Projeto de Análise de Algoritmos da disciplina MAC5711

Artur André
Diana Naranjo

Este projeto prevê uma aplicação que gera a árvore de comparação dos seguintes algoritmos de ordenação:
-SelectionSort
-MergeSort
-QuickSort
-HeapSort

1 - Instalação

A aplicação foi feita usando python. O uso é feito através de console com a linha de comando: "python comparisonTrees.py > OUT.txt" onde OUT é um arquivo de saída em texto plano. No arquivo OUT estão as árvores dos algoritmos de ordenação, citados acima, para números de 3, 4 e 8 digitos. Estes valores podem ser alterados conforme descrito na próxima seção.

2 - Descrição do algoritmo

Dados: **n** (tamanho do vetor de elementos distintos) e
alg (algoritmo de ordenação por comparação)
Encontrar: árvore de decisão correspondente ao algoritmo de ordenação de um vetor n elementos distintos.

O algoritmo implementado consiste em gerar a árvore a partir da execução do algoritmo de ordenação passado como parâmetro (**alg**) para cada uma das n! permutações do vetor de tamanho **n**. Ao fazer isto, a árvore irá conter todas as comparações feitas pelo algoritmo de entrada (**alg**).

Sendo **alg** o algoritmo de ordenação, **arr** uma das permutações e **ind** um vetor auxiliar com os índices do vetor **arr**:

Na execução de **alg** com o vetor **arr**, um caminho é criado desde o nó root (a primeira comparação feita) até uma das folhas. A folha é a representação dos índices do vetor **arr** ao final da ordenação. Para obter cada nó mediante a execução de **alg**, cada vez que o algoritmo precisa fazer uma comparação um nó é criado com os índices dos elementos que são comparados. Então se o primeiro elemento é menor do que o segundo o seguinte nó será filho esquerdo do nó atual, caso contrario, será o filho direito.

Quando os elementos de **arr** são trocados para realizar o ordenamento, os elementos do vetor **ind** também são trocados. Isto permite que quando o algoritmo termina o vetor **ind** é adicionado como folha à árvore de comparação. A exceção é quando **alg** é o mergesort, nesse caso, além de ter um vetor auxiliar para a ordenação no método merge, também é necessário um vetor auxiliar dos índices que armazene a ordem deles ao final da execução do método.

Para realizar uma modificação árvore, cada vez que uma comparação é feita a árvore é consultada, se um nó já existir com tal comparação nesse nível então esse nó é retornado, caso

contrario o nó é criado, adicionado à árvore e então retornado.

3 - Detalhes do código

O arquivo principal é o "comparisonTrees.py" nele é possível se configurar o número de elementos que cada algoritmo terá que ordenar e desta forma gerar suas respectivas árvores de comparações. Para a aplicação atual o vetor "n_vals" está configurado com os valores 3, 4 e 8, de forma que as árvores geradas são as dos algoritmos ordenando sequencias de 3, 4 e 8 dígitos respectivamente.

3.1 - comparisonTrees.py

Este é o ponto de entrada da aplicação através dele são chamados os demais módulos da aplicação.

Todos os módulos são baseados nos arquivos "sort.py" e "cmpTree.py". Para cada algoritmo é criada uma árvore (objeto Tree no arquivo cmpTree.py) e esta é populada de acordo com as comparações e índices finais obtidos da execução dos algoritmos de ordenamento sobre todas as permutações possíveis do vetor do tamanho indicado como parâmetro de entrada. Isto é executado no método createCmpTree da classe Sort que é a classe base. Por cada um das permutações um caminho é criado desde o nó root ate uma folha onde o vetor dos índices do vetor ordenado é apresentado.

Como todos os algoritmos de ordenamento fazem algumas operações do mesmo jeito, uma classe Sort foi criada. A classe contém os métodos comuns, como swap, onde os elementos e índices são trocados e isLess onde o nó de comparação é criado e a comparação em si é feita. Cada algoritmo é então executado usando suas próprias especificidades e os métodos da classe base, de forma a criar a árvore. Quando o primeiro elemento da comparação é menor que o segundo, um nó à esquerda do nó atual é criado, caso contrario um nó à direita do nó atual é criado.

3.2 - SelectionSort

O algoritmo de selectionSort está contido no arquivo selectionSort.py. Nele todo nó será comparado com o elemento mínimo encontrado, a cada iteração o elemento mínimo é substituído por algum elemento do vetor, de forma que todos os elementos do vetor são comparados em todas as permutações possíveis de n elementos.

3.3 - MergeSort

O algoritmo de mergeSort está contido no arquivo mergeSort.py. Nele são criados dois vetores auxiliares, um que irá conter o vetor ordenado de saída e outro que contém os índices usados nas comparações que irão compor o vetor auxiliar de saída. O array auxiliar de índices é necessário já que o MergeSort é o único algoritmo que não faz trocos entre elementos.
O algoritmo começa usualmente tendo como entrada os índices inicial e final do vetor, (assim como os vetores auxiliares), em seguida ele recursivamente quebra o vetor ao meio e ao final na fase de merge ele ordena os vetores fazendo as devidas comparações e construindo a árvore através do vetor de índices auxiliar.

3.4 - QuickSort

O algoritmo de quickSort está contido no arquivo quickSort.py. Ele faz uso de uma estrutura de dados do tipo pilha. Para sua inicialização a pilha é carregada com o primeiro e último índices do vetor. Até que a pilha esteja vazia o algoritmo iterativamente inicia as variáveis "ini" e "fim" com os dois valores mais no topo da pilha (últimos a entrarem) representando os índices de ínicio e fim de

cada passo do algoritmo. O algoritmo é executado para cada permutação de tamanho n comparando a cada iteração todos os valores entre "ini" e "fim" e organizando-os de maneira crescente. Ao final do processo a árvore está preenchida com todos os nós de comparação.

3.5 - HeapSort

O algoritmo de heapSort está contido no arquivo heapSort.py. Ele irá criar uma árvore quase cheia de tamanho n. Em seguida a raiz da árvore será trocada com cada um dos nós da árvore, de forma que todas as comparações possíveis com n elementos distintos no vetor serão feitas durante a etapa de fixdown do heapsort.