Projeto e Análise de Algoritmos

Método Guloso

Prof. Rodrigo de Barros Paes

[*rodrigo@ic.ufal.br*](mailto:rodrigo@ic.ufal.br)

[*https://sites.google.com/site/ldsicufal/disciplinas/projeto-e-analise-de-algoritmos*](https://sites.google.com/site/ldsicufal/disciplinas/projeto-e-analise-de-algoritmos)

Projeto de algoritmos

- Não existe a “bala de prata”
- O melhor que podemos fazer é pensar em princípios
- Exemplo
  - Dividir e conquistar
  - Guloso
    - Dijkstra
  - Programação dinâmica

Introdução

Muitas soluções para problemas de otimização ocorrem em uma sequência de passos

Um algoritmo guloso faz a escolha que parece melhor a cada momento

Toma decisões locais ótimas, na esperança de um ótimo global

Nem sempre leva a soluções ótimas (global)

Exemplo: Dijkstra

- O algoritmo tem somente uma chance de escolher o caminho
- Ele nunca volta pra analisar outra possibilidade
- [Explicação]

Comparando com Dividir e conquistar

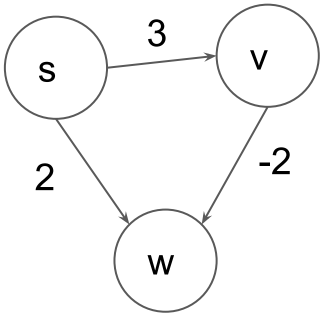
- D & Q
  - Dividir o problema em subproblemas
  - Resolve os subproblemas recursivamente
  - Combinar os resultados dos subproblemas em uma solução do problema original
  - Muitas vezes não é fácil de pensar e de implementar uma solução D&Q
  - Calcular a complexidade pode ser complicado
    - Recorrências, Teorema Mestre
  - Geralmente é fácil ver que ele dá respostas corretas

- Guloso
  - Fácil de aplicar e implementar
  - Calcular a complexidade geralmente é trivial
  - O grande problema é provar que o seu algoritmo guloso realmente está correto
  - É muito comum desenvolver um guloso que dê respostas erradas!!

Exemplo de guloso dando resposta errada

Usar Dijkstra para grafos com arestas com pesos negativos.

- Qual a resposta do algoritmo para o menor caminho entre s -> w?
- Qual seria a resposta correta?



A. 2 e 2
B. 2 e 0
C. 1 e 2
D. 2 e 1

Um problema de agendamento

O problema

- Um recurso compartilhado (ex: um processador)
- Muitas tarefas a serem realizadas (ex: processos)

Em que ordem devemos executar as tarefas?

Assuma que cada tarefa tem:

- prioridade (w)
- duração (l)

Conceito: tempo de completar uma tarefa

- O tempo C_j de terminar uma tarefa j é
  - Soma da duração de j com todas as durações das tarefas anteriores

Exemplo

- l_1=1, l_2=2, l_3=3

Agendamento:

#1	#2	#3

Quais os valores de C1, C2 e C3?

A. 1, 2, 3
B. 3, 5, 6
C. 1, 3, 6
D. 1, 4, 6

Mas como decidir a tarefa a ser executada?

Temos duas variáveis: w e l

Temos também esse conceito de tempo de completar uma tarefa C

Precisaremos de um critério objetivo para tomar uma decisão do que faz um agendamento melhor que o outro

Uma função objetiva

Iremos combinar o w e o l em uma função. Assim, nosso objetivo passará a ser de minimizar essa função,

Função:

- Soma ponderada dos tempos de finalização das tarefas

$$\min \sum_{j=1}^{n} w_j C_j$$

Uma função objetiva $\min \sum_{j=1}^{n} w_j C_j$

Voltando ao exemplo, seja cada tarefa definida como (l, w).

Suponha as 3 tarefas:

1. (1, 3)
2. (2, 2)
3. (3, 1)

Suponha que o agendamento foi 1, 2, 3. Logo, qual o valor da função objetiva?

