Busca binária e Mergesort

Universidade Federal do Ceará - Campus de Crateús

Roberto Cabral
rbcabral@crateus.ufc.br

16 de Outubro de 2017

Estrutura de Dados

Busca em vetor ordenado

- Um vetor de inteiros $v[0 \ldots n-1]$ é crescente se:
  - $v[0] \leq v[1] \leq \ldots \leq v[n-1]$
- Um vetor de inteiros $v[0 \ldots n-1]$ é decrescente se:
  - $v[0] \geqslant v[1] \geqslant \ldots \geqslant v[n-1]$.
- Um vetor está ordenado se é crescente ou decrescente.
- Estudaremos como encontrar um valor x em um vetor ordenado.

O problema da busca em vetor ordenado

- Em lugar de perguntar onde x está no vetor $v[0 \dots n-1]$, é mais útil e mais conveniente perguntar onde x deveria estar.
- Nosso problema pode ser formulado assim:

Problema

dado um inteiro x e um vetor crescente $v[0 \dots n-1]$, encontrar um índice j tal que:
$v[j-1] < x \leq v[j]$.

- De posse de j, é muito fácil resolver o problema enunciado na introdução do capítulo: basta comparar x com $v[j]$.

O problema da busca em vetor ordenado

- Para tirar o máximo proveito dessa formulação do problema, devemos permitir que a solução j assuma os valores 0 e n.
- Nesses dois casos, a expressão $v[j-1] < x \leq v[j]$ deve ser interpretada com bom senso.
  - se $j == 0$, a expressão se reduz a $x \leq v[0]$.
  - se $j == n$, a expressão se reduz a $v[n-1] < x$.

- No exemplo a seguir, se x vale 5, a solução j do problema é 4; se x vale 8, a solução é 7; e se x vale 12, a solução é 12.

0										$n-1$
1	2	3	4	5	6	7	8	9	10	11

Busca sequencial

- Comecemos com um algoritmo óbvio, que examina um a um todos os elementos do vetor.

```
int buscaSequencial (int x, int n, int *v) {
  int j = 0;
  while (j < n && v[j] < x)
    j++;
  return j;
}
```

- Quantas iterações a função faz? Ou melhor, quantas comparações faz entre x e elementos de v?
- No pior caso, x é comparado com cada elemento do vetor e portanto o número total de comparações é n.
- O consumo de tempo da função é proporcional ao número de comparações que envolvem x, e portanto proporcional a n no pior caso.

Busca binária

- Existe um algoritmo muito mais rápido que a busca sequencial. Ele é análogo ao método que se usa para encontrar um nome em uma lista telefônica impressa. É claro que a ideia só funciona porque o vetor está ordenado.

```
int buscaBinaria (int x, int n, int *v) {
  int e, m, d;
  e = -1; d = n;
  while (e < d-1){
    m = (e + d)/2;
    if (v[m] < x) e = m;
    else d = m;
  }
  return d;
}
```

Busca binária

- Simples e limpo!
- Os nomes das variáveis não foram escolhidos por acaso: e lembra "esquerda", m lembra "meio" e d lembra "direita".
- O resultado da divisão por 2 na expressão $(e + d)/2$ é automaticamente truncado, pois as variáveis são do tipo `int`.
- Por exemplo, se e vale 6 e d vale 9, a expressão $(e + d)/2$ vale 7.

0					e			d		$n-1$
1	2	3	4	5	6	8	8	8	8	9

Corretude

- Para entender a função `buscaBinaria`, basta verificar que no início de cada repetição do `while`, imediatamente antes da comparação de e com $d-1$, vale a relação:

$$v[e] < x \leq v[d]$$

- Essa relação é, portanto, invariante.
- Note a semelhança entre o invariante e o objetivo $v[j-1] < x \leq v[j]$ que estamos perseguindo.
- O invariante vale, em particular, no início da primeira iteração: basta imaginar que $v[-1]$ vale menos infinito e $v[n]$ vale mais infinito.

