数据结构课程设计



班级： 1618403

学号： 161840227

姓名： 韦 鑫

指导教师：孙 涵

目录

1.采用的数据结构 ………………………………………… 3

2.算法设计思想 …………………………………………… 3

3.关键代码 ………………………………………………… 4

4.测试数据和结果 ………………………………………… 20

5.算法的时间复杂度及其改进方法 ……………………… 24

6.结束语 …………………………………………………… 24

一、采用的数据结构

多叉链树

struct person

{

char name[50]; //姓名

date birth; //生日

bool hun; //婚否

char addr[60]; //地址

bool jian; //健在否

date death; //死亡日期

int chilnum; //孩子数

int dai; //代数

person \*chil[maxn]; //孩子指针数组

person \*pa; //父节点

};

二、算法设计思想

①建立家谱树：家族成员信息每行记录一个成员的相关信息，读取时特别需要主要孩子的获取：在创建成员节点并收入除孩子外的信息后，给每个孩子创建节点，并收入名字，将成员的孩子指针(可以动态分配，但在本实验中我使用了定长的指针数组)指向这些孩子，将孩子的父亲指针指向该成员。在收录下一个成员信息前，先用名字（这里不考虑重名）对该成员节点进行搜索，因为他可能是之前节点的孩子，若没有该节点，那么再创建。

②显示家谱：本次实验无法实现题目所述的图形显示家谱图，我采用层次遍历进行显示。

③成员信息查询：通过名字：递归搜索；通过代数：层次遍历；通过出生日期：递归搜索

④两成员关系查询：首先以名字搜索这两位成员的节点，调取他们的代数信息，通过代数的插确定他们的关系，这里不考虑过于复杂的亲属关系。只考虑了父母、伯父母、兄弟、堂兄弟、祖父等。

⑤删除成员：通过名字找到该成员节点，若该成员不是根节点，则整理父母节点中的孩子指针数组，让孩子数减一，再调用destroy()函数对该成员节点进行递归销毁。在此我特判，若删除的点为根节点，直接将整棵树递归删除。

⑥修改成员信息：通过名字进行搜索，再进行修改。

⑦将家谱树中的成员信息写入Excel文件：每个成员的信息按照’\t’为分隔，每个成员间以’\n’为分隔便可将数据存入Excel中。

三、关键代码

//运行环境：Visual Studio 2017

#include<iostream>

#include<stdio.h>

#include<cstdlib>

#include<cctype>

#pragma warning(disable:4996)

using namespace std;

const int maxn = 30;

const int maxv = 1000;

struct date

{

int year, moon, day;

};

struct person

{

char name[50]; //姓名

date birth; //生日

bool hun; //婚否

char addr[60]; //地址

bool jian; //健在否

date death; //死亡日期

int chilnum; //孩子数

int dai; //代数

person \*chil[maxn]; //孩子指针数组

person \*pa; //父节点

};

person \*p;

int tot = 0;//家谱总人数

//设置日期

date set\_date(char arr[])

{

int cnt, y, m, d;

cnt = y = m = d = 0;

for (int i = 0; arr[i] != '\0'; i++)

{

if (arr[i] == '/')cnt++;

else

{

if (cnt == 0)

{

y = y \* 10 + arr[i] - '0';

}

else if (cnt == 1)

{

m = m \* 10 + arr[i] - '0';

}

else

{

d = d \* 10 + arr[i] - '0';

}

}

}

date t;

t.year = y;

t.moon = m;

t.day = d;

return t;

}

//输出日期（格式为 yy-mm-dd）

void print\_date(date a)

{

printf("%d-%d-%d", a.year, a.moon, a.day);

}

//利用名字搜索某节点（这里假设名字不重复）

person \*name\_search(person \*root,char name[])

{

if (!root)return NULL;

if (strcmp(root->name, name) == 0) return root;

for (int i = 0; i < root->chilnum; i++)

