Capítulo 10

Notas sobre minorantes de complexidade

Neste capítulo faremos uma introdução às aplicações da teoria da Informação aos problemas de complexidade mínima, isto é, à procura de *minorantes* da complexidade da classe dos algoritmos que resolvem um determinado problema.

Antes de definir o que é um minorante ou um majorante de complexidade de um problema temos que convencionar o modelo de custos adoptado e a forma de o medir. Por exemplo

- Modelo de custos: tempo, espaço, número de comparações...
- Medição: pior caso, caso médio...

Definição 17 *Consideremos um problema P e seja n o comprimento dos dados. Diz-se que $f(n)$ é um* majorante da complexidade *de P para um certo modelo de custos $t(n)$ se existe um algoritmo A que resolve P com um custo de execução $t(n)$ que satisfaz $t(n) \leq f(n)$. Formalmente*

$$\exists\ A, \forall n :\ t(A(n)) \leq f(n)$$

onde, por exemplo no modelo do tempo no pior caso, é $t = \max_{|x|=n} t(A(x))$.

Note-se que para mostrar que $f(n)$ é majorante da complexidade de P basta encontrar um algoritmo específico que resolve P com um custo, no pior caso, que não excede $f(n)$.

Definição 18 *Consideremos um problema P e seja n o comprimento dos dados. Diz-se que $g(n)$ é um* minorante da complexidade *de P para um certo modelo de custos $t(n)$ se qualquer algoritmo*

que resolve P tem um custo $t(n)$ que satisfaz $t(n) \geq f(n)$ para uma infinidade de valores de n. Formalmente

$$\forall\ A,\ \exists_\infty n, \exists x,\ |x| = n :\ t(A(x)) \geq f(n)$$

□

Seja P um problema dado e seja $f(n)$ a complexidade (por exemplo, em termos de ordem de grandeza) do algoritmo mais eficiente A que resolve esse problema (eventualmente pode haver mais do que um); chamemos a $f(n)$ a complexidade óptima do problema. Normalmente o algoritmo óptimo A é desconhecido mas, tanto do ponto de vista teórico como do prático, é importante conhecer o mais exactamente possível — isto é, limitar a um intervalo tão pequeno quanto possível — a complexidade óptima $f(n)$.

A complexidade de qualquer algoritmo que solucione P é obviamente um majorante da complexidade óptima.

Determinar minorantes de $f(n)$ é muito mais difícil uma vez que, para se mostrar que $g(n)$ é um minorante de $f(n)$ é necessário considerar *todos* os algoritmos para P e mostrar que nenhum deles tem uma eficiência melhor do que $f(n)$; como o número desses algoritmos é infinito, um ataque directo à determinação de minorantes está fora de questão.

A Teoria da Informação pode ser utilizada para a determinação de minorantes de complexidade. A ideia básica, em linhas gerais é a seguinte. Para que um algoritmo consiga solucionar todas as "instâncias" de um determinado comprimento é necessário que obtenha, da "instância" em questão, informação suficiente para poder determinar a solução correcta. Por outras palavras, o facto de se "seleccionar" deterministicamente, de entre todas as soluções possíveis, a solução correcta, obriga a uma obtenção de informação da "instância" que representa, por si só, uma tarefa que qualquer algoritmo que resolva o problema dado tem que executar.

Neste trabalho discutiremos primeiro problemas em que a informação é obtida por pesagens em balanças de pratos. Seguidamente consideraremos a comparação entre valores numéricos como fonte de informação em problemas como a determinação de máximo, o "merge" de 2 sequências e a ordenação.

10.1 Exemplos introdutórios

10.1.1 Um problema simples

Existem 3 bolas, sendo 2 de peso igual e uma mais pesada; podemos fazer apenas uma "pesagem" numa balança de pratos que nos dá 3 resultados possíveis: ou cai para a esquerda ou para a direita

ou fica equilibrada. Qual a bola mais pesada? A resposta é simples: comparemos com a balança o peso da bola 1 com a bola 2:

- Se forem iguais, a bola 3 é a mais pesada de todas.
- Se a bola 1 pesa mais que a 2, então a bola 1 é a mais pesada de todas.
- Se a bola 2 pesa mais que a 1, então a bola 2 é a mais pesada de todas.

