Centro Universitário Católico Salesiano Auxilium - Araçatuba SP
UniSALESIANO

Implementação de algoritmos de ordenação

Elaboração
Eduardo Antonio Rodrigues RA: 212419
Leandro Roberto Alexandre RA: 212374

Orientação: Prof. Mayk Fernando Choji

Araçatuba
2022

Merge Sort e Quick Sort

Definição

Merge Sort:

O que é: Merge Sort é um algoritmo eficiente de ordenação por divisão e conquista. Se o objetivo é ordenar um array comparando seus elementos, do ponto de vista assintótico, n∗logn é o limite inferior.

Quick Sort:

O que é: Quick Sort é um algoritmo eficiente de ordenação por divisão e conquista. Apesar de ser da mesma classe de complexidade do Merge Sort, o Quick Sort é na prática o mais veloz, pois suas constantes são menores. Contudo, é importante destacar de antemão que, em seu pior caso, o Quick Sort é O(n2)

Descrição do problema

Esse trabalho propõe levantar dados por meio de testes para validar a eficiência do método de organização de um vetor que contém uma lista de números desorganizados. Com base nisso, foi validado a organização por meio das funções Merge Sort e Quick Sort.

Solução

Em seguida, será apresentada uma análise dos valores testados e os resultados obtidos.

Análise:

Criou-se um vetor contendo 6 elementos:

```
int vetor[] = {8, 7, 6, 9, 5, 10, 11};
```

Com isso os vetores passou em testes nas duas funções Merge Sort e Quick Sort conforme as os códigos a seguir:

Merge Sort

```
void merge(int *a, int g, int q, int r) {
    int num1;
    int num2;
    int i;
    int j;
    int k;
    int *ladoDireito;
    int *ladoEsquedo;
    int somaMinEsquerdo = 0;
    int somaMaxEsquerdo = 0;
    int somaMinDireito = 0;
    int somaMaxDireito = 0;
    num1 = q - g + 1;
    num2 = r - q;
    ladoEsquedo = new int[num1 + 1];
    ladoDireito = new int[num2 + 1];
    int numeroComp; // numeroComp
    int numeroTroca; // numeroTroca
    numeroTroca = 0;
    numeroComp = 0;
    for (i = 0; i < num1; i++) {
        ladoEsquedo[i] = a[g + i];
        numeroTroca++;
    }
    for (j = 0; j < num2; j++) {
        ladoDireito[j] = a[q + j+1];
        numeroTroca++;
    }
    ladoEsquedo[num1] = INT_MAX;
    ladoDireito[num2] = INT_MAX;
```

```
   printf("---- MERGE LADO ESQUERDO: ----");
   for (i = 0; i < num1; i++)

   {
       //cout << ladoEsquedo[i] << "  ";
       somaMinEsquerdo = somaMinEsquerdo + ladoEsquedo[0];
       somaMaxEsquerdo = somaMaxEsquerdo + ladoEsquedo[num1-1];
       cout << endl;
       cout<<"incremento minimo "<<ladoEsquedo[0]<<endl;
       cout<<"incremento maximo "<<ladoEsquedo[num1-1]<<endl;
   }


      float mediaMinEsq, mediaMaxEsq;
      mediaMinEsq = somaMinEsquerdo/2;
      mediaMaxEsq = somaMaxEsquerdo/2;
      cout<<"Total Minimo: "<<somaMinEsquerdo<<endl;
      cout<<"Total Maximo: "<<somaMaxEsquerdo<<endl;
      cout<<"Media Minimo: "<<mediaMinEsq<<endl;
      cout<<"Media Maximo: "<<mediaMaxEsq<<endl;
      cout<<endl;
      cout<<endl;

   cout << endl;
   printf("---- MERGE LADO DIREITO: ----");
   for (j = 0; j < num2; j++)
   {
       // cout << ladoDireito[j] << "  ";
       somaMinDireito = somaMinDireito + ladoDireito[0];
       somaMaxDireito = somaMaxDireito + ladoDireito[num2-1];
       cout << endl;
       cout<<"incremento minimo "<<ladoDireito[0]<<endl;
       cout<<"incremento maximo "<<ladoDireito[num2-1]<<endl;
   }
       float mediaMinDir, mediaMaxDir;
      mediaMinDir = somaMinDireito/2;
      mediaMaxDir = somaMaxDireito/2;
      cout<<"Total Minimo: "<<somaMinDireito<<endl;
      cout<<"Total Maximo: "<<somaMaxDireito<<endl;
      cout<<"Media Minimo: "<<mediaMinDir<<endl;
```

```
        float mediaMinDir, mediaMaxDir;
      mediaMinDir = somaMinDireito/2;
      mediaMaxDir = somaMaxDireito/2;
      cout<<"Total Minimo: "<<somaMinDireito<<endl;
      cout<<"Total Maximo: "<<somaMaxDireito<<endl;
      cout<<"Media Minimo: "<<mediaMinDir<<endl;
      cout<<"Media Maximo: "<<mediaMaxDir<<endl;
       cout<<"//////////////////"<<endl;


   cout << endl;

   i = 0;
   j = 0;
   //int numeroComp =0;
   for (k = g; k <= r; k++) {

           numeroComp++;
       if (ladoEsquedo[i] <= ladoDireito[j]) {
           a[k] = ladoEsquedo[i];
           i++;
           numeroTroca++;
       } else {
           a[k] = ladoDireito[j];
           j++;
           numeroTroca++;
       }

   }
      cout<<"Quantidade de Comparações Merge: "<< numeroComp<<endl;
      cout<<"Quantidade do trocas Merge:  "<< numeroTroca<<endl;
}
```