{ person \*p = name\_search(root->chil[i], name);

if (p) return p;

}

return NULL;

}

//利用生日查询成员名单

void birth\_search(person \*root, int y, int m, int d)

{

if (root->birth.year == y && root->birth.moon == m && root->birth.day == d) printf("%s\n", root->name);

else

{

for (int i = 0; i < root->chilnum; i++)

{

birth\_search(root->chil[i], y, m, d);

}

}

}

//读取文件建立家谱树

void read(person \*&root)

{

FILE \*fp;

if ((fp = fopen("family\_tree.txt", "r")) == NULL)

{

printf("打开文件失败！\n");

system("pause");

exit(-1);

}

char buff[4096];//缓冲区

char dai[4],name[64], birth[64], hun[8], addr[64], jian[8], death[64], chil[2048];

fgets(buff, 4096, fp);//表头不读

fgets(buff, 4096, fp);

while (!feof(fp))

{

int i = 0,cnt = 0;

//代

while (buff[i] != '\t') dai[cnt++] = buff[i++];

dai[cnt] = '\0';

i++;cnt = 0;

int d = 0,j = 0;

while (dai[j] != '\0') d = d \* 10 + dai[j++] - '0';

//姓名

while (buff[i] != '\t') name[cnt++] = buff[i++];

name[cnt] = '\0';

i++;cnt = 0;

//出生日期

while (buff[i] != '\t') birth[cnt++] = buff[i++];

birth[cnt] = '\0';

i++;cnt = 0;

//婚否

while (buff[i] != '\t') hun[cnt++] = buff[i++];

hun[cnt] = '\0';

i++;cnt = 0;

//地址

while (buff[i] != '\t') addr[cnt++] = buff[i++];

addr[cnt] = '\0';

i++;cnt = 0;

//健在否

while (buff[i] != '\t') jian[cnt++] = buff[i++];

jian[cnt] = '\0';

i++;cnt = 0;

//死亡日期

while (buff[i] != '\t') death[cnt++] = buff[i++];

death[cnt] = '\0';

i++;cnt = 0;

//建立新节点

person \*p = NULL;

if (tot == 0)//录入根节点

{

root = (person \*)malloc(sizeof(person));

for (j = 0; j < maxn; j++) root->chil[j] = NULL;

root->pa = NULL;

root->dai = d; //代

strcpy(root->name, name); //姓名

strcpy(root->addr, addr); //地址

if (strcmp(hun, "是") == 0) root->hun = 1; //婚否

else root->hun = 0;

if (strcmp(jian, "健在") == 0) root->jian = 1; //健在否

else root->jian = 0;

if (!root->jian) root->death = set\_date(death); //死亡日期

root->birth = set\_date(birth); //出生日期

root->chilnum = 0; //孩子个数

}

else//不是根的话则先通过名字找到对应的节点，这里我们假设名字不重复

{

p = name\_search(root, name);

//填写信息

p->dai = d;

strcpy(p->name, name);

strcpy(p->addr, addr);

if (strcmp(hun, "是") == 0) p->hun = 1;

else p->hun = 0;

if (strcmp(jian, "健在") == 0) p->jian = 1;

else p->jian = 0;

if (!p->jian) p->death = set\_date(death);

p->birth = set\_date(birth);

p->chilnum = 0;

}

//孩子指针数组

strcpy(chil, &buff[i]);

i = 0;

do

{

if (strcmp(chil, "无\n") == 0)break;

else

{

cnt = 0;

char temp[50];

while (chil[i] != ' ' && chil[i] != '\n')

{

temp[cnt++] = chil[i++];

}

temp[cnt] = '\0';

//printf("%s\n", temp);

person \*child = (person \*)malloc(sizeof(person));

for (j = 0; j < maxn; j++) child->chil[j] = NULL;

child->chilnum = 0;

strcpy(child->name, temp);

if (tot == 0)//若这些孩子节点的父节点是根

{

child->pa = root;

