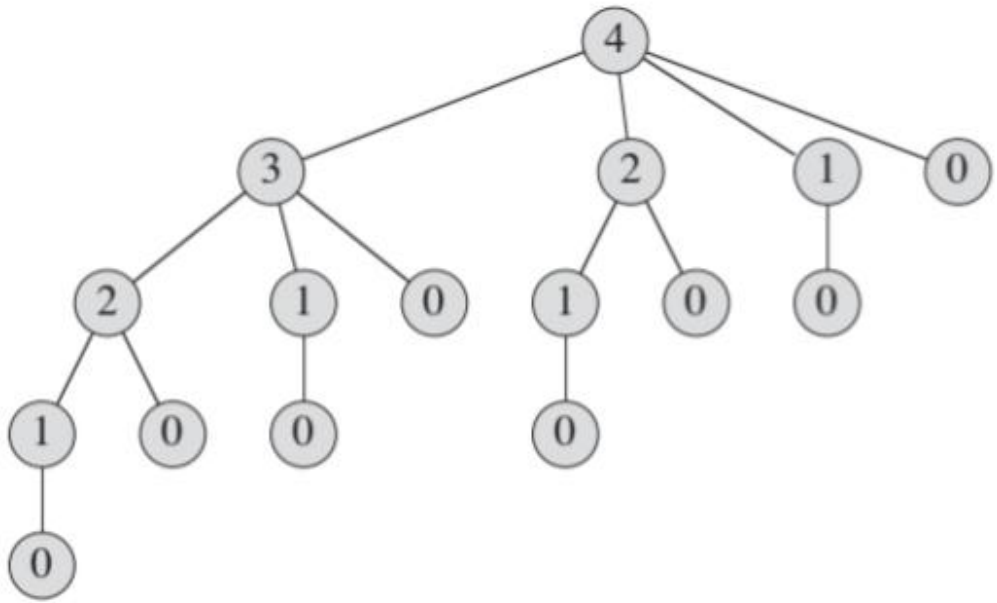
Reescrevendo r_n:

$$r_n = \max_{1 \leq i \leq n} (p_i + r_{n-i}) \qquad \text{onde } r_0 = 0$$

```
CUT-ROD(p, n)
1  if n == 0
2       return 0
3  q = -∞
4  for i = 1 to n
5       q = max(q, p[i] + CUT-ROD(p, n - i))
6  return q
```

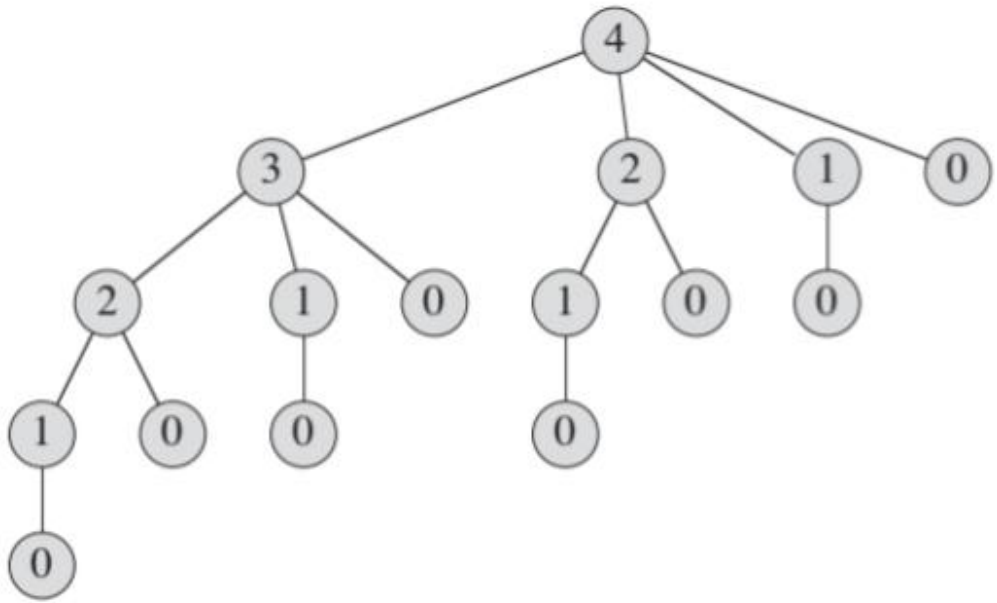
length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

Solução Recursiva



Árvore de recursões para $n = 4$.

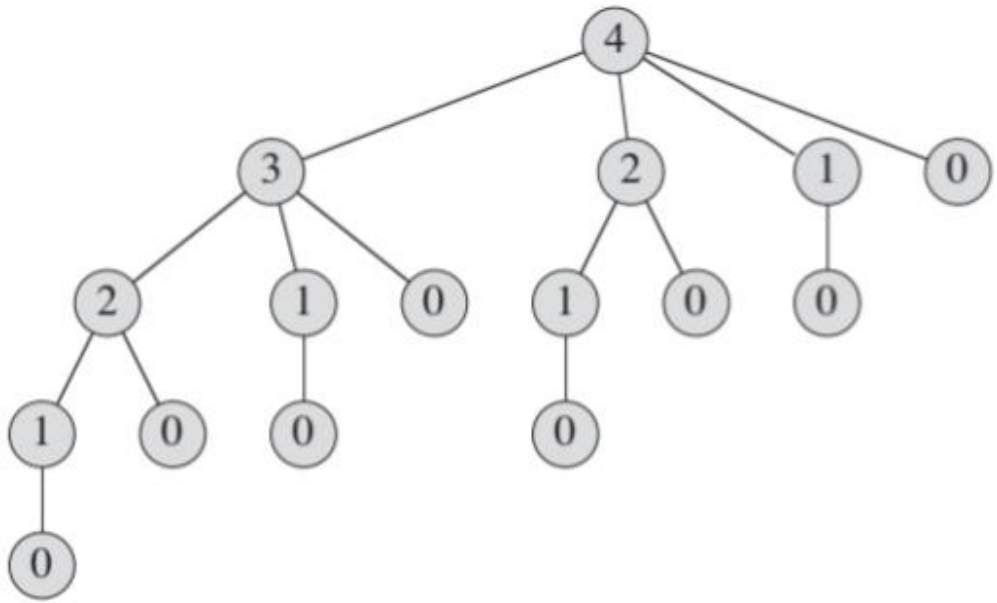
Solução Recursiva



Árvore de recursões para $n = 4$.

- A árvore de recursões tem 2^n nós

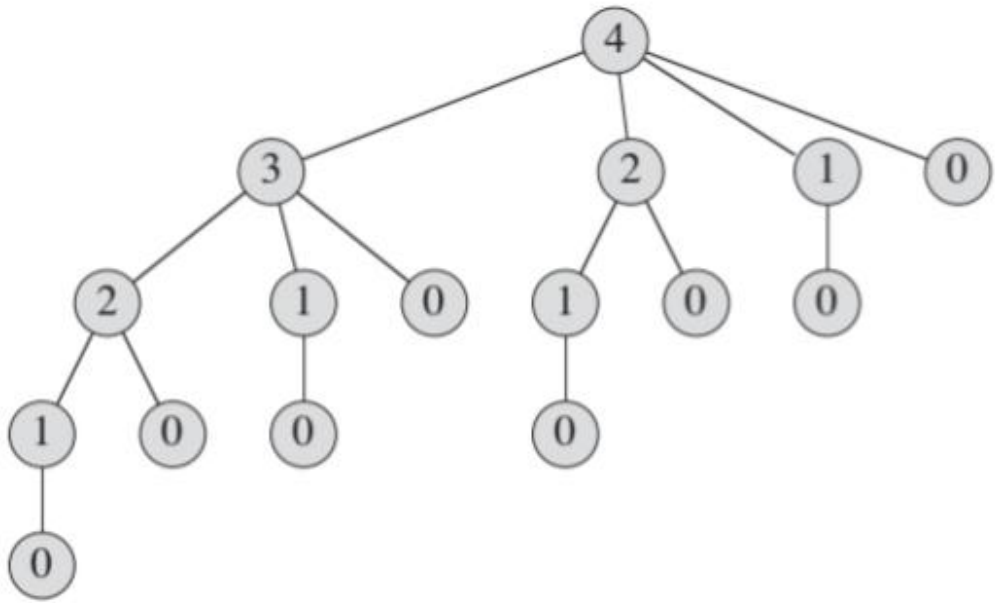
Solução Recursiva



Árvore de recursões para $n = 4$.

