Algoritmos Gulosos

Prof. Daniel Kikuti

Universidade Estadual de Maringá

Visão geral

- Técnica de projeto de algoritmos.
- Aplicada em vários problemas de otimização.
- Mais simples e mais eficientes que Programação Dinâmica.
- Escolha míope (melhor escolha no momento).
- Nem sempre produzem soluções ótimas.

Elementos de algoritmos gulosos

Subestrutura ótima

Um subproblema exibe **subestrutura ótima** se uma solução ótima para um problema contém dentro dela soluções ótimas para subproblemas.

Propriedade da escolha gulosa

Podemos construir uma solução ótima global fazendo escolhas ótimas locais (gulosas – sem considerar os resultados dos subproblemas).

Problema de Seleção de Atividades

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. **Introduction to Algorithms**, Third Edition. The MIT Press. Chapter 16.

Seleção de atividades

O problema

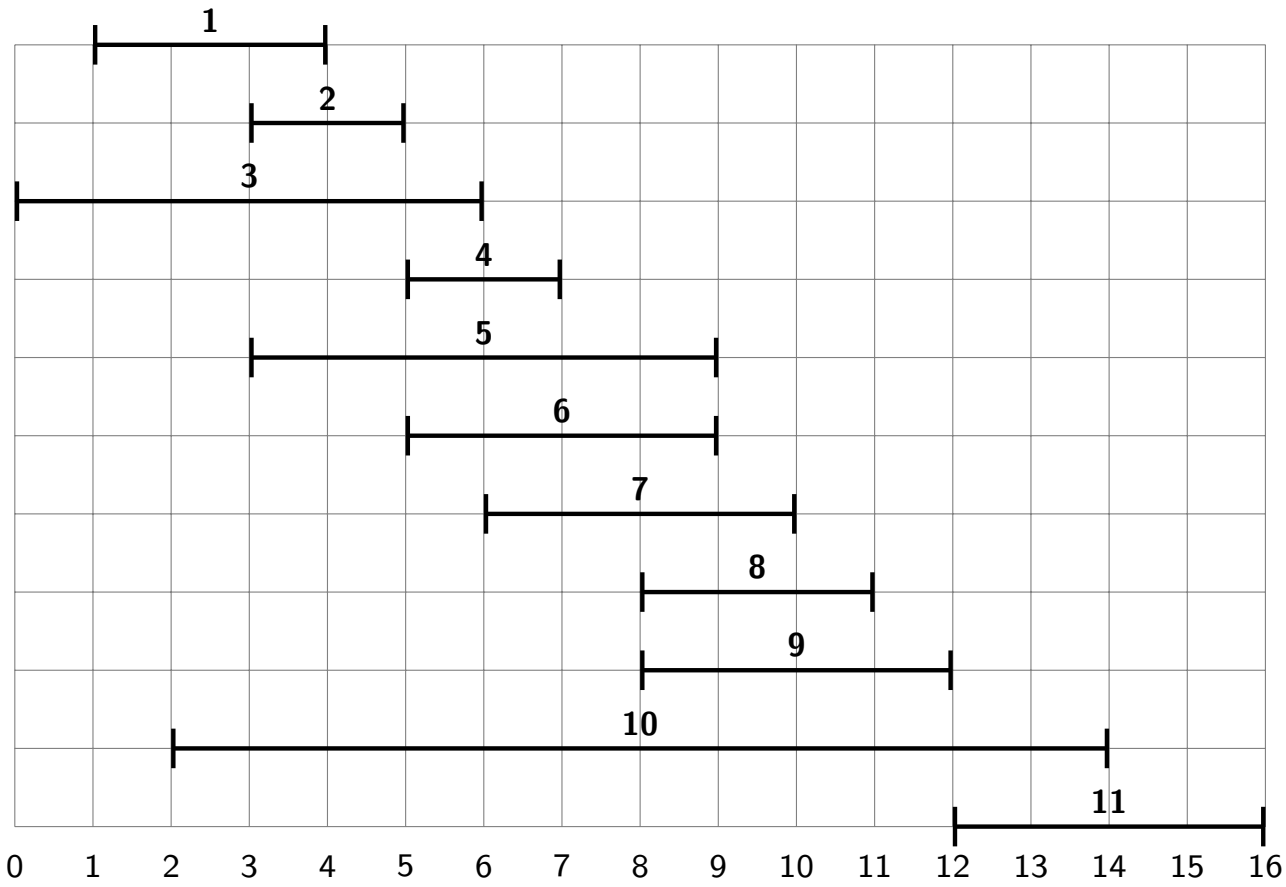
Dado um conjunto de n atividades $S = \{a_1, a_2, \cdots, a_n\}$ que requerem o uso exclusivo de um recurso comum (sala de aula, processador, etc.) e os respectivos tempos de início e término para cada atividade $[s_i, f_i)$, selecionar o maior conjunto de atividades mutuamente compatíveis, isto é, atividades a_i e a_j tais que $s_i \geq f_j$ ou $s_j \geq f_i$.

