

算法与数据结构设计报告

（ 2020 / 2021学年 第 一 学期）

题 目：字符串最长公共子序列求解问题

基于角色的简易家谱管理系统

|  |  |
| --- | --- |
| **专 业** | **计算机科学与技术** |
| **学 生 姓 名** | **张颖** |
| **班 级 学 号** | **B18030406** |
| **指 导 教 师** | **徐小龙** |
| **指 导 单 位** | **计算机科学与技术系** |
| **日 期** | **2020.10.26-2020.11.08** |

|  |  |  |
| --- | --- | --- |
| **支撑指标点** | **评价准则** | **计分（每项10分）** |
| 课程目标1：文献调研与资料收集能力，问题发现、研究、分析与解决能力（20分） | 1、能够掌握算法与数据结构设计的相关基础知识，并能够针对求解的工程问题，收集资料进行合理的分析与设计 |  |
| 2、通过调研，能够选择合适的程序设计语言与编程开发平台，对求解的工程问题进行编程实现 |  |
| 课程目标2：通过课程设计，培养学生综合应用算法和数据结构等知识解决工程问题的实践能力（20分） | 3、能够给出数据结构和算法的设计描述，给出关键算法的流程图或伪代码，并给出各算法之间的结构关系描述 |  |
| 4、具备一定的人机交互设计意识，人机交互设计合理、友好，操作简便 |  |
| 课程目标3：培养解决工程问题的开发工具运用能力，能够利用程序设计软件或系统对问题求解进行模拟和实现，能够设计测试数据验证问题解决方法的正确性，并能够对问题解决方法的性能和效率进行分析（40分） | 5、具备一定的算法与数据结构设计分析能力，能够完成课题要求的各项任务和指标 |  |
| 6、能够结合计算机软硬件资源，合理选用算法、数据结构、数据存储方式等技术手段，对求解的工程问题进行有效建模和求解 |  |
| 7、具备一定自学能力与探索创新意识，能够充分利用教科书及其资源（如网络等）自学新知识与新技能 |  |
| 8、掌握调试方法与工具，对程序开发过程中出现的问题进行分析、跟踪与调试，并能够进行充分测试 |  |
| 课程目标4：选择同类课题的学生能够通过讨论和交流解决课程设计中的难题，能在实验报告中准确阐述课程设计的内容，能够清晰陈述观点和回答问题（20分） | 9、能够正确、完整地回答指导教师关于课题的问询，反映其对课题内容，以及相关的工程基础知识具有较好的理解和掌握 |  |
| 10、具备一定的语言表达能力与文字处理能力，能够结合复杂工程问题撰写报告，报告内容和实验数据详实，格式规范 |  |
| 算法与数据结构设计能力测评总分 | |  |
| **指导教师： 年 月 日** | | |
| **备注：** | | |

**字符串的最长公共子序列求解问题**

**一、课题内容和要求**

本课题目标系统“字符串最长公共子序列求解”的功能框架图如图1所示。

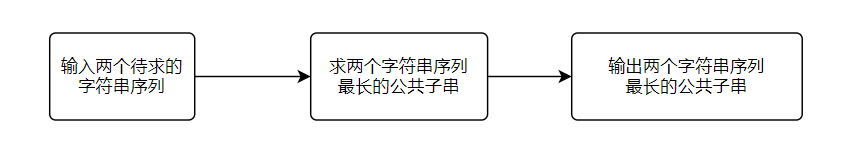


图1 功能框架图

（1）最长公共子串为在两个待求字符串中从左到右都出现的连续的最长字符串序列

（2）输入两个待求解的序列进行求解最长公共子串

（3）最长公共子串不唯一时支持输出全部公共子串

**二、数据结构说明**

struct longestCommonSubstring{ //最长公共子串

dp: Array, //用来存放x[i]和y[j]为止两个序列的最长公共子串长度

result: Array, //最长公共子串

res: Array, //公共子串长度 – 其中之一参数字符串映射表

max: String, //最长公共子串长度

};

**三、算法设计**

1 求最长公共子串长度算法

使用动态规划的算法，用dp[i][j] 表示Xi 和 Yj 的LCS的长度（直接保存最长公共子序列的中间结果不现实，需要先借助LCS的长度）。其中X = {x1 ... xm}，Y ={y1...yn}，Xi = {x1 ... xi}，Yj={y1... yj}。可得递归公式如下：