Quick Sort

```
//Inicia o QuickSort e ordena o vetor
void quickSort(int vetor[], int Nmenor, int Nmaior)
{
    if (Nmenor < Nmaior)
    {
        int pi = particao(vetor, Nmenor, Nmaior);
        quickSort(vetor, Nmenor, pi - 1);
        quickSort(vetor, pi + 1, Nmaior);
        nCompararativo++;


    }
}
```

Resultado:

Merge Sort

```
/////////////////
---- MERGE LADO ESQUERDO: ----
incremento minimo 6
incremento maximo 9

incremento minimo 6
incremento maximo 9

incremento minimo 6
incremento maximo 9

incremento minimo 6
incremento maximo 9
Total Minimo: 24
Total Maximo: 36
Media Minimo: 12
Media Maximo: 18



---- MERGE LADO DIREITO: ----
incremento minimo 5
incremento maximo 11

incremento minimo 5
incremento maximo 11

incremento minimo 5
incremento maximo 11
Total Minimo: 15
Total Maximo: 33
Media Minimo: 7
Media Maximo: 16
/////////////////

Quantidade de Comparacoes Merge: 7
Quantidade do trocas Merge:  14
Quantidade de Comparacoes  Merge Sort:  1
Vetor ordenado:
[5 6 7 8 9 10 11 ]
---------------

--------------------------------
```

Quick Sort

```
--------------------------------------------------------------------------
RESULTADO:
Vetor sem ordenar:
[8 7 6 9 5 10 11 ]
Vetor ordenado em ordem crescente:
[5 6 7 8 9 10 11 ]
Comparacoes: 5
Total trocas -> 18
Total minimo -> 91
Total maximo -> 102
Media minimo -> 45
Media maximo -> 51
--------------------------------------------------------------------------

--------------------------------
Process exited after 2.554 seconds with return value 0
Pressione qualquer tecla para continuar. . .
```

Discussão após Análise dos métodos:

1. Mesmo os algoritmos Merge Sort e Quick Sort sendo de mesma complexidade em média (i.e. ambos são O(nlg(n)), seu teste mostrou alguma diferença?

R: Conforme os testes realizando houve sim diferenças conforme os dados obtidos. Primeiramente, o número de comparação do Merge Sort foi de 7 quanto do Quick Sort de 5, além do mais, o número de trocas foram de 14 e 18 respectivamente. O Total de mínimo e máximo para o Merge Sort foi de 15 e 33 respectivamente, para o Quick foram uma soma do total de mínimos 91 e 102 a soma dos máximos. Por fim, a média mínimo de 7 média máximo de 16 para o Merge, ao contrário do Quick Sort, a média mínima foi de 45 e máximo 51.

2. Como seus resultados se relacionam com a análise teórica das complexidades desses algoritmos? Explique essas diferenças.

R: Com base dos resultados obtidos nos testes, a função Merge Sort e o Quick Sorte possuem vantagens e desvantagens ao se aplicarem em diferentes cenários. No caso do Merge Sort, conforme a análise teórica recursiva, sua melhor atuação em questão de desempenho se dá em lidar com vetores com elementos aleatórios, em contrapartida, ao lidar com elemento possivelmente ordenado, poderá ter um caída no seu desempenho, podendo realizar trocas irrelevantes. Por outro lado, o uso da função Quick Sort, usando o mesmo cenário que o Merge, o seu tempo de execução diminui, porém, possuí uma maior quantidade de comparações e trocas quanto ao Merge Sort.