Ordenação

André Gustavo dos Santos

Departamento de Informática
Universidade Federal de Viçosa

INF 492 - 2012/1

Aplicações

Busca eficiente

Como testar de forma eficiente se um elemento s está num conjunto S ?

- Se S está ordenado, aplicar busca binária

Esta é certamente a aplicação mais comum da ordenação, mas existem muitas outras!!!

Aplicações

Teste de ocorrência única

Como testar se os n elementos de um dado conjunto S são todos distintos?

- Ordenar os itens; assim qualquer item repetido estará um ao lado do outro
- Testar se S[i] $\neq$ S[i+1] para $1 \leq i < n$

Apagar duplicados

Como apagar itens deixando apenas um de cada elemento repetido de S ?

- Ordenar os itens
- Percorrer os itens com dois índices: *ult*, para a última posição no array sem duplicados; e *i* para a próxima posição considerada
- Se S[ult] $\neq$ S[i], incrementar *ult* e copiar S[i] para S[ult]

Aplicações

Prioridade

Suponha um conjunto de tarefas a fazer, cada uma com uma data de entrega

- Ordenar as tarefas pela data de entrega (ou algo parecido) e as tarefas serão feitas na ordem certa
- Obs: se houver muitas inserções e retiradas de tarefas (como lembretes num calendário) melhor usar fila de prioridades. Mas se as tarefas não mudam durante a execução, basta ordenar

k-ésimo item

Como encontrar o k-ésimo maior item em um conjunto S ?

- Ordenar os itens, e o elemento estará em S[k]
- Isso pode ser usado também para encontrar (de uma forma não tão eficiente) o primeiro, último e a mediana

Aplicações

Frequência

Qual o item mais frequente num conjunto de itens S (i.e., a moda) ?

- Ordenar os itens e percorrer contando o número de ocorrências (os iguais estarão um ao lado do outro)

Reconstrução da ordem original

Como restaurar a ordem original de um conjunto de itens depois deles terem sido misturados para alguma aplicação ?

- Adicionar um campo nos dados de cada item, valendo i para o i-ésimo item
- Mover esse campo junto com o item sempre que o item for movido na aplicação
- No final, ordenar por esse campo e a ordem original será restaurada

Aplicações

União / Interseção

Como calcular união ou interseção de dois conjuntos de itens ?

- Ordenar os dois conjuntos de itens
- Repetidamente pegar o menor elemento entre os primeiros das listas, colocar no novo conjunto dependendo da operação e apagar o primeiro da lista apropriada

Encontrar um par

Como testar se há x e y em S tal que $x + y = z$ para algum valor alvo z ?

- Ordenar os elementos, e percorrer nos dois sentidos
- Note que enquanto S[i] cresce com i, seu possível par j tal que S[j]=z-S[i] decresce
- Então basta decrementar j apropriadamente enquanto incrementa i

Algoritmos de Ordenação

- Bubblesort, Selectionsort, Insertionsort, Heapsort, Mergesort, Quicksort, Shellsort, Radixsort, DistributionSort, Caminhamento in-order, ...
- Pra que tantos? Pra que saber tantas maneiras de fazer a mesma coisa? Aliás, pra que saber alguma se existe biblioteca com função já implementada?
- A razão principal é que as idéias por trás desses métodos reaparecem como idéias de algoritmos para diversos outros problemas. Por exemplo, estruturas de dados eficientes (heap, árvores) e estratégia dividir-para-conquistar.

Selectionsort

- O array tem uma parte ordenada e outra não ordenada
- Em cada iteração encontra o menor elemento da parte não ordenada e o move para o final da parte já ordenada.

