Introdução à Ciência da Computação II

Introdução à Busca em Memória Primária

Prof. Ricardo J. G. B. Campello

Créditos

- Os slides a seguir são adaptações e extensões dos originais em Pascal gentilmente cedidos por:
  - Prof. Rudinei Goularte

Sumário

- Introdução
- Termos Relacionados
- Tipos de Busca
- Busca Seqüencial
- Busca Binária (Recursiva)

3

Introdução

- Importância de estudar busca:
  - Tarefa muito comum em computação em geral
  - Fundamental para recuperar dados
    - de estruturas de dados em memória primária (**ALGI** & **ICCII**)
    - de estruturas de dados em arquivos (**ALGII**)
- Várias estruturas de dados e algoritmos disponíveis
  - cada um dos quais mais conveniente para uma situação específica
- Certos métodos de organização de dados permitem algoritmos de busca mais eficientes

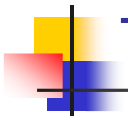
4

Introdução

- O problema de busca:

  "Dado um conjunto de elementos, onde cada um é identificado por uma **chave**, o objetivo da busca é localizar, nesse conjunto, o elemento que corresponde a uma chave específica"

5



Termos Relacionados

- **Estrutura de Dados (ED):**
  - termo genérico que se refere à alguma topologia para organização e armazenamento de dados
  - no presente contexto, pode ter outras denominações
    - por exemplo: **contêiner**, **tabela de busca**, etc
  - trata-se de um conjunto de **elementos**, usualmente também denominados de **itens** ou **registros**
    - inter-relacionados de acordo com a topologia de cada ED
    - por exemplo: listas, árvores, etc

6

Termos Relacionados

- **Chave:**
  - Existe ao menos uma chave associada a cada item, usada para diferenciar os itens entre si
    - Chave interna: chave está contida dentro do item, na forma de uma variável específica (p. ex. campo de um *struct* em C)
    - Chave externa: chaves estão contidas em uma tabela de chaves separada (índice) que inclui ponteiros para os itens

7

Termos Relacionados

- **Chave:**
  - Existe ao menos uma chave associada a cada item, usada para diferenciar os itens entre si
    - Chave primária: exclusiva para cada item (CPF, No. USP, etc)
    - Chave secundária: demais chaves (sobrenome, etc)
- **Algoritmo de busca:**
  - rotina que aceita um valor de chave **v** como argumento e tenta recuperar o item (ou itens) cuja chave é **v**

8

Tipos de Busca

- Depende da ED:
  - Um vetor de itens
  - Uma lista ligada de itens
  - Uma árvore de itens
  - etc
- Depende da Localização:
  - Totalmente na memória principal (**busca interna**)
  - Totalmente na memória auxiliar (busca externa)
  - Dividida entre ambas

9

Busca em Memória Interna

- As principais técnicas de busca em memória interna:
  - Busca Seqüencial (**ICCII** & **ALGI**)
  - Busca Binária (**ICCII** & **ALGI**)
  - Hashing (**ICCII**)
  - Busca em Árvores (**ALGI**)
- O objetivo é encontrar um dado item com o menor **custo computacional** possível (maior rapidez, eficiência)
  - Cada técnica possui vantagens e desvantagens

10

Busca Seqüencial

- A **busca seqüencial** é a forma mais simples de busca
- É aplicável a uma ED organizada linearmente:
  - vetor ou lista ligada
- Algoritmo rudimentar de busca seqüencial em um vetor A, com N posições, sendo x ao mesmo tempo o item e a chave procurada (vide caso mais geral em ALGI...):

```
for(i=0; i<N; i++)
      if (A[i] == x) return i;    {chave encontrada}
return -1;                        {busca falhou}
```

11

Busca Seqüencial

- Eficiência Computacional:

  (em número de comparações)
  - Se o item procurado for o primeiro?
  - Se o item procurado for o último?
  - Se a busca for mal sucedida?
  - Se for igualmente provável que o item apareça em qualquer posição?

