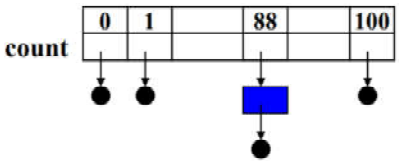
# 基数排序

2019年6月30日 20:54

桶排序

假设我们有 N 个学生，他们的成绩是0到100之间 的整数（于是有 M = 101 个不同的成绩值个不同的成绩值）。如 何在线性时间内将学生按成绩排序?




如果 $M >> N$
该怎么办?

```
void Bucket_Sort(ElementType A[], int N)
{  count[]初始化;
   while (读入1个学生成绩grade)
      将该生插入count[grade]链表;
   for ( i=0; i<M; i++ ) {
      if ( count[i] )
         输出整个count[i]链表;
   }
}
```

$T(N, M) = O(M+N)$

基数排序

假设我们有 N = 10 个整数，每个整数的值在0到 999之间（于是有 M = 1000 个不同的值）。还有 可能在线性时间内排序吗? 输 序列 入 : 64, 8, 216, 512, 27, 729, 0, 1, 343, 125

输入序列: 64, 8, 216, 512, 27, 729, 0, 1, 343, 125

$T=O(P(N+B))$

用“次位优先”（Least Significant Digit）

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Bucket | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| Pass 1 | 0 | 1 | 512 | 343 | 64 | 125 | 216 | 27 | 8 | 729 |
| Pass 2 | 0 | 512 | 125 | | 343 | | 64 | | | |
| | 1 | 216 | 27 | | | | | | | |
| | 8 | | 729 | | | | | | | |
| Pass 3 | 0 | 125 | 216 | 343 | | 512 | | 729 | | |
| | 1 | | | | | | | | | |
| | 8 | | | | | | | | | |
| | 27 | | | | | | | | | |
| | 64 | | | | | | | | | |

多关键字排序

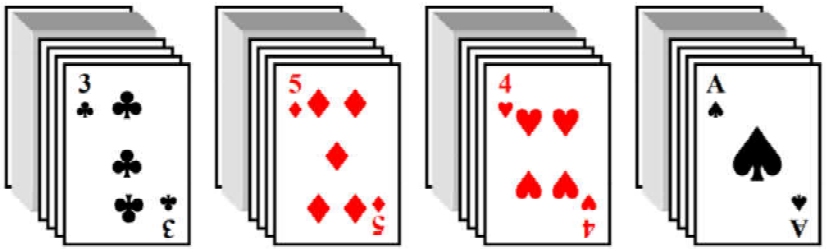
## 一副扑克牌是按2种关键字排序的

$K^0$ [花色] ♣ < ♦ < ♥ < ♠

$K^1$ [面值] 2 < 3 < 4 < 5 < 6 < 7 < 8 < 9 < 10 < J < Q < K < A

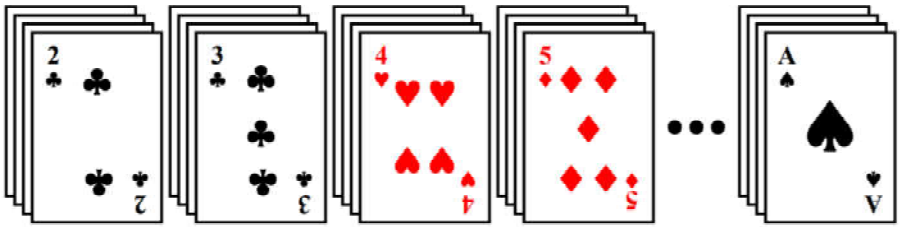
有序结果： 2♣ ... A♣ 2♦ ... A♦ 2♥ ... A♥ 2♠ ... A♠

☞ 用“主位优先”( Most Significant Digit ) 排序：为花色建4个桶



在每个桶内分别排序，最后合并结果。

- 用“次位优先”( Least Significant Digit ) 排序：为面值建13个桶



- 将结果合并，然后再为花色建4个桶

- 问题：LSD任何时候都比MSD快吗？

代码：

```
/* 基数排序 - 次位优先 */

/* 假设元素最多有MaxDigit个关键字，基数全是同样的Radix */

#include<iostream>
using namespace std;
#define MaxDigit 4
#define Radix 10
#define ElementType int
/* 桶元素结点 */
typedef struct Node* PtrToNode;
struct Node {
        int key;
        PtrToNode next;
};

/* 桶头结点 */
struct HeadNode {
        PtrToNode head, tail;
};
typedef struct HeadNode Bucket[Radix];

int GetDigit(int X, int D)
{ /* 默认次位D=1, 主位D<=MaxDigit */
        int d, i;
```

```
        for (i = 1; i <= D; i++) {
                d = X % Radix;
                X /= Radix;
        }
        return d;
}

void LSDRadixSort(ElementType A[], int N)
{ /* 基数排序 - 次位优先 */
        int D, Di, i;
        Bucket B;
        PtrToNode tmp, p, List = NULL;

        for (i = 0; i < Radix; i++) /* 初始化每个桶为空链表 */
                B[i].head = B[i].tail = NULL;
        for (i = 0; i < N; i++) { /* 将原始序列逆序存入初始链表List */
                tmp = (PtrToNode)malloc(sizeof(struct Node));
                tmp->key = A[i];
                tmp->next = List;
                List = tmp;
        }
        /* 下面开始排序 */
        for (D = 1; D <= MaxDigit; D++) { /* 对数据的每一位循环处理 */
                /* 下面是分配的过程 */
                p = List;
                while (p) {
                        Di = GetDigit(p->key, D); /* 获得当前元素的当前位数字 */
                        /* 从List中摘除 */
                        tmp = p; p = p->next;
                        /* 插入B[Di]号桶尾 */
                        tmp->next = NULL;
                        if (B[Di].head == NULL)
                                B[Di].head = B[Di].tail = tmp;
                        else {
                                B[Di].tail->next = tmp;
                                B[Di].tail = tmp;
                        }
                }
                /* 下面是收集的过程 */
                List = NULL;
                for (Di = Radix - 1; Di >= 0; Di--) { /* 将每个桶的元素顺序收集入List */
                        if (B[Di].head) { /* 如果桶不为空 */
                                /* 整桶插入List表头 */
                                B[Di].tail->next = List;
                                List = B[Di].head;
                                B[Di].head = B[Di].tail = NULL; /* 清空桶 */
                        }
                }
        }
        /* 将List倒入A[]并释放空间 */
        for (i = 0; i < N; i++) {
                tmp = List;
                List = List->next;
                A[i] = tmp->key;
                free(tmp);
```

```
        }
}

int main()

{
        int n;
        cin >> n;
        int* a = new int[n];
        for (int i = 0; i < n; i++)
        {
                cin >> a[i];
        }
        LSDRadixSort(a, n);

        for (int i = 0; i < n; i++)
        {
                cout << a[i]<<" ";
        }
        delete[] a;
        return 0;
}
```