- Suponha que as atividades estão ordenadas por tempo de término: $f_1 \leq f_2 \leq \cdots \leq f_{n-1} \leq f_n$.

Exemplo

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

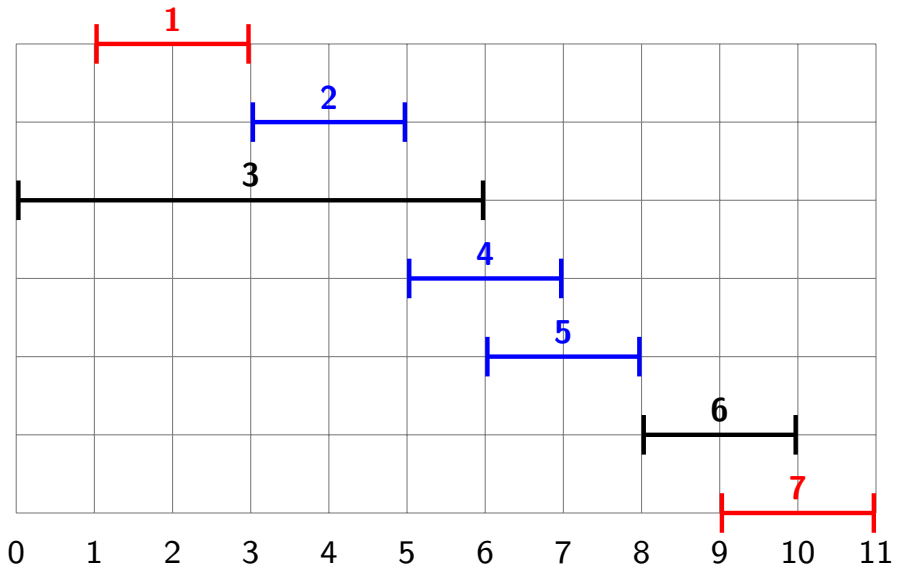
Exemplo



- Exemplo de solução viável: $\{a_3, a_9, a_{11}\}$.
- Exemplo de soluções ótimas: $\{a_1, a_4, a_8, a_{11}\}$ e $\{a_2, a_4, a_9, a_{11}\}$.

Notação

Seja S_{ij} o conjunto de atividades que começam após o término da atividade a_i e terminam antes do início da atividade a_j.



Exemplo: $S_{1,7} = \{a_2, a_4, a_5\}$.

Caracterizando a estrutura ótima

- Queremos determinar um conjunto máximo de atividades mutuamente compatíveis com S_{ij} e tal subconjunto máximo é A_{ij}, que inclui alguma atividade a_k.
- Incluindo a_k em uma solução ótima, ficamos com dois subproblemas: S_{ik} e S_{kj}.
- Sejam $A_{ik} = A_{ij} \cap S_{ik}$ e $A_{kj} = A_{ij} \cap S_{kj}$, então $A_{ij} = A_{ik} \cup \{a_k\} \cup A_{kj}$.
- Portanto, o conjunto de tamanho máximo A_{ij} de atividades mutuamente exclusivas em S_{ij} consiste em $|A_{ij}| = |A_{ik}| + 1 + |A_{kj}|$ e deve ser ótimo (argumento de recortar e colar).

