

Estrutura de Dados

Pesquisa sequencial e binária

Agenda

- Introdução
- Pesquisa sequencial
- Pesquisa binária
- Exercícios

Introdução

- Foi visto
 - Como armazenar itens em uma estrutura
- Ainda vamos ver
 - Como ordenar os itens em uma estrutura
- Agora veremos como localizar um item em uma estrutura
 - Tarefa difícil quando há uma grande quantidade de dados

Introdução

- Manteremos a ideia de 1 registro para cada informação
- Cada registro possui uma chave, que será o campo de pesquisa
- Objetivo
 - Encontrar uma ou mais ocorrências de registros com chaves iguais à chave de pesquisa

Introdução

- Também continuaremos utilizando TADs – tipos abstratos de dados
 - TDicionario
 - Analogia com um dicionário
 - As chaves são as palavras
 - Os registros são as informações associadas à palavra pesquisada, como definição e sinônimos
- Operações
 - Inicialização
 - Pesquisar um ou mais registros
 - Inserir novos registros
 - Remover um registro específico

Pesquisa sequencial

- Método de pesquisa simples
- A partir do 1º registro, pesquise sequencialmente até encontrar a chave requisitada
- Armazenamento por vetor

Pesquisa sequencial

- Busca (*find*)
 - Retorna o índice do registro que contém a chave X
 - Caso não exista a chave X no dicionário, é retornado **-1**
 - Esta implementação sempre retorna apenas 1 registro (o primeiro)
 - Então não suporta mais de um registro com a mesma chave

Pesquisa sequencial

- Estrutura

```
typedef struct {  
    int chave;  
    int info; //exemplo de informação a ser armazenada  
    int info2; //exemplo de informação a ser armazenada  
} TItem;
```

```
typedef struct {  
    TItem *v;  
    int n, max;  
} TDicionario;
```


Pesquisa sequencial

- Operações

```
void iniciarDicionario (TDicionario *d) {  
    d->n = 0;  
    d->max = 10;  
    d->v = (TItem*) malloc (sizeof(TItem) * d->max);  
}
```

```
void inserir (TDicionario *d, TItem x) {  
    if (d->n == d->max) {  
        d->max *= 2;  
        d->v = (TItem*) realloc (d->v, sizeof(TItem) * d->max);  
    }  
  
    d->v[d->n++] = x;  
}
```

Pesquisa sequencial

- Operações

```
int busca (TDicionario *d, int chave) {  
    int i;  
  
    for (i = 0; i < d->n; i++)  
        if (d->v[i].chave == chave)  
            return i;  
  
    return -1;  
}
```

Pesquisa sequencial

- Teste

```
int main (void) {
    TDicionario d;
    int i;

    printf ("Criando dicionário e inserindo itens...\n");
    iniciarDicionario (&d);
    for (i = 0; i < 21; i++) {
        TItem item;
        item.chave = i * 2;
        item.info = rand();
        item.info2 = rand();
        inserir (&d, item);
    }
    printf ("%d itens foram inseridos e o vetor tem alocado %d espaços.\n", d.n, d.max);

    printf ("Buscando itens\n");
    for (i = 0; i < 5; i++) {
        int chaveSorteada = rand() % 50;
        printf ("Procurando chave %d... ", chaveSorteada);

        int indice = busca (&d, chaveSorteada);
        if (indice == -1)
            printf ("não encontrada\n");
        else
            printf ("encontrada no índice %d\n", indice);
    }
}
```

Pesquisa sequencial

- Análise
 - Melhor caso: 1 comparação
 - Pior caso: n comparações
- Pesquisa sequencial é bom apenas para conjunto de dados pequenos

Pesquisa binária

- Podemos otimizar a pesquisa se pressupormos que o conjunto de dados está ordenado
- Assim, podemos utilizar do método de pesquisa binária