- A árvore de recursões tem 2^n nós
- Cada nó i corresponde à execução com entrada de tamanho $n - i$

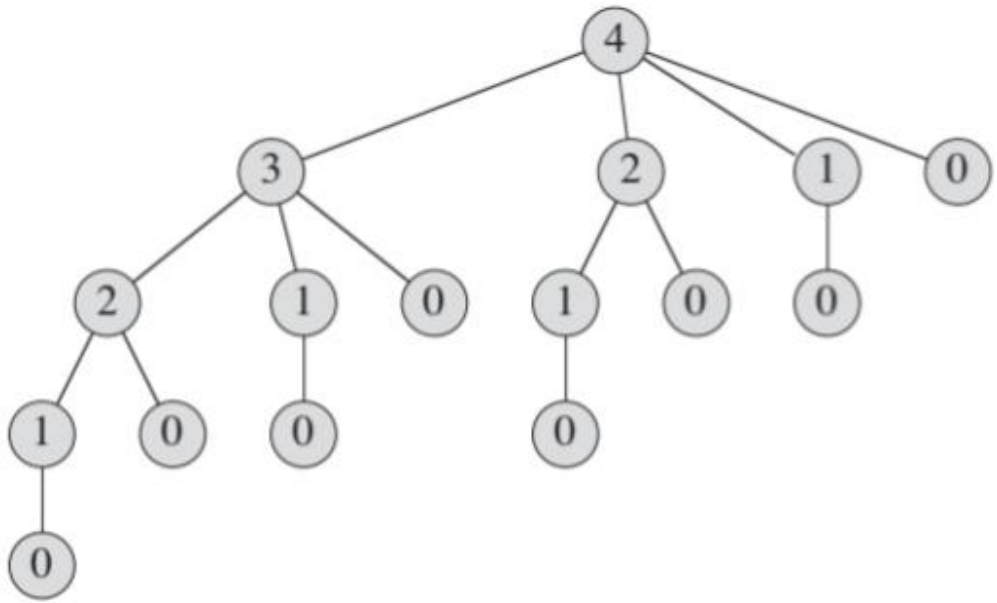
Solução Recursiva



Árvore de recursões para $n = 4$.

- A árvore de recursões tem 2^n nós
- Cada nó i corresponde à execução com entrada de tamanho $n - i$ $i = 1, ..., n$

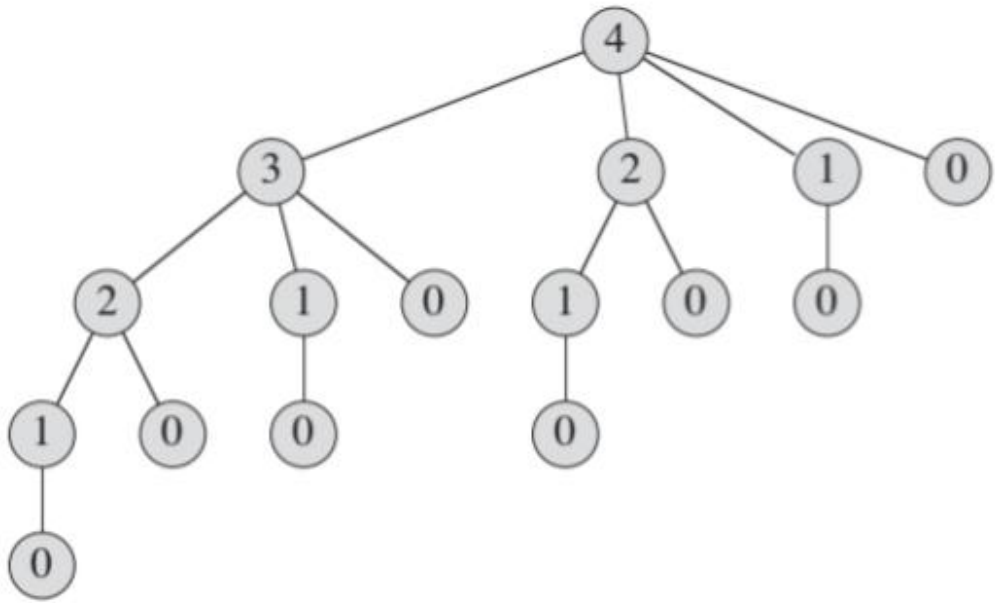
Solução Recursiva



Árvore de recursões para $n = 4$.

- A árvore de recursões tem 2^n nós
- Cada nó i corresponde à execução com entrada de tamanho $n - i$ $i = 1, ..., n$
- Seja $k = n - i$.

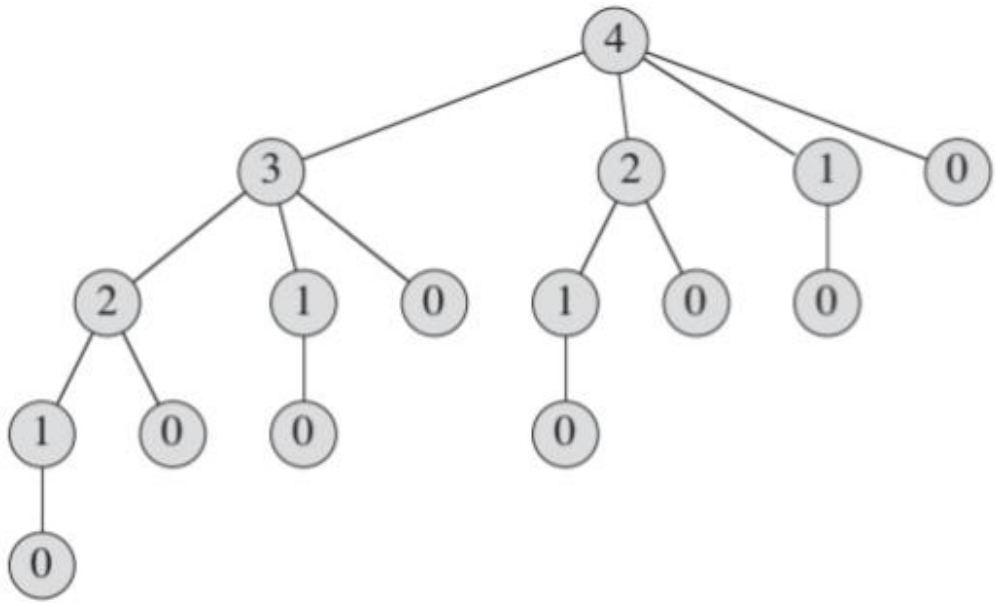
Solução Recursiva



Árvore de recursões para $n = 4$.

- A árvore de recursões tem 2^n nós
- Cada nó i corresponde à execução com entrada de tamanho $n - i$ $i = 1, ..., n$
- Seja $k = n - i$. O problema é resolvido repetidas vezes para cada um dos subproblemas de tamanho k, $0 \leq k < 4$

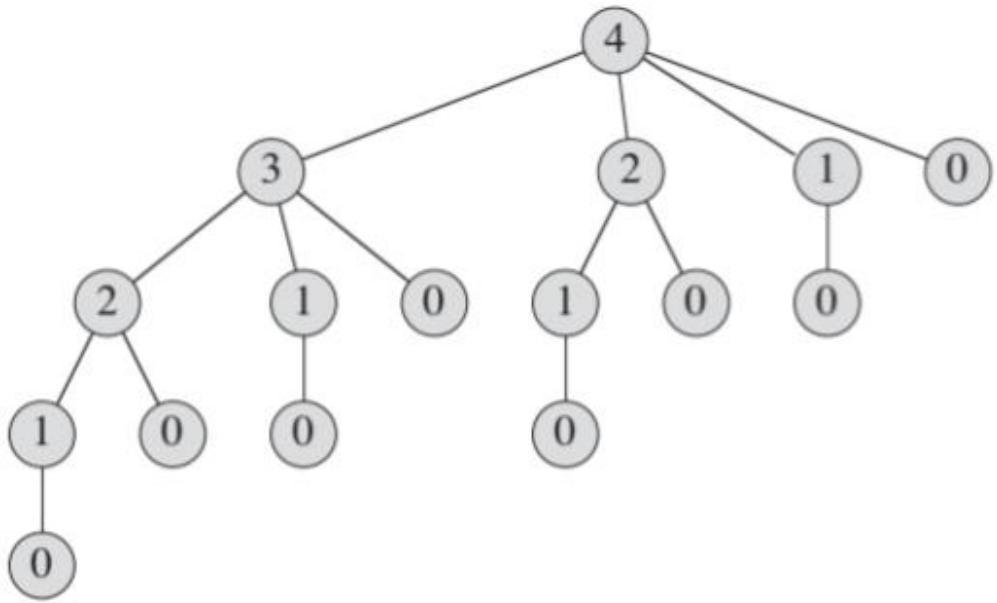
Solução Recursiva



Árvore de recursões para $n = 4$.

- A árvore de recursões tem 2^n nós
- Cada nó i corresponde à execução com entrada de tamanho $n - i$ $i = 1, ..., n$
- Seja $k = n - i$. O problema é resolvido repetidas vezes para cada um dos subproblemas de tamanho k, $0 \leq k < 4$
- Ideia: Só usar recursão **uma única vez** para cada subproblema de tamanho k.

Solução Recursiva



Árvore de recursões para $n = 4$.

- A árvore de recursões tem 2^n nós
- Cada nó i corresponde à execução com entrada de tamanho $n - i$ $i = 1, ..., n$
- Seja $k = n - i$. O problema é resolvido repetidas vezes para cada um dos subproblemas de tamanho k, $0 \leq k < 4$
- Ideia: Só usar recursão **uma única vez** para cada subproblema de tamanho k.
- Importante: Se a quantidade de problemas diferentes for **polinomial**, temos um algoritmo polinomial!

