Algoritmos e Estrutura de Dados

Busca binária

Baseado no material dos Profs:
Carlos F. S. Costa
Paulo R. de Oliveira

Prof. Nilton Luiz Queiroz Jr.

Pesquisa em memória primária

- Buscas são as tarefas mais frequentemente encontradas em programação;
- Existem diversos algoritmos que podem ser usados para realizar tal tarefa;

Pesquisa em memória primária

- Alguns casos usam-se estruturas dinâmicas para se fazer as pesquisas:
  - Listas Ligadas;
  - Árvores Binárias de Busca;
  - Árvores AVL;
- Alguns casos se costuma fazer busca em memória primária usando outras estruturas;
  - Em geral vetores

Busca em vetores desordenados

- Existem diversas maneiras de se buscar um valor em um vetor;
- O que determina qual maneira será a escolhida são algumas características do problema:
  - Quantidade de dados envolvidos;
  - Frequência com que ocorrem operações de:
    - Inserção;
    - Remoção;

Busca em vetor

- Podemos dividir a busca em vetor em duas principais maneiras:
  - Busca sequencial;
  - Busca binária;
- Para as implementações dessas buscas vamos assumir as estruturas de listas estáticas que foram vistas anteriormente

Busca sequencial

- Método de Pesquisa mais simples;
  - A partir do primeiro elemento, procura-se o elemento desejado de maneira sequencial;
  - A execução para quando se encontra o elemento procurado;
  - Ao encontrar o elemento retorna-se o índice do elemento;
  - Caso o elemento não seja encontrado, retorna-se um valor que está fora dos índices do vetor;
    - Por exemplo:
      - Valor -1 para vetores que iniciam no índice 0;
      - Valor 0 para vetores que iniciam no índice 1;

Busca sequencial

- Desse modo, podemos implementar a busca sequencial levando em consideração três elementos:
  - O vetor;
  - O tamanho do vetor;
  - O valor procurado;
- É implementada da mesma forma que uma busca em uma lista;

Busca sequencial

- De acordo com as entradas podemos calcular os custos em cada um dos casos:
  - Melhor caso: O elemento que é procurado é encontrado logo na primeira pergunta, ou seja, ocupa a primeira posição do vetor;
    - Tempo de execução: O(1);
  - Pior caso: O elemento não é encontrado no vetor, ou seja, é necessário comparar o elemento buscado com todos outros do vetor;
    - Tempo de execução O(n);
  - Caso médio: Para o caso médio assumimos uma distribuição de probabilidade onde todo elemento do vetor tem a mesma chance de ser o elemento procurado;
    - Tempo de execução O(n);

Busca binária

- Uma outra maneira de se fazer busca de elementos é com a busca binária;
- Diferente da busca sequencial que os elementos não dependem de uma ordem, a busca binária requer que os elementos do vetor estejam ordenados pela chave;
    - Ou pelo campo procurado;

Busca binária

- A ideia da busca binária é eliminar metade dos elementos do vetor a cada pesquisa restantes a serem pesquisado a cada iteração;
- Ou seja:
  - Dado um vetor:
    - Verifica-se se o elemento está no meio;
    - Se ele estiver então o valor foi encontrado;
    - Caso não esteja:
      - Se o valor for menor, procura-se o elemento na primeira metade;
      - Se o valor for maior, procura-se o elemento na segunda metade;
    - Se repete esse processo até que o elemento seja encontrado ou reste apenas um registro com chave diferente da procurada;

Exemplo

- Faça uma busca binária pelo número 4, mostre onde se encontram as variáveis esquerda, direita e meio a cada iteração

3	5	9	12	15	21	31	32
0	1	2	3	4	5	6	7

Exemplo

e			m				d
3	5	9	12	15	21	31	32
0	1	2	3	4	5	6	7

Exemplo

e	m	d					
3	5	9	12	15	21	31	32
0	1	2	3	4	5	6	7

Exemplo

m
e
d

3	5	9	12	15	21	31	32
0	1	2	3	4	5	6	7

Exemplo

d e

3	5	9	12	15	21	31	32
0	1	2	3	4	5	6	7

Exemplo 2

- Faça a busca binária pelo valor 21;

3	5	9	12	15	21	31	32
0	1	2	3	4	5	6	7

Exemplo 2

e			m				d
3	5	9	12	15	21	31	32
0	1	2	3	4	5	6	7

Exemplo 2

				e	m		d
3	5	9	12	15	21	31	32
0	1	2	3	4	5	6	7

Exemplo 2

3	5	9	12	15	21	31	32
0	1	2	3	4	5	6	7

Encontrou o valor 21 na posição 5;

Implementação

- Para a implementação da busca binária iremos considerar as estruturas de listas estáticas ja implementadas anteriormente;

```
struct tipo_item{
        int chave;
        /*outros campos*/
};
struct tipo_lista{
        int tam;
        struct tipo_item dados[TAM];
};
```

- É importante ter em mente que os dados desta lista devem estar previamente ordenados;
    - Ou seja, **lista.dados[i].chave <= lista.dados[i+1].chave**, para todo valor de i entre 0 e tamanho da lista-1;

Busca binária

```
int busca_binaria(struct tipo_lista *l, int chave){
    int e,d,m;
    e=0;
    d=l->tam-1;
    m=(e+d)/2;
    while(e<=d){
        printf("l[%d]=%d\n",m,l->dados[m].chave);
        if(l->dados[m].chave==chave){
            return m;
        }else if(l->dados[m].chave < chave){
            e=m+1;
        }else{
            d=m-1;
        }
        m=(e+d)/2;
    }
    return -1;
}
```

Busca binária

- O tempo de execução da busca binária está ligado a forma que as variáveis *direita* e *esquerda* são alteradas;
  - Suponha um vetor com n elementos
  - No início da primeira iteração *direita - esquerda* vale n-1;
  - Na segunda interação, *direita - esquerda* tem um valor próximo a n/2;
  - Na quarta iteração, *direita - esquerda* tem um valor próximo a n/4;
  - Na k-ésima iteração esse valor é de $n/2^k$;
  - Quando k passar de $\log_2 n$ o tamanho do vetor será menor que 1 e p algoritmo irá encerrar;
  - Assim, o número de iterações no algoritmo é por volta de $\log_2 n$, então temos um tempo no pior caso de $O(\log_2 n)$;
- Apesar do rápido acesso, manter o vetor ordenado pode ser caro;

Exercícios

1. Faça a busca binária dos elementos seguintes elementos no vetor abaixo, mostrando todos elementos que foram comparados, qual a posição do início e qual a posição do fim para cada elemento buscado:
   a. 23;
   b. 5;
   c. 29;

5	7	8	15	22	25	29	30

2. Implemente uma versão recursiva da busca binária.