

Apostila Introdutória de Algoritmos

Guilherme D. da Fonseca
Celina M. H. de Figueiredo

Versão rascunho para o curso de Análise de Algoritmos da Unirio
2009.

18 de Setembro de 2009

conjunto de vértices cuja excentricidade é mínima. Como a árvore tem pelo menos 3 vértices, o centro não será removido nesse processo. \square

Nosso algoritmo é, então, bastante simples. A cada iteração removemos todas as folhas da árvore. Quando sobrar apenas 1 ou 2 vértices, retornamos este(s) vértice(s) como o centro.

A complexidade de tempo do algoritmo é linear. Primeiro, construímos uma lista com todas as folhas. Em seguida, removemos todas as folhas, construindo uma lista das novas folhas criadas nesse processo. Ou seja, ao removermos uma folha f , verificamos se o vértice adjacente a f passa a ter grau 1. Em caso afirmativo, adicionamos o vértice adjacente a f na lista de folhas criadas. Repetimos este procedimento até restarem apenas 1 ou 2 folhas. Como a complexidade de tempo de cada etapa de remoção de folhas é linear no número de folhas removidas e nenhum vértice é removido mais de uma vez, a complexidade de tempo é linear no número de vértices.

4.2. Seleção do k -ésimo

Vários algoritmos se baseiam em dividir um conjunto S em dois conjuntos S_1 e S_2 de aproximadamente o mesmo tamanho. Muitas vezes, é útil adicionar a propriedade que os elementos de S_1 são menores que os elementos de S_2 . Porém, para fazermos esta divisão, precisamos determinar o elemento mediano de S , ou seja, o elemento de posição $\lfloor |S|/2 \rfloor$ em S ordenada. Para encontrarmos o elemento mediano, uma alternativa é ordenarmos S e, em seguida, pegarmos o elemento de posição $\lfloor |S|/2 \rfloor$. Esta alternativa leva tempo $O(n \lg n)$. Será que podemos fazer melhor?

Para resolvermos este problema, vamos primeiro torná-lo um pouco mais geral. Trataremos, então, do problema de determinar o k -ésimo menor elemento de S . Assim, fazendo $k = \lfloor |S|/2 \rfloor$, obtemos o elemento mediano.

PROBLEMA 11. *Dados um conjunto S e um inteiro k , determinar o k -ésimo menor elemento de S .*

A solução deste problema usa a técnica de simplificação de modo bastante complexo, por isso, apresentaremos a solução em partes. Inicialmente, vamos supor que temos acesso a uma função pronta de mediana aproximada, com complexidade de tempo linear no tamanho de S . Esta função recebe como entrada um conjunto S e retorna um elemento $x \in S$ tal que pelo menos 30% dos elementos de S são menores ou iguais a x e pelo menos 30% dos elementos de S são maiores ou iguais a x . Estamos considerando que S representa um conjunto, portanto não tem elementos repetidos. É fácil adaptar os algoritmos para funcionarem no caso de elementos repetidos.

Podemos usar a mediana aproximada de um conjunto para dividir este conjunto em duas partes, S_1 e S_2 , ‘aproximadamente’ de mesmo tamanho, com a propriedade que os elementos de S_1 são menores que os elementos de S_2 . Se desejamos encontrar o k -ésimo menor elemento, sabemos que S_1 contém este elemento se e só se $|S_1| \geq k$. Caso $|S_1| < k$, temos que o k -ésimo menor elemento de S está em S_2 . Não só isso, como podemos fazer afirmações ainda mais fortes. Caso $|S_1| \geq k$, então o k -ésimo menor elemento de S é o k -ésimo menor elemento de S_1 . Caso $|S_1| < k$, então o k -ésimo menor elemento de S é o $(k - |S_1|)$ -ésimo elemento de S_2 .

