内排序

目录

[内排序 1](#_Toc450589414)

[线性排序 2](#_Toc450589415)

[快速排序的优化 3](#_Toc450589416)

### 线性排序

**桶排序**

**适用于数据的域比较小时，统计所有可能出现的数出现的次数，然后从小到大一次放入数组里面，得到一个有序的数组, 代码如下：**

void BucketSort()

{

memset(B, 0, sizeof(B));

memset(C, 0, sizeof(C));

for (int i = 1; i <= N; i++)

C[A[i]]++;

int j = 0;

for (int i = 1; i <= MAXK; i++)

while (C[i] >= 1)

{

C[i]--;

B[++j] = i;

}

output("BucketSort");

}

**计数排序**

**和桶排序的思想差不多，只不过计数排序提前计算出元素应该存在的位置的下标，然后根据下标储存数据，代码如下：**

void CountSort()

{

memset(B, 0, sizeof(B));

memset(C, 0, sizeof(C));

for (int i = 1; i <= N; i++)

C[A[i]]++;

for (int i = 2; i <= MAXK; i++)

C[i] += C[i - 1];

for (int i = N; i >= 1; i--)

{

B[C[A[i]]] = A[i];

C[A[i]]--;

}

output("Count Sort");

}

## 快速排序的优化

未经优化的快速排序代码：

void \_QuickSort(int L, int R)

{

if (L >= R) return;

int X = rand() % (R - L + 1) + L;

// int X = R;

swap(a[X], a[R]);

int StorePosition = L;

for (int i = L; i < R; i++)

if (a[i] < a[R])

swap(a[i], a[StorePosition++]);

swap(a[StorePosition], a[R]);

\_QuickSort(L, StorePosition - 1);

\_QuickSort(StorePosition + 1, R);

}

**优化1**

if (R - L <= 5) //较小数据使用选择排序

{

for (int i = L; i <= R; i++)

{

int minvalue = 0x7f7f7f7f;

int mini = -1;

for (int j = i; j <= R; j++)

{

if (a[j] < minvalue)

{

minvalue = a[j];

mini = j;

}

}

swap(a[i], a[mini]);

}

return;

}

**优化2**

int X1 = rand() % (R - L + 1) + L;

int X2 = rand() % (R - L + 1) + L;

int X3 = rand() % (R - L + 1) + L;

int X;

if (a[X1] < a[X2] && a[X2] < a[X3])

X = X2;

else if (a[X3] < a[X1] && a[X1] < a[X2])

X = X1;

else

X = X3;

swap(a[X], a[R]); //三取中间

**优化3**

int StorePosition = L, EqAdd = 0;

for (int i = L; i < R; i++)

{

if (a[i] < a[R])

{

if (StorePosition + EqAdd != i)

swap(a[StorePosition], a[StorePosition + EqAdd]);

swap(a[i], a[StorePosition++]);

}

else if (a[i] == a[R]) //相同的数存放在比它小的数的后面

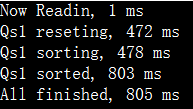
swap(a[i], a[StorePosition + (EqAdd++)]);

}

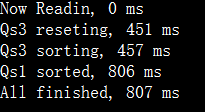
swap(a[StorePosition + EqAdd], a[R]);

**对于1000000的随机生成的范围在（1， 1000000）的数据排序**

**未经优化的快速排序：**



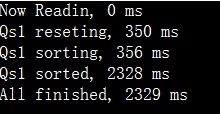
**加入三种优化的快速排序：**



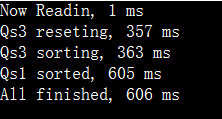
**因为两种快排都加入了随机化的因素，所以每次运行的时间会有差别，但总体来讲，对于随机产生的数据，重复不太多的情况下，加入的三种优化效果不明显。**

**对于 1000000 个范围在（1，1000）的随机数据进行排序：**

**未加入优化的算法：**



**加入三种优化的算法：**



**可以看出对于重复元素很多的情况下，优化3能够很好的优化快速排序的时间复杂度，对于其他两种优化来看，效果不明显。**