Pesquisa binária

- Ideia
 - Comparar a chave com o registro que está na posição do meio
 - Se a chave for menor, então o registro procurado está na primeira metade
 - Se a chave for maior, então o registro procurado está na segunda metade
 - Repita até que a chave seja encontrada ou que verifique que a chave não existe no conjunto

Pesquisa binária

- Exemplo
- Conjunto de dados

-5	1	3	5	8	10	15	16
----	---	---	---	---	----	----	----

- Chave a ser encontrada: 15

Pesquisa binária

- Exemplo
- Conjunto de dados

-5	1	3	5	8	10	15	16
----	---	---	---	---	----	----	----

- Chave a ser encontrada: 15
- Buscas

-5	1	3	5	8	10	15	16
----	---	---	---	---	----	----	----

Pesquisa binária

- Exemplo
- Conjunto de dados

-5	1	3	5	8	10	15	16
----	---	---	---	---	----	----	----

- Chave a ser encontrada: 15
- Buscas

-5	1	3	5	8	10	15	16
----	---	---	---	---	----	----	----

8	10	15	16
---	----	----	----

Pesquisa binária

- Exemplo
- Conjunto de dados

-5	1	3	5	8	10	15	16
----	---	---	---	---	----	----	----

- Chave a ser encontrada: 15
- Buscas

-5	1	3	5	8	10	15	16
----	---	---	---	---	----	----	----

8	10	15	16
---	----	----	----

15	16
----	----

Pesquisa binária

- Busca - recursiva

```
int busca (TDicionario *d, int chave) {
    if (d->n == 0)
        return -1;
    else
        return buscaBinaria (d, 0, d->n-1, chave);
}

int buscaBinaria (TDicionario *d, int esq, int dir, int chave) {
    int meio = (esq + dir) / 2;

    if (d->v[meio].chave != chave && esq >= dir)
        return -1;
    else if (chave > d->v[meio].chave)
        return buscaBinaria(d, meio+1, dir, chave);
    else if (chave < d->v[meio].chave)
        return buscaBinaria(d, esq, meio-1, chave);
    else
        return meio;
}
```

Pesquisa binária

- Busca - iterativa

```
int buscaIterativa (TDicionario *d, int chave) {
    int i, esq, dir;

    if (d->n == 0)
        return -1;

    esq = 0;
    dir = d->n-1;

    do {
        i = (esq + dir) / 2;
        if (chave > d->v[i].chave)
            esq = i + 1;
        else
            dir = i - 1;
    } while (chave != d->v[i].chave && esq <= dir);

    if (chave == d->v[i].chave)
        return i;
    else
        return -1;
}
```

Pesquisa binária

- Análise
 - A cada iteração o conjunto é dividido ao meio
 - Comparações: $\log n$
- O custo para manter o conjunto ordenado pode ser alto
 - A pesquisa binária não deve ser usada em aplicações muito dinâmicas

Exercícios

- Implemente os dois tipos de pesquisa
 - Sequencial
 - Binária
- Conte quantas comparações são realizadas em cada tipo de pesquisa para a mesma chave no mesmo conjunto de dados
- Altere a pesquisa sequencial para suportar mais de um registro com a mesma chave
 - A busca deve requisitar qual chave deve ser retornada (a primeira, ou a segunda, ..., ou a n-ésima chave)

```
int busca (TDicionario *d, int chave, int n) {  
    ...  
}
```

Estrutura de Dados

Material elaborado por:
Thiago Meirelles Ventura

Baseado em:

- Ascencio, A. F. G; Araújo, G. S. Estruturas de Dados. Pearson, 2011.
- Cormen, T. H.; Leiserson, C. E.; Rivest, R. L.; Stein, C. Algoritmos: teoria e prática. Elsevier, 2002.
- Aulas do Prof. Reinaldo Silva Fortes (<http://www.decom.ufop.br/reinaldo/>)
- Demaine, E., Devadas, S. Introduction to Algorithms (MIT OpenCourseWare), <http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-006-introduction-to-algorithms-fall-2011>