A. 11
B. 24
C. 15

Uma função objetiva

$$\min \sum_{j=1}^{n} w_j C_j$$

1. (1, 3)
   $\boxed{C_1 = 1}$
2. (2, 2)
   C2= C1 + 2
   = 1 + 2
   $\boxed{= 3}$
3. (3, 1)
   C3 = C2 + 3
   = 3 + 3
   $\boxed{=6}$

= 3 * 1 + 2 * 3 + 1 * 6
= 3 + 6 + 6
= 15

Como fazer um algoritmo para resolver esse problema?

Lembre-se

Queremos: $$\min \sum_{j=1}^{n} w_j C_j$$

Obviamente, não queremos testar todas as combinações de agendamento.

Será que conseguiremos desenvolver um algoritmo guloso?

Refletindo

- O problema contém uma sequência de passos
- Você agenda uma tarefa, depois outra …
    - Essa é uma das características dos gulosos
- Precisaremos de um pouco de sorte :-)

- Vamos ser otimistas!!!
    - Suponhamos que exista um guloso

Como começar?

Que tal começar com casos pequenos e especiais onde saberíamos exatamente a resposta?

Caso especial 1

Todos os pesos (w) são iguais!

Quem você agendaria pra executar primeiro?

A. Tarefas com menor duração (l)
B. Tarefas com maior duração (l)

Caso especial 1

Exemplo:

tarefa : (l, w)

#1 (1, 1)

#2 (2, 1)

#3 (3, 1)

Agendamento 1: (#1, #2, #3)

- C_1=1
- C_2=1+2=3
- C_3=3+3=6

- W1C1 + W2C2 +W3C3
  = 1*1 + 1*3 + 1*6
  = 10

Agendamento 2: (#3, #2, #1)

- C_1=3
- C_2=3+2=5
- C_3=5+1=6

- W1C1 + W2C2 + W3C3
  = 1*3 + 1*5 + 1*6
  = 14

Ou seja, no caso de pesos iguais, devemos favorecer a tarefa de menor duração (l)

Caso especial 2

Todas as durações são iguais

Quem deve executar primeiro?

A. Tarefas com maior prioridade
B. Tarefas com menor prioridade

Caso especial 2

Exemplo:

tarefa : (l, w)

#1 (1, 1)

#2 (1, 2)

#3 (1, 3)

Agendamento 1: (#1, #2, #3)

- C_1=1
- C_2=1+1=2
- C_3=2+1=3

- W1C1 + W2C2 +W3C3
  = 1*1 + 2*2 + 3*3
  = 14

Agendamento 2: (#3, #2, #1)

- C_1=1
- C_2=1+1=2
- C_3=2+1=3

- W1C1 + W2C2 + W3C3
  = 3*1 + 2*2 + 1*3
  = 10

Ou seja, no durações iguais, devemos favorecer a tarefa de maior prioridade (w)

Próximo passo

Para os casos especiais, já aprendemos algo, mas e para o caso geral?

E se $w_i > w_j$ (deveríamos favorecer i) mas $l_i > l_j$ (deveríamos favorecer j) ?

De novo, vamos ser otimistas :-)

Ideia

Que tal tentar agregar esses dois parâmetros em um único valor

De forma que a gente tente manter os dois princípios que aprendemos:

- Maior peso (w) vem primeiro
- Menor duração (l) vem primeiro

Se conseguirmos isso (e tivermos um pouco de sorte), o agendamento é simplesmente ordenar as tarefas de acordo com esse valor mágico!

Pense você em como isso poderia ser feito

Ideia 1: diferença entre w e l

$$f(w, l) = w - l$$

Ideia 2: razão entre w e l

$$g(w, l) = w / l$$

- Se esses algoritmos geram agendamentos diferentes, então se houver algum correto, não poderá ser os dois!

- Será que a gente consegue já descartar algum?

