Notas de Aula – Algoritmos e Estrutura de Dados 1 (AE42CP) – Algoritmos de Pesquisa (parte 1)
Prof. Jefferson T. Oliva

Busca é uma tarefa muito comum em computação, principalmente por causa da operação de recuperação de informação. Por exemplo, suponha que você trabalha em um crediário e quer verificar se um cliente está em dia com as suas contas. Apenas digitando o nome ou um documento não basta. Em outras palavras, a partir de uma dessas informações, o registro deverá ser procurado.

Vários outros métodos e estruturas de dados podem ser empregados para se fazer busca: ex.: biblioteca, lista telefônica, registros de clientes de loja.

Certos métodos de organização/ordenação de dados podem tornar o processo de busca mais eficiente.

Problema da 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.

Existem métodos de pesquisa que foram propostos para a aplicação em memória primária, secundária ou ambos (podendo haver combinação).

Existem vários métodos de busca, mas não existe um que se destaca em comparação com os outros, pois cada um tem as suas virtudes e fraquezas. O desempenho de algoritmos de busca depende de vários fatores, como tamanho e organização de dados. Por exemplo, o conjunto está ordenado? O conjunto é grande? Onde estão os dados?

Antes da apresentação de alguns métodos conhecidos, veremos a terminologia básica relacionada à pesquisa.

Terminologia Básica

Tabela ou Arquivo: termos genéricos, pode ser qualquer estrutura de dados usada para armazenamento e organização de coleções de dados.

Uma tabela é um conjunto de elementos, chamados registros.

Existe uma chave associada a cada registro. Uma chave pode ser:
- interna: contida dentro do arquivo em uma localização específica;
- externa: contida em uma tabela de chaves separada do arquivo que inclui ponteiros para os registros;
- primária: para todo arquivo existe pelo menos um conjunto exclusivo de chaves. Nesse caso, dois registros não podem ter o mesmo valor de chave;
- secundária: chaves não primárias, que não precisam ter seus valores exclusivos. Elas podem ser utilizadas para agrupar registros (por exemplo, os alunos podem ser agrupados por cursos). Em outras palavras, um registro de aluno pode ter RA e código de curso.

Uma tabela pode ser representada de várias formas: arranjo, lista encadeada, árvore, tabela hash, etc.

Uma tabela pode estar localizada totalmente na memória primária, totalmente na memória secundária, ou entre ambas

Algumas das operações básicas em TAD:
- Inserção
- Remoção
- Busca (ou pesquisa)

Por que falei dessas outras operações em TAD? Por que elas geralmente dependem da operação de busca.

Algoritmo de busca, formalmente, é o algoritmo que aceita um argumento ***a*** e tenta encontrar o registro cuja chave seja ***a***.

Principais algoritmos de busca:
Pesquisa sequencial
Pesquisa sequencial indexada
Pesquisa binária
Pesquisa por interpolação
Pesquisa em árvores (veremos detalhadamente na disciplina AEDII)
Hashing (veremos detalhadamente na disciplina AEDII)

Ao aplicar um método de busca, o objetivo é encontrar um registro com o menor custo possível.

Como dito anteriormente, cada técnica possui vantagens e desvantagens.

Nessa aula serão apresentados dois métodos: sequencial e sequencial indexada.

Pesquisa Sequencial

Método mais simples de busca: cada chave pode ser comparada sequencialmente, a partir do primeiro registro, até que a chave procurada seja encontrada ou a tabela seja percorrida completamente.

É aplicável a uma tabela organizada como um vetor ou uma lista encadeada.

Percorre-se registro por registro em busca da chave.

Ver slides de 15 ao 20.

Algoritmo de busca sequencial:

```
int busca_sequencial1(int x, int v, int n){
        int i;

        for (i = 0; i < n; i++)
                if (x == v[i])
                        return i;

        return -1;
}
```

Uma outra forma de implementar a busca sequencial é por meio do uso de um sentinela. O sentinela garante que o elemento procurado será encontrado, o que elimina uma expressão condicional:

```
int busca_sequencial2(int x, int v, int n){
        int i;
        v[n] = x;

        for (i = 0; x != v[i]; i++);

        if (i < n)
                return i;
        else;
                return -1;
}
```

Limitação do uso de vetores: tamanho fixo.

Alternativa: listas encadeadas.