...

```
selection_sort(int s[], int n)
{
  int i,j;    /* contadores      */
  int min;    /* índice do menor */

  for (i=0; i<n; i++) {
    min=i;
    for (j=i+1; j<n; j++)
      if (s[j] < s[min])
        min=j;
    troca(&s[i],&s[min]);
  }
}
```

Selectionsort

Características

- Faz muitas comparações, mas é muito eficiente no número de movimentações, somente $n - 1$ trocas (o mínimo possível no pior caso)
- Se for usada fila de prioridades para selecionar o menor elemento, passa de $O(n^2)$ para $O(n \log n)$ - vira um heapsort

Insertionsort

- O array tem uma parte ordenada e outra não ordenada
- Em cada iteração o próximo elemento da parte não ordenada é movido para a posição apropriada na parte já ordenada.

...

```
insertion_sort(int s[], int n)
{
  int i,j; /* contadores */

  for (i=1; i<n; i++) {
    j=i;
    while ((j>0) && (s[j] < s[j-1])) {
      troca(&s[j],&s[j-1]);
      j = j-1;
    }
  }
}
```

Insertionsort

Características

- Uma *inversão* numa permutação p é um par de elementos fora de ordem, isto é, i e j tais que $i < j$ mas $p[i] > p[j]$. Cada troca do inserção cancela exatamente uma inversão, e nenhum outro elemento é movido, então o número de trocas é igual ao número de inversões.
- Desta forma, como um array quase ordenado possui poucas inversões, o método é bastante eficiente nesse caso
- É um método *estável*: mantém a posição relativa de elementos iguais

Aplicação de métodos estáveis

- Se, de toda forma, os elementos iguais vão aparecer um ao lado do outro depois da ordenação, qual a vantagem de manter a ordem relativa usando um método estável?
- A vantagem aparece quando há mais campos nos dados a serem ordenados, e deseja-se preservar a ordem já existente de algum campo, enquanto se reordena pelo outro.
- Isto é particularmente interessante quando o critério de ordenação envolve múltiplos campos para resolver empates.

Exemplo de problema

Enunciado

Dado o número de medalhas de ouro, de prata e de bronze obtidas por cada país numa olimpíada, criar um ranking dos países pelo número de medalhas de ouro, utilizando a prata como desempate, e em seguida o bronze caso persista o empate (e se persistir pela ordem alfabética)

Exemplo de entrada

Brasil	3	4	8
Portugal	1	1	0
Cuba	2	11	11
EUA	36	38	36
Jamaica	6	3	2
China	51	21	28
Bélgica	1	1	0
Noruega	3	5	1

Saída esperada

China	51	21	28
EUA	36	38	36
Jamaica	6	3	2
Noruega	3	5	1
Brasil	3	4	8
Cuba	2	11	11
Bélgica	1	1	0
Portugal	1	1	0

Solução 1

Utilizar um critério complexo de comparação

Critério de comparação

```
if (x.ouro > y.ouro ||
    (x.ouro == y.ouro && x.prata > y.prata) ||
    (x.ouro == y.ouro && x.prata == y.prata && x.bronze > y.bronze) ||
    (x.ouro == y.ouro && x.prata == y.prata && x.bronze == y.bronze
       && x.nome < y.nome)

            // então x deve aparecer antes de y
```

Isso serve para qualquer método de ordenação que compara itens

Solução 2

Ordenar várias vezes, começando pelo critério **menos** importante

Ordenação

- ordenar de forma crescente pelo nome do país;
- ordenar de forma decrescente pelas medalhas de bronze;
- ordenar de forma decrescente pelas medalhas de prata;
- ordenar de forma decrescente pelas medalhas de ouro;

Só funciona se for usado um método estável!

Pois quem empata no bronze continua em ordem alfabética. Quem empata na prata continua em ordem de bronze. E quem empata no ouro continua em ordem de prata.

Solução 2

I) Depois de ordenar pelo nome

Bélgica	1	1	0
Brasil	3	4	8
China	51	21	28
Cuba	2	11	11
EUA	36	38	36
Jamaica	6	3	2
Noruega	3	5	1
Portugal	1	1	0

Por enquanto parece uma bagunça

II) Depois de ordenar pelo bronze

EUA	36	38	36
China	51	21	28
Cuba	2	11	11
Brasil	3	4	8
Jamaica	6	3	2
Noruega	3	5	1
Bélgica	1	1	0
Portugal	1	1	0

Estável, mantém Bélgica e Portugal na ordem alfabética

III) Depois de ordenar pela prata

EUA	36	38	36
China	51	21	28
Cuba	2	11	11
Noruega	3	5	1
Brasil	3	4	8
Jamaica	6	3	2
Bélgica	1	1	0
Portugal	1	1	0

Só mais um passo, e...

