Instituto Federal do Sudeste de Minas Gerais

Sistemas para Internet

Estruturas de Dados I

Trabalho Prático 2

Professor Wender Cota

Aluno: Mateus Ferreira Silva
Turma: 2011

Introdução

Trabalho com objetivo de avaliar os desempenhos dos algoritmos de ordenação apresentados na tabela abaixo. Utilizando uma avaliação por tempo de execução.

Algoritmos
Bubblesort
Selectsort
Insertsort
Shellsort
Heapsort
Quicksort Recursivo (pivô = elemento do meio)
Quicksort Recursivo (pivô = media dos extremos mais a mediana)
Quicksort Recursivo com Inserção
Quicksort não Recursivo

As entradas serão de vetores inteiros aleatórios, ordenados crescentemente e ordenados decrescentemente, com tamanhos de 5000, 10000, 50000, 100000, 500000 e 1000000.

Serão armazenados em um arquivo (*Vetores.txt*) os vetores aleatórios e em seguida utilizarei o método de ordenação Quicksort Recursivo com pivô sendo o elemento do meio para ordena-los de forma crescente e decrescente.

Ao final, teremos um quadro comparativo sobre os algoritmos.

Algoritmos

Bubblesort:

```
void bubbleSort(tipo_item *v, int tamanho)
{
    int i, ultima_troca, ultima_posicao;
    tipo_item aux;
    ultima_posicao=tamanho;
    do{
        ultima_troca = 0;
        for( i = 0 ; i < ultima_posicao-1 ; i++ )
            if (v[i].chave > v[i+1].chave)
            {
                aux = v[i];
                v[i] = v[i+1];
                v[i+1] = aux;
                ultima_troca = i+1;
            }
        ultima_posicao = ultima_troca;
    }while(ultima_posicao>0);
}
```

Algoritmos

Selectsort:

```
void selectSort(tipo_item *v, int tamanho)
{
    int i, j, posicao_menor;
    tipo_item aux;
    for (i = 0; i < tamanho - 1; i++)
    {
        posicao_menor = i;
        for (j = i + 1 ; j < tamanho; j++)
        if (v[j].chave < v[posicao_menor].chave)
            posicao_menor = j;
        aux = v[posicao_menor];
        v[posicao_menor] = v[i];
        v[i] = aux;
    }
}
```

Algoritmos

Insertsort:

```
void insertSort(tipo_item *v, int tamanho)
{
    int i,j;
    tipo_item aux;
    for (i = 1; i < tamanho; i++)
    {
        aux = v[i];
        j = i - 1;
        while ((j >= 0)&&(aux.chave<v[j].chave))
        {
            v[j + 1] = v[j];
            j--;
        }
        v[j + 1] = aux;
    }
}
```

Algoritmos

Shellsort:

```
void shellSort(tipo_item *v, int tamanho)
{
    int i, j;
    int h = 1;
    tipo_item aux;
    do
        h = h * 3 + 1;
    while (h < tamanho);
    do {     h /= 3;
             for( i = h ; i < tamanho ; i++ )
             {
                  aux = v[i];
                  j = i;
                  while (v[j-h].chave > aux.chave)
                  {
                       v[j] = v[j-h];
                       j -= h;
                       if (j < h)
                            break;
                  }
                  v[j] = aux;
             }
    } while (h != 1);
}
```

Algoritmos

Heapsort:

```
void heapSort(tipo_item *v, int tamanho)
{
    int i;
    tipo_item aux;

    for(i = tamanho/2; i >= 0; i--)
        constroi(v,i,tamanho-1);
    for(i = tamanho-1; i > 0; i--)
     {
        aux = v[0];
        v[0] = v[i];
        v[i] = aux;
        constroi(v,0,i-1);
     }
}

void constroi(tipo_item *v, int posicao, int fim)
{
     int trocou, h, i;
     tipo_item aux;
     i = posicao;
     trocou = 1;
     do
     {
        if(2 * i > fim)
           trocou = 0;
        else
        {
           if(2 * i + 1 > fim)
              h = 2 * i;
           else
              if(v[2*i].chave > v[2*i+1].chave)
                 h = 2 * i;
              else
                 h = 2 * i + 1;
           if(v[i].chave < v[h].chave)
           {
              aux = v[i];
              v[i] = v[h];
              v[h] = aux;
              i = h;
           }
           else
              trocou = 0;
        }
     }while(trocou);
}
```

Algoritmos

Quicksort Recursivo (pivô = elemento do meio):

```
void quickSort(tipo_item *v, int tamanho)
{
     particao(v,0,tamanho-1);
}

void particao(tipo_item *v, int esq, int dir)
{
     int i,k;
     tipo_item aux, pivo;
     pivo = v[(esq+dir)/2];
     i = esq;
     k = dir;
     do {
          while(v[i].chave < pivo.chave)
               i++;
        while(v[k].chave > pivo.chave)
               k--;
          if (i <= k)
          {
               aux = v[i];
               v[i] = v[k];
               v[k] = aux;
               i++;
               k--;
          }
     }while(i <= k);
     if (esq < k)
          particao(v,esq,k);
      if (i < dir)
          particao(v,i,dir);
}
```

Algoritmos

Quicksort Recursivo (pivô = media dos extremos mais a mediana):

```
void quickSort(tipo_item *v, int tamanho)
{
     particao(v,0,tamanho-1);
}

void particao(tipo_item *v, int esq, int dir)
{
     int i,k;
     tipo_item aux, pivo;
     pivo = v[(esq+dir+(esq+dir)/2)/3];
     i = esq;
     k = dir;
     do {
          while(v[i].chave < pivo.chave)
               i++;
        while(v[k].chave > pivo.chave)
               k--;
          if (i <= k)
          {
               aux = v[i];
               v[i] = v[k];
               v[k] = aux;
               i++;
               k--;
          }
     }while(i <= k);
     if (esq < k)
          particao(v,esq,k);
     if (i < dir)
          particao(v,i,dir);
}
```