Neste caso, o número de soluções que o problema pode ter é 3 (qualquer uma das bolas pode ser a mais pesada) e o número de resultados possíveis das pesagens é também 3. Em geral podemos enunciar o seguinte Teorema.

Teorema 30 (Princípio da informação necessária) *Para que seja sempre possível chegar à solução é condição necessária que $a \leq b$ onde a representa o número de soluções do problema e b representa o número de casos que é possível distinguir com a informação obtida pelo algoritmo.*

Se são permitidas k pesagens, o número de resultados possíveis das pesagens é 3^k.

Problema

Existem 14 bolas, todas iguais, com excepção de uma que pode ser mais ou menos pesada que as outras; com 3 pesagens determinar qual a diferente e se é mais leve ou pesada que as outras.

Neste caso $a = 28$ pois qualquer uma das 14 bolas pode ser a mais leve ou a mais pesada e $b = 3^3 = 27$; como $a > b$, podemos afirmar que não existe solução!

Problema

Existem 5 bolas todas de pesos diferentes; determinar, com um máximo de 4 pesagens, a sequência das bolas ordenadas pelos seus pesos.

Temos $a = 5! = 120$, (todas as permutações das 5 bolas) e $b = 3^4 = 81$; mais uma vez o problema não tem solução. Qual o número mínimo de pesagens para que o problema *possa* ter solução? Admitamos que os pesos são tais que a balança nunca fica equilibrada (o que é consistente com o facto de os pesos serem todos diferentes). Cada pesagem só dá origem a 2 decisões possíveis. Assim temos que p, o número de pesagens, deve satisfazer $2^p \geq 120$, ou seja

$$p \geq \lceil \log 120 \rceil = 7$$

10.1.2 O problema das 12 bolas

O Teorema 30 anterior não só é útil para determinar casos de impossibilidade mas pode ser também usado para seleccionar as pesagens que podem constituir uma solução. Vejamos um problema com solução e procuremo-la.

> **Problema**
>
> Existem 12 bolas, todas iguais, excepto uma (que pode ser mais ou menos pesada que as outras); com 3 pesagens determinar a diferente e se é mais leve ou pesada.

Neste caso $a = 24$ e $b = 3^3 = 27$ pelo que é $a \leq b$ e o problema poderá ter ou não solução; o Teorema 30 só exprime uma condição necessária para haver solução.

Procuremos uma possível solução. Utilizaremos apenas pesagens com igual número de bolas em cada prato. Se, por exemplo, colocassemos 4 bolas no prato da esquerda e 3 no da direita, e a balança a caísse para a esquerda, não tiravamos qualquer conclusão: qualquer uma destas 7 bolas podia ser a mais pesada, a mais leve, ou podiam ser todas iguais!

Primeira pesagem

Temos $a = 24$ e $b = 27$; depois da primeira pesagem ficará $b' = 9$ onde b' representa o valor de b para o problema subsequente. Se colocassemos 1 bola em cada prato e a balança a ficasse equilibrada, restavam 20 hipóteses (10 bolas) para serem decididas em 2 pesagens; como $20 > 9$, isso seria impossível, pelo Teorema 30. A mesma conclusão se tira se usarmos 2 (pois $16 > 9$) ou 3 (pois $12 > 9$) bolas em cada prato. Se usarmos 5 bolas em cada prato e a balança a cair para a esquerda, teríamos que decidir uma de 10 hipóteses (uma das 5 bolas da esquerda pode ser mais a pesada, ou uma das 5 bolas da direita pode ser a mais leve) com 2 pesagens; ora $10 > 9$ e isso é impossível; com 6 bolas em cada prato ainda é pior. Conclusão: teremos que colocar 4 bolas em cada prato, seja $(1, 2, 3, 4) - (5, 6, 7, 8)$. Temos a considerar os casos

1. Fica equilibrado

   Uma das restantes 4 bolas, 9, 10, 11 ou 12 é a diferente; temos 8 hipóteses e $8 < 9$, tudo bem, só pode haver um resultado de pesagem desperdiçado. Façamos seguidamente a pesagem $(9, 10, 11) - (1, 2, 3)$, notando que 1, 2 e 3 são bolas iguais.