//printf("pa- %s \n", child->pa->name);

root->chil[root->chilnum] = child;

root->chilnum++;

}

else

{

child->pa = p;

p->chil[p->chilnum] = child;

p->chilnum++;

}

if (chil[i] == '\n')break;

i++;

}

} while (chil[i] != '\n');

tot++;

fgets(buff, 4096, fp);

}

fclose(fp);

}

//输出节点p的相关信息

void print\_info(person \*p)

{

if (p)

{

printf("姓名：%s\n", p->name);

printf("出生日期：");

print\_date(p->birth);

printf("\n");

printf("地址：%s\n", p->addr);

printf("婚否：");

if (p->hun) printf("已婚\n");

else printf("未婚\n");

printf("健在否：");

if (p->jian) printf("健在\n");

else printf("已故\n");

printf("死亡日期：");

if (!p->jian)

{

print\_date(p->death);

printf("\n");

}

else printf("无\n");

printf("父母：%s\n", p->pa->name);

printf("孩子：");

if (p->chilnum == 0)printf("无");

for (int i = 0; i < p->chilnum; i++)

{

printf("%s ", p->chil[i]->name);

}

printf("\n");

}

}

//层次遍历输出第n代人的信息

void layer(person \*root,int n)

{

if (root)

{

person \*Q[maxv];//队列

int front = 0, rear = 0,lastrear = 0,d = 1;

Q[rear++] = root;

lastrear = rear;//用于分层

while (front < rear)

{

person \*fnt = Q[front++];

if (d == n)

{

print\_info(fnt);

printf("======================\n");

}

//孩子入队

for (int i = 0; i < fnt->chilnum; i++)

{

Q[rear++] = fnt->chil[i];

}

if (front == lastrear)

{

lastrear = rear;

d++;

}

}

printf("\n");

}

}

//修改基本信息

void modify(person \*p)

{

char birth[64], death[64], hun[8], jian[8];

printf("姓名：");

scanf("%s", p->name);

printf("出生日期(yy/mm/dd)：");

scanf("%s", birth);

p->birth = set\_date(birth);

printf("地址：");

scanf("%s", p->addr);

printf("婚否(是/否)：");

scanf("%s", hun);

if (strcmp(hun, "是") == 0) p->hun = 1;

else p->hun = 0;

printf("健在否(健在/已故)：");

scanf("%s", jian);

if (strcmp(jian, "健在") == 0) p->jian = 1;

else p->jian = 0;

if (p->jian == 0)

{

printf("死亡日期(yy/mm/dd)：");

scanf("%s", death);

p->death = set\_date(death);

}

}

//输入两人名字确定其关系

void relat(person \*root, char name1[], char name2[])

{

person \*p1, \*p2, \*temp = (person \*)malloc(sizeof(person));

p1 = name\_search(root, name1);

p2 = name\_search(root, name2);

if (!p1 && !p2) printf("家谱中无 %s 、%s\n", name1, name2);

else if (!p1 && p2)printf("家谱中无 %s\n", name1);

else if (p1 && !p2) printf("家谱中无 %s\n", name2);

else

{

int d;

//让p1为长辈，代的数量越小则辈分越大

if (p1->dai >= p2->dai)

{

d = p1->dai - p2->dai;

temp = p1;

p1 = p2;

p2 = temp;

}

else d = p2->dai - p1->dai;

switch (d)