Deste modo, temos o algoritmo da figura 4.2 que encontra o k -ésimo menor elemento de S . Para analisarmos sua complexidade, escrevemos a recorrência

$$T(n) = T(7/10n) + n$$

Pode-se provar por indução que $T(n) = O(n)$. Para ganhar mais intuição sobre este limite, note que, na primeira iteração do algoritmo, são examinados n elementos. Na segunda iteração, são examinados no máximo $7/10n$ elementos. Na i -ésima iteração, são examinados no máximo $(7/10)^{i-1}n$ elementos. Esses valores formam uma progressão geométrica de termo inicial n e razão $7/10$, portanto, mesmo que somássemos infinitos termos, o que não é o caso, a soma não excederia $n/(1 - 7/10) = 10n/3 = O(n)$.

Entrada: S : Conjunto de números reais. k : Número inteiro tal que $1 \leq k \leq |S|$.Saída:O k -ésimo menor elemento de S .Observações:

$\text{MedianaAproximada}(S)$: Função que retorna um elemento $x \in S$ tal que pelo menos 30% dos elementos de S são menores ou iguais a x e pelo menos 30% dos elementos de S são maiores ou iguais a x .

 $\text{Selecionar}(S, k)$ Se $|S| < 10$ Ordene S e retorne $S[k]$ $x \leftarrow \text{MedianaAproximada}(S)$ $S_1 \leftarrow$ elementos de S menores ou iguais a x $S_2 \leftarrow$ elementos de S maiores que x Se $k \leq |S_1|$ Retorne $\text{Selecionar}(S_1, k)$

Senão

Retorne $\text{Selecionar}(S_2, k - |S_1|)$

FIGURA 4.2. Primeira parte da solução do problema 11

Como podemos construir a função que calcula a mediana aproximada? Imagine que agrupamos os elementos de S , arbitrariamente, em subconjuntos de 5 elementos. Vamos considerar que $|S|$ é múltiplo de 5 para simplificarmos nossa análise, porém caso $|S|$ não seja múltiplo de 5, não há problema em deixarmos um subconjunto com menos de 5 elementos. Podemos ordenar cada um destes sub-conjuntos de 5 elementos em tempo $O(1)$, assim ordenando todos os sub-conjuntos em tempo $O(|S|)$. Criamos, então, um conjunto M com as medianas de cada um dos sub-conjuntos de 5 elementos. Vale o seguinte teorema:

TEOREMA 4.2. *A mediana de M é maior ou igual a pelo menos 30% dos elementos de S e menor ou igual a pelo menos 30% dos elementos de S .*

DEMONSTRAÇÃO. Estamos considerando que $|S|$ é múltiplo de 5. Vamos chamar de x a mediana de M . Como x é a mediana de M , pelo menos metade dos elementos de M são menores ou iguais a x . Para cada $y \in M$, há pelo menos outros dois elementos de S que são menores que y , pois y é mediana de um conjunto de 5 elementos de S . O tamanho de M é $|S|/5$. Assim,

$$3 \cdot \frac{1}{2} \cdot \frac{|S|}{5}$$

elementos de S são menores ou iguais a x . O mesmo argumento prova que pelo menos 30% dos elementos de S são maiores ou iguais a x . A prova deste teorema está representada graficamente na figura 4.3. \square

Com este teorema, temos um algoritmo para obter a mediana aproximada, ilustrado na figura 4.4. Porém, encontramos uma situação bastante atípica. Nosso algoritmo para selecionar o k -ésimo chama nosso algoritmo de mediana aproximada (além de chamar a si próprio recursivamente) e nosso algoritmo de mediana aproximada chama nosso algoritmo de encontrar o k -ésimo de modo a obter a mediana exata de um conjunto cinco vezes menor. Vamos agora provar algo surpreendente. A complexidade de tempo de nosso algoritmo de selecionar o k -ésimo menor elemento é linear! Vejamos a recorrência abaixo, que define sua complexidade de tempo:

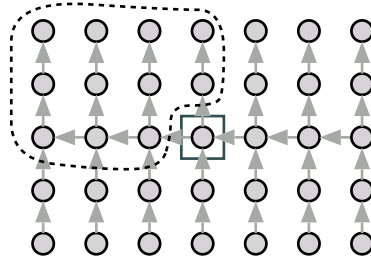


FIGURA 4.3. Conjunto de elementos com setas indicando relação ‘maior que’.

