

Projeto e Análise de Algoritmos I

Pedro Hokama

- [cls] Algoritmos: Teoria e Prática (Terceira Edição) Thomas H. Cormen, Charles Eric Leiserson, Ronald Rivest e Clifford Stein.
 - [timr] Algorithms Illuminated Series, Tim Roughgarden
 - Desmistificando Algoritmos, Thomas H. Cormen.
 - Algoritmos, Sanjoy Dasgupta, Christos Papadimitriou e Umesh Vazirani
 - Stanford Algorithms
<https://www.youtube.com/playlist?list=PLXFMmlk03Dt7Q0xr1PIAriY5623cKiH7V>
<https://www.youtube.com/playlist?list=PLXFMmlk03Dt5EMI2s2WQBsLsZ17A5HEK6>
 - Conjunto de Slides dos Professores Cid C. de Souza, Cândida N. da Silva, Orlando Lee, Pedro J. de Rezende
 - Conjunto de Slides do Professores Cid C. de Souza para a disciplina MO420
- Qualquer erro é de minha responsabilidade.

1 / 33

2 / 33

Paradigmas de Projeto de Algoritmos

- Divisão e Conquista
- Aleatorização
- Projeto por Indução
- **Algoritmos Gulosos**
- Programação Dinâmica
- etc, etc...

Algoritmos Gulosos (Gananciosos)

- Informalmente, um algoritmo guloso toma uma decisão que parece ser a melhor possível naquele momento (Escolha Gulosa).
- Não faz uma análise global para tomar essa decisão.
- Não se “arrepende” dessa decisão.

3 / 33

4 / 33

- Vantagens:
 - ▶ Normalmente é fácil criar um critério guloso.
 - ▶ Normalmente é fácil de implementar.
 - ▶ Normalmente a complexidade é baixa e fácil de analisar.
- Desvantagens:
 - ▶ A maioria dos critérios gulosos não vai levar a uma solução correta.
 - ▶ Provar que um critério guloso leva a solução pode ser trabalhoso.

5 / 33

Problema do Escalonamento Ponderado

- Suponha que você tenha uma máquina que faz alguma atividade.
- Trabalhos diferentes, cada trabalho i tem um tempo de execução l_i , e uma prioridade w_i , prioridade maior indica uma tarefa mais importante.
- Seja I o conjunto de todos os trabalhos, e c_i o tempo de conclusão do trabalho i .
- Em que ordem devo executar os trabalhos para minimizar

$$\sum_{i \in I} c_i \cdot w_i$$

7 / 33

- Estamos interessados em algoritmos que encontrem a solução correta para um dado problema.
- Algoritmos Gulosos também podem ser usados como heurísticas para encontrar boas soluções para problemas de otimização. Não necessariamente encontra a melhor solução, nem por isso estão incorretos.

6 / 33

Problema do Escalonamento Ponderado - Exemplo

- Três trabalhos $l_a = 1$, $l_b = 2$ e $l_c = 3$

0	1	2	3	4	5	6
a	b			c		

- Qual o tempo de término de cada atividade? $c_a = 1$, $c_b = 3$ e $c_c = 6$.
- Se as prioridades forem, $w_a = 3$, $w_b = 2$ e $w_c = 1$. Quanto é:

$$\sum_{i \in I} c_i \cdot w_i = 1 \cdot 3 + 3 \cdot 2 + 6 \cdot 1 = 15$$

8 / 33

Problema do Escalonamento Ponderado

- Poderíamos criar um algoritmo força-bruta para encontrar a ordem que minimiza $\sum_{i \in I} c_i \cdot w_i$.
- Testar todas as permutações possíveis.
- Mas existem $n!$ permutações.
- Podemos fazer melhor?

9 / 33

Problema do Escalonamento Ponderado - Critério Guloso

- Vamos tentar criar alguns critérios gulosos.
- Caso 1: Todos os trabalhos tem comprimento igual, mas prioridades diferentes. Qual seria um bom critério guloso? Maior ou menor prioridade na frente?
- Caso 2: Todas as prioridades são iguais, mas o tempo de execução é diferente. Qual seria um bom critério guloso? Curtas ou Longas primeiro?
- Caso 3: Se você tiver um trabalho curto com alta prioridade, e um longo com baixa prioridade? Qual executar primeiro?
- Caso 4: Trabalho longo com alta prioridade e outro curto com baixa prioridade?

