CI1238 - Otimização

Aula 14 - Programação Dinâmica (Parte 1)

Professor Murilo V. G. da Silva

Departamento de Informática

Universidade Federal do Paraná

19/11/2024

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.
  - Semelhante a técnica de dividir para conquistar, mas com um **diferença**:
  - No caso de PD, subproblemas **podem ter intersecção**
  - Idéia central: Quando for possível, não resolver o mesmo subproblema mais de uma vez. $\rightarrow$ armazenar solução já computada

note: na técnica dividir para conquistar não há subproblemas repetidos, então não existe esta dificuldade para se lidar.

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

- Para este problema vamos ver a técnica em bastante detalhe
- Bibliografia: [CIRS09] – *Introduction to Algorithms*, Capítulo 15

- Em seguida veremos a mesma técnica para vários problemas
- Bibliografia: [CIRS09] – *Algorithms*, Capítulo 6.

O Problema do Corte de Hastes

- Problema: Temos uma haste de comprimento n e queremos cortar em pedaços menores para vender os pedaços
- Tanto n quanto o tamanho de cada pedaço são números inteiros
- Pedaços de tamanhos diferentes tem valores diferentes (valores em $\mathbb{R}$)

Exemplo de instância: Cada p_i é o valor de venda de uma haste de comprimento i:

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

PROBLEMA DAS HASTES (HASTES):
Instância: (p, n), onde p é um vetor contendo n valores reais

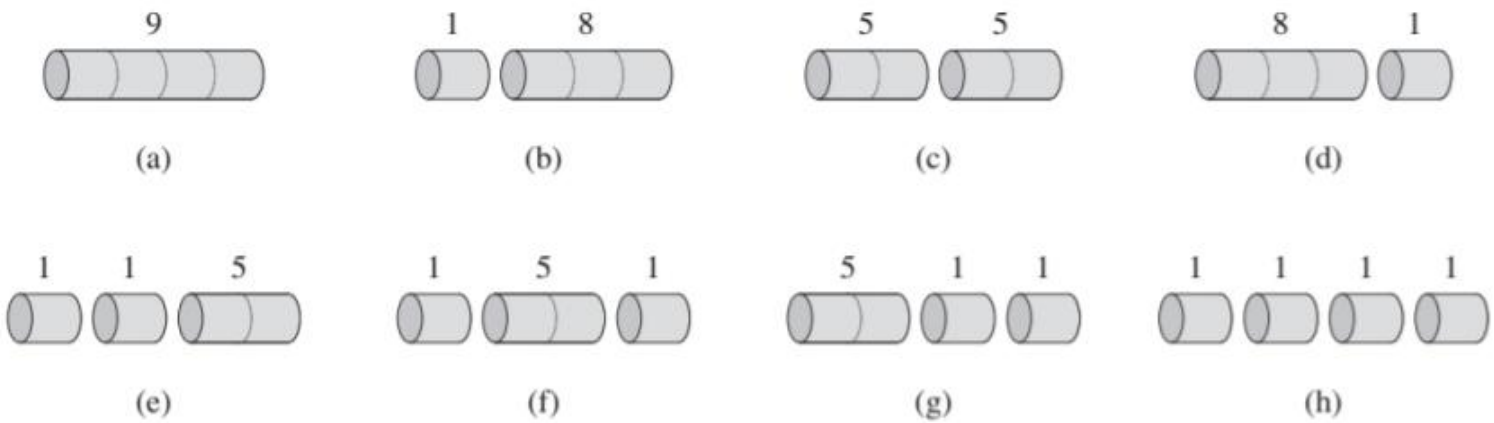
Solução: lista inteiros $[i_1, ..., i_k]$ que indica o tamanho de cada pedaço de haste, i.e., sujeito a $n = i_1 + i_2 + ... + i_k$ tal que $r_n = p_{i_1} + p_{i_2} + ... + p_{i_k}$ seja máximo.

Note: A solução é um conjunto de índices;
Vamos usar r_n para indicar o valor ótimo para uma instância de tamanho n, i.e., o valor da função objetivo aplicada à solução $[i_1, ..., i_k]$

O Problema do Corte de Hastes

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

Para o vetor do slide anterior, considere o subproblema em que $n = 4$:



- 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.

- A solução do subproblema: esquema de corte (c)

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 10 subproblemas da instância acima: $n = 1, 2, ..., 10$

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

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

Solução recursiva para um problema mais simples

PROBLEMA DO VALOR DAS HASTES (VALOR-HASTES):
Instância: (p, n), onde p é um vetor contendo n valores reais

Solução: Valor r_n tal que existe $[i_1, ..., i_k]$ tal que $n = i_1 + ... + i_k$ e $r_n = p_{i_1} + ... + p_{i_k}$.