Solução via Programação Dinâmica

Seja $c[i,j]$ o tamanho de uma solução ótima para S_{ij}, ou seja: $c[i,j] = |A_{ij}|$.

$$c[i,j] = \begin{cases} 0 & \text{se } S_{ij} = \emptyset \\ \max\limits_{i \leq k \leq j} \{c[i,k] + 1 + c[k,j]\} & \text{se } S_{ij} \neq \emptyset. \end{cases}$$

Partindo desta formulação recursiva, poderíamos verificar que há sobreposição de problemas e poderíamos usar Programação Dinâmica.

Pensando na formulação recursiva

$$c[i,j] = \begin{cases} 0 & \text{se } S_{ij} = \emptyset \\ \max\limits_{i \leq k \leq j} \{c[i,k] + 1 + c[k,j]\} & \text{se } S_{ij} \neq \emptyset. \end{cases}$$

- E se pudéssemos escolher uma atividade para acrescentar à solução ótima sem ter que resolver primeiro todos os subproblemas?
- Mas qual seria esta escolha?
  - Atividade que começa primeiro?
  - Atividade que requer o menor intervalo de tempo?
  - Atividade com menor número de conflitos?
  - Atividade com menor tempo de término?

Escolha gulosa

Após a escolha gulosa da atividade a_k, restará somente um subproblema composto por atividades que começam após o término de a_k.

Substrutura ótima

- Seja $S_k = \{a_i \in S : s_i \geq f_k\}$ (atividades que começam após o término de a_k).
- Escolhendo a_1, S_1 permanecerá como o único problema a ser resolvido.
- Se a_1 estiver na solução ótima, uma solução ótima para o problema original consistirá de a_1 e todas as atividades em uma solução ótima para o subproblema S_1.

Demonstrando a escolha gulosa

Teorema

Considere um subproblema qualquer não vazio S_k. Seja a_m uma atividade em S_k com o menor tempo de término. Então, a_m estará incluída em algum subconjunto de tamanho máximo de atividades mutuamente compatíveis de s_k.

Demonstrando a escolha gulosa

Teorema

Considere um subproblema qualquer não vazio S_k. Seja a_m uma atividade em S_k com o menor tempo de término. Então, a_m estará incluída em algum subconjunto de tamanho máximo de atividades mutuamente compatíveis de s_k.

Demonstração

Seja A_k um subconjunto de tamanho máximo de atividades mutuamente compatíveis em S_k, e seja a_j a atividade em A_k que tem o menor tempo de término.

- $a_j = a_m$: não há o que demonstrar.
- $a_j \neq a_m$: considere o conjunto $A'_k = A_k - \{a_j\} \cup \{a_m\}$. As atividades em A'_k são disjuntas, pois a_j é a primeira atividade a terminar em A_k e $f_m \leq f_j$. Visto que $|A'_k| = |A_k|$, concluímos que A'_k é um subconjunto de tamanho máximo de atividades mutuamente compatíveis de S_k e inclui a_m.

Abordagem gulosa

1. Escolher uma atividade para colocar na solução ótima.
2. Resolver o problema de escolher atividade entre as que são compatíveis com as já escolhidas.

Comparação com a primeira formulação

	# de subproblemas	# de escolhas
Primeira	2	$j - i - 1$
Gulosa	1	1

Algoritmo guloso recursivo

Parâmetros

- As atividades estão ordenadas por tempo de término.
- s/f contém os tempos de início/término de cada atividade.
- k define o subproblema S_k a ser resolvido.
- n é o tamanho do problema original.
- Inserir uma atividade fictícia a_0 com $f_0 = 0$, tal que o problema S_0 é todo o conjunto de atividades S. A chamada inicial é $\textsc{Recursive-Activity-Selector}(s, f, 0, n)$.