Tentando “quebrar” o algoritmo guloso

Vamos tentar achar um exemplo onde os algoritmos produzirão saídas diferentes

Exemplo:
#id: (l, w)
#1: (5, 3)
#2: (2, 1)

Algoritmo
#1: 3 - 5 = -2
#2: 1 - 2 = -1
Logo, a ordem será:
#2, #1

Algoritmo 2:
#1: 3/5 = 0,6
#2: 1/2 = 0,5
Logo, a ordem será:
#1, #2

Um deles está errado

Aplicando a função:

$$\sum_{j=1}^{n} w_j C_j$$

$c_1 = 2$
$c_2 = 2+5=7$
$w_1c_1 + w_2c_2$
$1*2 + 3*7$
$= 23$

$c_1 = 5$
$c_2 = 5+2=7$
$w_1c_1 + w_2c_2$
$3*5 + 1*7$
$= 22$

Melhor resposta!

Conclusões até agora

Nosso algoritmo 1 não está correto

E quanto ao algoritmo 2? Até agora, não podemos dizer nada!

Será que ele está sempre correto?

Vamos tentar provar!

Prova de corretude do algoritmo 2

Prova

Tentaremos provar que o algoritmo 2 (ordena de forma decrescente de acordo com a razão w_j / l_j) está sempre correto!

Estratégia de prova: troca de argumentos!

Prova

Por enquanto vamos ignorar os empates.

Ideia

- Dada uma entrada arbitrária de n tarefas
- Vamos tentar provar por absurdo
- Seja σ = agendamento guloso ótimo
  - Se σ for falso, significa que existe um outro agendamento σ* que é ótimo
  - Se durante a prova mostrarmos que não existe esse σ*, então σ será ótimo
- Vamos tentar achar um agendamento ainda melhor que σ*
  - Se isso acontecer, significa que σ* não era ótimo
  - Ou seja, não existirá um σ* que é melhor que σ.
  - Isso significa que σ é verdadeiro
    - Ou seja, o guloso é ótimo

Prova

Seja:

- $\forall j$: w_j/l_j são distintos
- As tarefas serão nomeadas de 1 a n de acordo com a razão (e não de acordo com a entrada):
  - $w_1/l_1 > w_2/l_2 > \ldots > w_n/l_n$

Logo: o agendamento guloso σ nada mais é que a sequência de tarefas 1, 2, 3, …, n

σ =

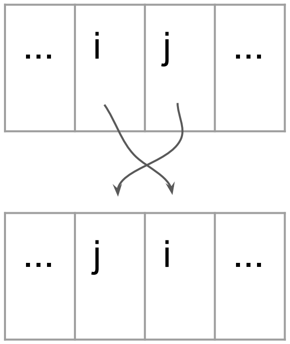
1	2	3	...	n

Prova

- Vamos supor agora que existe esse σ*, que é melhor que σ
- Se σ* ≠ σ, então deve existir um par de tarefas consecutivas i, j em σ* tal que i > j
  - Perceba que isso não ocorre em σ
- Suponha que nós iremos **trocar** a ordem de i e j em σ*

σ* =

novo agendamento =



i > j

O que aconteceria com os tempos de completar as tarefas (C) :

a) que não seja i ou j
b) i
c) j

Prova

$\sigma^* =$ | ... | i | j | ... |

novo agendamento = | ... | j | i | ... |

$i > j$

O que aconteceria com os tempos de completar as tarefas (C) :

a) que não seja i ou j
b) i
c) j

1. Sem informações suficientes / sobe / desce
2. Sem informações suficientes / desce / sobe
3. Não são afetadas / sobe / desce
4. Não são afetadas / desce / sobe

- *Se j passa a ser executada antes de i, então i vai demorar mais exatamente* C_j
- *E j vai demorar menos exatamente* C_i
- *Quanto aos pedaços antes e depois, simplesmente não são afetados*

