//开放地址法构造的哈希表的运算算法

#include <stdio.h>

#include<stdlib.h>

#include<time.h>

#define MaxSize 100 //定义最大哈希表长度

#define NULLKEY -1 //定义空关键字值

#define DELKEY -2 //定义被删关键字值

typedef int KeyType; //关键字类型

typedef struct

{

KeyType key; //关键字域

int count; //探测次数域

} HashTable; //哈希表类型

void InsertHT(HashTable ha[], int& n, int m, int p, KeyType k) //将关键字k插入到哈希表中

{

int i, adr;

adr = k % p; //计算哈希函数值

if (ha[adr].key == NULLKEY || ha[adr].key == DELKEY)//k可以直接放在哈希表中

{

ha[adr].key = k;

ha[adr].count = 1;

}

else //发生冲突时采用线性探测法解决冲突

{

i = 1; //i记录k发生冲突的次数

do

{

adr = (adr + 1) % m; //线性探测

i++;

} while (ha[adr].key != NULLKEY && ha[adr].key != DELKEY);

ha[adr].key = k; //在adr处放置k

ha[adr].count = i; //设置探测次数

}

n++; //总关键字个数增1

}

void CreateHT(HashTable ha[], int& n, int m, int p) //创建哈希表

{

for (int i = 0;i < m;i++) //哈希表置空的初值

{

ha[i].key = NULLKEY;

ha[i].count = 0;

}

n = 0;

//for (int i = 0;i < n1;i++)

// InsertHT(ha, n, m, p, keys[i]); //插入n个关键字

}

int SearchHT(HashTable ha[], int m, int p, KeyType k) //在哈希表中查找关键字k

{

int i = 1, adr;

adr = k % p; //计算哈希函数值

while (ha[adr].key != NULLKEY && ha[adr].key != k)

{

i++; //累计关键字比较次数

adr = (adr + 1) % m; //线性探测

}

if (ha[adr].key == k) { //查找成功

//printf("成功：关键字%d，比较%d次\n", k, i);

return 1;

}

else { //查找失败

//printf("失败：关键字%d，比较%d次\n", k, i);

return 0;

}

}

bool DeleteHT(HashTable ha[], int& n, int m, int p, KeyType k) //删除哈希表中关键字k

{

int adr;

adr = k % p; //计算哈希函数值

while (ha[adr].key != NULLKEY && ha[adr].key != k)

adr = (adr + 1) % m; //线性探测

if (ha[adr].key == k) //查找成功

{

ha[adr].key = DELKEY; //删除关键字k

return true;

}

else //查找失败

return false; //返回假

}

double ASL(HashTable ha[], int n, int m, int p) //求平均查找长度

{

int i, j;

int succ = 0, unsucc = 0, s;

for (i = 0;i < m;i++)

if (ha[i].key != NULLKEY)

succ += ha[i].count; //累计成功时总关键字比较次数

//printf(" 成功情况下ASL(%d)=%g\n", n, succ \* 1.0 / n);

return succ \* 1.0 / n;

for (i = 0;i < p;i++)

{

s = 1; j = i;

while (ha[j].key != NULLKEY)

{

s++;

j = (j + 1) % m;

}

unsucc += s;

}

//printf(" 不成功情况下ASL(%d)=%g\n", n, unsucc \* 1.0 / p);

return unsucc \* 1.0 / p;

}

void DispHT(HashTable ha[], int n, int m, int p) //输出哈希表

{

int i, j;

int succ = 0, unsucc = 0, s;

printf("哈希表:\n");

printf(" 哈希表地址:\t");

for (i = 0;i < m;i++)

printf(" %3d", i);

printf(" \n");

printf(" 哈希表关键字:\t");

for (i = 0;i < m;i++)

if (ha[i].key == NULLKEY)

printf(" "); //输出3个空格

else

printf(" %3d", ha[i].key);

printf(" \n");

printf(" 探测次数:\t");

for (i = 0;i < m;i++)

if (ha[i].key == NULLKEY)

printf(" "); //输出3个空格

else

printf(" %3d", ha[i].count);

printf(" \n");

ASL(ha, n, m, p);

}

//随机产生0~10000范围内的整数

void my\_randHashTable(HashTable ha[],int m,int p)

{

//设置种子

srand((unsigned)time(NULL));

int n = 0;

while (n < m)

{

int k = rand() % 10000;

//printf("随机数是:%d\n", k);

int t = SearchHT(ha, m,p,k);

if (t == 0)

{

InsertHT(ha, n, m, p, k);

printf("%d %g\n", n, ASL(ha, n, m, p));

}

}

}

int main()

{

int keys[] = { 16,74,60,43,54,90,46,31,29,88,77 };

int n, m = 97, p = 97, k;

HashTable ha[MaxSize];

CreateHT(ha, n, m, p);

my\_randHashTable(ha, m, p);

return 0;

/\*HashTable ha[MaxSize];

printf("(1)创建哈希表\n"); CreateHT(ha, n, m, p, keys, 11);

printf("(2)显示哈希表:\n"); DispHT(ha, n, m, p);

k = 29;

printf("(3)查找"); SearchHT(ha, m, p, k);

k = 31;

printf("(4)删除：关键字%d\n", k);

DeleteHT(ha, n, m, p, k);

printf("(5)显示哈希表:\n"); DispHT(ha, n, m, p);

printf("(6)查找"); SearchHT(ha, m, p, k);

printf("(7)插入：关键字%d\n", k);

InsertHT(ha, n, m, p, k);

printf("(8)显示哈希表:\n"); DispHT(ha, n, m, p);

printf("\n");

return 1;\*/

}

分析:

最初哈希表，没有发生冲突的时候，平均查找长度为1;

随着哈希表的长度增加，冲突无可避免，既然无可避免，那就只能解决冲突，这样就会导致查找次数增加，这就会导致平均查找长度增加，这就是为什么曲线图有上升趋势的原因；

有趣的是，曲线在某些地方也会有一个小区间是下降的，那是因为在那个区间哈希表长度不断增加，但是冲突很少或者没有发生，那样平均下来平均查找长度就会下降，但不会低于1。