$\textsc{Recursive-Activity-Selector}(s, f, k, n)$

```
1 m ← k + 1
2 while m ≤ n and s[m] < f[k] do
3 |   m ← m + 1
4 if m ≤ n then
5 |   return {a_m} ∪ Recursive-Activity-Selector(s, f, m, n)
6 else return ∅
```

Algoritmo guloso recursivo

RECURSIVE-ACTIVITY-SELECTOR(s, f, k, n)

```
1 m ← k + 1
2 while m ≤ n and s[m] < f[k] do
3 |   m ← m + 1
4 if m ≤ n then
5 |   return {a_m} ∪ RECURSIVE-ACTIVITY-SELECTOR(s, f, m, n)
6 else  return ∅
```

Complexidade

- Considerando que as atividades já estão ordenadas, a chamada RECURSIVE-ACTIVITY-SELECTOR$(s, f, 0, n)$ consome tempo $\Theta(n)$.
- Considerando todas as chamadas recursivas feitas, cada atividade é examinada exatamente uma vez no teste do laço **while** (linha 2).

Algoritmo guloso iterativo

GREEDY-ACTIVITY-SELECTOR(s, f)

```
1 n ← s.length
2 A ← {a_1}
3 k ← 1
4 for m ≤ 2 to n do
5 |   if s[m] ≥ f[k] then
6 |   |   A ← A ∪ {a_m}
7 |   |   k ← m
8 return A
```

O problema de escalonamento de tarefas

O problema

Dado um conjunto de n **tarefas** $S = \{1, 2, \cdots, n\}$ que requerem o uso exclusivo de um recurso comum (processador, por exemplo).
Cada tarefa i possui:

- **comprimento** t_i (tempo necessário para executá-la);
- **prioridade** p_i (ou peso).

Definimos o tempo de término c_i para a tarefa i como sendo a soma de todos os tempos de término das tarefas antecedentes a i, incluindo t_i.
Objetivo: Minimizar a soma ponderada dos tempos de término:

$$\min \sum_{i=1}^{n} p_i c_i.$$

Exemplo

Suponha que temos 3 tarefas com os seguintes comprimentos e pesos:

- $t_1 = 1,\ t_2 = 2,\ t_3 = 3.$
- $p_1 = 3,\ p_2 = 2,\ p_3 = 1.$

Pergunta 1

Se as tarefas são escalonadas na ordem $\{1, 2, 3\}$, quais são os tempos de término de $c_1,\ c_2$ e c_3?

Pergunta 2

Qual o valor da soma ponderada?

Pergunta 3

De quantas maneiras distintas podemos escalonar n tarefas?

Exemplo

Suponha que temos 3 tarefas com os seguintes comprimentos e pesos:

- $t_1 = 1,\ t_2 = 2,\ t_3 = 3.$
- $p_1 = 3,\ p_2 = 2,\ p_3 = 1.$

Pergunta 1

Se as tarefas são escalonadas na ordem $\{1, 2, 3\}$, quais são os tempos de término de $c_1,\ c_2$ e c_3?
Resposta: 1, 3 e 6.

Pergunta 2

Qual o valor da soma ponderada?
Resposta: $3 * 1 + 2 * 3 + 1 * 6 = 15.$

Pergunta 3

De quantas maneiras distintas podemos escalonar n tarefas?
Resposta: $n!$ maneiras.

Analisando a relação tempo vs. prioridade

$$\min \sum_{i=1}^{n} p_i c_i.$$

1. Se todas as tarefas possuem a mesma prioridade, qual a melhor maneira de escalonar as tarefas?
   Exemplo: $t_1 = 1$, $t_2 = 2$, $t_3 = 3$ e $p_i = 1$.

2. E se todas as tarefas possuem o mesmo comprimento, qual a melhor maneira de escalonar as tarefas?
   Exemplo: $p_1 = 3$, $p_2 = 2$, $p_3 = 1$ e $t_i = 2$.

Analisando a relação tempo vs. prioridade

$$\min \sum_{i=1}^{n} p_i c_i.$$

1. Se todas as tarefas possuem a mesma prioridade, qual a melhor maneira de escalonar as tarefas?
   Exemplo: $t_1 = 1$, $t_2 = 2$, $t_3 = 3$ e $p_i = 1$.
   Menor tempo primeiro.
2. E se todas as tarefas possuem o mesmo comprimento, qual a melhor maneira de escalonar as tarefas?
   Exemplo: $p_1 = 3$, $p_2 = 2$, $p_3 = 1$ e $t_i = 2$.
   Maior prioridade primeiro.

Resolvendo conflito

Caso geral

E se $p_i > p_j$ e $t_i > t_j$? Qual tarefa deve ser escalonada primeiro?