Entrada:

S : Conjunto de números reais, tal que $|S|$ é múltiplo de 5.

Saída:

Elemento $x \in S$ tal que pelo menos 30% dos elementos de S são menores ou iguais a x e pelo menos 30% dos elementos de S são maiores ou iguais a x .

MedianaAproximada(S)

Se $|S| < 10$

Ordene S e retorne $S[\lfloor |S|/2 \rfloor]$

Particione S em sub-conjuntos com aproximadamente 5 elementos

Ordene os sub-conjuntos

Crie o conjunto M das medianas dos sub-conjuntos de S

Retorne Seleccionar($M, \lfloor |M|/2 \rfloor$)

FIGURA 4.4. Segunda parte da solução do problema 11

$$T(n) = n + T(7n/10) + T(n/5).$$

A recorrência da complexidade de tempo do algoritmo tem três termos. O primeiro corresponde a complexidade de $O(n)$ necessária para particionar o conjunto S em S_1 e S_2 , assim como ordenar os sub-conjuntos de 5 elementos. O segundo corresponde a chamar o algoritmo recursivamente para S_1 ou S_2 . O terceiro termo corresponde a encontrar a mediana das medianas dos conjuntos de 5 elementos.

Como sabemos estar interessados em obter um algoritmo de complexidade de tempo linear, podemos provar por indução que $T(n) = \Theta(n)$. Vamos supor $T(n) = cn$. Por indução,

$$\begin{aligned} T(n) &= n + T(7n/10) + T(n/5) \\ &= n + 7cn/10 + cn/5 \\ &= n(1 + 7c/10 + c/5) \\ &= n(1 + 9c/10). \end{aligned}$$

Portanto, $c = 10$ e $T(n) = 10n$.

Caso não tivéssemos idéia de um palpite para fazermos nossa prova por indução, poderíamos pensar que $T(n)$, não sendo uma função sub-linear, deve ser convexa, pelo menos para n suficientemente grande. Neste caso vale que $T(a + b) \geq T(a) + T(b)$. Com isto, temos:

$$T(n) \leq n + T(9n/10).$$

Esta recorrência se parece bastante com a vista no início da sessão e é claramente linear pelo argumento da soma dos termos da progressão geométrica (agora com razão $9/10$).

Embora nosso algoritmo de mediana aproximada necessite de $|S|$ ser múltiplo de 5 para garantir a cota de 30%, caso $|S|$ não seja múltiplo de 5, a nossa cota será alterada apenas pela adição de uma constante, não alterando a complexidade de tempo do nosso algoritmo de seleção do k -ésimo menor elemento.

O algoritmo visto não é um exemplo típico de simplificação, pois resolve não um, mas dois sub-problemas em cada chamada. Porém, preferimos colocá-lo nesta sessão porque a motivação do projeto do algoritmo, como foi apresentado aqui, se baseia em uma idéia de simplificação.

4.3. Ponte do Fecho Convexo

No exercício 3.6, deve-se escrever um algoritmo que determina o fecho convexo de um conjunto S de n pontos no plano em tempo $O(n \lg h)$, onde h é o número de pontos do fecho convexo. Este algoritmo usa uma função que, dados um conjunto S de n pontos no plano e uma reta vertical r , obten as arestas do fecho convexo de S que interceptam r (figura 4.5(a)), em tempo $O(n)$. Nesta sessão, descreveremos esta função.