{

case 0:

if (p2->pa == p1->pa) printf("%s 与 %s 是兄弟\n", p1->name, p2->name);

else printf("%s 与 %s 是堂兄弟\n", p1->name, p2->name);

break;

case 1:

if (p2->pa == p1) printf("%s 是 %s 的父母\n", p1->name, p2->name);

else printf("%s 是 %s 的伯父母\n", p1->name, p2->name);

break;

case 2:

if (p2->pa->pa == p1) printf("%s 是 %s 的祖父母\n", p1->name, p2->name);

else printf("%s 是 %s 的伯祖父母\n", p1->name, p2->name);

break;

case 3:

if (p2->pa->pa->pa == p1) printf("%s 是 %s 的曾祖父母\n", p1->name, p2->name);

else printf("%s 是 %s 的伯曾祖父\n", p1->name, p2->name);

break;

case 4:

if (p2->pa->pa->pa->pa == p1) printf("%s 是 %s 的高祖父母\n", p1->name, p2->name);

else printf("%s 是 %s 的伯高祖父\n", p1->name, p2->name);

break;

}

}

}

//删除某成员(若有后代，一并删除)

//递归销毁以p为根的子树

void destroy(person \*&p)

{

if (!p)return;

for (int i = 0; i < p->chilnum; i++)

{

destroy(p->chil[i]);

}

//printf("%s\n", root->name);

free(p);

p = NULL;

tot--;

}

//删除某成员

void del\_member(person \*&root)

{

char name[64],ch;

printf("输入要删除成员的姓名：");

scanf("%s", name);

person \*p = name\_search(root, name);

if (p)

{

print\_info(p);

getchar();

printf("确定删除？(y/n)：");

scanf("%c", &ch);

bool flag = 1;//特判若删除整棵家谱树

if (toupper(ch) == 'Y')

{

//当p不为根时，整理p父母节点的孩子指针数组并将孩子数减一

if (p != root)

{

flag = 0;

person \*pa = p->pa;

int i = 0;

//找到要删除的位置

while (pa->chil[i] != p) i++;

//左移

for (int j = i; j < pa->chilnum - 1; j++)

{

pa->chil[j] = pa->chil[j + 1];

}

pa->chilnum--;

}

destroy(p);

if (flag) root = NULL;

}

}

else printf("家谱中无此人！\n");

}

//某成员添加孩子

void add\_child(person \*&root)

{

char name[64];

printf("请输入要添加孩子的成员姓名：");

scanf("%s", name);

person \*p = name\_search(root, name);

if (p)

{

print\_info(p);

person \*child = (person \*)malloc(sizeof(person));

for (int i = 0; i < maxn; i++)

{

child->chil[i] = NULL;

}

printf("===================\n");

printf("要添加的孩子信息\n");

modify(child);

p->chil[p->chilnum++] = child; //往孩子指针数组中添加元素

child->pa = p; //指向父节点

child->dai = p->dai + 1; //代数在父节点代数的基础上加1

child->chilnum = 0; //孩子数为0

}

else printf("家谱中无此人！\n");

}

//测试输出

void test(person \*root)

{

if (root)

{

person \*Q[maxv];//队列

int front = 0, rear = 0, lastrear = 0;

Q[rear++] = root;

lastrear = rear;//用于分层

while (front < rear)

{

person \*fnt = Q[front++];

printf("%s ", fnt->name);

//孩子入队

for (int i = 0; i < fnt->chilnum; i++)

{

Q[rear++] = fnt->chil[i];

}

if (front == lastrear)

{

printf("\n");

lastrear = rear;

}

}

printf("\n");

}

}

//可视化输出(先用层次遍历替代吧。。test)

void view\_print(person \*root)

{

}

//按层序遍历的顺序写入文件

void output(person \*root)

{

FILE \*fp;

if ((fp = fopen("update\_family\_tree.xls","w")) == NULL)

{

printf("文件打开失败！\n");

exit(-1);

}

//写入表头

fprintf(fp, "代\t姓名\t出生日期\t婚否\t地址\t健在否\t死亡日期\t孩子\n");

person \*Q[maxv];//队列

int front = 0, rear = 0;

Q[rear++] = root;

while (front < rear)

{

person \*fnt = Q[front++];