   (a) Se cai para a esquerda, ou 9 ou 10 ou 11 é a diferente e é mais pesada ($3 \leq 3$); com uma nova pesagem entre 9 e 10 a situação fica resolvida.

   (b) Se cai para a direita é análogo: ou 9 ou 10 ou 11 é a diferente e é mais leve ($3 \leq 3$); com uma nova pesagem entre 9 e 10 a situação fica resolvida.

(c) Se fica equilibrada, a bola 12 é diferente; comparada com, por exemplo, a bola 1, determinamos se é mais leve ou mais pesada (aqui a balança a não pode ficar equilibrada: é o resultado desperdiçado).

2. Cai para a esquerda

   Há 8 hipóteses: ou 1 ou 2 ou 3 ou 4 é a mais pesada; ou 5 ou 6 ou 7 ou 8 é a mais leve ($8 \leq 9$) Façamos a pesagem $(1, 2, 5) - (3, 4, 6)$.

   (a) Cai para a esquerda: 1 ou 2 é a mais pesada ou 6 é a mais leve; uma nova pesagem decide a situação $(1) - (2)$.

   (b) Cai para a direita; análogo: 3 ou 4 é a mais pesada ou 5 é a mais leve; fazemos a pesagem $(3) - (4)$.

   (c) Fica equilibrada: a mais leve é 7 ou 8; pesemos $(7) - (8)$ (não pode dar igual).

3. Cai para a direita

   Raciocínio análogo ao caso 2.

Exercício 97 *O problema análogo com 13 bolas é insolúvel (por isso é que se diz que 13 dá azar). Provar esse facto.*

10.2 Entropia, informação e minorantes de complexidade

10.2.1 Introdução

Se existem n hipóteses para uma situação e p_i é a probabilidade de o facto i ser verdadeiro a entropia deste estado de conhecimento é o seguinte valor que nunca é negativo

$$S = -\sum_{i=1}^{n} p_i \log p_i$$

Esta entropia mede a falta de informação ou incerteza deste estado de conhecimento; admitimos que os logaritmos são na base 2 embora isso não seja importante aqui. Uma entropia grande significa que é necessária uma grande quantidade de informação para descrever a situação. Uma entropia nula quer basicamente dizer que o conhecimento da situação é completo.

Vejamos em 2 situações o valor tomado pela entropia.

– Se temos a certeza que a hipótese 1 é válida, $p_1 = 1$ e $p_i = 0$ para $i \geq 2$. Resulta que o valor limite de S quando todos os p_i, com $i \geq 2$, tendem para 0 (e, consequentemente, p_1 tende para 1)

$$S = 1 \times \log 1 = 0$$

levando em consideração que $\lim_{p \to 0} p \log p = 0$.

– Se qualquer das n hipóteses é igualmente provável, $p_i = 1/n$ para todos os $1 \leq i \leq n$. Temos pois

$$S = -n \frac{1}{n} \log \frac{1}{n} = -\log \frac{1}{n} = \log n$$

Por exemplo, se $n = 16$, vem $S = 4$; a entropia (usando o logaritmo na base 2) mede, o número de bits de informação necessários para especificar completamente uma das hipóteses.

A fórmula da entropia não "caiu do céu". Shannon em *the Mathematical Theory of Information* demonstra que a *única* função $S(p_1, \cdots, p_n)$ que satisfaz os seguintes pressupostos razoáveis

– S é contínua em todos os p_i.

– Se todos os $p_i = 1/n$, a função $S(n)$ deverá crescer monotonicamente com n. Mais hipóteses (=acontecimentos) representam mais incerteza.

– Se uma escolha se desdobrar em duas escolhas sucessivas, a função ao original S deve ser a soma pesada dos S de cada escolha (ver o trabalho referido de Shannon para mais pormenores).