Prova: analisando o custo-benefício da troca

$\sigma^* =$ | ... | i | j | ... | $i > j$

novo agendamento = | ... | j | i | ... |

- *O impacto será somente sobre i e j. Logo faremos o custo benefício dessas duas tarefas*
- *i vai piorar (custo)*
- *j vai melhorar (benefício)*

- ***Custo*** *será $w_i l_j$ (em relação ao valor anterior, essa tarefa i vai subir exatamente lj)*
- *O **benefício** será $w_j l_i$ (em relação ao valor anterior, essa tarefa vai melhorar li)*

Lembrando:
Em σ :
$w_1/l_1 > w_2/l_2 > \ldots > w_n/l_n$

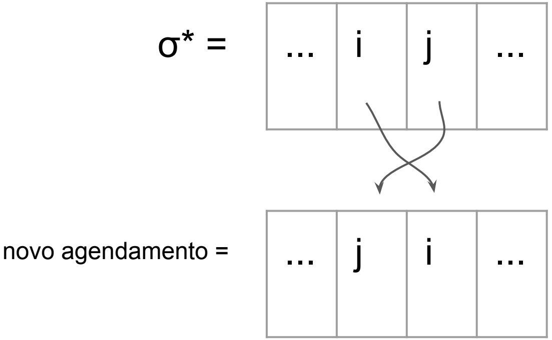
e dissemos que os índices eram:

1, 2, 3, 4, 5
Ou seja, quanto maior o índice menor a razão.

Logo, se $i > j$

$$\frac{w_i}{l_i} < \frac{w_j}{l_j}$$

Prova: analisando o custo-benefício da troca



i > j

$$\frac{w_i}{l_i} < \frac{w_j}{l_j}$$

$$w_i l_j < w_j l_i$$

custo < benefício

- ***Custo*** *será* $w_i l_j$
- *O* ***benefício*** *será* $w_j l_i$

Ou seja:

- σ* era um agendamento ótimo, ainda melhor que σ
- Mas trocamos uma posição do agendamento σ* e obtivemos uma solução ainda melhor
- Logo, σ* não é ótimo! O que contradiz o nosso argumento e portanto, provamos por absurdo!
    - Ou seja, σ* não existe! Fazendo com que nosso σ seja uma solução ótima

Mais um detalhe: os empates

Começamos a prova assumindo que $\forall j$: w_j/l_j são distintos

Ou seja, ignoramos os empates para simplificar

Agora vamos deixar de ignorá-los

Provando (agora com empates)

Provar que o algoritmo 2 que ordena as tarefas de forma não crescente (ou seja, podem conter empates ou mesmo serem todos iguais) está sempre correto!

Ideia

- Dada uma entrada arbitrária de n tarefas
- Seja σ = agendamento guloso ótimo
- Seja σ* qualquer outro agendamento
- Vamos tentar mostrar que σ é no mínimo tão bom quanto qualquer outro agendamento σ*
- Ou seja, σ será ótimo

Provando (cont.)

- As tarefas serão nomeadas de 1 a n de acordo com a razão:
  - $w_1/l_1 \geq w_2/l_2 \geq \ldots \geq w_n/l_n$
- Vamos tentar mostrar que σ é pelo menos tão bom quanto σ*
- Considere um agendamento qualquer σ*
  - Se σ* = σ, logo não há nada a fazer, pois eles seriam iguais e, portanto, provaríamos o argumento.
  - Senão,
    - Então existe um par i, j em σ* tal que i > j
    - Logo:

$$\frac{w_i}{l_i} \leq \frac{w_j}{l_j}$$

$$w_i l_j \leq w_j l_i$$

Ou seja, trocar i e j gera um benefício de líquido de $w_j l_i - w_i l_j \geq 0$

Ou seja, trocar só pode melhorar ou deixar como está, mas nunca piora.

Provando (cont.)