Programação Dinâmica usando Memorização

Ideia chave: Usar vetor para guardar soluções de subproblemas já computados

```
MEMOIZED-CUT-ROD(p, n)
1  let r[0..n] be a new array
2  for i = 0 to n
3      r[i] = -∞
4  return MEMOIZED-CUT-ROD-AUX(p, n, r)
```

```
MEMOIZED-CUT-ROD-AUX(p, n, r)
1  if r[n] ≥ 0
2      return r[n]
3  if n == 0
4      q = 0
5  else q = -∞
6      for i = 1 to n
7          q = max(q, p[i] + MEMOIZED-CUT-ROD-AUX(p, n - i, r))
8  r[n] = q
9  return q
```

No código acima, o vetor r guarda as soluções já computadas

Programação Dinâmica "Bottom-up'

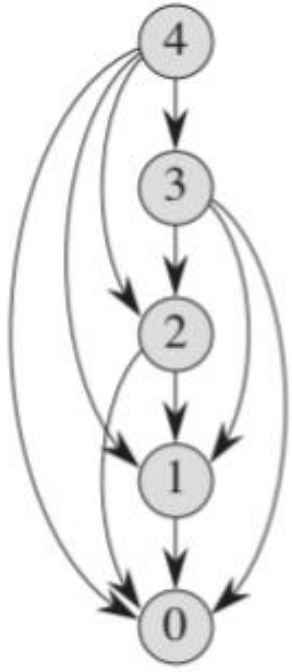
Versão "Bottom-up":

```
BOTTOM-UP-CUT-ROD(p, n)
1  let r[0..n] be a new array
2  r[0] = 0
3  for j = 1 to n
4       q = -∞
5       for i = 1 to j
6            q = max(q, p[i] + r[j - i])
7       r[j] = q
8  return r[n]
```

Ideia: Começar do menor subproblema para o maior subproblema

Programação Dinâmica "Bottom-up'

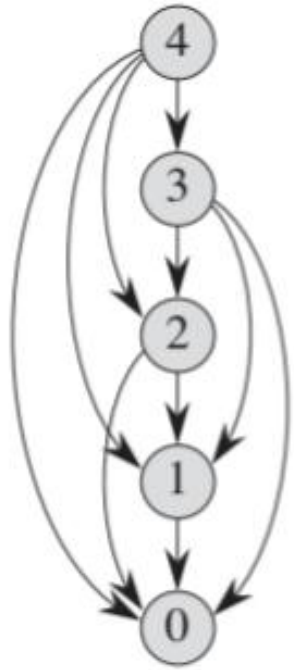
Grafo de subproblemas:



- Cada nó é um subproblema (assim como na árvore de recursões)

Programação Dinâmica "Bottom-up'

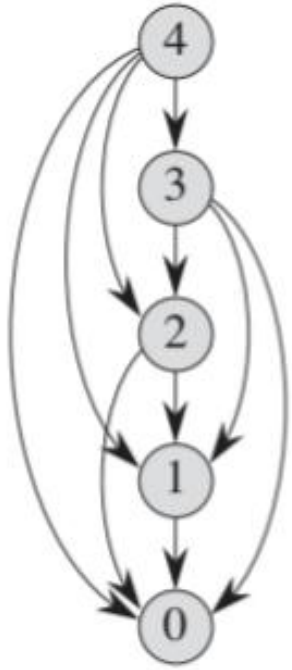
Grafo de subproblemas:



- Cada nó é um subproblema (assim como na árvore de recursões)
- Arco (x, y) indica que suproblema x depende da resolução do subproblema y

Programação Dinâmica "Bottom-up'

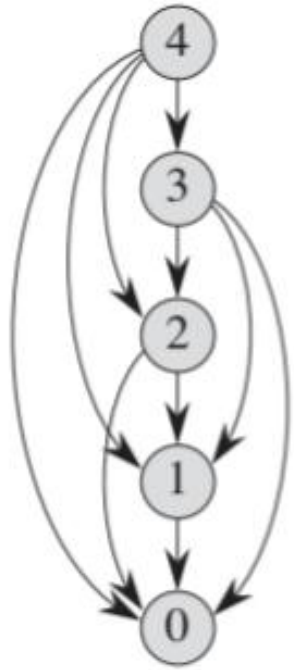
Grafo de subproblemas:



- Cada nó é um subproblema (assim como na árvore de recursões)
- Arco (x, y) indica que suproblema x depende da resolução do subproblema y
- Este grafo pode ser visto como uma versão da árvore de recursões em que "colapsamos" nós idênticos.

Programação Dinâmica "Bottom-up'

Grafo de subproblemas:



- Cada nó é um subproblema (assim como na árvore de recursões)
- Arco (x, y) indica que suproblema x depende da resolução do subproblema y
- Este grafo pode ser visto como uma versão da árvore de recursões em que "colapsamos" nós idênticos.
- Tipicamente a complexidade de algoritmos de PD é linear no tamanho deste grafo, ou seja, $\Theta(|V| + |E|)$

Programação Dinâmica "Bottom-up'

Guardando as soluções (não apenas o valor ótimo):

```
EXTENDED-BOTTOM-UP-CUT-ROD(p, n)
 1  let r[0..n] and s[0..n] be new arrays
 2  r[0] = 0
 3  for j = 1 to n
 4      q = -∞
 5      for i = 1 to j
 6          if q < p[i] + r[j - i]
 7              q = p[i] + r[j - i]
 8              s[j] = i
 9      r[j] = q
10  return r and s
```

```
PRINT-CUT-ROD-SOLUTION(p, n)
1  (r, s) = EXTENDED-BOTTOM-UP-CUT-ROD(p, n)
2  while n > 0
3      print s[n]
4      n = n - s[n]
```

Programação Dinâmica "Bottom-up'

Guardando as soluções (não apenas o valor ótimo):

```
EXTENDED-BOTTOM-UP-CUT-ROD(p, n)
 1  let r[0..n] and s[0..n] be new arrays
 2  r[0] = 0
 3  for j = 1 to n
 4      q = -∞
 5      for i = 1 to j
 6          if q < p[i] + r[j - i]
 7              q = p[i] + r[j - i]
 8              s[j] = i
 9      r[j] = q
10  return r and s
```

```
PRINT-CUT-ROD-SOLUTION(p, n)
1  (r, s) = EXTENDED-BOTTOM-UP-CUT-ROD(p, n)
2  while n > 0
3      print s[n]
4      n = n - s[n]
```

i	0	1	2	3	4	5	6	7	8	9	10
$r[i]$	0	1	5	8	10	13	17	18	22	25	30
$s[i]$	0	1	2	3	2	2	6	1	2	3	10

PD - esboço de algoritmos para outros exemplos

Caminhos mínimo em grafos direcionados acíclicos (ponderados):

PD - esboço de algoritmos para outros exemplos

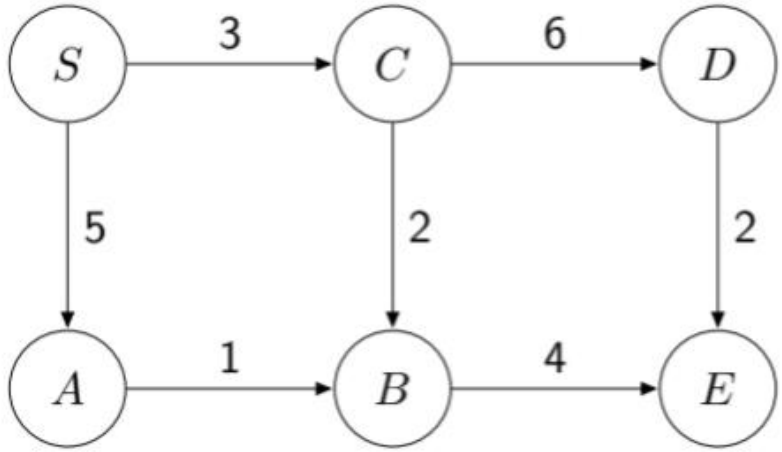
Caminhos mínimo em grafos direcionados acíclicos (ponderados):

- Dado $v \in V(G)$, encontrar caminho mínimo de v para cada outro vértice de G

PD - esboço de algoritmos para outros exemplos

Caminhos mínimo em grafos direcionados acíclicos (ponderados):

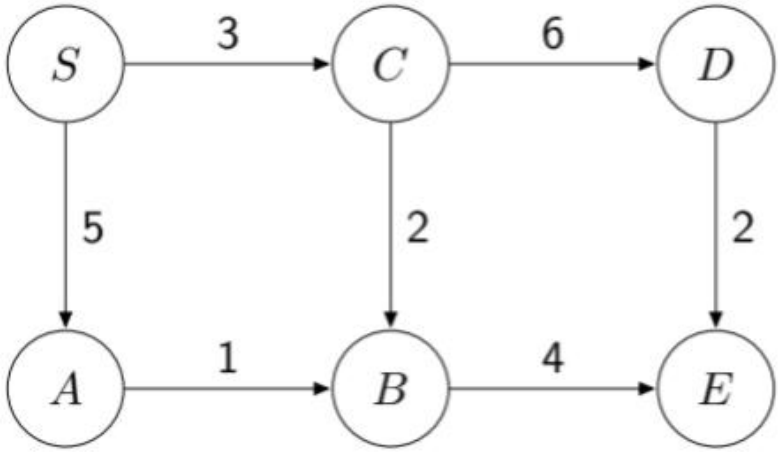
- Dado $v \in V(G)$, encontrar caminho mínimo de v para cada outro vértice de G