é do tipo

$$S = -k \sum_{i=1}^{n} p_i \log p_i$$

onde $k > 0$.

10.2.2 Informação e os problemas de pesagens

Uma pesagem do tipo considerado pode ter 3 resultados; seja:

n o número de hipóteses possíveis antes da pesagem.

n_1, n_2 e n_3 os números de hipóteses que restam após cada pesagem.

Supondo probabilidades iguais para as situações possíveis, a entropia é diminuida de, respectivamente, $\log(n/n_1)$, $\log(n/n_2)$ e $\log(n/n_3)$. Ora, como pretendemos limitar o número máximo de pesagens em todas as situações, a situação mais favorável seria quando a divisão fosse disjunta

(em hipóteses) e igual, isto é $n_1 = n_2 = n_3 = n/3$. Nesta hipótese a pesagem daria um "trit" de informação:

$$\text{Um trit} = \log_2 3 \text{ bits} \approx 1.585 \text{ bits}$$

Em termos de informação expressa em trits, a possível solução do problema das 12 bolas (ver o teorema30) resulta da desigualdade:

$$3 \text{ trits} \geq \log_3 24 \text{trits} \approx 2.893 \text{ trits}$$

Nota: Nota: a desigualdade $a \leq b$ deve ser válida em todos os nós da árvore.

Exercício 98 *Construir a árvore da solução dada para as 12 bolas, representando em cada nó:*

- *A pesagem efectuada.*
- *A entropia (antes da pesagem) expressa em trits.*
- *A entropia máxima, dado o número que falta de pesagens (0, 1, 2 ou 3).*

10.3 Minorantes de algoritmos de ordenação

Anteriormente determinamos minorantes do número de pesagens que é necessário efectuar com vista a determinar qual a bola de um determinado conjunto que é diferente das outras. Vamos agora aplicar o mesmo tipo de técnicas – baseadas na Teoria da Informação – para determinar minorantes de complexidade de problemas de ordenação. O leitor deve rever o conceito de "modelo externo dos dados", ver página 37.

Vejamos agora um problema que tem a ver com os importantes algoritmos de ordenação.

Problema (Ordenação)
Sejam n bolas de pesos diferentes; pretende-se colocá-las por ordem crescente de pesos usando apenas operações de comparação entre 2 bolas, isto é, pesagens com uma bola num prato e outra noutro. Note-se que a balança a nunca fica equilibrada uma vez que não existem 2 bolas de igual peso.

Aplicando directamente o Teorema 30 temos.

O número de hipóteses possíveis é $a = n!$.

O número de situações que se podem discriminar com c comparações é $b = 2^c$.

Resulta que qualquer algoritmo de ordenação que obtenha a informação através de comparações deve satisfazer $a \leq b$, ou seja,

$$2^c \geq n!$$

ou ainda

$$c \geq \lceil \log(n!) \rceil$$

Exercício 99 *Determine minorantes de c para os primeiros valores de n (por exemplo entre 1 e 6) e procure algoritmos de ordenação que se aproximem tanto quanto possível desses valores (no caso mais desfavorável).*

Usando a fórmula de Stirling como aproximação para o factorial

$$n! \approx \sqrt{2\pi n} \left(\frac{n}{e}\right)^n$$

temos (podiamos ser mais rigorosos; o sinal $\approx$ pode ser substituido por $>$ e este facto pode ser utilizado no que se segue).

$$2^c \geq \sqrt{2\pi n} \left(\frac{n}{e}\right)^n$$

ou seja (os logaritmos são, como é usual, na base 2)

$$c \geq \frac{1}{2} \log(2\pi n) + n \log n - n \log e$$

Vemos pois que qualquer algoritmo de ordenação baseado em comparações deve fazer (no pior caso) pelo menos cerca de $n \log n$ comparações, isto é deve ser pelo menos de ordem $O(n \log n)$.