Suponha, por exemplo, σ*:

1	2	4	3	5	7	6

Inverte:

1	2	3	4	5	7	6

De acordo com o slide anterior, essa nova solução é melhor ou tão boa quanto o σ* anterior

Suponha que aplicamos a lógica até não haver mais i>j

1	2	3	4	5	7	6

- A cada iteração o novo σ* será tão bom quanto o anterior (ou melhor)
- A cada inversão, existirá uma inversão a menos pra fazer
- No pior caso existirão n/2 inversões
- E no final iremos sempre parar com um agendamento igual a σ
- Ou seja, a cada iteração, só fomos melhorando e no final o último será igual ao σ

- LOGO:
- σ é pelo menos tão bom quanto qualquer σ*
- Ou seja, ele é ótimo!

Seleção de atividades

Exemplo: Activity-selection

Qual o máximo de atividades que podem acontecer em um determinado local?

O local só comporta uma atividade por vez.

Cada atividade possui um horário de início e outro de fim

Exemplo

Atividade (i)	1	2	3	4	5
Início (s_i)	10	8	9	8	7
Fim (f_i)	13	11	10	10	9

intervalo: $[s_i, f_i)$

Qual o máximo de atividades podem ocorrer sem conflitos?

Colisão

Atividade (i)	1	2	3	4	5
Início (s_i)	10	8	9	8	7
Fim (f_i)	13	11	10	10	9

intervalo: $[s_i, f_i)$

Atividades i e j são compatíveis quando:

$s_i >= f_j \ || \ s_j >= f_i$

Ou seja, uma só começa depois que a outra termina.

Guloso

Critério: agendar a atividade que termina mais cedo e que não colide com as já agendadas.

Suponha o conjunto S, representando as atividades agendadas com essa estratégia, e queremos adicionar mais uma atividade:

- A última atividade será: max ($f_i \mid i \in S$)
- $f_{\text{último}} <= s_{\text{nova}} <= f_{\text{nova}}$
  - Segundo a estratégia, obrigatoriamente, a nova atividade começará depois do último do conjunto.

LOGO:

Novo agendamento também não terá conflitos

Guloso

Como vamos usar o fim da atividade para escolher, vamos reordenar a entrada pelo término

Atividade (i)	1	2	3	4	5
Início (s_i)	10	8	9	8	7
Fim (f_i)	13	11	10	10	9

O(nlgn)

Atividade (i)	5	4	3	2	1
Início (s_i)	7	8	9	8	10
Fim (f_i)	9	10	10	11	13

Sendo guloso

Atividade (i)	5	4	3	2	1
Início (s_i)	7	8	9	8	10
Fim (f_i)	9	10	10	11	13
	SIM				

Sendo guloso

Atividade (i)	5	4	3	2	1
Início (s_i)	7	8	9	8	10
Fim (f_i)	9	10	10	11	13
	SIM	NÃO			

Sendo guloso

Atividade (i)	5	4	3	2	1
Início (s_i)	7	8	9	8	10
Fim (f_i)	9	10	10	11	13
	SIM	NÃO	SIM		

Sendo guloso

Atividade (i)	5	4	3	2	1
Início (s_i)	7	8	9	8	10
Fim (f_i)	9	10	10	11	13
	SIM	NÃO	SIM	NÃO	

Sendo guloso

Atividade (i)	5	4	3	2	1
Início (s_i)	7	8	9	8	10 ✓
Fim (f_i)	9	10	10 ✓	11	13
	SIM	NÃO	SIM	NÃO	SIM

Sendo guloso

Atividade (i)	5	4	3	2	1
Início (s_i)	7	8	9	8	10
Fim (f_i)	9	10	10	11	13
	SIM	NÃO	SIM	NÃO	SIM

O(n)

Algoritmo guloso para o problema

```
in  = intervalos em ordem crescente de término
ans = ∅
last = -∞
while in ≠ ∅
{
    (s,f) = next_and_remove(in)
    if (last <= s)
    {
        ans = ans ∪ (s,f)
        last = f
    }
}
return ans;
```

Será que essa solução é ótima?

Caso base: A atividade de **menor término** pertence a uma solução ótima!