//写入代、姓名、出生日期

fprintf(fp, "%d\t%s\t%d/%d/%d\t", fnt->dai, fnt->name, fnt->birth.year, fnt->birth.moon, fnt->birth.day);

//写入婚否

if (fnt->hun) fprintf(fp, "是\t");

else fprintf(fp, "否\t");

//写入地址

fprintf(fp, "%s\t", fnt->addr);

//写入健在否

if (fnt->jian) fprintf(fp, "健在\t");

else fprintf(fp, "已故\t");

//写入死亡日期

if (fnt->jian) fprintf(fp, "无\t");

else fprintf(fp, "%d/%d/%d\t", fnt->death.year, fnt->death.moon, fnt->death.day);

//写入孩子的同时将孩子指针入队

for (int i = 0; i < fnt->chilnum; i++)

{

Q[rear++] = fnt->chil[i];

fprintf(fp, "%s", fnt->chil[i]->name);

if (i != fnt->chilnum - 1) fprintf(fp, " ");

else fprintf(fp, "\n");

}

if (fnt->chilnum == 0) fprintf(fp, "无\n");

}

fclose(fp);

printf("写入文件成功！\n");

}

int main()

{

person \*root,\*p = NULL;

read(root);

while (1)

{

system("cls");

printf

(

"1.显示家谱\n"

"2.信息查询\n"

"3.关系查询\n"

"4.删除成员\n"

"5.添加孩子\n"

"6.修改信息\n"

"7.写入文件\n"

"8.退出"

"请输入你的选择："

);

int ch,flag = 0;

scanf("%d", &ch);

system("cls");

switch (ch)

{

case 1://显示家谱

test(root);

break;

case 2://信息查询

printf

(

"1.按姓名查询\n"

"2.按代数查询\n"

"3.按生日查询\n"

"请输入你的选择："

);

int c;

scanf("%d", &c);

system("cls");

switch (c)

{

case 1:

char name[64];

printf("请输入姓名：");

scanf("%s", name);

p = NULL;

p = name\_search(root, name);

if (!p) printf("家谱中无 %s\n", name);

else

{

print\_info(p);

if (p->pa)//输出父母信息

{

printf("\n=======父母信息=======\n");

print\_info(p->pa);

}

if (p->chilnum != 0)

{

printf("\n=======孩子信息=======\n");

for (int i = 0; i < p->chilnum; i++)

{

printf("======================\n");

print\_info(p->chil[i]);

}

}

}

break;

case 2:

int n;

printf("请输入代数：");

scanf("%d", &n);

layer(root, n);

break;

case 3:

int y, m, d;

printf("年：");

scanf("%d",&y);

printf("月：");

scanf("%d", &m);

printf("日：");

scanf("%d", &d);

printf("输出成员名单：\n");

birth\_search(root, y, m, d);

printf("\n");

break;

}

break;

case 3://关系查询

char name1[64], name2[64];

printf("姓名1：");

scanf("%s", name1);

printf("姓名2：");

scanf("%s", name2);

relat(root, name1, name2);

break;

case 4://删除成员

del\_member(root);

break;

case 5://添加孩子

add\_child(root);

break;

case 6://修改信息

char name[64];

printf("请输入需修改的姓名：");

scanf("%s", name);

p = NULL;

p = name\_search(root, name);

if (p)

{

printf("修改前\n");

print\_info(p);

printf("======================\n");

modify(p);

}

else printf("家谱中无 %s\n", name);

break;

case 7://写入文件

output(root);

break;

case 8://退出

flag = 1;

break;

}

if (flag) break;