IV) Depois de ordenar pelo ouro

China	51	21	28
EUA	36	38	36
Jamaica	6	3	2
Noruega	3	5	1
Brasil	3	4	8
Cuba	2	11	11
Bélgica	1	1	0
Portugal	1	1	0

...*voilà!* (mantém desempate Noruega/Brasil p/ prata)

Funções prontas em C

Quicksort

```
#include <cstdlib>

void qsort(void *base, size_t nel, size_t width,
           int (*compara) (const void*, const void*));
```

- Ordena os primeiros `nel` elementos do array apontado por `base`, cada elemento de tamanho `width` bytes
- O último argumento é uma função que indica a ordem desejada: recebe apontadores para dois elementos e retorna:
  - negativo se o 1º deve vir antes do 2º
  - positivo se o 1º deve vir depois do 2º
  - zero se são iguais

Funções prontas em C

Exemplo

```
#include <cstdlib>

int comparaint(const void *x, const void *y)
{
  if (*(int*)x > *(int*)y) return 1;
  if (*(int*)x < *(int*)y) return -1;
  return 0;
}

int main()
{
  int a[100];
  ...
  qsort(a,100,sizeof(int),comparaint);
  ...
}
```

Funções prontas em C

Exemplo (em vez de comparar, retorna a diferença)

```
#include <cstdlib>

int comparaint(const void *x, const void *y)
{
  return *(int*)x - *(int*)y;
}

int main()
{
  int a[100];
  ...
  qsort(a,100,sizeof(int),comparaint);
  ...
}
```

Funções prontas em C

Solução 3 para o quadro de medalhas

```
int comparapais(const void *x, const void *y)
{
  pais xx = *(pais*)x;
  pais yy = *(pais*)y;

  if (xx.ouro > yy.ouro) return -1;      //Primeiro critério, ouro
  if (xx.ouro < yy.ouro) return 1;
  if (xx.prata > yy.prata) return -1;    //Se não diferenciou, usa prata
  if (xx.prata < yy.prata) return 1;
  if (xx.bronze > yy.bronze) return -1; //Se não diferenciou, usa bronze
  if (xx.bronze < yy.bronze) return 1;
  return strcmp(xx.nome,yy.nome);        //Se não, usa ordem alfabética
}

int main()
{
  pais lista[100];
  ...
  qsort(lista,n,sizeof(pais),comparapais);
  ...
}
```

Funções prontas em C

Busca Binária

```
#include <cstdlib>

void * bsearch(const void * key,
               void * base, size_t nel, size_t width,
               int (*compara) (const void*, const void*));
```

- Procura a chave `key` entre os primeiros `nel` elementos do array apontado por `base`, cada elemento de tamanho `width` bytes
- Retorna um apontador para um elemento do array cuja chave corresponda à chave `key` (ou NULL se não encontrado)
- O último argumento é uma função que indica a ordem, como no qsort

Funções prontas em C++

sort

```
template <class RandomAccessIterator>
  void sort ( RandomAccessIterator first, RandomAccessIterator last );

template <class RandomAccessIterator, class Compare>
  void sort ( RandomAccessIterator first, RandomAccessIterator last,
              Compare comp );
```

- `first`, `last`: iteradores (`iterator`)
- `comp`: função que recebe dois valores e retorna `true` se o 1º vem antes do 2º na ordem considerada, e `false` nos demais casos

- Ordena os elementos do intervalo [`first`,`last`) em ordem crescente
- Elementos comparados com `operador<` na 1ª versão e `comp` na 2ª
- Não há garantia de elementos iguais terminarem na mesma posição relativa