Demonstração:

- A atividade de menor término
  - $a = \min \{ f_i \mid i = 1, \dots, n\}$
- O agendamento máximo pode ser qualquer conjunto de atividades
  - $M \subset \{i=1, \dots, n\}$
- Suponha a atividade de menor término do agendamento máximo
  - $a' = \min \{f_i, i \in M\}$
- Vamos tentar mostrar que se removermos de a´ de M e incluirmos a, o agendamento também será máximo
  - $(M - \{a'\}) \cup \{a\}$ é máximo

Caso base: demonstração

- Suponha uma outra atividade i pertencente a M - {a´}
  - $i \in M - \{a'\}$
  - Como a´ é a atividade de menor término de M, e M não tem conflitos, então ela termina antes do início de qualquer outra atividade de M
    - $f_{a'} < s_i$
  - "a" é a atividade com menor término de todas, logo:
    - $f_a <= f_{a'} < s_i$
  - Logo não existe conflito entre a e i
- Assim, se removermos a´ e incluirmos a o agendamento continua válido
  - $(M - \{a'\}) \cup \{a\}$
- E por fim, as cardinalidades permanecem iguais
  - $| (M - \{a'\}) \cup \{a\} | = | M |$

Caso base demonstrado!

Passo indutivo

Teorema

- Considere um subproblema qualquer não vazio: S
- Seja m uma atividade em S com o menor tempo de término
- Então, m estará incluída em algum subconjunto de tamanho máximo de atividades compatíveis de S

Demonstração

- Seja K um subconjunto de tamanho máximo em S
- Seja j a atividade em K com menor tempo de término
- Se j == m
  - Então acabamos, pois essa afirmação é o mesmo que dizer que m está em um subconjunto máximo (K)
- Senão
  - Considere K´ = K - {j} ∪ {m} , ou seja, pegamos o subconjunto máximo, tiramos a atividade com menor tempo de término e colocamos uma outra atividade m
  - As atividades de K´ são disjuntas pois
    - j é a primeira atividade a terminar em K
    - mas m é primeira a terminar do conjunto maior (S)
    - Logo: $f_m <= f_j$
  - A cardinalidade não se altera: |K| = |K´|
  - Ou seja, K´ é um subconjunto máximo de S e inclui m
- Conclusão: a atividade menor término (m) estará incluída em algum subconjunto de tamanho máximo de atividades compatíveis de S

Demonstração

Ainda sobre a conclusão:

Podemos escolher repetidamente a atividade que termina primeiro, manter somente as atividades compatíveis com essa atividade, e repetir o processo até não restar mais nenhuma atividade.

Elementos da estratégia gulosa

Etapas

1. Expressar o problema de forma a fazer uma escolha e ficar com um subproblema pra resolver
2. Provar que sempre existe uma solução ótima para o problema original quando se usa a escolha gulosa
3. Tendo feito a escolha gulosa, mostrar que o subproblema restante:
    a. se combinarmos uma solução ótima para o subproblema com a escolha gulosa que fizemos, chegamos a uma solução ótima para o problema original

Exemplo de problema onde o guloso não funciona

Problema da mochila 0-1

Um ladrão assalta uma loja e encontra n itens. Quais itens ele deve levar para maximizar o valor do roubo?

Peso (w)	10	20	30
Valor (v)	60	100	120

A mochila tem uma capacidade máxima de 50Kg

Estratégia gulosa #1

Escolher o item de maior valor por quilo

i	0	1	2
Peso (w)	10	20	30
Valor (v)	60	100	120
v/w	6	5	4

Escolha	Mochila
0	50 - 10 = 40

Escolha	Mochila
1	40 - 20 = 20

Não seria possível escolher o 2, logo o guloso nos levou à solução: 0, 2

Não foi ótima!

Por que não funcionou?

Ao considerar incluir um item na mochila, temos que comparar a solução para o subproblema que inclui o item com a solução para o subproblema que não inclui esse mesmo item.

Muitos subproblemas sobrepostos!!

Para isso, vamos usar programação dinâmica!