Leandro de Assis Madeira

Estrutura de Dados – Prof.ª Dr.ª Andreia Cristina Grisolio Machion

Fatec Ipiranga

Tarefa da Aula do dia 18/03/2021

**Algoritmos de Ordenação**

***Insertion Sort*:**

void insertion\_sort(int \* v, int n){

    int i, j, x;

    for(j=1; j<n; j++){

        x = v[j];

        for(i = j-1; i >= 0 && v[i] > x; i--){

            v[i+1] = v[i];

        }

        v[i+1] = x;

    }

}

***Selection Sort*:**

void selection\_sort(int \* v, int n){

    int i, j, x;

    for(i=0; i<n-1; i++){

        int min = i;

        for(j=i+1; j<n; j++){

            if(v[j] < v[min]){

                min = j;

            }

        }

        x = v[i];

        v[i] = v[min];

        v[min] = x;

    }

}

**Comparando os tempos de execução:**



**Hardware utilizado:**

* Processador: Intel® Core™ i3-5005U CPU @ 2.00 GHz
* Memória Instalada (RAM): 4,00 GB

**Conclusão:**

Através da atividade, pôde-se perceber e compreender de forma mais clara o funcionamento dos algoritmos de ordenação. Foram abordados na atividade os algoritmos *Selection Sort* e *Insertion Sort* (além do algoritmo *Bubble Sort*, visto em aula, que foi comparado aos outros dois a título de curiosidade).

O funcionamento do algoritmo *Selection Sort* se dá através de uma ordenação que consiste em passar o menor dos termos sempre para a primeira posição do vetor, o segundo menor termo para a segunda posição do vetor e assim sucessivamente, ordenando os termos restantes. A lógica é construída através de dois laços de repetição *for*, o mais externo começando do índice 0, somando uma unidade a cada iteração ao final do vetor, e o mais interno começando do índice externo acrescido de uma unidade. Essencialmente, após o menor termo assumir a primeira posição do vetor, o termo que antes ocupava a primeira posição é alocado à posição onde o menor termo do vetor se encontrava, e assim sucessivamente para os termos subsequentes.

Já o algoritmo *Insertion Sort* realiza a ordenação percorrendo as posições dentro do vetor começando pelo índice 1, comparando-o com a posição anterior e a posição seguinte, conferindo se o termo que se encontra no índice atual é maior que o anterior e menor que o seguinte. É comum comparar esse algoritmo com a organização das cartas do baralho, onde cada carta nova deve ser alocada de forma ordenada à mão do jogador.

Podemos concluir, então, que o algoritmo *Insertion Sort* realiza a ordenação dos termos de forma quadrática, ou seja, no pior caso, o número de execuções de comparação “” é descrito por , ou seja, um pouco menos que a metade de , onde o melhor caso (quando o vetor já está quase ordenado) assume um número de comparações proporcional a . Já o algoritmo *Selection Sort*, no pior caso, realiza um número de comparações “” descrito por , porém, esse número de comparações permanece mesmo no melhor caso, assim, o consumo de tempo deste algoritmo é sempre proporcional à .

Portanto, através do experimento realizado na atual atividade, conclui-se que os algoritmos *Insertion Sort* e *Selection Sort* possuem uma boa aplicabilidade em vetores com um número de termos relativamente pequeno. Foi possível observar, também, que, entre esses dois algoritmos, o desempenho permanece praticamente igual em vetores de até 200000 termos, apresentando pouca diferença nos tempos de execução, que se intensifica apenas quando testado em vetores com um número maior de termos. De tal forma, torna-se preferível a utilização do algoritmo *Insertion Sort* por realizar menos execuções de comparação e, com isso, poupar mais tempo.

**Referências:**

<<https://www.ime.usp.br/~pf/algoritmos/aulas/ordena.html>> Acesso: 20/03/2021;

<<https://pt.khanacademy.org/computing/computer-science/algorithms/insertion-sort/a/insertion-sort>> Acesso: 20/03/2021;

<<https://pt.khanacademy.org/computing/computer-science/algorithms/sorting-algorithms/a/selection-sort-pseudocode>> Acesso: 20/03/2021.