PROBLEMA 12. *Dados um conjunto S de n pontos no plano e uma reta vertical r , encontre as arestas do fecho convexo de S que interceptam r .*

A reta r interceptará duas arestas do fecho convexo, sendo uma do fecho convexo superior e outra do fecho convexo inferior. Nos concentraremos em obter a aresta do fecho convexo superior que intercepta r , também chamada de “ponte”. A obtenção da aresta do fecho convexo inferior que intercepta r é análoga. Também consideraremos que r realmente intercepta o fecho convexo de S , pois isto só não acontece caso todos os pontos de S estejam do mesmo lado de r .

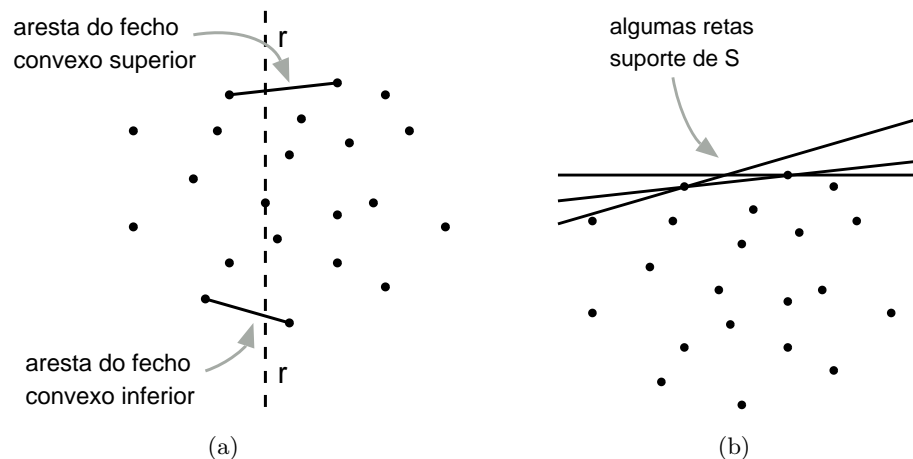


FIGURA 4.5. (a) Ponte do fecho convexo. (b) Algumas retas suporte de S .

Desejamos obter a ponte usando um algoritmo de simplificação. Nosso algoritmo procederá eliminando, a cada iteração, vértices que não são candidatos a serem um dos dois vértices da ponte. Começamos agrupando os pontos de S , arbitrariamente, em pares $(p_1, q_1), \dots, (p_{\lfloor n/2 \rfloor}, q_{\lfloor n/2 \rfloor})$. Consideramos que, nos pares (p_i, q_i) , p é o ponto mais à esquerda e q o mais à direita. Caso $|S|$ seja ímpar, um dos pontos fica sozinho e não concorre a ser descartado na iteração atual.

Como podemos fazer para descobrirmos pontos que, com certeza, não são candidatos a serem vértices da ponte? Definimos uma reta suporte de S como uma reta ρ que contém pelo menos um ponto de S e todos os demais pontos de S estão abaixo da reta ρ (figura 4.5(b)). Dada uma inclinação, é fácil determinar a única reta suporte com esta inclinação. Digamos que nos seja fornecida uma reta suporte qualquer. Caso a reta suporte contenha tanto pontos à esquerda da reta vertical r quanto à direita (provavelmente um ponto de cada lado), então a reta suporte contém a ponte e nosso problema está resolvido. Normalmente, porém, isto não acontecerá. Digamos que a reta suporte ρ contém apenas um ou mais pontos de S que estão à direita da reta vertical r .

Poderíamos rodar esta reta no sentido anti-horário, como um embrulho para presente, até encontrarmos a ponte. Não vamos fazer isto, porque o tempo gasto não seria linear. Mas segue desta observação um teorema importantíssimo para o nosso algoritmo:

TEOREMA 4.3. *Se ρ é uma reta suporte de S que contém apenas pontos à direita de r , então a ponte de S que intercepta r tem coeficiente angular maior que o de ρ . Analogamente, se ρ é uma reta suporte de S que contém apenas pontos à esquerda de r , então a ponte de S que intercepta r tem coeficiente angular menor que o de ρ .*