Exercício 100 *Considere o algoritmo de ordenação "mergesort". Compare para* $n = 1, 2, 4, \ldots, 1024$ *o minorante teórico do número de comparações (*$\lceil \log(n!) \rceil$*) com o majorante número de comparações calculado para o algoritmo (ver apontamentos teóricos de Análise de Algoritmos).*

Concluimos pois que, em termos de ordens de grandeza (a afirmação não é válida se considerarmos os valores exactos do número de comparações), são conhecidos algoritmos de ordenação óptimos. Um exemplo é o "heapsort"; o "quicksort" não é óptimo, uma vez que as nossas conclusões são em termos do comportamento no pior caso (para o qual o "quicksort" é $O(n^2)$).

O teorema 30 dá para cada problema um limite mínimo da complexidade dos algoritmos respectivos; pode ser expresso, por outras palavras, da forma seguinte: o algoritmo tem que obter pelo menos tanta informação quanta é necessária para distinguir o caso particular do problema em questão. Para certos problemas — como o da ordenação — existem algoritmos próximos deste limite teórico; para outros, os melhores (teoricamente possiveis) algoritmos estão muito longe dele e há que recorrer a outros métodos, em geral mais complicados, para obter melhores minorantes.

10.4 Algoritmos de ordenação em que o custo é o número de trocas

Vamos considerar o problema de ordenar um vector de n elementos, usando como medida de custo o *número de trocas* que são efectuadas. O vector que se pretende ordenar só pode ser alterado através de trocas. Cada troca tem o custo 1; as outras operações têm o custo 0.

Um exemplo prático seria o seguinte: temos n livros numa prateleira e pretendemos colocá-los por ordem. Só há um custo: a troca entre 2 livros; todos os outras operações (como as comparações) têm custo 0.

Quantas trocas são necessárias?

Teorema 31 *$n-1$ trocas são suficientes.*

Dem. Basta exibir um algoritmo que efectue $n-1$ trocas. O leitor pode facilmente verificar que o método de ordenação "selecção do mínimo" está nessa situação. □

Teorema 32 *$n-1$ trocas são necessárias.*

Antes de vermos a demonstração deste resultado vamos definir o que é o grafo associado ao vector que se pretende ordenar.

Definição 19 *O grafo dirigido associado a um vector com n elementos (distintos) tem n vértices e tem um arco (i,j) sse, com a ordenação, o elemento que está na posição i do vector vai para a posição j.*

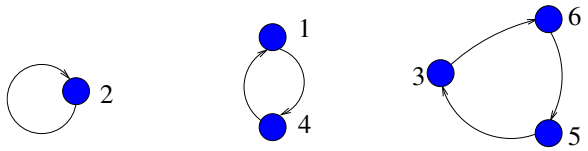
Por exemplo, o grafo correspondente ao vector `[40,20,600,10,30,50]` pode ser representado por

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 4 & 2 & 6 & 1 & 3 & 5 \end{pmatrix}$$

Nota importante. Os inteiros nos diagramas e nos grafos designam sempre posições (e nunca valores); por exemplo, o elemento que está na posição 5 vai para a posição 3.

A tabela anterior também pode ser representado pelo "produto de ciclos"

$$(1,4)(2)(3,6,5)$$



Repare-se que um vector está ordenado se o grafo associado tiver n ciclos, cada um deles com um só elemento.

O que acontece ao grafo se trocarmos um par de valores do vector? Sejam i e j as posições em que se encontram inicialmente esses valores.

- Se essas posições, i e j, estão no mesmo ciclo, esse ciclo desdobra-se em 2 ciclos. Exercício. demonstre esta afirmação, começando por trocar no exemplo dado as posições 3 e 5, bem como as posições 1 e 4.
- Se essas posições, i e j, estão em ciclos diferentes, esses 2 ciclos juntam-se num só ciclo. Exercício. demonstre esta afirmação.

Vamos agora provar o Teorema anterior.

Dem. Considere-se um vector cujo grafo tenha um só ciclo, por exemplo

`[2,3,4,...,n,1]`

Exercício. Verifique que só há 1 ciclo. Temos então

- Cada troca aumenta no máximo o número de ciclos de 1 (também pode reduzir de 1).
- O vector inicial dado tem 1 ciclo.
- O vector ordenado tem n ciclos.