Corretude

- No início de cada iteração, em virtude do invariante, temos $v[e] < v[d]$ e portanto $e < d$, uma vez que o vetor é crescente.
- No início da última iteração, temos $e \geqslant d - 1$ e portanto $e == d - 1$.
- A relação invariante garante agora que, ao devolver d, a função está se comportando como prometeu!
- Em cada iteração temos $e < m < d$. Logo, tanto $d - m$ quanto $m - e$ são estritamente menores que $d - e$.
- Portanto, a sequência de valores da expressão $d - e$ é estritamente decrescente. É por isso que a execução do algoritmo termina, mais cedo ou mais tarde.

Desempenho da Busca Binária

- Quantas iterações a função `buscaBinaria` executa?
- No início da primeira iteração, $d - e$ vale aproximadamente n.
- No início da segunda, vale aproximadamente $n/2$.
- No início da terceira, aproximadamente $n/4$.
- No início da k+1-ésima, aproximadamente $n/2^k$.
- Quando k passar de $\mathtt{log}\ n$, o valor da expressão $n/2^k$ fica menor que 1 e o algoritmo para.
- Logo, o número de iterações é aproximadamente $\mathtt{log}\ n$.

Versão recursiva da busca binária

- Para formular uma versão recursiva da busca binária é preciso generalizar ligeiramente o problema, trocando $v[0..n-1]$ por $v[a..b]$.
- A ponte entre a formulação básica e a generalizada é uma "função-embalagem" `buscaBinaria2` que apenas repassa todo o serviço para uma função recursiva `buscaBinR`.

```
int buscaBinaria2 (int x, int n, int *v) {
  return buscaBinR(x, -1, n, v);
}
```

Versão recursiva da busca binária

```
/* Recebe um vetor crescente v[e+1..d-1]
 e um inteiro x tal que v[e] < x <= v[d]
 e devolve um índice j em e+1..d tal que
 v[j-1] < x <= v[j].*/

static int buscaBinR (int x, int e, int d, int *v) {
  if (e == d-1) return d;
  else {
    int m = (e + d)/2;
    if (v[m] < x)
      return buscaBinR(x, m, d, v);
    else
      return buscaBinR(x, e, m, v);
  }
}
```

- Qual a profundidade da recursão na função buscaBinR? Ou seja, quantas vezes buscaBinR chama a si mesma?

Seção 1

Mergesort

Intercalação de vetores ordenados

- Antes de resolver nosso problema principal (Ordenar um vetor v) é preciso resolver o seguinte problema da intercalação (=*merge*): dados vetores crescentes $v[p..q-1]$ e $v[q..r-1]$, rearranjar $v[p..r-1]$ em ordem crescente.

p					$q-1$	q				$r-1$
1	3	3	4	5	6	2	6	6	8	9

- É fácil resolver o problema em tempo proporcional ao quadrado de $r-p$!
- Mas é possível fazer algo bem melhor! Como?
- Para fazer isso, será preciso usar uma área de trabalho, digamos $w[0..r-p-1]$, do mesmo tipo (e mesmo tamanho) que o vetor v[p..r-1].

Intercalação de vetores ordenados

```c
/* A função recebe vetores crescentes v[p..q-1]
 e v[q..r-1] e rearranja v[p..r-1] em ordem
 crescente.*/

static void intercala1(int p, int q, int r, int *v) {
  int i, j, k, *w;
  w = (int*)malloc((r-p) * sizeof (int));
  i = p; j = q;
  k = 0;
  while (i < q && j < r) {
    if (v[i] <= v[j])  w[k++] = v[i++];
    else  w[k++] = v[j++];
  }
  while (i < q)  w[k++] = v[i++];
  while (j < r)  w[k++] = v[j++];
  for (i = p; i < r; ++i)  v[i] = w[i-p];
  free (w);
}
```

Intercalação de vetores ordenados - Desempenho

- A função `intercala1` consiste essencialmente em movimentar elementos do vetor v de um lugar para outro (primeiro de v para w e depois de w para v).
- Ela consome tempo proporcional ao número de comparações entre elementos do vetor.
- Esse número é menor que $r - p$. Podemos dizer, então, que o consumo de tempo da função no pior caso é: proporcional ao número de elementos do vetor.