10 / 33

Problema do Escalonamento Ponderado - Critério Guloso

- Vamos tentar criar uma pontuação para guiar a nossa escolha.
- Essa pontuação tem que ser crescer se a prioridade for maior.
- E tem que diminuir quanto mais longa.
- Podemos pensar em pelo menos 2 tentativas claras:
- Tentativa 1: $w_i - l_i$
- Tentativa 2: $\frac{w_i}{l_i}$

11 / 33

Problema do Escalonamento Ponderado - Critério Guloso

- | | | | | |
|-----|-----|-----------|-----|-----------|
| a | $ $ | $l_a = 5$ | $ $ | $w_a = 3$ |
| b | $ $ | $l_b = 2$ | $ $ | $w_b = 1$ |
- Tentativa 1: $w_i - l_i$
 - Tentativa 2: $\frac{w_i}{l_i}$

Tentativa 1:

$$w_a - l_a = 3 - 5 = -2$$

$$w_b - l_b = 1 - 2 = -1$$

Vai colocar b primeiro

$$c_a = 7 \text{ e } c_b = 2$$

$$\text{F.O.} = 7 \cdot 3 + 2 \cdot 1 = 23$$

Tentativa 2:

$$\frac{w_a}{l_a} = \frac{3}{5}$$

$$\frac{w_b}{l_b} = \frac{1}{2}$$

Vai colocar a primeiro

$$c_a = 5 \text{ e } c_b = 7$$

$$\text{F.O.} = 5 \cdot 3 + 7 \cdot 1 = 22$$

12 / 33

Problema do Escalonamento Ponderado - Corretude

- Suponha que renomeamos os trabalhos de forma que

$$\frac{w_1}{l_1} > \frac{w_2}{l_2} > \frac{w_3}{l_3} > \dots > \frac{w_n}{l_n}$$

- O critério da Tentativa 2 irá escalonar os trabalhos nessa ordem $\sigma^* = 1, 2, 3, \dots, n$.
- Suponha que exista uma outra sequencia ótima $\sigma \neq \sigma^*$ tal que $F.O.(\sigma) < F.O.(\sigma^*)$

13 / 33

Problema do Escalonamento Ponderado - Corretude

$$l_j \cdot w_i - l_i \cdot w_j$$

- Sabemos que se $i > j$, então

$$\begin{aligned} \frac{w_i}{l_i} &< \frac{w_j}{l_j} \\ \frac{(l_i \cdot l_j)w_i}{l_i} &< \frac{(l_i \cdot l_j)w_j}{l_j} \\ l_j \cdot w_i &< l_i \cdot w_j \end{aligned}$$

- portanto o que é economizado é maior que o gasto. Portanto a F.O. desse novo escalonamento é MENOR. Então σ não pode ser ótimo.

15 / 33

Problema do Escalonamento Ponderado - Corretude

- Em σ vai existir duas atividades em sequencia, i, j tal que i é executado antes de j mas $i > j$.

$$F.O.(\sigma) = A + c_i \cdot w_i + c_j \cdot w_j + B$$

- Suponha que trocamos essas duas atividades.

$$\begin{aligned} \text{novo custo} &= A + (c_i + l_j) \cdot w_i + (c_j - l_i) \cdot w_j + B \\ &= A + c_i \cdot w_i + l_j \cdot w_i + c_j \cdot w_j - l_i \cdot w_j + B \end{aligned}$$

- A diferença então é

$$l_j \cdot w_i - l_i \cdot w_j$$

14 / 33

Problema do Escalonamento Ponderado - Complexidade

- Podemos calcular todas as razões em $O(n)$
- Ordenar em tempo $O(n \log n)$
- Portanto a complexidade total do algoritmo é $O(n \log n)$

16 / 33

Representação de Caracteres

- Tradicionalmente caracteres são representados em binários através de uma codificação de tamanho fixo, ou seja, todo caracteres tem o mesmo número de bits, tradicionalmente 8 bits.
- Para simplificar, considere um alfabeto bem pequeno, de 6 letras { a, b, c, d, e, f }.
- Para representar esse alfabeto poderíamos usar uma representação de 3 bits.

a	b	c	d	e	f
000	001	010	011	100	101

- Para representar a palavra “babaca” precisaríamos de 18 bits:

001000001000010000

- Para representar um texto de 100.000 caracteres, precisaríamos de 300.000 bits.