Então qualquer sucessão de trocas que ordene o vector `[2,3,4,...,n,1]` tem, pelo menos, $n-1$ trocas. □

10.5 Minorantes de algoritmos de determinação do maior elemento

Um outro problema, mais simples que o da ordenação é o seguinte.

Problema (Máximo)

Determinar o maior de entre n valores distintos.

Neste caso a aplicação do Teorema 30 fornece um minorante de complexidade muito fraco: são precisas pelo menos $\log n$ comparações (o número de soluções possíveis é n); ora é possível mostrar que são necessárias $n-1$ comparações!

Exercício 101 *Temos n bolas; uma pesa 1, outra 2,... e a outra 2^{n-1} Mostre que é possível determinar a mais pesada em não mais de cerca de $\log n$ pesagens (aproximando-se do limite dado pelo teorema).*

Exercício 102 *Analise a complexidade mínima do seguinte problema. Existem n elementos distintos v_1, v_2,...,v_n ordenados por ordem crescente; é dado um valor x e pergunta-se: qual o menor i tal que $v_i > x$ (sendo $i = 0$ se x é menor que todos os elementos)? Por outras palavras, onde deveria x ser colocado na lista dos elementos?*

Notas sobre a solução

Aqui existem $n+1$ respostas possíveis; o teorema diz-nos que o número de comparações deve satisfazer: $2^c \geq n+1$, ou seja $c \geq \lceil \log(n+1) \rceil$.

Existe um algoritmo — o da pesquisa binária — que faz exactamente este número de comparações; não pode haver melhor!

Exercício 103 *Na seguinte situação determine (i) quantas comparações são necessárias pelo Teorema 30? (ii) E na realidade? (iii) Qual a entropia da situação inicial?*

- *A "pensa" num inteiro secreto entre 1 e 1000; B tenta adivinhar qual é, dizendo de cada vez um número; A responde: é maior, menor ou é esse.*

Teorema 33 (Majorante da complexidade) *$n-1$ comparações são suficientes para determinar o maior de n elementos distintos.*

Dem. Existe um algoritmo muito simples de determinação do maior elemento que estabelece este majorante. □

É interessante verificar que $n-1$ comparações são também necessárias!

Teorema 34 (Minorante da complexidade) *São*[1] *necessárias $n-1$ comparações para determinar o maior de n elementos distintos.*

Dem. Consideremos um qualquer algoritmo que determina o maior dos n elementos. Seja G um grafo não dirigido constituído por n vértices (associados aos elementos do vector) e com um arco entre os vértices i e j sse os elementos i e j do vector foram comparados entre si. Esse grafo evolui ao longo da execução do algoritmo.

No início, o grafo tem n componentes conexos, cada um com 1 vértice. Cada vez que se faz uma comparação o número de componentes conexos

- ou se mantém, se i e j estão no mesmo componente conexo,
- ... ou diminui de uma unidade, se i e j estão em componentes conexos distintos.

Suponhamos que havia 2 ou mais componentes no final da execução do algoritmo e seja x a resposta do algoritmo; seja ainda C_1 o componente a que pertence x e C_2 outro componente. O valor x não foi comparado com qualquer elemento de C_2 (não houve qualquer comparação entre os elementos de C_1 e os elementos de C_2). Assim, se aumentarmos todos os valores de C_2 de um mesmo valor suficientemente grande, a resposta x está errada, uma vez que o maior elemento está em C_2; e o resultado das comparações efectuadas pelo algoritmo é exactamente o mesmo.

Assim, no final o grafo tem que ter exactamente 1 componente conexa, pelo que tem que efectuar pelo menos $n-1$ comparações. □

10.6 Determinação do segundo maior elemento

Consideremos agora o problema de determinar o segundo maior elemento de um vector com n elementos distintos.

Um minorante de complexidade é fácil de estabelecer.

Teorema 35 (Minorante da complexidade) *São necessárias $n-1$ comparações para determinar o segundo maior elemento de entre n elementos distintos.*

[1] Usando o modelo de custos do pior caso, como sempre neste Capítulo. Isto é, poderíamos reformular o enunciado: "Para qualquer n podem ser necessárias...".