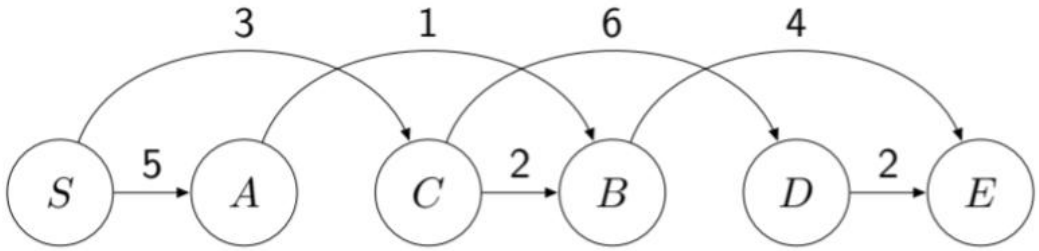


PD - esboço de algoritmos para outros exemplos

Caminhos mínimo em grafos direcionados acíclicos (ponderados):

- Dado $v \in V(G)$, encontrar caminho mínimo de v para cada outro vértice de G





PD - esboço de algoritmos para outros exemplos

Caminhos mínimo em grafos direcionados acíclicos (ponderados):

PD - esboço de algoritmos para outros exemplos

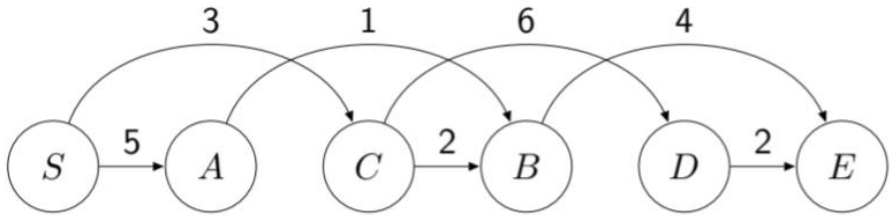
Caminhos mínimo em grafos direcionados acíclicos (ponderados):

- Dado $v \in V(G)$, encontrar caminho mínimo de v para cada outro vértice de G

PD - esboço de algoritmos para outros exemplos

Caminhos mínimo em grafos direcionados acíclicos (ponderados):

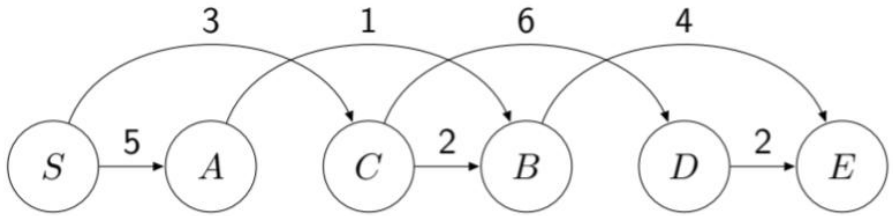
- Dado $v \in V(G)$, encontrar caminho mínimo de v para cada outro vértice de G



PD - esboço de algoritmos para outros exemplos

Caminhos mínimo em grafos direcionados acíclicos (ponderados):

- Dado $v \in V(G)$, encontrar caminho mínimo de v para cada outro vértice de G



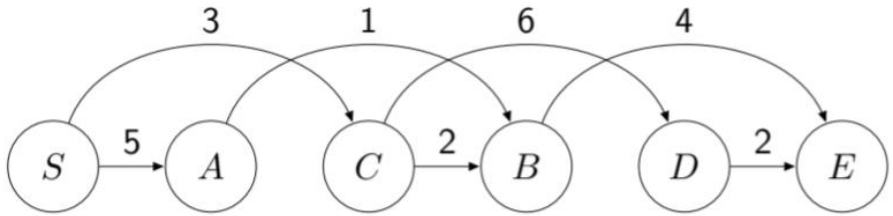
ShortestPathDAG (G, l, s)

1: $\mathrm{d}[s] = 0$
2: **for all** $v \in V(G)$ em ordem topológica **do**
3: $\quad \mathrm{d}[v] = \min_{u \in pred(v)} \{d(u) + l(u, v)\}$
4: **end for**

PD - esboço de algoritmos para outros exemplos

Caminhos mínimo em grafos direcionados acíclicos (ponderados):

- Dado $v \in V(G)$, encontrar caminho mínimo de v para cada outro vértice de G



ShortestPathDAG (G, l, s)

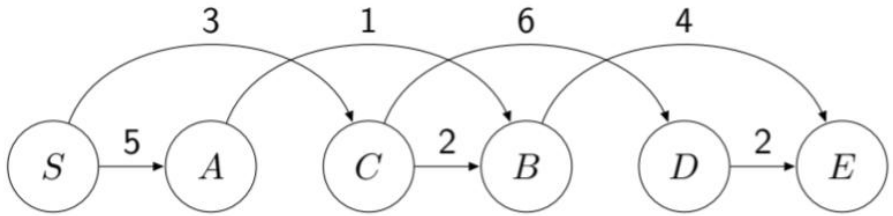
1: $\mathrm{d}[s] = 0$
2: **for all** $v \in V(G)$ em ordem topológica **do**
3: $\quad \mathrm{d}[v] = \min_{u \in pred(v)} \{d(u) + l(u, v)\}$
4: **end for**

- O algoritmo usa S como vértice inicial

PD - esboço de algoritmos para outros exemplos

Caminhos mínimo em grafos direcionados acíclicos (ponderados):

- Dado $v \in V(G)$, encontrar caminho mínimo de v para cada outro vértice de G



ShortestPathDAG (G, l, s)

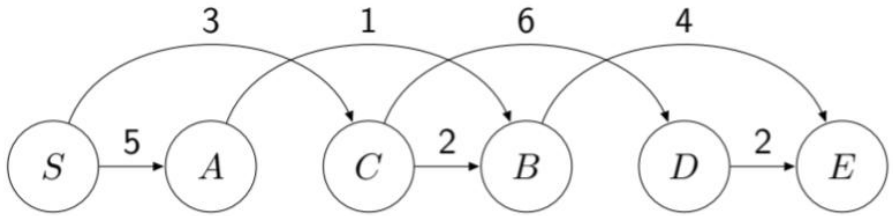
1: $\mathrm{d}[s] = 0$
2: **for all** $v \in V(G)$ em ordem topológica **do**
3: $\quad \mathrm{d}[v] = \min_{u \in pred(v)} \{d(u) + l(u, v)\}$
4: **end for**

- O algoritmo usa S como vértice inicial
- E se quisermos que o vértice inicial seja outro vértice qualquer?

PD - esboço de algoritmos para outros exemplos

Caminhos mínimo em grafos direcionados acíclicos (ponderados):

- Dado $v \in V(G)$, encontrar caminho mínimo de v para cada outro vértice de G



ShortestPathDAG (G, l, s)

1: $\mathrm{d}[s] = 0$
2: **for all** $v \in V(G)$ em ordem topológica **do**
3: $\quad \mathrm{d}[v] = \min_{u \in pred(v)} \{d(u) + l(u, v)\}$
4: **end for**

- O algoritmo usa S como vértice inicial
- E se quisermos que o vértice inicial seja outro vértice qualquer?
- E se quisermos os caminhos máximos?

PD - esboço de algoritmos para outros exemplos

Subsequência crescente máxima:

PD - esboço de algoritmos para outros exemplos

Subsequência crescente máxima:

PD - esboço de algoritmos para outros exemplos

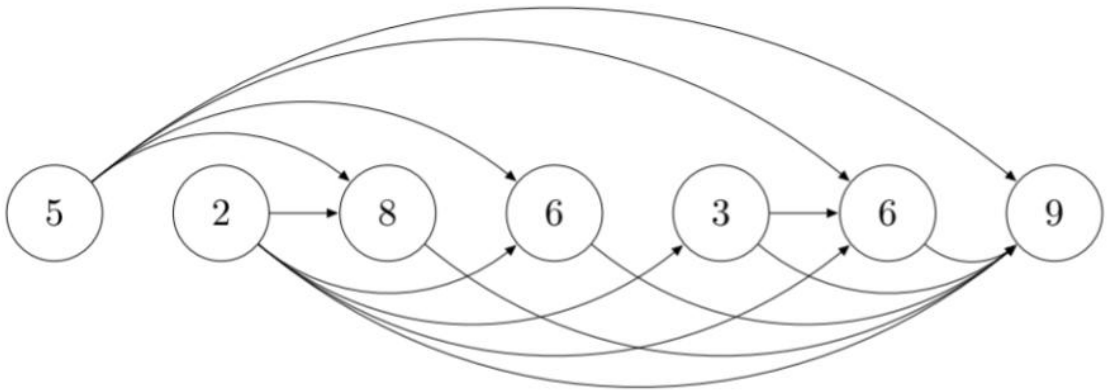
Subsequência crescente máxima:

Solução: 2, 3, 6, 9

PD - esboço de algoritmos para outros exemplos

Subsequência crescente máxima:

Solução: 2, 3, 6, 9

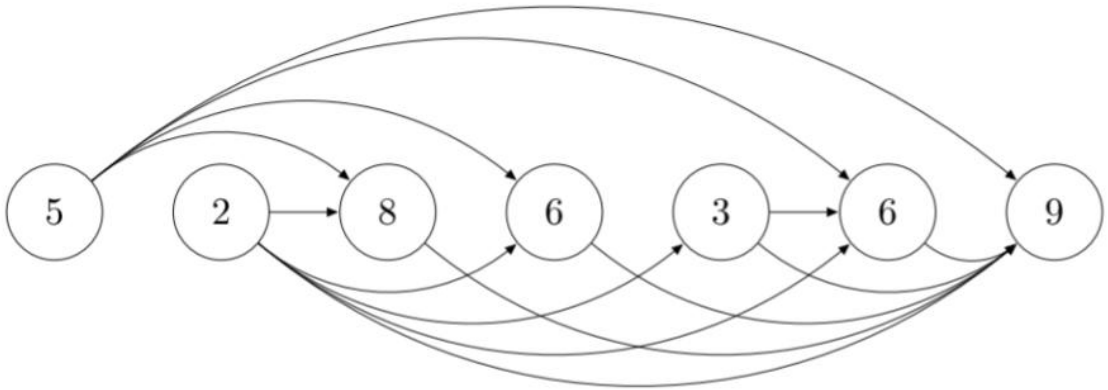


PD - esboço de algoritmos para outros exemplos

Subsequência crescente máxima:

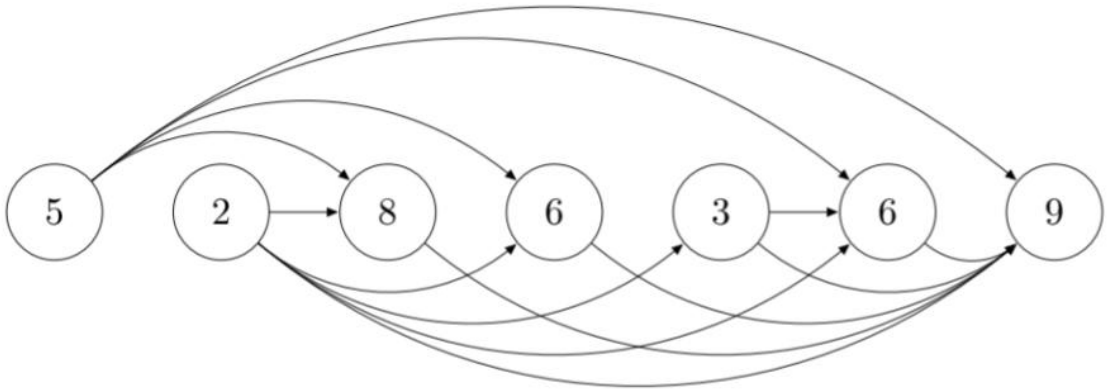
PD - esboço de algoritmos para outros exemplos

Subsequência crescente máxima:



PD - esboço de algoritmos para outros exemplos

Subsequência crescente máxima:



LongestSubseq (G, L)

1: $\mathsf{L}[s] = 1$ /* s é o primeiro vértice da ordenação */
2: **for all** $v \in V(G) \setminus \{s\}$ na ordem dada (topológica) **do**
3: $\quad \mathsf{L}[v] = \max_{u \in pred(v)} \{L(u) + 1\}$
4: **end for**

PD - esboço de algoritmos para outros exemplos

Distância de Edição

PD - esboço de algoritmos para outros exemplos

Distância de Edição (também chamado de "alinhamento de sequências"):

PD - esboço de algoritmos para outros exemplos

Distância de Edição (também chamado de "alinhamento de sequências"):

Qual a "distância de edição" entre as as strings SILVA e SILVER?

PD - esboço de algoritmos para outros exemplos

Distância de Edição (também chamado de "alinhamento de sequências"):

Qual a "distância de edição" entre as as strings SILVA e SILVER?
Resp: Distância 2.

PD - esboço de algoritmos para outros exemplos

Distância de Edição (também chamado de "alinhamento de sequências"):

Qual a "distância de edição" entre as as strings SILVA e SILVER?
Resp: Distância 2.

Na segunda string faça:

PD - esboço de algoritmos para outros exemplos

Distância de Edição (também chamado de "alinhamento de sequências"):

Qual a "distância de edição" entre as as strings SILVA e SILVER?
Resp: Distância 2.

Na segunda string faça:

- Remova R
- Troque E por A.

PD - esboço de algoritmos para outros exemplos

Distância de Edição (também chamado de "alinhamento de sequências"):

Qual a "distância de edição" entre as as strings SILVA e SILVER?
Resp: Distância 2.

Na segunda string faça:

- Remova R
- Troque E por A.

Ou na primeira string faça:

PD - esboço de algoritmos para outros exemplos

Distância de Edição (também chamado de "alinhamento de sequências"):

Qual a "distância de edição" entre as as strings SILVA e SILVER?
Resp: Distância 2.

Na segunda string faça:

- Remova R
- Troque E por A.

Ou na primeira string faça:

- Troque A por E.
- Insira R

PD - esboço de algoritmos para outros exemplos

Distância de Edição (também chamado de "alinhamento de sequências"):

Qual a "distância de edição" entre as as strings SILVA e SILVER?
Resp: Distância 2.

Na segunda string faça:

- Remova R
- Troque E por A.

Ou na primeira string faça:

- Troque A por E.
- Insira R

Operações: remover, inserir, editar (trocar)

PD - esboço de algoritmos para outros exemplos

Distância de Edição

PD - esboço de algoritmos para outros exemplos

Distância de Edição (também chamado de "alinhamento de sequências"):

PD - esboço de algoritmos para outros exemplos

Distância de Edição (também chamado de "alinhamento de sequências"):

Qual a "distância de edição" entre as as strings ACCTG e CACGTG?

PD - esboço de algoritmos para outros exemplos

Distância de Edição (também chamado de "alinhamento de sequências"):

Qual a "distância de edição" entre as as strings ACCTG e CACGTG?

Resp: Distância 2.

PD - esboço de algoritmos para outros exemplos

Distância de Edição (também chamado de "alinhamento de sequências"):

Qual a "distância de edição" entre as as strings ACCTG e CACGTG?

Resp: Distância 2.

PD - esboço de algoritmos para outros exemplos

Distância de Edição

PD - esboço de algoritmos para outros exemplos

Distância de Edição (também chamado de "alinhamento de sequências"):

PD - esboço de algoritmos para outros exemplos

Distância de Edição (também chamado de "alinhamento de sequências"):

	␣	A	C	C	T	G
␣						
C						
A						
C						
G						
T						
G						

PD - esboço de algoritmos para outros exemplos

Distância de Edição

PD - esboço de algoritmos para outros exemplos

Distância de Edição (também chamado de "alinhamento de sequências"):

PD - esboço de algoritmos para outros exemplos

Distância de Edição (também chamado de "alinhamento de sequências"):

	␣	A	C	C	T	G
␣	0	1	2	3	4	5
C	1	1	1	2	3	4
A	2	1	2	2	3	4
C	3	2	1	2	3	4
G	4	3	2	2	3	3
T	5	4	3	3	2	3
G	6	5	4	4	3	2

PD - esboço de algoritmos para outros exemplos

Distância de Edição

PD - esboço de algoritmos para outros exemplos

Distância de Edição (também chamado de "alinhamento de sequências"):

PD - esboço de algoritmos para outros exemplos

Distância de Edição (também chamado de "alinhamento de sequências"):

	⊔	A	C	C	T	G
⊔	0	1	2	3	4	5
C	1	1	1	2	3	4
A	2	1	2	2	3	4
C	3	2	1	2	3	4
G	4	3	2	2	3	3
T	5	4	3	3	2	3
G	6	5	4	4	3	2

$$\begin{aligned}\mathsf{ED}[i,j] = \min\{&\mathsf{ED}[i-1,j]+1,\\ &\mathsf{ED}[i,j-1]+1,\\ &\mathsf{ED}[i-1,j-1]+\ \mathsf{diff}(x[i],y[j])\}\end{aligned}$$

PD - esboço de algoritmos para outros exemplos

Mochila (com repetição e capacidades inteiras):

PD - esboço de algoritmos para outros exemplos

Mochila (com repetição e capacidades inteiras):

Item	Peso	Valor
1	6	\$30
2	3	\$14
3	4	\$16
4	2	\$19

Capacidade: $W = 9$

PD - esboço de algoritmos para outros exemplos

Mochila (com repetição e capacidades inteiras):

Item	Peso	Valor
1	6	\$30
2	3	\$14
3	4	\$16
4	2	\$19

Capacidade: $W = 9$

Ideia: $K[w] =$
"Maior valor pesando w"

PD - esboço de algoritmos para outros exemplos

Mochila (com repetição e capacidades inteiras):

Item	Peso	Valor
1	6	\$30
2	3	\$14
3	4	\$16
4	2	\$19

Capacidade: $W = 9$

Ideia: *K*[*w*] =
"Maior valor pesando *w*"

MochilaR (K)

1: $K[0] = 0$
2: **for** $w = 1$ até W **do**
3: $\quad K[v] = \max_{\text{item } i}\{K[w - w_i] + v_i\}$
4: **end for**

PD - esboço de algoritmos para outros exemplos

Mochila (capacidades inteiras):

PD - esboço de algoritmos para outros exemplos

Mochila (capacidades inteiras):

Item	Peso	Valor
1	6	\$30
2	3	\$14
3	4	\$16
4	2	\$19

Capacidade: $W = 9$

PD - esboço de algoritmos para outros exemplos

Mochila (capacidades inteiras):

Item	Peso	Valor
1	6	\$30
2	3	\$14
3	4	\$16
4	2	\$19

Capacidade: $W = 9$

Ideia: $K[w, i] =$
"Maior valor pesando w usando itens 1,2,...,i"

PD - esboço de algoritmos para outros exemplos

Mochila (capacidades inteiras):

PD - esboço de algoritmos para outros exemplos

Mochila (capacidades inteiras):

Ideia: $K[w, i] =$
"Maior valor pesando w usando itens 1,2,...,i"

Mochila (K)