Vamos continuar supondo que ρ contém apenas pontos à direita de r . O outro caso é análogo. Digamos que um dos nossos pares de pontos (p_i, q_i) defina um segmento de coeficiente angular *menor* que o coeficiente angular de ρ . Neste caso, podemos dizer seguramente que q_i , o vértice da direita do par, não é um dos vértices da ponte. Vamos justificar com cuidado este fato, em princípio não muito óbvio. Suponha, por absurdo, que q_i seja um vértice da ponte. O coeficiente angular da ponte tem que ser menor ou igual ao coeficiente angular de (p_i, q_i) , pois caso contrário p_i estaria acima da reta que contém a ponte. Isto é absurdo, pois sabemos que a ponte tem coeficiente angular maior que ρ , que por sua vez tem coeficiente angular maior que (p_i, q_i) .

Deste modo, dada uma reta suporte ρ que contenha apenas pontos a *direita* de r , podemos descartar os vértices da direita de todos os pares (p_i, q_i) com coeficientes angulares menores que o de ρ . Analogamente, dada uma reta suporte ρ que contenha apenas pontos à *esquerda* de r , podemos descartar os vértices da esquerda de todos os pares (p_i, q_i) com coeficientes angulares maiores que o de ρ .

Não falamos até agora sobre como obter a inclinação conveniente para nossa reta suporte. Queremos que tanto o número de segmentos com coeficientes angulares maiores que o da reta suporte quanto com coeficientes angulares menores que o da reta suporte sejam grandes, pois não sabemos, em princípio, se nossa reta suporte conterá pontos à direita ou à esquerda de ρ . Usando o algoritmo da sessão anterior, podemos escolher a inclinação mediana dentre os segmentos (p_i, q_i) . Deste modo, descartaremos um dos pontos de metade dos segmentos, assim descartando $1/4$ do total de pontos. Como, a cada iteração, uma fração constante dos pontos é descartada, pelo argumento já apresentado de progressão geométrica, a complexidade de tempo do algoritmo é linear no número de pontos da entrada. O pseudo-código deste algoritmo está na figura 4.7.

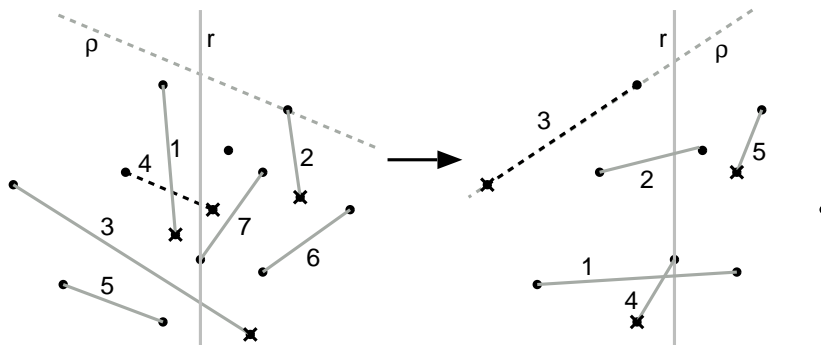


FIGURA 4.6. Duas iterações do algoritmo para encontrar a ponte. Segmentos numerados segundo os coeficientes angulares.

4.4. Resumo e Observações Finais

A técnica de simplificação consiste em reduzir um problema com uma entrada grande ao mesmo problema com uma entrada menor. A simplificação é um caso particular do paradigma de divisão e conquista, onde só é necessário resolver recursivamente um único problema menor. Quando o problema é pequeno o suficiente, podemos resolvê-lo diretamente.