0, i =0 或j = 0

dp[i,j] dp[i - 1][j - 1] + 1, i,j > 0, xi = yi

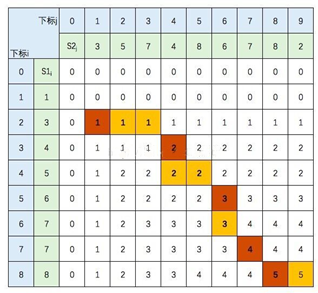


图2 最长公共子序列长度算法示例图

2 求最长公共子串算法

建好dp表后，将dp表中每行/每列中最大的数值（公共子串的长度）映射到行/列代表的参数上，并多增加一个0在数组末尾方便操作，最后遍历这个数组，判断当前项的值是否等于最长子串长度，是则进行切割子串并添入结果数组。

**四、详细设计**

function longestCommonSubstring(x,y) {

let dp = [], result = [],res = [];

//初始化dp数组

for(let index = 0; index<=x.length ;index++){

let arr = new Array(y.length + 1).fill(0)

dp.push (arr)

}

//填表

for(let i = 1; i <= x.length; i++){

for(let j = 1; j <= y.length; j++){

if(x[i - 1] == y[j - 1]){

dp[i][j] = dp[i - 1][j - 1] + 1;

}

}

}

//将dp映射到其中一参数上，

for(let index = 1 ;index<dp.length;index++){

res.push(Math.max(...dp[index]))

}

res.push(0)

let max = Math.max(...res)

//找最长公共子串

for(let index =0;index< res.length;index++){

if(res[index] == max){

result.push(x.slice(index - res[index] + 1,index + 1))

}

}

return result;