Algoritmos

Quicksort Recursivo com Inserção:

```
void quickSortInsert(tipo_item *v, int tamanho)
{
     particaoInsert(v,0,tamanho-1);
}

void particaoInsert(tipo_item *v, int esq, int
dir)
{
     int i,k;
     tipo_item aux, pivo;
     pivo = v[(esq+dir+(esq+dir)/2)/3];
     i = esq;
     k = dir;
     do {
          while(v[i].chave < pivo.chave)
               i++;
        while(v[k].chave > pivo.chave)
               k--;
          if (i <= k)
          {
               aux = v[i];
               v[i] = v[k];
               v[k] = aux;
               i++;
               k--;
          }
       }while(i <= k);
     if( (k-i) <= 20 )
     {
         insertSortLimitado(v,i,k);
     }
     if (esq < k)
          particaoInsert(v,esq,k);
     if (i < dir)
          particaoInsert(v,i,dir);
}

void insertSortLimitado(tipo_item *v, int esq,
int dir)
{
     int i,j;
     tipo_item aux;
     for (i = esq+1; i < dir; i++)
     {
          aux = v[i];
          j = i - 1;
          while ( ( j >= esq ) && ( aux.chave <
v[j].chave ) )
          {
               v[j + 1] = v[j];
               j--;
          }
          v[j + 1] = aux;
     }
}
```

Algoritmos

Quicksort não Recursivo:

```
void QuickSortNaoRec(tipo_item *v, int tamanho)
{
     pilha p; tipo_item_pilha item;
     int esq, dir, i, j;

     cria_pilha(&p);
     esq = 0;
     dir = tamanho-1;
     item.dir = dir;
     item.esq = esq;
     push(&p,item);
     do
     {
       if (dir > esq)
        {
          ParticaoNaoRec(v,esq,dir,&i, &j);
          if ((j-esq)>(dir-i))
          {
             item.dir = j;
             item.esq = esq;
             push(&p,item);
             esq = i;
          }
          else
          {
              item.esq = i;
              item.dir = dir;
              push(&p,item);
              dir = j;
          }
        }
        else
        {
         pop(&p,&item);
         dir = item.dir;
         esq = item.esq;
        }
     }while (!pilha_vazia(p));
}
void ParticaoNaoRec(tipo_item *v,int Esq, int Dir,int
*i, int *j)
{
  tipo_item pivo, aux;
  *i = Esq;
  *j = Dir;
  pivo = v[(*i + *j)/2];
  do
  {
    while (pivo.chave > v[*i].chave) (*i)++;
    while (pivo.chave < v[*j].chave) (*j)--;
    if (*i <= *j)
    {
      aux = v[*i];
      v[*i] = v[*j];
      v[*j] = aux;
      (*i)++;
      (*j)--;
    }
  }while (*i <= *j);
```

Comparações

Vetores aleatórios:

Tam.Vetor / Método	bubbleSort	selectSort	insertSort	shellSort	heapSort	quickSort	quickSortNum3	quickSortInsert	QuickSortNaoRec
5000	0,359	0,047	0,118	0,003	0,004	0,001	0,001	0,001	0,002
10000	1,451	0,188	0,490	0,007	0,008	0,003	0,002	0,002	0,006
50000	39,636	7,551	12,570	0,044	0,057	0,017	0,016	0,018	0,029
100000	160,650	51,995	58,047	0,107	0,129	0,037	0,037	0,037	0,062
500000	4234,105	1484,935	1621,342	0,766	0,961	0,216	0,211	0,213	0,335
1000000	9235,026	5977,743	6881,168	1,782	2,257	0,465	0,472	0,475	0,579

*Tempo em segundos

Vetores ordenados crescentemente:

Tam.Vetor / Método	bubbleSort	selectSort	insertSort	shellSort	heapSort	quickSort	quickSortNum3	quickSortInsert	QuickSortNaoRec
5000	0,000	0,047	0,001	0,001	0,003	0,000	0,001	0,001	0,002
10000	0,000	0,187	0,001	0,002	0,007	0,001	0,001	0,001	0,002
50000	0,000	7,519	0,002	0,017	0,047	0,007	0,007	0,007	0,017
100000	0,001	51,761	0,000	0,035	0,099	0,016	0,016	0,016	0,037
500000	0,009	1485,997	0,015	0,229	0,580	0,105	0,106	0,112	0,154
1000000	0,022	6069,020	0,040	0,450	1,197	0,237	0,237	0,249	0,331

*Tempo em segundos

Comparações

Vetores ordenados decrescentemente:

Tam.Vetor / Método	bubbleSort	selectSort	insertSort	shellSort	heapSort	quickSort	quickSortNum3	quickSortInsert	QuickSortNaoRec
5000	0,593	0,062	0,244	0,002	0,003	0,001	0,000	0,000	0,001
10000	2,387	0,249	0,970	0,004	0,007	0,001	0,002	0,001	0,002
50000	63,247	9,563	26,525	0,021	0,044	0,007	0,007	0,007	0,018
100000	255,462	54,787	122,2211	0,046	0,095	0,017	0,018	0,018	0,040
500000	5753,211	1506,713	3401,263	0,283	0,607	0,110	0,115	0,117	0,170
1000000	12001,126	6312,652	13842,432	0,554	1,258	0,255	0,259	0,271	0,356

*Tempo em segundos

Mateus Ferreira Silva
Barbacena – Minas Gerais, 02 de dezembro de 2011