**Lab4报告文档**

1. **设计思路**
2. 代码中设计了hashNode结构体来存储数据，该结构体中还有状态信息status,表示删除delete或者存在exist，另外一种状态empty用NULL表示(我的哈希表是用vector数组来表示的，动态分配内存构建数组也可以，因为数组中存的数据是hashNode\*，初始化数组时都初始化为NULL)。
3. 因为要处理中文防止乱码，把数据txt文档改成ANSI编码，相应的，代码采用GB2312编码方式。
4. **哈希函数**

1、原本的方法：

同改进后的方法一样先得到key,F(x)也相同，只是在发生碰撞时key不变，只让i++ 。用原本的方法在把学生（共46个）都插入后，再插入4个，发现会因为可存入空间仅剩几个，哈希函数一直死循环得不到仅剩的可存入空间的index。而用改进后的方法就不会出现死循环情况。

完整求index代码(hashkey即为该元素在数组内的index):

        for(int i=0;i<name.size();i++)

        {

            key+=name[i];

        }

        key=int(fabs(key))%capacity;

        hashkey=(key+i\*i)%capacity;

        while(hash[hashkey])

        {

            if(hash[hashkey]->name==name&&hash[hashkey]->status==Exist)

                return hash[hashkey];

            i++;

            hashkey=(key+i\*i)%capacity;//只有这一行和改进后的方法不同

        }

跳出这个while循环之后得到的hashkey就是最终的index。具体见代码。

2、改进后的方法：

Xiugaifor(int i=0;i<student->name.size();i++)

            {

                key+=student->name[i];

                cout<<student->name[i];

            }

            key=fabs(key);

            cout<<key<<' ';

            hashkey=(key%capacity+i\*i)%capacity;

先遍历名字name，把每个字符当成整数加在一起，再取绝对值（因为是中文字符，有时候会得到负数）然后%capacity即为这个名字的key。

设F(x)=(x+i\*i)%capcity，i先置为0

先让Hash(name)=F(key),当发生碰撞就让key=F(key),并且i++，再取F(key)直到没有碰撞就是最终的Hash(name)。

因为我在测试代码中尝试插入节点直到该哈希表全部充满（注：这里说的全部充满不代表每一个位置都是Exist不存在Empty，而是只留一个Empty。因为如果不留下一个Empty的地方，查找这个哈希表中一个不存在的值时就永远无法停止，因为search中只有找到Empty或者找到这个学生，寻找操作才会停止），

完整求index代码(hashkey即为该元素在数组内的index):

        for(int i=0;i<name.size();i++)

        {

            key+=name[i];

        }

       key=int(fabs(key))%capacity;

        hashkey=(key+i\*i)%capacity;

        while(hash[hashkey])

        {

            if(hash[hashkey]->name==name&&hash[hashkey]->status==Exist)

                return hash[hashkey];

            i++;

            hashkey=(hashkey+i\*i)%capacity;

        }

跳出这个while循环之后得到的hashkey就是最终的index。具体见代码。

**三、其他**

1、每个名字的平均搜索次数其实与hash的表格大小（代码中记为capacity）有关，即与负载因子有关，一般哈希表的负载因子大于0.75就要扩容，但是本次代码并没有实现扩容（本人Lab的代码在哈希表全部充满后就不能再insert了，以及这里的全部充满的意思已在上文解释），但是给出了扩容的想法：

当负载因子大于0.75时就进行扩容，可以把之前已存入哈希表中的数据取出，扩容成之前的2倍，把之前的数据求得新的哈希值再放入扩容后的新的哈希表。

2、求出46个学生放入容量为50的哈希表里查询平均次数为2.13043次