1: $K[0, i] = 0$
2: $K[w, 0] = 0$
3: **for** $i = 1$ até W **do**
4: **for** $w = 1$ até W **do**
5: **if** $(w_i > w)$ **then**
6: $K[w, i] = K[w, i - 1]$
7: **else**
8: $K[w, i] = \max_{\text{item } i}(K[w, i - 1], K[w - w_i, i - 1] + v_i)$
9: **end if**
10: **end for**
11: **end for**

PD - esboço de algoritmos para outros exemplos

Caixeiro Viajante:

PD - esboço de algoritmos para outros exemplos

Caixeiro Viajante:

TSP (G) /* $V(G) = \{1, 2, ..., n\}$ */

1: $S = \{1\}$

2: $\forall S \subseteq V(G)$ em ordem de tamanho crescente ≥ 2

3: **for all** $j \in S$, $j \neq 1$ **do** $L[S, j] = \min_{\substack{i \in S \\ i \neq j}} \{L[S \setminus \{j\}, i] + l(i, j)\}$

4: **end for**

Algoritmos Gulosos

Algoritmos Gulosos

Árvore Geradora Mínima:

Algoritmos Gulosos

Árvore Geradora Mínima:

- Dado grafo ponderado conexo G

Algoritmos Gulosos

Árvore Geradora Mínima:

- Dado grafo ponderado conexo G
- Encontrar subgrafo T de G tal que

Algoritmos Gulosos

Árvore Geradora Mínima:

- Dado grafo ponderado conexo G
- Encontrar subgrafo T de G tal que
  - T é árvore

Algoritmos Gulosos

Árvore Geradora Mínima:

- Dado grafo ponderado conexo G
- Encontrar subgrafo T de G tal que
  - T é árvore (i.e., conexo e acíclico)

Algoritmos Gulosos

Árvore Geradora Mínima:

- Dado grafo ponderado conexo G
- Encontrar subgrafo T de G tal que
  - T é árvore (i.e., conexo e acíclico)
  - $V(T) = V(G)$

Algoritmos Gulosos

Árvore Geradora Mínima:

- Dado grafo ponderado conexo G
- Encontrar subgrafo T de G tal que
  - T é árvore (i.e., conexo e acíclico)
  - $V(T) = V(G)$
  - Soma dos pesos de $E(T)$ é mínimo

Algoritmos Gulosos

Árvore Geradora Mínima:

- Dado grafo ponderado conexo G
- Encontrar subgrafo T de G tal que
  - T é árvore (i.e., conexo e acíclico)
  - $V(T) = V(G)$
  - Soma dos pesos de $E(T)$ é mínimo

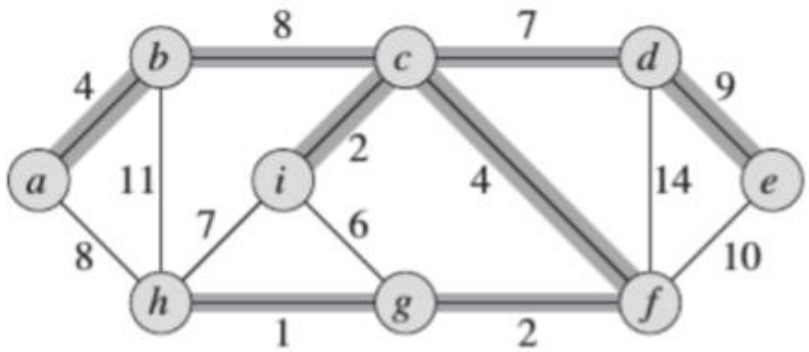


Figura: (Cormen, Leiserson, Rivest, and Stein, 2009, fig 23.1, p. 625)

Algoritmos Gulosos

Árvore Geradora Mínima: Algoritmo de Kruskal

Algoritmos Gulosos

Árvore Geradora Mínima: Algoritmo de Kruskal

KRUSKAL (G)

1: Ordene as arestas de G e coloque em Q
2: Inicialize $T = (V, \varnothing)$
3: **while** $|E(T)| < n - 1$ **do**
4: Remova primeiro elemento de Q e coloque em e
5: **if** $T + e$ é acíclico **then**
6: Insira e em T
7: **else**
8: Descarte e
9: **end if**
10: **end while**

Algoritmos Gulosos

Árvore Geradora Mínima: Algoritmo de Prim

Algoritmos Gulosos

Árvore Geradora Mínima: Algoritmo de Prim

PRIM (G)

1: Fixe um vértice v (escolhido arbitrariamente)
2: Inicialize $T = (\{v\}, \varnothing)$
3: **while** $|E(T)| < n - 1$ **do**
4: Escolha a aresta $e = \{u, v\}$ de menor peso incidente a $V(T)$
5: **if** e tem as duas extremidades em $V(T)$ **then**
6: Descarte e
7: **else**
8: Insira vértices u, v e aresta e em T
9: **end if**
10: **end while**

Algoritmos Gulosos

Cobertura por Conjuntos (CC)

Algoritmos Gulosos

Cobertura por Conjuntos (CC)

- **Entrada:** Um conjunto B e m subconjuntos $S_1, ..., S_m \subseteq B$.

Algoritmos Gulosos

Cobertura por Conjuntos (CC)

- **Entrada:** Um conjunto B e m subconjuntos $S_1, ..., S_m \subseteq B$.
- **Saída:** O menor número de subconjuntos S_i tal que a união dos conjuntos S_i seja B.

Algoritmos Gulosos

Cobertura por Conjuntos (CC)

- **Entrada:** Um conjunto B e m subconjuntos $S_1, ..., S_m \subseteq B$.
- **Saída:** O menor número de subconjuntos S_i tal que a união dos conjuntos S_i seja B.

Algoritmo Guloso: A cada passo escolha o conjunto S_i que contenha mais elementos ainda não inclusos nos conjuntos já escolhidos

Algoritmos Gulosos

Cobertura por Conjuntos (CC)

- **Entrada:** Um conjunto B e m subconjuntos $S_1, ..., S_m \subseteq B$.
- **Saída:** O menor número de subconjuntos S_i tal que a união dos conjuntos S_i seja B.

Algoritmo Guloso: A cada passo escolha o conjunto S_i que contenha mais elementos ainda não inclusos nos conjuntos já escolhidos

- Seja n o tamanho de B e k o tamanho da solução ótima

Algoritmos Gulosos

Cobertura por Conjuntos (CC)

- **Entrada:** Um conjunto B e m subconjuntos $S_1, ..., S_m \subseteq B$.
- **Saída:** O menor número de subconjuntos S_i tal que a união dos conjuntos S_i seja B.

Algoritmo Guloso: A cada passo escolha o conjunto S_i que contenha mais elementos ainda não inclusos nos conjuntos já escolhidos

- Seja n o tamanho de B e k o tamanho da solução ótima
- Estratégia gulosa garante uma solução de tamanho no máximo $k \cdot \ln n$

Algoritmos Gulosos

Cobertura por Conjuntos (CC)

- **Entrada:** Um conjunto B e m subconjuntos $S_1, ..., S_m \subseteq B$.
- **Saída:** O menor número de subconjuntos S_i tal que a união dos conjuntos S_i seja B.

Algoritmo Guloso: A cada passo escolha o conjunto S_i que contenha mais elementos ainda não inclusos nos conjuntos já escolhidos

- Seja n o tamanho de B e k o tamanho da solução ótima
- Estratégia gulosa garante uma solução de tamanho no máximo $k \cdot \ln n$

Algoritmos Gulosos

Cobertura por Conjuntos (CC)

- **Entrada:** Um conjunto B e m subconjuntos $S_1, ..., S_m \subseteq B$.
- **Saída:** O menor número de subconjuntos S_i tal que a união dos conjuntos S_i seja B.

Algoritmo Guloso: A cada passo escolha o conjunto S_i que contenha mais elementos ainda não inclusos nos conjuntos já escolhidos

- Seja n o tamanho de B e k o tamanho da solução ótima
- Estratégia gulosa garante uma solução de tamanho no máximo $k \cdot \ln n$

Cobertura por Vértices (CV)

Algoritmos Gulosos

Cobertura por Conjuntos (CC)

- **Entrada:** Um conjunto B e m subconjuntos $S_1, ..., S_m \subseteq B$.
- **Saída:** O menor número de subconjuntos S_i tal que a união dos conjuntos S_i seja B.

Algoritmo Guloso: A cada passo escolha o conjunto S_i que contenha mais elementos ainda não inclusos nos conjuntos já escolhidos

- Seja n o tamanho de B e k o tamanho da solução ótima
- Estratégia gulosa garante uma solução de tamanho no máximo $k \cdot \ln n$

Cobertura por Vértices (CV)

- **Entrada:** Um grafo $G = (V(G), E(G))$.

Algoritmos Gulosos

Cobertura por Conjuntos (CC)

- **Entrada:** Um conjunto B e m subconjuntos $S_1, ..., S_m \subseteq B$.
- **Saída:** O menor número de subconjuntos S_i tal que a união dos conjuntos S_i seja B.

Algoritmo Guloso: A cada passo escolha o conjunto S_i que contenha mais elementos ainda não inclusos nos conjuntos já escolhidos

- Seja n o tamanho de B e k o tamanho da solução ótima
- Estratégia gulosa garante uma solução de tamanho no máximo $k \cdot \ln n$

Cobertura por Vértices (CV)

- **Entrada:** Um grafo $G = (V(G), E(G))$.
- **Saída:** Conjunto S de vértices de tamanho mínimo tal que $\forall e \in E(G)$, a aresta e tem pelo menos uma ponta em S

Algoritmos Gulosos

Cobertura por Conjuntos (CC)

- **Entrada:** Um conjunto B e m subconjuntos $S_1, ..., S_m \subseteq B$.
- **Saída:** O menor número de subconjuntos S_i tal que a união dos conjuntos S_i seja B.

