Algoritmy a dátové štruktúry

Triedenia (1)

12.10.2010

Triedenia

- Dôvod: zrýchlenie nasledujúcich častí
- Čo triedime: záznamy podľa kľúča (najjednoduchšie čisto celočíselné kľúče)
- Formálne: Máme postupnosť a(1),a(2) .. a(n) a chceme jej permutáciu a(i_1),a(i_2) .. a(i_n) takú, že pre každé j platí $a(i\_j) \leq a(i\_(j+1))$

Triedenia - vlastnosti

- Stabilné triedenie: keď nezmení relatívnu postupnosť zhodných objektov
- Vnútorné vs. vonkajšie triedenie
- Zložitosť: počet porovnaní, počet výmien

SelectionSort

```
SelectionSort(A){
    for (i = 0; i < (n - 1); i++) {
      imin = i;    amin=a[imin];
      for (j = (i + 1); j < n; j++) {
        if (a[j] < amin) {
          imin = j; // nový index minima
          amin=a[imin]; // nová hodnota minima
        } }
    if (imin==i) { continue; } // minimum je na mieste
    a[imin]↔ a[i] // napríklad a[imin]=a[i]; a[i]=amin;
  }
```

BubbleSort

```
void BubbleSort(int a[], int n){
  int k,t,last_swap;
  int bound = n-1;
  while (bound) {
    last_swap = 0;
    for ( k=0; k<bound; k++ )
      t = a[k]; /* t is a maximum of A[0]..A[k] */
      if ( t > a[k+1] ) {  a[k] = a[k+1]; a[k+1] = t; last_swap = k;  }
    }
    bound=last_swap; /*elements after bound are already sorted */
    }
}
```

BubbleSort (2)

Two properties of the correct implementation of bubblesort:

1. If there are no exchanges, then the sorting is complete .
2. Each next pass ends at the last swap of the previous pass.

InsertionSort

```
void InsertionSort(int a[], int N){
int i, j, current;
      for (i=1; i < N; i++){
        current = a[i];
        j = i; # koniec utriedenej časti
        while ((j > 0) && (a[j-1] > current)){
          a[j] = a[j-1]; j = j - 1;
        }
        a[j] = current;
      }
}
```

InsertionSort

C implementation usually contains two nested loops:

- External for loop marked the boundary of the sorted area. It has one continue statement for enlarging the sorted area in case the next element is larger then that last element in the sorted zone.
- Internal for loop that inserts one element into the appropriate place of sorted area working from the last (bottom) element and shifting elements down until smaller element is found. It has internal break statement for exiting the loop if smaller element is found.

HeapSort

- Použitie haldy, kde na vrchu je maximum
- V prvom kroku musíme vytvoriť takúto haldu

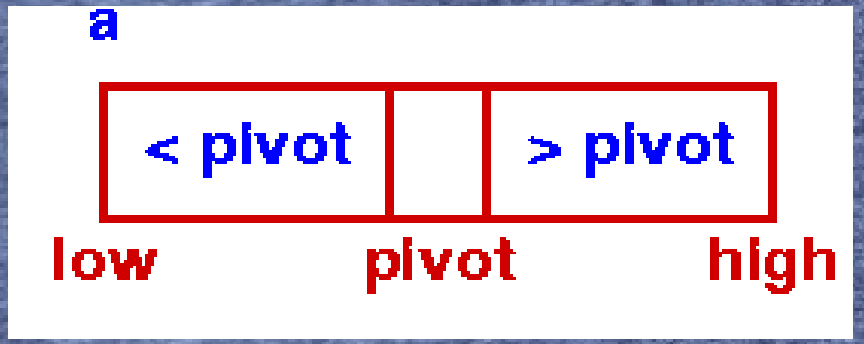
```
for (i=N/2; i>0; i--) UpravHaldu(A,i);
```

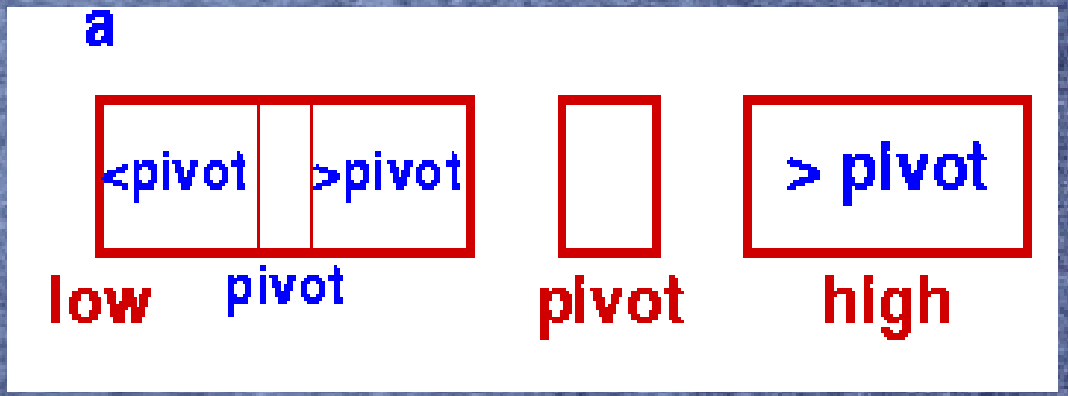
- Následne vyberáme maximum a dávame na koniec – zmenšujeme haldu

```
for (i=N; i>1; i--){ A[1]↔A[i]; N--; UpravHaldu(A,1);}
```

QuickSort

```
Quicksort( void *a, int low, int high )
  {
  int pivot;
  /* Termination condition! */
  if ( high > low )
    {
    pivot = partition( a, low, high );
    quicksort( a, low, pivot-1 );
    quicksort( a, pivot+1, high );
    }
  }
```





QuickSort

- Rozdeľuj a panuj

MergeSort

```
Mergesort(int a[], low, high){
   int mid;
 if(low<high){
   mid=(low+high)/2;
   mergesort(a,low,mid);
   mergesort(a,mid+1,high);
   merge(a,low,high,mid);
 }
return(0);
}
```

```
Merge(int a[], low, high, int mid)
{ int i, j, k, c[50];
i=low; j=mid+1; k=low;
while((i<=mid)&&(j<=high)){
  if(a[i]<a[j]){ c[k]=a[i]; k++;i++;}
  else { c[k]=a[j]; k++; j++;}
}
while(i<=mid){c[k]=a[i];k++;i++;}
while(j<=high){c[k]=a[j];k++;j++}
for(i=low;i<k;i++){ a[i]=c[i];}
}
```

Zložitosť triedení

Cvičenia

- Porovnanie triedení
- Zložitosť triedení na rôznych vstupoch
- Najnevyváženejší AVL strom