17 / 33

Porém não funcionaria!

a	b	c	d	e	f
0	1	00	01	10	11

- Não é possível decodificar uma palavra. Por exemplo:

1010000

- Poderia ser “babaca”. Mas também poderia ser “eeca” ou “bdaaaa”
- O Problema é que o código de um caractere é prefixo de outro caractere.

19 / 33

Compressão de Dados

- Suponha agora que queremos comprimir o texto, de forma a ocupar uma quantidade de bits menor.
- Uma ideia é utilizar uma codificação de tamanho variável, ao invés da de tamanho fixo.

- Considere a seguinte codificação (Spoiler: Não vai funcionar)

a	b	c	d	e	f
0	1	00	01	10	11

- Para representar a palavra “babaca” precisaríamos agora de apenas de 7 bits:

1010000

- Para representar um texto de 100.000 caracteres, precisaríamos de menos de 200.000 bits.

18 / 33

Solução

- Codificação de tamanho variável livre de prefixo.

a	b	c	d	e	f
0	101	100	111	1101	1100

- Dessa forma uma codificação não será ambígua.

101010101000

- Com essa codificação a palavra “babaca” precisa de 12 caracteres.

20 / 33

Solução

- Utilizando essa codificação em um texto de 100.000 caracteres, em que cada um aparece com a seguinte frequência:

a	b	c	d	e	f
0	101	100	111	1101	1100
45%	13%	12%	16%	9%	5%

- Seriam necessários:

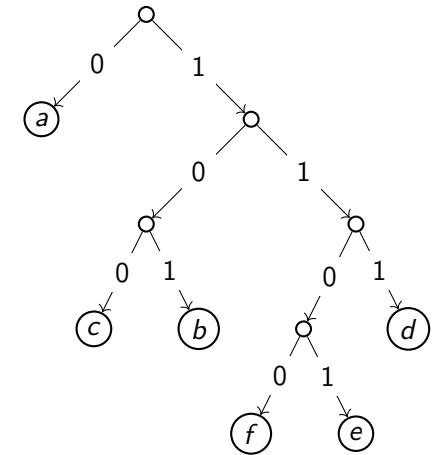
$$(45 \times 1 + 13 \times 3 + 12 \times 3 + 16 \times 3 + 9 \times 4 + 5 \times 4) \times 1000 = 224.000$$

- Compressões de até 90%
- Como escolher uma boa codificação?

21 / 33

Código de Huffman

- Criaremos uma árvore binária que representa uma codificação.
- A cada ramificação para a esquerda temos um bit 0 e cada ramificação a direita temos um bit 1.
- As folhas representam os caracteres.



22 / 33

Código de Huffman

- Construindo a árvore.

a	b	c	d	e	f
45%	13%	12%	16%	9%	5%

23 / 33

Código de Huffman

Algoritmo 1: Algoritmo de Huffman

Entrada: Um alfabeto C e suas frequências

Saída: Um código de tamanho variável livre de prefixo

- Seja Q uma fila de prioridade, inicialmente vazia.;
 - para** cada c em C **faça**
 - Q.insere(c);
 - enquanto** |Q| > 1 **faça**
 - Criar novo Nó V;
 - V.esq = ExtraiMin(Q);
 - V.dir = ExtraiMin(Q);
 - V.freq = V.esq.freq + V.dir.freq;
 - Q.inserir(V);
 - devolva** Árvore construída;
-

24 / 33

Complexidade

- Criar a fila de prioridade usando um Heap, pode ser feito em $O(n \log n)$ (na verdade dá para fazer até em $O(n)$).
- No laço da linha 3, cada iteração reduz o tamanho da fila em 1, então serão executadas $O(n)$ iterações.
- Dentro do laço, 2 extrações e uma inserção, que podem todas ser feitas em $O(\log n)$.
- Portanto a complexidade total do algoritmo é $O(n \log n)$

25 / 33

Escolha gulosa

Lema (Escolha gulosa)

Sejam x e y os nós correspondentes aos dois caracteres com as menores frequências em C . Então:

- existe **codificação ótima** em que x e y são folhas irmãs
- elas estão tão distantes da raiz quanto qualquer folha

Demonstração:

- considere uma árvore ótima T
- sejam a e b duas folhas irmãs mais profundas
- e sejam x e y as duas folhas de menor frequência
- construa uma árvore T' trocando a e x
- e depois uma árvore T'' trocando b por y
- vamos mostrar que T'' também é uma **árvore ótima**