Algoritmo Guloso: A cada passo escolha o conjunto S_i que contenha mais elementos ainda não inclusos nos conjuntos já escolhidos

- Seja n o tamanho de B e k o tamanho da solução ótima
- Estratégia gulosa garante uma solução de tamanho no máximo $k \cdot \ln n$

Cobertura por Vértices (CV)

- **Entrada:** Um grafo $G = (V(G), E(G))$.
- **Saída:** Conjunto S de vértices de tamanho mínimo tal que $\forall e \in E(G)$, a aresta e tem pelo menos uma ponta em S

Note que CV é um caso particular de CC:

Algoritmos Gulosos

Cobertura por Conjuntos (CC)

- **Entrada:** Um conjunto B e m subconjuntos $S_1, ..., S_m \subseteq B$.
- **Saída:** O menor número de subconjuntos S_i tal que a união dos conjuntos S_i seja B.

Algoritmo Guloso: A cada passo escolha o conjunto S_i que contenha mais elementos ainda não inclusos nos conjuntos já escolhidos

- Seja n o tamanho de B e k o tamanho da solução ótima
- Estratégia gulosa garante uma solução de tamanho no máximo $k \cdot \ln n$

Cobertura por Vértices (CV)

- **Entrada:** Um grafo $G = (V(G), E(G))$.
- **Saída:** Conjunto S de vértices de tamanho mínimo tal que $\forall e \in E(G)$, a aresta e tem pelo menos uma ponta em S

Note que CV é um caso particular de CC:

- Basta fazer $B = E(G)$ e para $v_1, v_2, ...,$ fazer $S_i = \partial(v_i)$

Algoritmos Gulosos

Cobertura por Conjuntos (CC)

- **Entrada:** Um conjunto B e m subconjuntos $S_1, ..., S_m \subseteq B$.
- **Saída:** O menor número de subconjuntos S_i tal que a união dos conjuntos S_i seja B.

Algoritmo Guloso: A cada passo escolha o conjunto S_i que contenha mais elementos ainda não inclusos nos conjuntos já escolhidos

- Seja n o tamanho de B e k o tamanho da solução ótima
- Estratégia gulosa garante uma solução de tamanho no máximo $k \cdot \ln n$

Cobertura por Vértices (CV)

- **Entrada:** Um grafo $G = (V(G), E(G))$.
- **Saída:** Conjunto S de vértices de tamanho mínimo tal que $\forall e \in E(G)$, a aresta e tem pelo menos uma ponta em S

Note que CV é um caso particular de CC:

- Basta fazer $B = E(G)$ e para $v_1, v_2, ...,$ fazer $S_i = \partial(v_i)$
- Portanto algoritmo guloso (escolher vértice de maior grau) encontra cobertura de tamanho $k \cdot \ln |V(G)|$

Algoritmos Gulosos

Cobertura por Conjuntos (CC)

- **Entrada:** Um conjunto B e m subconjuntos $S_1, ..., S_m \subseteq B$.
- **Saída:** O menor número de subconjuntos S_i tal que a união dos conjuntos S_i seja B.

Algoritmo Guloso: A cada passo escolha o conjunto S_i que contenha mais elementos ainda não inclusos nos conjuntos já escolhidos

- Seja n o tamanho de B e k o tamanho da solução ótima
- Estratégia gulosa garante uma solução de tamanho no máximo $k \cdot \ln n$

Cobertura por Vértices (CV)

- **Entrada:** Um grafo $G = (V(G), E(G))$.
- **Saída:** Conjunto S de vértices de tamanho mínimo tal que $\forall e \in E(G)$, a aresta e tem pelo menos uma ponta em S

Note que CV é um caso particular de CC:

- Basta fazer $B = E(G)$ e para $v_1, v_2, ...,$ fazer $S_i = \partial(v_i)$
- Portanto algoritmo guloso (escolher vértice de maior grau) encontra cobertura de tamanho $k \cdot \ln |V(G)|$ (sendo k o tamanho da cobertura ótima)

Algoritmos Gulosos

Cobertura por Conjuntos (CC)

- **Entrada:** Um conjunto B e m subconjuntos $S_1, ..., S_m \subseteq B$.
- **Saída:** O menor número de subconjuntos S_i tal que a união dos conjuntos S_i seja B.

Algoritmo Guloso: A cada passo escolha o conjunto S_i que contenha mais elementos ainda não inclusos nos conjuntos já escolhidos

- Seja n o tamanho de B e k o tamanho da solução ótima
- Estratégia gulosa garante uma solução de tamanho no máximo $k \cdot \ln n$

Cobertura por Vértices (CV)

- **Entrada:** Um grafo $G = (V(G), E(G))$.
- **Saída:** Conjunto S de vértices de tamanho mínimo tal que $\forall e \in E(G)$, a aresta e tem pelo menos uma ponta em S

Note que CV é um caso particular de CC:

- Basta fazer $B = E(G)$ e para $v_1, v_2, ...,$ fazer $S_i = \partial(v_i)$
- Portanto algoritmo guloso (escolher vértice de maior grau) encontra cobertura de tamanho $k \cdot \ln |V(G)|$ (sendo k o tamanho da cobertura ótima)
- Entretanto, existe algoritmo polinomial com garantia $2k$

Algoritmos Gulosos

Cobertura por Conjuntos (CC)

- **Entrada:** Um conjunto B e m subconjuntos $S_1, ..., S_m \subseteq B$.
- **Saída:** O menor número de subconjuntos S_i tal que a união dos conjuntos S_i seja B.

Algoritmo Guloso: A cada passo escolha o conjunto S_i que contenha mais elementos ainda não inclusos nos conjuntos já escolhidos

- Seja n o tamanho de B e k o tamanho da solução ótima
- Estratégia gulosa garante uma solução de tamanho no máximo $k \cdot \ln n$

Cobertura por Vértices (CV)

- **Entrada:** Um grafo $G = (V(G), E(G))$.
- **Saída:** Conjunto S de vértices de tamanho mínimo tal que $\forall e \in E(G)$, a aresta e tem pelo menos uma ponta em S

Note que CV é um caso particular de CC:

- Basta fazer $B = E(G)$ e para $v_1, v_2, ...,$ fazer $S_i = \partial(v_i)$
- Portanto algoritmo guloso (escolher vértice de maior grau) encontra cobertura de tamanho $k \cdot \ln |V(G)|$ (sendo k o tamanho da cobertura ótima)
- Entretanto, existe algoritmo polinomial com garantia $2k$ (fator de aprox. 2)

Algoritmos Gulosos

Cobertura por Conjuntos (CC)

- **Entrada:** Um conjunto B e m subconjuntos $S_1, ..., S_m \subseteq B$.
- **Saída:** O menor número de subconjuntos S_i tal que a união dos conjuntos S_i seja B.

Algoritmo Guloso: A cada passo escolha o conjunto S_i que contenha mais elementos ainda não inclusos nos conjuntos já escolhidos

Algoritmos Gulosos

Cobertura por Conjuntos (CC)

- **Entrada:** Um conjunto B e m subconjuntos $S_1, ..., S_m \subseteq B$.
- **Saída:** O menor número de subconjuntos S_i tal que a união dos conjuntos S_i seja B.

Algoritmo Guloso: A cada passo escolha o conjunto S_i que contenha mais elementos ainda não inclusos nos conjuntos já escolhidos

- Seja n o tamanho de B e k é o tamanho da solução ótima

Algoritmos Gulosos

Cobertura por Conjuntos (CC)

- **Entrada:** Um conjunto B e m subconjuntos $S_1, ..., S_m \subseteq B$.
- **Saída:** O menor número de subconjuntos S_i tal que a união dos conjuntos S_i seja B.

Algoritmo Guloso: A cada passo escolha o conjunto S_i que contenha mais elementos ainda não inclusos nos conjuntos já escolhidos

- Seja n o tamanho de B e k é o tamanho da solução ótima
- Fato: A cada passo pelo menos $1/k$ dos elementos restantes é coberto

Algoritmos Gulosos

Cobertura por Conjuntos (CC)

- **Entrada:** Um conjunto B e m subconjuntos $S_1, ..., S_m \subseteq B$.
- **Saída:** O menor número de subconjuntos S_i tal que a união dos conjuntos S_i seja B.

Algoritmo Guloso: A cada passo escolha o conjunto S_i que contenha mais elementos ainda não inclusos nos conjuntos já escolhidos

- Seja n o tamanho de B e k é o tamanho da solução ótima
- Fato: A cada passo pelo menos $1/k$ dos elementos restantes é coberto

Seja n_i o número de elementos descobertos depois de i passos

Algoritmos Gulosos

Cobertura por Conjuntos (CC)

- **Entrada:** Um conjunto B e m subconjuntos $S_1, ..., S_m \subseteq B$.
- **Saída:** O menor número de subconjuntos S_i tal que a união dos conjuntos S_i seja B.