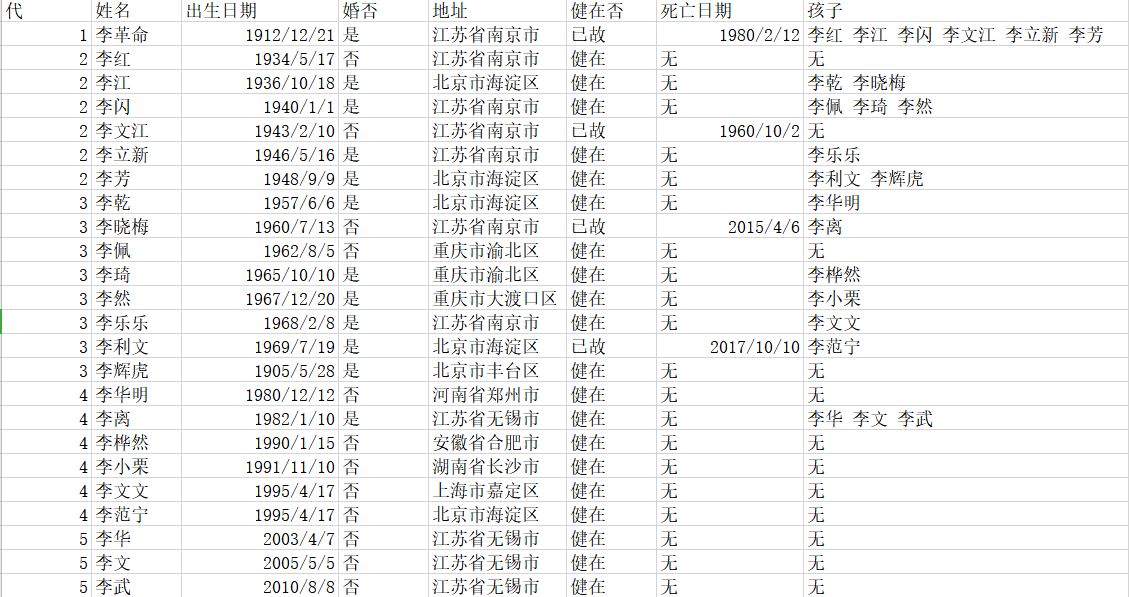
system("pause");

}

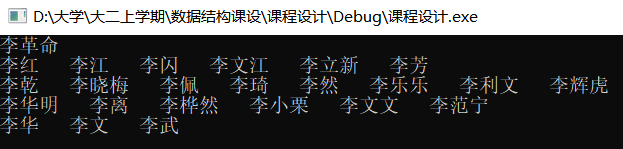
return 0;