27 / 33

Custo de uma codificação

Algumas notações

- T é uma árvore binária representando uma codificação
- $d_T(c)$ é a distância da raiz até o nó do caractere c
- $f(c)$ é a frequência do caractere c

Tamanho do texto codificado

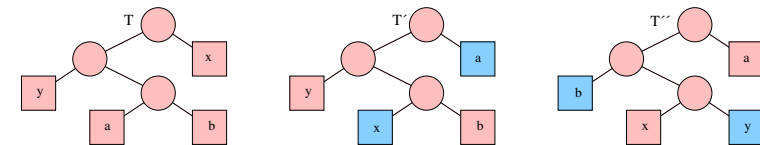
- o número de bits de um texto codificado por T é

$$B(T) = \sum_{c \in C} f(c) d_T(c).$$

- dizemos que $B(T)$ é o **custo** de T

26 / 33

Escolha gulosa (cont)



- aplicando a definição de B e cancelando termos idênticos,

$$\begin{aligned} B(T) - B(T') &= \sum_{c \in C} f(c) d_T(c) - \sum_{c \in C} f(c) d_{T'}(c) \\ &= f(x) d_T(x) + f(a) d_T(a) - f(x) d_{T'}(x) - f(a) d_{T'}(a) \\ &= f(x) d_T(x) + f(a) d_T(a) - f(x) d_T(a) - f(a) d_T(x) \\ &= (f(a) - f(x))(d_T(a) - d_T(x)) \geq 0 \end{aligned}$$

- assim, $B(T) \geq B(T')$ e, analogamente, $B(T') \geq B(T'')$
- como T é ótima, T'' também é ótima e o enunciado segue

28 / 33

Lema (Subestrutura ótima)

Suponha que

- x e y são os dois caracteres com menores frequências
 - $C' = C \setminus \{x, y\} \cup \{z\}$ é alfabeto com $f(z) = f(x) + f(y)$
 - T' é uma árvore ótima para o alfabeto C'
 - T é obtida de T' substituindo-se z por duas folhas x e y
- Então T é uma árvore ótima para C .

29 / 33

Comparando os custos de T e T'

- se $c \in C \setminus \{x, y\}$, então

$$f(c)d_T(c) = f(c)d_{T'}(c)$$

- como os códigos de x e y têm um bit a mais que z , sabemos que

$$\begin{aligned} f(x)d_T(x) + f(y)d_T(y) &= (f(x) + f(y))(d_{T'}(z) + 1) \\ &= f(z)d_{T'}(z) + (f(x) + f(y)) \end{aligned}$$

- portanto, $B(T) = B(T') + f(x) + f(y)$

30 / 33

Subestrutura ótima (cont)

Vamos mostrar que T é uma árvore ótima para C

- considere uma árvore ótima T^* para C
- pelo lema, supomos que x e y são folhas irmãs em T^*
- construa $\hat{T}$ a partir de T^* trocando x e y por z
- fazendo $f(z) = f(x) + f(y)$, temos $B(\hat{T}) = B(T^*) - f(x) - f(y)$
- como $\hat{T}$ é viável para C' e T' é ótima, $B(T') \leq B(\hat{T})$
- daí,

$$\begin{aligned} B(T) &= B(T') + f(x) + f(y) \\ &\leq B(\hat{T}) + f(x) + f(y) \\ &= B(T^*) - f(x) - f(y) + f(x) + f(y) \\ &= B(T^*) \end{aligned}$$

- então, de fato, T é ótima para C

31 / 33

Correção do algoritmo de Huffman

Teorema

Huffman constrói uma codificação ótima.

Demonstração:

- seja T a codificação devolvida pelo algoritmo
- associamos cada nó de T a um caractere distinto
- vamos mostrar que T é ótima por indução em $|C|$

32 / 33

Correção do algoritmo de Huffman (cont)

Considere $|C| = 1$

- nesse caso, o algoritmo devolve um único nó
- que é uma solução ótima

Agora, suponha que $|C| \geq 2$

- sejam x e y os caracteres escolhidos na primeira iteração
- seja z um caractere com $f(z) = f(x) + f(y)$
- considere um conjunto de caracteres $C' = C \setminus \{x, y\} \cup \{z\}$
- a árvore devolvida pelo algoritmo para C' é $T' = T - x - y$
- como $|C'| < |C|$, por hipótese de indução, T' é ótima para C'
- então, pela subestrutura ótima, T é ótima para C