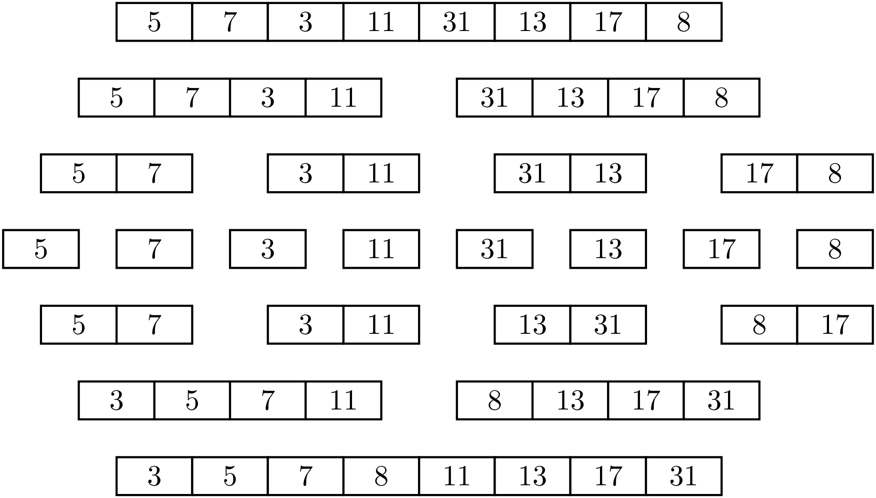
Técnica de ordenação de divisão e conquista

- Esta técnica consiste em dividir um problema maior recursivamente em problemas menores até que ele possa ser resolvido diretamente.
- A solução do problema inicial é dada através da combinação dos resultados de todos os problemas menores computados.
- A técnica soluciona o problema através de três fases:
    - Divisão: o problema maior é dividido em problemas menores.
    - Conquista: cada problema menor é resolvido recursivamente.
    - Combinação: os resultados dos problemas menores são combinados para se obter a solução do problema maior.

Merge Sort

- O Merge Sort foi proposto por John von Neumann em 1945.
- O algoritmo Merge Sort é baseado em uma operação de intercalação (*merge*) que une dois vetores ordenados para gerar um terceiro vetor também ordenado.
- O algoritmo pode ser construído a partir dos seguintes passos:
  - Divisão: o vetor é dividido em dois subvetores de tamanhos aproximadamente iguais.
  - Conquista: cada subvetor é ordenado recursivamente.
  - Combinação: os dois subvetores ordenados são intercalados para se obter o vetor final ordenado.

Merge Sort


5 7 3 11 31 13 17 8
5 7 3 11
31 13 17 8
5 7
3 11
31 13
17 8
5
7
3
11
31
13
17
8
5 7
3 11
13 31
8 17
3 5 7 11
8 13 17 31
3 5 7 8 11 13 17 31

Merge Sort

```
/* A função mergesort rearranja o vetor
 v[p..r-1] em ordem crescente.*/

void mergesort (int p, int r, int *v){
  if (p < r-1) {
    int q = (p + r)/2;
    mergesort (p, q, v);
    mergesort (q, r, v);
    intercala1(p, q, r, v);
    }
}
```

Marge Sort

- Ao aplicarmos a função `mergesort` a um vetor $v[0..n-1]$. O tamanho do vetor é reduzido à metade a cada passo da recursão.
- Na primeira rodada, a instância original do problema é reduzida a duas menores:
  - $v[0..n/2-1]$ e $v[n/2..n-1]$.
- Na segunda rodada, temos quatro instâncias:
  - $v[0..n/4-1], v[n/4..n/2-1], v[n/2..3n/4-1]$ e $v[3n/4..n-1]$.
- E assim por diante, até que, na última rodada, cada instância tem no máximo 1 elemento.
- O número total de rodadas é aproximadamente $\texttt{log}\ n$.
- Em cada rodada, a função intercala executa $O(n)$ movimentações de elementos do vetor $v[0..n-1]$. Assim, o número total de movimentações para ordenar $v[0..n-1]$ é aproximadamente $n\ \texttt{log}\ n$.

Marge Sort

Como ficaria a versão iterativa do Merge sort? (Para casa!)

Busca binária e Mergesort

Universidade Federal do Ceará - Campus de Crateús

Roberto Cabral
rbcabral@crateus.ufc.br

16 de Outubro de 2017

Estrutura de Dados