}

四、测试数据和结果

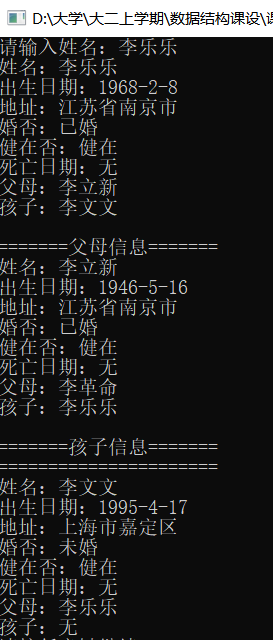


显示家谱：

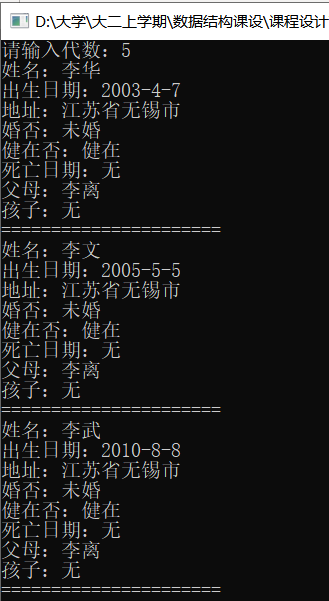


成员查找：

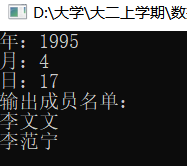
1. 按姓名



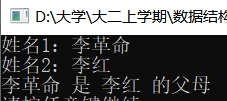
1. 按代数



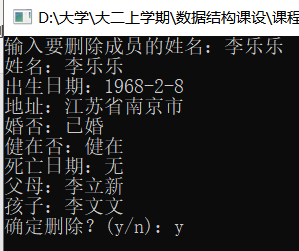
1. 按出生日期

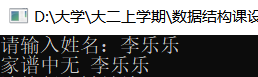


关系查询：

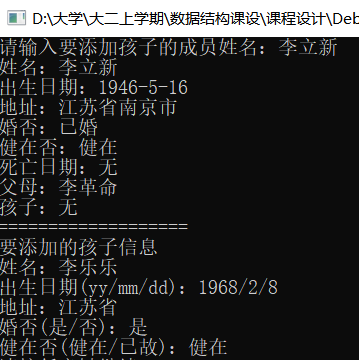


删除成员：

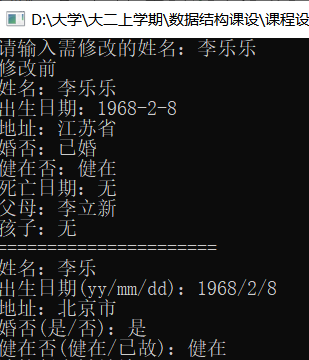




添加孩子：

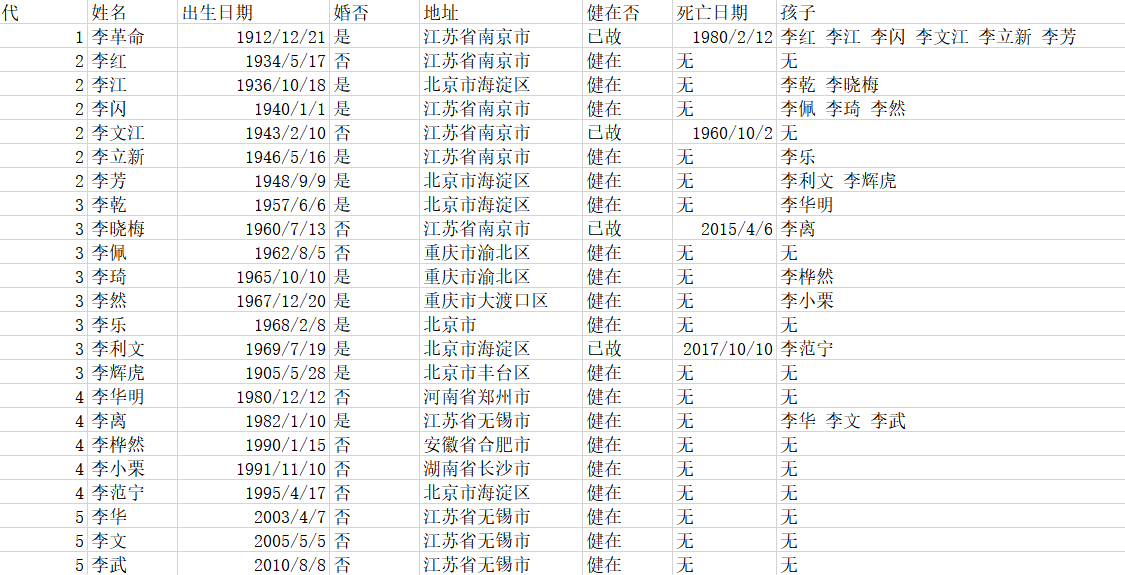


修改信息：



写入文件：





五、算法的时间复杂度即改进方法

主要的功能为查询

查询：O(m\*logn) m为最大孩子数

六、结束语

代码共674行

我个人认为该实验中除了无法用图形显示家谱树外，整体上写得还不错，另外孩子指针数组若以动态分配方式的话会更好。该实验我没有使用递归操作灵活的孩子-兄弟表示的存储结构，而是采用了多叉链树，我认为后者更好理解，在大脑中也能直观的呈现。