Reescrevendo r_n:

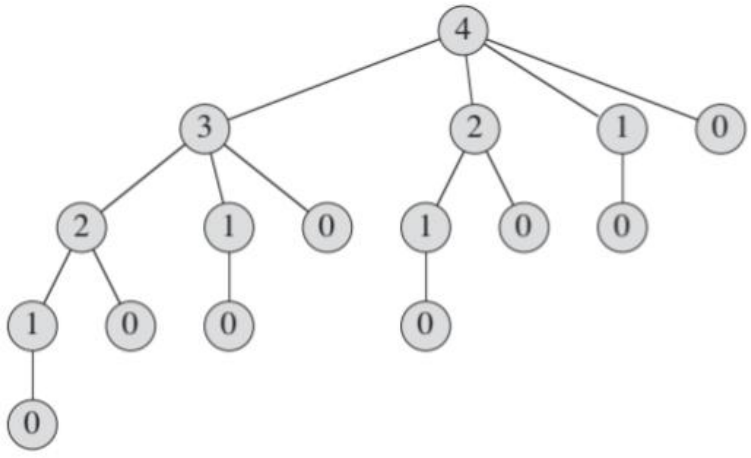
$$r_n = \max_{1 \le i \le n} (p_i + r_{n-i}) \qquad \text{onde } r_0 = 0$$

```
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

Valores de q: $-\infty$ 0 1 5 8 10 13 17 18 22 25 30

Solução recursiva para um problema mais simples



Árvore de recursões para $n = 4$.

- A árvore de recursões tem 2^n nós
  Cada nó k corresponde à execução com entrada de tamanho k
- Observe que o problema é resolvido repetidas vezes para cada subproblema de tamanho k
- Ideia: Só usar recursão **uma única vez** para cada subproblema de tamanho k.
- Importante: Em problemas de PD em geral, se a quantidade de problemas diferentes for **polinomial** e o tempo de computar a base for polinomial, temos um algoritmo polinomial!

Programação Dinâmica usando Memoização

Usando vetor *r* para guardar valor 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)
```

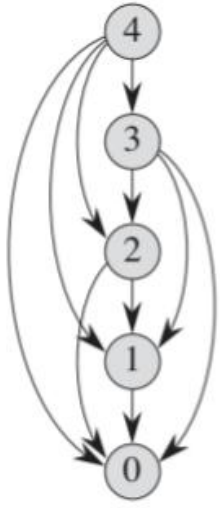
```
MEMOIZED-CUT-ROD-AUX(p, n, r)
1  if r[n] ≥ 0
2      return r[n]
3  if n == 0
4      q = 0
5  else q = -∞
6      for i = 1 to n
7          q = max(q, p[i] + MEMOIZED-CUT-ROD-AUX(p, n - i, r))
8  r[n] = q
9  return q
```

Versão "Bottom-up" do mesmo algoritmo

Algoritmo *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]
```

Grafo $G = (V, E)$ de subproblemas



- Problemas resolvidos do menor para o maior
- Cada vértice é um subproblema de tamanho $j \rightarrow$ solução armazenada em $r[j]$
- 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 cada vértice é "colapsado" em um único vértice.

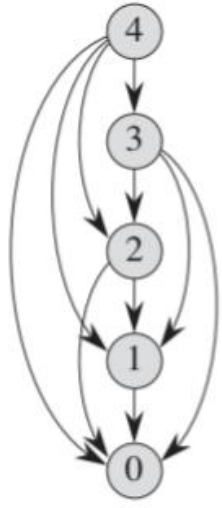
- Importante: Em geral não se constrói o grafo, trata-se de um grafo implícito!

Programação Dinâmica "Bottom-up'

Algoritmo bottom up

Grafo $G = (V, E)$ de subproblemas

```
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]
```



- Na versão *Bottom-up*, observe que o grafo é percorrido de baixo para cima
  - Repetição externa (linha 3): "sobe" a para resolver subproblemas maiores
    Repetição interna (linha 5): "desce" olhando subproblemas já resolvidos
- Tipicamente: a complexidade em PD é polinomial em $|V| + |E|$,
  isto é, no tamanho do grafo.

Resolvendo o problema original HASTES

Solução para HASTES-VALOR:

```
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]
```

Solução para HASTES:

```
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"

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

```
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