Análise e Projeto de Algoritmos - Trabalho 02

Maurício El Uri

Universidade Federal do Pampa

1.6 Onde está a bolinha de gude?

Entrada de uma certa quantidade de bolinhas com números e números a serem consultados.

O algoritmo deve consultar nas bolinhas e retornar se cada consulta foi encontrada e onde foi encontrada na lista de bolinhas.

Solução desenvolvida

Classe abstrata JogoBolinha
Classe JogoBF
Classe JogoDC
Classe BubbleSort
Classe TextualGame

JogoBF

Método executaConsulta → T(n) é O(bolinhas)

```
private void executaConsulta(int consulta) {
        int resultado = -1;
        for (int i = 1; i < getBolinhas().size(); i++) {
                if (consulta == getBolinhas().get(i - 1)) {
                resultado = i - 1;
                }
        }
        setaResultado(consulta, resultado);
}
```

JogoBF

Método executa → T(n) é O(consulta*bolinhas)

```
public void executa() {
        for (int consulta : getConsultas()) {
                executaConsulta(consulta);
        }
}
```

JogoBF

Complexidade total, processamento por força bruta:

T(n) → O(consultas*bolinhas)

Merge Sort

Método mergeSort → T(n) é O(nlogn)

```
private ArrayList<Integer> mergeSort(int esq, int dir) {
        if (esq != dir) {
                int meio = ((esq + dir) / 2);
                ArrayList<Integer> listEsq;
                ArrayList<Integer> listDir;
                listEsq = mergeSort(esq, meio);
                listDir = mergeSort(meio + 1, dir);
                return merge(listEsq, listDir);
        }
        ArrayList<Integer> listOrdenada = new ArrayList<>();
        listOrdenada.add(list.get(esq));
        return listOrdenada;
}
```

Merge Sort

Função merge → T(n) é O(n)

```
        private ArrayList<Integer>
merge(ArrayList<Integer> listEsq,
        ArrayList<Integer> listDir) {

        ArrayList<Integer> listOrdenada = new
ArrayList<>();
        ListIterator itrEsq = listEsq.listIterator();
        ListIterator itrDir = listDir.listIterator();

        while (itrEsq.hasNext() && itrDir.hasNext()) {
        int itemEsq = (int) itrEsq.next();
        int itemDir = (int) itrDir.next();
        if (itemEsq > itemDir) {
                listOrdenada.add(itemDir);
                itrEsq.previous();
        } else {
                listOrdenada.add(itemEsq);
                itrDir.previous();
        }
        }
        while (itrEsq.hasNext()) {
        listOrdenada.add((int) itrEsq.next());
        }
        while (itrDir.hasNext()) {
        listOrdenada.add((int) itrDir.next());
        }
        return listOrdenada;
        }
```

JogoDC

Função buscaBinaria → T(n) é O(logn)

```
private int buscaBinaria(ArrayList<Integer> bolinhas, int minimo, int maximo, int consulta) {
        int meio = ((maximo + minimo) / 2);
        if (bolinhas.get(meio) == consulta) {
        return getBolinhas().indexOf(
                bolinhas.get(meio));
        }
        if (minimo >= maximo) {
                return -1;
        } else if (bolinhas.get(meio) < consulta) {
                return buscaBinaria(bolinhas, meio + 1, maximo, consulta);
        } else {
                return buscaBinaria(bolinhas, minimo, meio - 1, consulta);
        }
}
```

JogoDC

Método executa → T(n) é O(nlogn)

```
public void executa() {
        ArrayList<Integer> listaOrdenada = ordenaBolinhas();
        for (int consulta : getConsultas()) {
                int resultado = buscaBinaria(listaOrdenada, 0,
                        listaOrdenada.size() - 1, consulta);
                setaResultado(consulta, resultado);
        }
}
```

Conclusões

- Podemos concluir que a ordenação não significou peso significativo no algoritmo de Divisão e Conquista;
- JogoBF tem complexidade $O(n^2)$;
- O método de Busca binária (sem a ordenação) tem complexidade O(nlogn);
- A escolha de um algoritmo para ordenação eficiente fez toda a diferença no desempenho em Divisão e Conquista.

Muito obrigado!

Análise e Projeto de Algoritmos - Trabalho 02

Maurício El Uri

Universidade Federal do Pampa