Algoritmo Guloso: A cada passo escolha o conjunto S_i que contenha mais elementos ainda não inclusos nos conjuntos já escolhidos

- Seja n o tamanho de B e k é o tamanho da solução ótima
- Fato: A cada passo pelo menos $1/k$ dos elementos restantes é coberto

Seja n_i o número de elementos descobertos depois de i passos (obs: $n_0 = n$)

Algoritmos Gulosos

Cobertura por Conjuntos (CC)

- **Entrada:** Um conjunto B e m subconjuntos $S_1, ..., S_m \subseteq B$.
- **Saída:** O menor número de subconjuntos S_i tal que a união dos conjuntos S_i seja B.

Algoritmo Guloso: A cada passo escolha o conjunto S_i que contenha mais elementos ainda não inclusos nos conjuntos já escolhidos

- Seja n o tamanho de B e k é o tamanho da solução ótima
- Fato: A cada passo pelo menos $1/k$ dos elementos restantes é coberto

Seja n_i o número de elementos descobertos depois de i passos (obs: $n_0 = n$)

Passo 1: Sobram $n_1 \leq (1 - 1/k)n_0$ elementos;

Algoritmos Gulosos

Cobertura por Conjuntos (CC)

- **Entrada:** Um conjunto B e m subconjuntos $S_1, ..., S_m \subseteq B$.
- **Saída:** O menor número de subconjuntos S_i tal que a união dos conjuntos S_i seja B.

Algoritmo Guloso: A cada passo escolha o conjunto S_i que contenha mais elementos ainda não inclusos nos conjuntos já escolhidos

- Seja n o tamanho de B e k é o tamanho da solução ótima
- Fato: A cada passo pelo menos $1/k$ dos elementos restantes é coberto

Seja n_i o número de elementos descobertos depois de i passos (obs: $n_0 = n$)

Passo 1: Sobram $n_1 \leq (1 - 1/k)n_0$ elementos;

Passo 2: Sobram $n_2 \leq (1 - 1/k)n_1$

Algoritmos Gulosos

Cobertura por Conjuntos (CC)

- **Entrada:** Um conjunto B e m subconjuntos $S_1, ..., S_m \subseteq B$.
- **Saída:** O menor número de subconjuntos S_i tal que a união dos conjuntos S_i seja B.

Algoritmo Guloso: A cada passo escolha o conjunto S_i que contenha mais elementos ainda não inclusos nos conjuntos já escolhidos

- Seja n o tamanho de B e k é o tamanho da solução ótima
- Fato: A cada passo pelo menos $1/k$ dos elementos restantes é coberto

Seja n_i o número de elementos descobertos depois de i passos (obs: $n_0 = n$)

Passo 1: Sobram $n_1 \leq (1 - 1/k)n_0$ elementos;

Passo 2: Sobram $n_2 \leq (1 - 1/k)n_1 \leq (1 - 1/k)^2 n_0$ elementos;

Algoritmos Gulosos

Cobertura por Conjuntos (CC)

- **Entrada:** Um conjunto B e m subconjuntos $S_1, ..., S_m \subseteq B$.
- **Saída:** O menor número de subconjuntos S_i tal que a união dos conjuntos S_i seja B.

Algoritmo Guloso: A cada passo escolha o conjunto S_i que contenha mais elementos ainda não inclusos nos conjuntos já escolhidos

- Seja n o tamanho de B e k é o tamanho da solução ótima
- Fato: A cada passo pelo menos $1/k$ dos elementos restantes é coberto

Seja n_i o número de elementos descobertos depois de i passos (obs: $n_0 = n$)

Passo 1: Sobram $n_1 \leq (1 - 1/k)n_0$ elementos;

Passo 2: Sobram $n_2 \leq (1 - 1/k)n_1 \leq (1 - 1/k)^2 n_0$ elementos;

Passo *t*: Sobram $n_t \leq (1 - 1/k)^t n_0$

Algoritmos Gulosos

Cobertura por Conjuntos (CC)

- **Entrada:** Um conjunto B e m subconjuntos $S_1, ..., S_m \subseteq B$.
- **Saída:** O menor número de subconjuntos S_i tal que a união dos conjuntos S_i seja B.

Algoritmo Guloso: A cada passo escolha o conjunto S_i que contenha mais elementos ainda não inclusos nos conjuntos já escolhidos

- Seja n o tamanho de B e k é o tamanho da solução ótima
- Fato: A cada passo pelo menos $1/k$ dos elementos restantes é coberto

Seja n_i o número de elementos descobertos depois de i passos (obs: $n_0 = n$)

Passo 1: Sobram $n_1 \leq (1 - 1/k)n_0$ elementos;

Passo 2: Sobram $n_2 \leq (1 - 1/k)n_1 \leq (1 - 1/k)^2 n_0$ elementos;

Passo *t*: Sobram $n_t \leq (1 - 1/k)^t n_0 = (1 - 1/k)^t n$ elementos;

Algoritmos Gulosos

Cobertura por Conjuntos (CC)

- **Entrada:** Um conjunto B e m subconjuntos $S_1, ..., S_m \subseteq B$.
- **Saída:** O menor número de subconjuntos S_i tal que a união dos conjuntos S_i seja B.

Algoritmo Guloso: A cada passo escolha o conjunto S_i que contenha mais elementos ainda não inclusos nos conjuntos já escolhidos

- Seja n o tamanho de B e k é o tamanho da solução ótima
- Fato: A cada passo pelo menos $1/k$ dos elementos restantes é coberto

Seja n_i o número de elementos descobertos depois de i passos (obs: $n_0 = n$)

Passo 1: Sobram $n_1 \leq (1 - 1/k)n_0$ elementos;

Passo 2: Sobram $n_2 \leq (1 - 1/k)n_1 \leq (1 - 1/k)^2 n_0$ elementos;

Passo *t*: Sobram $n_t \leq (1 - 1/k)^t n_0 = (1 - 1/k)^t n$ elementos;

Vamos provar que no passo $t = k \ln n$ todos elementos foram escolhidos

Algoritmos Gulosos

Cobertura por Conjuntos (CC)

- **Entrada:** Um conjunto B e m subconjuntos $S_1, ..., S_m \subseteq B$.
- **Saída:** O menor número de subconjuntos S_i tal que a união dos conjuntos S_i seja B.

Algoritmo Guloso: A cada passo escolha o conjunto S_i que contenha mais elementos ainda não inclusos nos conjuntos já escolhidos

- Seja n o tamanho de B e k é o tamanho da solução ótima
- Fato: A cada passo pelo menos $1/k$ dos elementos restantes é coberto

Seja n_i o número de elementos descobertos depois de i passos (obs: $n_0 = n$)

Passo 1: Sobram $n_1 \leq (1 - 1/k)n_0$ elementos;

Passo 2: Sobram $n_2 \leq (1 - 1/k)n_1 \leq (1 - 1/k)^2 n_0$ elementos;

Passo *t*: Sobram $n_t \leq (1 - 1/k)^t n_0 = (1 - 1/k)^t n$ elementos;

Vamos provar que no passo $t = k \ln n$ todos elementos foram escolhidos

$$n_t \leq (1 - 1/k)^t n$$

Algoritmos Gulosos

Cobertura por Conjuntos (CC)

- **Entrada:** Um conjunto B e m subconjuntos $S_1, ..., S_m \subseteq B$.
- **Saída:** O menor número de subconjuntos S_i tal que a união dos conjuntos S_i seja B.

Algoritmo Guloso: A cada passo escolha o conjunto S_i que contenha mais elementos ainda não inclusos nos conjuntos já escolhidos

- Seja n o tamanho de B e k é o tamanho da solução ótima
- Fato: A cada passo pelo menos $1/k$ dos elementos restantes é coberto

Seja n_i o número de elementos descobertos depois de i passos (obs: $n_0 = n$)

Passo 1: Sobram $n_1 \leq (1 - 1/k)n_0$ elementos;

Passo 2: Sobram $n_2 \leq (1 - 1/k)n_1 \leq (1 - 1/k)^2 n_0$ elementos;

Passo *t*: Sobram $n_t \leq (1 - 1/k)^t n_0 = (1 - 1/k)^t n$ elementos;

Vamos provar que no passo $t = k \ln n$ todos elementos foram escolhidos

$$n_t \leq (1 - 1/k)^t n < (e^{-1/k})^t n$$

Algoritmos Gulosos

Cobertura por Conjuntos (CC)

- **Entrada:** Um conjunto B e m subconjuntos $S_1, ..., S_m \subseteq B$.
- **Saída:** O menor número de subconjuntos S_i tal que a união dos conjuntos S_i seja B.

Algoritmo Guloso: A cada passo escolha o conjunto S_i que contenha mais elementos ainda não inclusos nos conjuntos já escolhidos

- Seja n o tamanho de B e k é o tamanho da solução ótima
- Fato: A cada passo pelo menos $1/k$ dos elementos restantes é coberto

Seja n_i o número de elementos descobertos depois de i passos (obs: $n_0 = n$)

Passo 1: Sobram $n_1 \leq (1 - 1/k)n_0$ elementos;

Passo 2: Sobram $n_2 \leq (1 - 1/k)n_1 \leq (1 - 1/k)^2 n_0$ elementos;

Passo *t*: Sobram $n_t \leq (1 - 1/k)^t n_0 = (1 - 1/k)^t n$ elementos;

Vamos provar que no passo $t = k \ln n$ todos elementos foram escolhidos

$$n_t \leq (1 - 1/k)^t n < (e^{-1/k})^t n = (e^{-1/k})^{\ln n \cdot k} n = \frac{1}{n} n = 1$$