Complexidade (pior caso): O(n)
- : quando a chave não é encontrada, ou seja, O(n)

Por mais que as versões dos algoritmos acima possuam a mesma complexidade, segunda versão executa menos instruções em comparação com a primeira:
- Busca sequencial 1: 3n + 3 instruções
- Busca sequencial 2: 2n + 5 instruções

Busca sequencial em tabela ordenada: a eficiência da operação de busca melhora se as chaves dos registros estiverem ordenadas, mas a principal dificuldade do método é que nem todas as tabelas estão ordenadas, ou seja, nesse cenário, a chave pode não ser encontrada, mesmo existindo dentro da tabela.

```
int busca_sequencial3(int x, int v[], int n){
        int i;

        for (i = 0; i < n && x > v[i]; i++);

        if ((i < n) && (v[i] == x))
                return i;
        else
                return -1;
}
```

Possui complexidade de O(n), mas pode executar até 4n + 8 instruções. Caso seja utilizada sentinela como na versão 2, a quantidade máxima de operações pode ser reduzida para 2n + 7 operações!

Outra versão da busca sequencial em tabela ordenada:

```
int busca_sequencial4(int x, int v[], int n){
        int i, j;

        for (i = 0, j = n - 1; (i < j) && (x > v[i]) && (x < v[j]); i++, j--);

        if ((i < n) && (v[i] == x))
                return i;
        else  if ((j >= 0) && (v[j] == x))
                return j;
        else
                return -1;
}
```

O código acima, também tem complexidade na ordem de O(n), mas executa, no máximo, 3,5n + 11 instruções.

Para aumentar eficiência: reordenar continuamente a tabela de modo que os registros mais acessados sejam deslocados para o início (recuperação recorrente de registros).
1 - Método mover-para-frente: sempre que uma pesquisa obtiver êxito, o registro recuperado é colocado no início da lista
2 - Método da transposição: um registro recuperado com sucesso é trocado com o registro imediatamente anterior.

Desvantagens do método mover-para-frente:
- Uma única recuperação não implica que o registro será frequentemente recuperado
- O método é mais custoso para arranjos em comparação com listas encadeadas

Principal vantagem do método mover-para-frente: possui resultados melhores para quantidades pequena e média de buscas.

Para uma grande quantidade de buscas, o método transposição é mais vantajoso, já que os registros recuperados com frequência estarão nas primeiras posições. Por outro lado, encontrar um registro pouco utilizado pode ser tão trabalhoso quanto em uma busca sequencial “convencional”.

Busca sequencial em tabela ordenada: a eficiência da operação de busca melhora se as chaves dos registros estiverem ordenadas. Porém, mantém as mesmas desvantagens para um arranjo.

Conclusão: a complexidade dos métodos de busca sequencial apresentados em aula é na ordem de O(n)!

Pesquisa Sequencial Indexada

Melhoria da busca sequencial.

Existe uma tabela auxiliar, chamada tabela de índices, além do próprio arquivo ordenado.

Cada elemento na tabela de índices contém uma chave (kindex) e um indicador do registro no arquivo que corresponde a esse índice. Faz-se a busca a partir do ponto indicado na tabela, sendo que a busca não precisa ser feita desde o começo.

Pode ser implementada como um vetor ou como uma lista encadeada.

Ver slides de 30 a 39.

Vantagem: os itens na tabela poderão ser examinados sequencialmente sem que todos os registros precisem ser acessados.

Desvantagens:
- A tabela tem que estar ordenada
- Exige espaço adicional para armazenar a(s) tabela(s) de índices

Passos para montar a tabela de índice:
- Se a tabela não estiver ordenada, ordene-a.
- Divida o número de elementos da tabela pelo tamanho do índice desejado: n/m, onde n é o tamanho da tabela e m o tamanho do índice.
- Para montar o índice, recuperam-se da tabela os elementos 0, 1 * (n / m), 2 * (n / m), … (m - 1) * (n / m)
- Cada elemento do índice representa n=m elementos da tabela

Para montar um índice secundário, aplica-se raciocínio similar sobre o índice primário.

Em geral, não são necessários mais do que 2 índices.
Na análise de complexidade, o pior caso é quando o item procurando não existe e é maior que o último elemento do arquivo, ou seja, a tabela de índice será completamente percorrida e o último "bloco" do arquivo coberto pela tabela de índice (ou seja, da posição (m - 1) * (n / m) até n – 1 no arquivo) também será percorrido, resultando na seguinte complexidade:

- O(max(m, n / m)): que é a notação utilizada de acordo com o que foi passado em uma aula anterior

ou

- O(m + n / n): outra opção para definirmos a complexidade da busca sequencial indexada, mas não está de acordo com o que foi passado na aula introdutória sobre complexidade de algoritmos.

Referências

Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C. Introduction to Algorithms. Third edition, The MIT Press, 2009.

Horowitz, E., Sahni, S. Rajasekaran, S. Computer Algorithms. Computer Science Press, 1998.

Rosa, J. L. G. Métodos de Busca. SCE-181 - Introdução à Ciência da Computação II. Slides. Ciência de Computação. ICMC/USP, 2018.

Ziviani, N. Projeto de Algoritmos - com implementações em Java e C++. Thomson, 2007.