Funções prontas em C++

sort

```
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;

bool mycomp (int i,int j) { return (i<j); }

int main () {
  int vi[] = {32,71,12,45,26,80,53,33};
  vector<int> v (vi, vi+8);                  // 32 71 12 45 26 80 53 33

  // usando comparação default (operator<):
  sort (v.begin(), v.begin()+4);             //(12 32 45 71)26 80 53 33

  // usando função criada
  sort (v.begin()+4, v.end(), mycomp);       // 12 32 45 71(26 33 53 80)

  // ordenando tudo
  sort (v.begin(), v.end());                 //(12 26 32 33 45 53 71 80)

  ...
}
```

Funções prontas em C++

stable_sort

Mesmo que a `sort`, porém usa método estável, garantindo preservar a ordem relativa de elementos de mesmo valor

Exemplo de problema

Enunciado

Polly não tem problema em arranjar um parceiro. Aliás, seu maior problema é na verdade selecionar os melhores pretendentes. Ela sabe que um programa simplificaria muito esta seleção.

Ela gosta muito de dançar, e sabe que seu parceiro ideal teria 180 cm de altura. Seu primeiro critério é achar alguém o mais próximo dessa altura. Não importa se a diferença é pra mais ou pra menos.

Entre aqueles de mesma altura, ela prefere o mais próximo de 75 Kg, sem passar disso. Entre todos os acima desse peso, ele prefere o mais leve.

Se duas ou mais pessoas empatarem nessas características, ela quer seus nomes em ordem alfabética de sobrenome, e se houver mesmo sobrenome, em ordem alfabética de nome.

Ela quer apenas os nomes dos pretendentes, ordenados de acordo com os critérios estabelecidos.

Problema e solução retirados do livro Programming Challenges

Exemplo de problema

Exemplo de entrada

```
George Bush       195  110
Harry Truman      180   75
Bill Clinton      180   75
John Kennedy      180   65
Ronald Reagan     165  110
Richard Nixon     170   70
Jimmy Carter      180   77
```

Saída esperada

```
Clinton, Bill
Truman, Harry
Kennedy, John
Carter, Jimmy
Nixon, Richard
Bush, George
Reagan, Ronald
```

Exemplo de problema - solução

```
#define TAMNOME      30    /* tamanho maximo do nome */
#define NPRETEND    100    /* numero maximo de pretendentes */

#define BESTALTURA  180    /* melhor altura, em centimetros */
#define BESTPESO     75    /* melhor peso, em kilos */

#include <cstdio>
#include <cstring>

typedef struct {
  char nome[TAMNOME];   /* nome do pretendente */
  char sobre[TAMNOME];  /* sobrenome do pretendente */
  int altura;           /* altura do pretendente */
  int peso;             /* peso do pretendente */
} pretend;

pretend p[NPRETEND];    /* dados dos pretendentes */
int npretends;          /* numero de pretendentes *
```

Exemplo de problema - solução

```
le_pretends()
{
  int altura, peso;

  npretends = 0;

  while (scanf("%s %s %d %d\n",p[npretends].nome,
         p[npretends].sobre,&altura,&peso) != EOF) {

    p[npretends].altura = abs(altura - BESTALTURA);
    if (peso > BESTPESO)
      p[npretends].peso = peso - BESTPESO;
    else
      p[npretends].peso = -peso;

    npretends++;
  }
}
```

Exemplo de problema - solução

```
int compara_pretends(pretend *a, pretend *b)
{
  int result; /* resultado da comparacao */

  if (a->altura < b->altura) return(-1);
  if (a->altura > b->altura) return(1);

  if (a->peso < b->peso) return(-1);
  if (a->peso > b->peso) return(1);

  if ((result=strcmp(a->sobre,b->sobre)) != 0) return result;

  return(strcmp(a->nome,b->nome));
}
```

Exemplo de problema - solução

```
main()
{
  int i; /* contador */

  le_pretends();

  qsort(p, npretends, sizeof(pretend), compara_pretends);

  for (i=0; i<npretends; i++)
    printf("%s, %s\n",p[i].sobre, p[i].nome);
}
```