Dem. A demonstração do Teorema 34 é aplicável. □

Relativamente ao majorante de complexidade temos o seguinte resultado.

Teorema 36 (Majorante fraco da complexidade) *Bastam* $2n-3$ *comparações para determinar o segundo maior de entre* n *elementos distintos.*

Dem. Podemos determinar o segundo maior elemento de entre n elementos distintos pelo seguinte processo: (i) determinamos o maior elemento ($n-1$ comparações); (ii) percorremos de novo o vector com exclusão do maior elemento para obter o segundo maior elemento ($n-2$ comparações). □

Temos um intervalo de desconhecimento: entre $n-1$ e $2n-3$. O seguinte resultado reduz substancialmente esse intervalo, apresentando um algoritmo que efectua pouco mais que $n-1$ comparações.

Teorema 37 (Majorante forte da complexidade) *Bastam* $n + \log n - 2$ *comparações para determinar o segundo maior de entre* n *elementos distintos.*

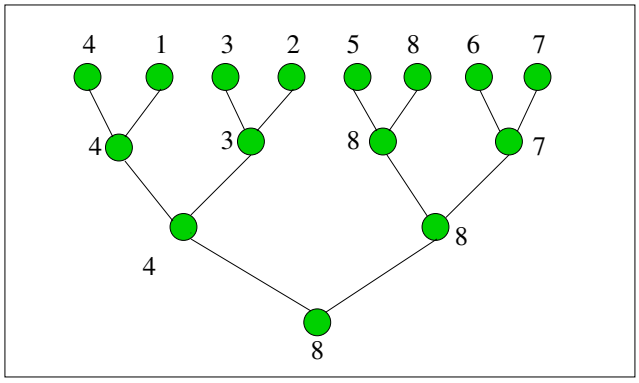
Dem. Consideremos o seguinte algoritmo em que supomos que n é uma potência de 2.

1) Os elementos são comparados aos pares ($n/2$ comparações);

2) Os maiores elementos de cada par são comparados aos pares ($n/4$ comparações);

... ... (até encontrar o maior elemento)

O número total de comparações é $n/2+n/4+\ldots+1 = n-1$ (demonstre esta igualdade). A figura seguinte ilustra a árvore resultante para o vector `[4,1,3,2,5,8,6,7]`.



Como o maior elemento foi necessariamente comparado com o segundo maior elemento (esta observação é atribuída a Lewis Carroll!), basta percorrermos a árvore da raíz para o topo com vista a determinar o segundo maior elemento; no exemplo da figura esse elemento é $\max\{4,7,5\} = 7$. O número de comparações é $\log n - 1$ (demonstre).

O número total de comparações é $(n-1)+(\log n-1) = n+\log n-2$. Quando n não é uma potência de 2, este resultado não é válido, mas o número de comparações é limitado por $n+\log n+c$ onde c é uma constante. □

10.7 Minorantes do problema de "Merge"

Consideremos agora o algoritmo do "merge".

Problema ("Merge")

São dadas 2 sequências de ordenadas de inteiros, uma com n e a outra com m elementos.

Pretende-se obter a sequência ordenada constituída pelos $m+n$ elementos.

Um algoritmo de resolver este problema é baseado na aplicação iterativa do seguinte passo. Os dois menores elementos das sequências dadas são comparados; o menor deles é retirado e colocado na sequência resultado. Quando uma das sequências dadas "chegar ao fim", o resto da outra é "passado" para a sequência resultado. Supondo todos os elementos distintos, é fácil ver que são feitas no máximo $m+n-1$ comparações.