Ideia

Atribuir uma pontuação para cada tarefa de modo que a tarefa com maior prioridade e menor comprimento receba uma pontuação maior.

Sugestões de funções de pontuação

Resolvendo conflito

Caso geral

E se $p_i > p_j$ e $t_i > t_j$? Qual tarefa deve ser escalonada primeiro?

Ideia

Atribuir uma pontuação para cada tarefa de modo que a tarefa com maior prioridade e menor comprimento receba uma pontuação maior.

Sugestões de funções de pontuação

1. Colocar as tarefas em ordem decrescente conforme a pontuação $p_i - t_i$.
2. Colocar as tarefas em ordem decrescente conforme a pontuação p_i/t_i.
3. ???

Analisando funções de pontuação

Suponha $t_1 = 5$, $p_1 = 3$ e $t_2 = 2$, $p_2 = 1$.
A soma ponderada dos tempos de conclusão produzidos pelas funções de pontuação são:

Analisando funções de pontuação

Suponha $t_1 = 5$, $p_1 = 3$ e $t_2 = 2$, $p_2 = 1$.
A soma ponderada dos tempos de conclusão produzidos pelas funções de pontuação são:

1. Função $p_i - t_i$

- pontuação da tarefa 1: $3 - 5 = -2$.
- pontuação da tarefa 2: $1 - 2 = -1$.

Portanto, executar tarefa 2 antes da tarefa 1. Custo total será $2 * 1 + 7 * 3 = 23$.

Analisando funções de pontuação

Suponha $t_1 = 5$, $p_1 = 3$ e $t_2 = 2$, $p_2 = 1$.
A soma ponderada dos tempos de conclusão produzidos pelas funções de pontuação são:

1. Função $p_i - t_i$

- pontuação da tarefa 1: $3 - 5 = -2$.
- pontuação da tarefa 2: $1 - 2 = -1$.

Portanto, executar tarefa 2 antes da tarefa 1. Custo total será $2 * 1 + 7 * 3 = 23$.

2. Função p_i/t_i

- pontuação da tarefa 1: $3/5 = 0.6$.
- pontuação da tarefa 2: $1/2 = 0.5$.

Portanto, executar tarefa 1 antes da tarefa 2. Custo total será $5 * 3 + 7 * 1 = 22$.

Escolha gulosa

Teorema

A escolha de tarefas ordenadas em ordem decrescente conforme a razão p_i/t_i está sempre correta.

Demonstração

- Seja δ o escalonamento guloso e δ^* um escalonamento ótimo.
- Assumiremos que todos p_i/t_i são distintos e renomearemos as tarefas de forma que:

$$p_1/t_1 > p_2/t_2 > \cdots > p_{n-1}/t_{n-1} > p_n/t_n.$$

- Então o escalonamento guloso será: $\delta = \{1, 2, \ldots, n\}$.

Escolha gulosa

Demonstração – continuação

Suponha que $\delta \neq \delta^*$, então existem tarefas consecutivas i e j com $i > j$ com posições invertidas. Se trocarmos a ordem de i e j em δ^* (mantendo as outras tarefas inalteradas), percebemos que:

- o tempo de conclusão de qualquer outra tarefa k permanece inalterado;
- o tempo de conclusão da tarefa i aumenta t_j unidades;
- o tempo de conclusão da tarefa j diminui t_i unidades;

Então o tempo de conclusão ponderado seria:

$$\sum_{k=1}^{i-1} p_k c_k + p_i(c_i + t_j) + p_j(c_j - t_i) + \sum_{k=j+1}^{n} p_k c_k.$$

Mas $i > j \Rightarrow \frac{p_i}{t_i} < \frac{p_j}{t_j} \Rightarrow p_i t_j < p_j t_i$, ou seja, o benefício é maior que o custo e portanto é possível melhorar δ^*, o que contradiz a otimalidade de δ^*.

Concluindo

Subestrutura ótima

Se a tarefa com maior p_i/t_i for removida da solução ótima, então a solução restante para $n-1$ tarefas é ótima.

Exercícios

- Mostre que o problema possui subestrutura ótima.
- Escrever o algoritmo para este problema e analisar sua complexidade.