12

Busca Seqüencial

- Eficiência Computacional:
  - Se o item for o primeiro (**melhor caso**):
    - 1 comparação
  - Se o item procurado for o último (**pior caso**):
    - N comparações
  - Se a busca for mal sucedida (**pior caso**):
    - N comparações
  - Se for igualmente provável que o item apareça em qualquer posição:
    - N/2 comparações, **em média**

13

Busca Seqüencial

- Para aumentar a eficiência:
  - Reorganizar continuamente a ED de modo que os itens mais acessados tendam a estar mais próximos do início. Por exemplo:
    - Método Mover-para-Frente: sempre que uma busca obtiver êxito, o item recuperado é colocado no início
    - Método Transposição: um item recuperado com sucesso é trocado com aquele imediatamente anterior

14

Busca Seqüencial

- Desvantagens do método Mover-para-Frente:
  - Uma única recuperação não implica que o item será freqüentemente acessado
    - pode implicar perda de eficiência para outros itens
  - É mais custoso computacionalmente no caso de buscas em vetores
    - demanda **lista ligada**

15

Busca Seqüencial

- Desvantagem do método de Transposição:
  - requer uma dada quantidade de reorganizações para que o tempo despendido com essas passe a ser compensado
  - apresenta, portanto, desempenho melhor para aplicações envolvendo quantidades maiores de buscas

16

Busca Seqüencial

- Busca seqüencial em ED linear ordenada:
  - A eficiência da operação de busca melhora se as chaves dos itens estiverem ordenadas
    - pela própria natureza do problema; ou
    - por imposição das operações de inserção e remoção de itens
  - Chave não encontrada não necessariamente implica o pior caso:
    - Em média, são necessárias "apenas" N/2 comparações (e não N) se as chaves estiverem ordenadas e a busca falhar
    - Porque... ?

17

Busca Binária

- É um meio mais eficiente de pesquisa do que a busca seqüencial
- É aplicável a uma ED organizada linearmente através de um vetor
- Requer que os dados estejam necessariamente ordenados por chave
- Idéia:
  - O argumento da busca é comparado à chave do item central do vetor
  - Se forem iguais, a busca terminará com sucesso
  - Caso contrário, a metade superior ou a metade inferior do vetor deverá ser pesquisada de modo análogo, a outra sendo desprezada

18

Busca Binária

- Algoritmo recursivo rudimentar de busca binária em um vetor A de inteiros, sendo o argumento x ao mesmo tempo o item e chave procurados entre A[L] e A[H]:

```
int BB(int A[], int, L, int H, int x){
   int mid;
   if (L > H) return -1;   {chave não existe}
   else {
      mid = (L + H)/2;
      if (x == A[mid]) return mid; {chave encontrada}
      else
         if (x < A[mid]) return BB(L, mid-1, x);
         else return BB(mid+1, H, x);
   }
}
```

19

Busca Binária

- Eficiência
  - Cada comparação reduz o número de possíveis candidatos (inicialmente N) por um fator de 2:
    - $\approx \log_2(N)$ comparações no pior caso
- Desvantagem:
  - Só pode ser usada para vetor (e ainda ordenado)
    - baseia-se nos índices do vetor como inteiros consecutivos
  - Por esta razão, não é adequada em situações com várias inserções e/ou remoções (vide listas em ALGI...)

20

Exercícios

- Programe uma função em C que receba um vetor A de inteiros, o tamanho N deste vetor, e um argumento de busca também inteiro x (que é ao mesmo tempo o item e a chave a ser procurada no vetor A) e realize uma **busca seqüencial** pela posição de x no vetor, ao mesmo tempo implementando a estratégia de **transposição**

21

Exercícios

- Explique em mais detalhes porque o número de comparações do pior caso em busca binária é proporcional a $\log_2(N)$, onde N é a quantidade de itens
- Incremente o **algoritmo BB** de forma que o vetor A armazene registros com dois campos, 1 deles a chave inteira x que já está sendo usada na versão apresentada e o outro uma string com até 30 caracteres. Retorne a string ao invés da localização do item no vetor

22

Para Saber Mais:

- Ziviani, N. *Projeto de Algoritmos*. Thomson, 2ª Edição, 2004.
- A. M. Tenembaum et al., *Data Structures Using C*, Prentice-Hall, 1990
- ...

23