Vejamos se este algoritmo se aproxima do limite teórico; o número de situações possíveis diferentes entre as duas sequências é:

$$\binom{m+n}{n}$$

Este número, igual a $(m+n)!/(n!m!)$, pode ser aproximado usando a fórmula de Stirling; vamos supor apenas o caso $m=n$.

$$\binom{2n}{n} = \frac{(2n)!}{(n!)^2} \approx (\pi n)^{-1/2} 2^{2n}$$

Sendo c o número de comparações efectuadas, deverá ser pois:

$$2^c \geq \left\lceil (\pi n)^{-1/2} 2^{2n} \right\rceil$$

ou seja:

$$c \geq \left\lceil 2n - \frac{1}{2}\log(\pi n) \right\rceil$$

Ora, como dissemos, o algoritmo do "merge" faz neste caso $2n-1$ comparações na pior hipótese. Vemos que, para n grande, o algoritmo é quase óptimo. Por exemplo, para $n=100$, o algoritmo faz 199 comparações, enquanto o minorante (confiando na aritmética do yap) é $\lceil 195.852 \rceil = 196$.

10.8 Conectividade de grafos

Consideremos o problema de determinar se um grafo não dirigido $G = (V, E)$ com n vértices é conexo.

Usamos o modelo exterior dos dados; o algoritmo faz perguntas do tipo "$(i, j) \in E$?", isto é, "existe o arco $i - j$?".

O seguinte resultado é trivial.

Teorema 38 *Um majorante de complexidade é* $n(n-1)/2$.

Dem. Na verdade, um grafo não dirigido fica inteiramente caracterizado pelos $n(n-1)/2$ elementos (0's ou 1's) abaixo da diagonal principal da matriz das incidências[2]. Assim, um algoritmo que efectue $n(n-1)/2$ perguntas pode obter informação completa sobre o grafo e determinar se ele é conexo. □

Vamos agora ver que é realmente necessário ler todos esses $n(n-1)/2$ bits para responder sempre de forma correcta.

Teorema 39 *Um minorante de complexidade é* $n(n-1)/2$.

Dem. Consideremos um qualquer algoritmo de conectividade que obtém informação sobre o grafo da forma indicada. Vamos mostrar que se o algoritmo não fizer todas as $n(n-1)/2$ perguntas "$(i, j) \in E$?", há sempre um grafo a que responde erradamente, isto é, que diz "conexo" quando o grafo não é conexo, ou vice-versa. Esse grafo onde o algoritmo falha é construído (por um "adversário"...) da forma seguinte

> Regra. A resposta a "$(i, j) \in E$?" é `não` a não ser que o grafo já desvendado[3] ficasse definitivamente desconexo[4].

Há uma propriedade (um invariante) que vai servir de base a esta demonstração.

> Invariante. Para qualquer par (i, j) ainda não desvendado, o grafo já desvendado não tem caminho entre i e j.
>
> Prova. Suponhamos que havia caminho no grafo já desvendado entre i e j e seja (i', j') o último arco já desvendado (resposta `sim`) desse caminho. Mas então, na altura em que se respondeu a "$(i', j') \in E$?" com `sim` não se seguiu a regra indicada atrás, uma

[2] Para efeitos de conectividade a existência de um arco entre um vértice e ele próprio é irrelevante.

[3] Entende-se por "grafo já desvendado" o grafo de n vértices que apenas ter arcos $i \to j$ se a resposta a uma pergunta "$(i, j) \in E$?" foi `sim`.

[4] Isso pode ser determinado analisando a conectividade do grafo que resulta de completar o grafo já desvendado com ligações entre todos pares ainda não desvendados.

vez que deveríamos ter respondido com `não`, pois haveria a possível ligação (i, j) que manteria (mais tarde) a conectividade do grafo já desvendado. □

Se um algoritmo não faz todas as $n(n-1)/2$ perguntas, consideremos um (i, j) não desvendado.

- Se o algoritmo diz "`conexo`", nós (o adversário...) apresentamos, como contra-exemplo, o grafo já desvendado sem qualquer outro arco. Na verdade, pelo invariante demonstrado atrás, não existe caminho entre i e j e o grafo não é conexo.
- Se o algoritmo diz "`não conexo`", essa resposta está errada, pois contradiz a estratégia de construção do grafo por parte do adversário.

Assim, um algoritmo que não faça todas as $n(n-1)/2$ perguntas não está correcto. □