Entrada: S : Conjunto de pontos no plano. r : Reta vertical que separa os pontos.Saída: (p, q) : Par de pontos da ponte.Observações: $\angle(p, q)$: Coeficiente angular do segmento (p, q) .Ponte(S, r) $R \leftarrow$ Conjunto de $\lfloor n/2 \rfloor$ segmentos $(p, q) \in S$ com $p.x < q.x$ $C_\rho \leftarrow$ coeficiente angular mediano dentre os segmentos de R $\rho \leftarrow$ reta suporte de coeficiente angular C_ρ Se ρ contém pontos à direita e à esquerda de r $p \leftarrow$ ponto de S mais à esquerda sobre ρ $q \leftarrow$ ponto de S mais à direita sobre ρ Retorne (p, q) Se ρ contém somente pontos de S à direita de r Para todo $(p, q) \in R$ Se $\angle(p, q) \leq C_\rho$ Remova de S o ponto q Retorne Ponte(S, r)Se ρ contém somente pontos de S à esquerda de r Para todo $(p, q) \in R$ Se $\angle(p, q) \geq C_\rho$ Remova de S o ponto p Retorne Ponte(S, r)

FIGURA 4.7. Solução do problema 12

No primeiro problema estudado, desejamos obter o centro de uma árvore. Simplificamos o problema através da remoção de todas as folhas da árvore, o que não altera o centro. Paramos quando a árvore obtida possuir apenas 1 ou 2 vértices, que são seu próprio centro.

Em seguida, examinamos o algoritmo para determinar o k -ésimo menor elemento de um conjunto, que engloba o caso particular de determinar o elemento mediano. Neste problema, conseguimos descartar 20% dos elementos a cada iteração do algoritmo. Para fazermos isso, entretanto, precisamos chamar o próprio algoritmo de seleção da mediana recursivamente.

Uma ponte do fecho convexo é a aresta do fecho convexo superior que intercepta uma reta vertical r . Consideramos o problema de dados um conjunto de n pontos e uma reta vertical r obter a ponte. Uma maneira trivial de resolver este problema seria determinando o fecho convexo do conjunto de pontos, o que leva tempo $\Theta(n \lg n)$. Porém, podemos resolvê-lo diretamente, gastando tempo $O(n)$. Para isso, usamos o algoritmo de cálculo da mediana de modo que conseguimos descartar um quarto dos pontos a cada iteração.

Exercícios

- 4.1) O maior divisor comum (mdc) de um par de números inteiros é o maior número que divide, sem deixar resto, os dois números do par. O algoritmo de Euclides encontra mdc de dois números inteiros por simplificação. Dados dois inteiros a, b , com $a \geq b$, se b divide a , então $mdc(a, b) = b$. Caso contrário, seja r o resto da divisão de a por b , então $mdc(a, b) = mdc(b, r)$. Prove que este algoritmo funciona corretamente.

- 4.2) Escreva um algoritmo para particionar um conjunto S com n elementos em m conjuntos S_1, \dots, S_m com $\lfloor n/m \rfloor$ ou $\lceil n/m \rceil$ elementos de modo que os elementos de S_i são menores que os elementos de S_{i+1} para i de 1 até $m - 1$. Uma solução trivial usa ordenação e tem complexidade de tempo $O(n \lg n)$. Porém, a sua solução deve ter complexidade de tempo $O(n \lg m)$.
- 4.3) Na sessão 3.3 vimos um algoritmo de divisão e conquista que encontra um conjunto independente máximo em uma árvore com pesos nos vértices. Resolva, usando simplificação, a versão mais simples do problema onde não há pesos nos vértices.
- *4.4) Dados um conjunto de desigualdades lineares de duas variáveis, da forma $ax + by \leq c$, e uma outra função linear $f(x, y)$, escreva um algoritmo que encontre o valor de (x, y) que maximiza $f(x, y)$ e satisfaz todas as desigualdades. Seu algoritmo deve ter complexidade de tempo linear no número de desigualdades. Este problema é chamado de programação linear com duas variáveis.
- *4.5) No algoritmo da figura 4.2, substitua a chamada a MedianaAproximada pela escolha de um elemento aleatório de S , com distribuição uniforme. Prove que o valor esperado da complexidade de tempo do algoritmo se mantém linear em $|S|$.