}

**五、测试数据及其结果分析**

测试1：输入的两组序列只有一队最长公共子序列时的测试情况

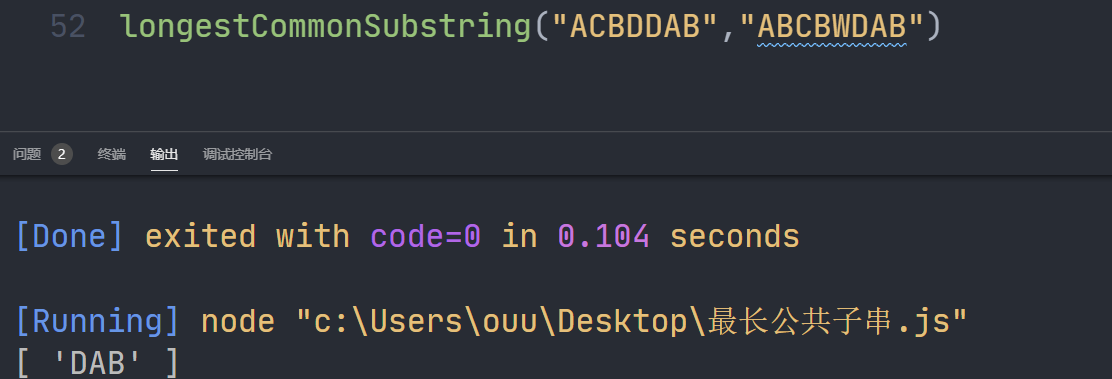


图3 第一次测试数据

测试2：输入的两组序列有多队最长公共子序列的测试情况

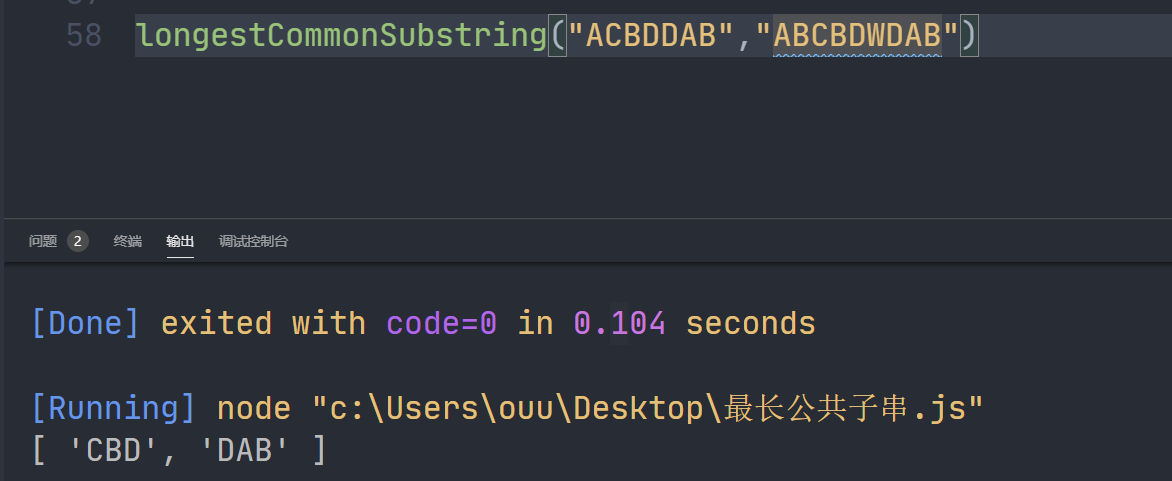


图4 第二次测试数据

**六、算法设计和程序调试过程中的问题**

**问题1**：最开始在输入的两组序列只有一队最长公共子串时，输出了所有的公共子串

**解决方法**：遍历公共子串数组，移除长度不等于最长公共子串的长度的公共子串

**问题2**：感觉问题1的解决办法比较繁琐，要经过两次循环，想优化算法

**解决方法**：直接在切割公共子串的时候就只切割长度为最长公共子串的公共子串

**问题3**：求出最长公共子串长度的时候不明白要怎么切割公共子串

**解决方法**：将dp表映射到参数上，即在映射表中参数的下标表示的值为当前字符的公共子串长度，就可以利用这个来切割公共子串了

**基于角色的简易家谱管理系统**

**一、课题内容和要求**

本课题目标系统“基于角色的简易家谱管理系统”的功能框架图如图1所示。

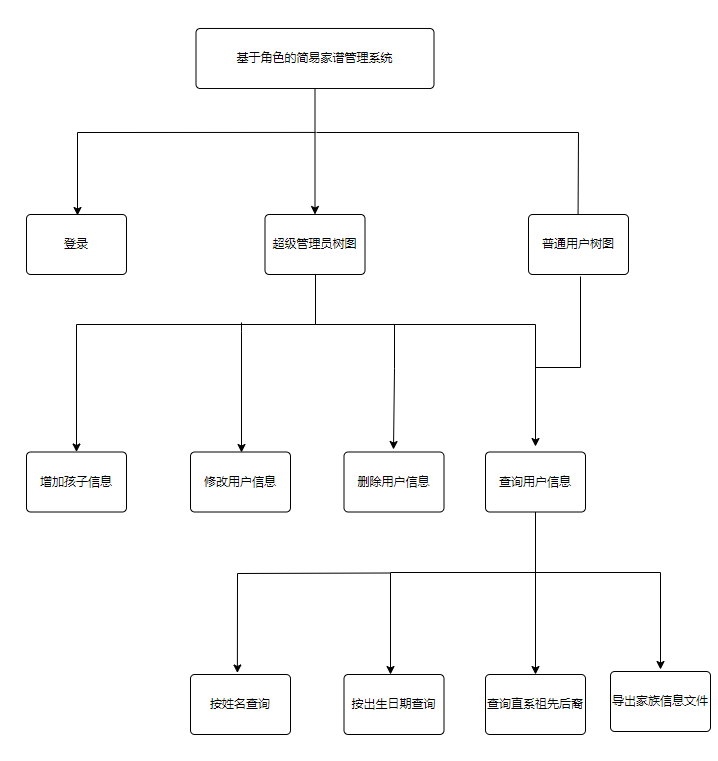


图1 功能框架图

（1）需要设置普通用户、超级管理员不同角色，不同角色登录后的权限各不相同，普通用户可以进行查询；超级管理员有对所有成员增加、删除和修改的权限。

（2）家谱中成员的信息中均应包含姓名、出生日期、婚否、地址、健在否、死亡日期（若其已死亡）等 ，也可附加其它信息，并存储于文本文件中。

（3）查询功能。可按照姓名查询，输出成员信息（包括其本人、父亲、孩子的信息、所在辈分）；按照出生日期查询成员名单。

（4）按出生日期对家谱中所有人排序。

（5）打开家谱时，提示当天生日的健在成员。

（6）输入任意两个姓名，能够查询这两个姓名之间的是否具有直系祖先后裔关系。

（7）以图形方式显示家谱。

**二、数据结构说明**

struct person{ //用户个人信息

name: "", //用户姓名

birthday: "", //用户出生日期

address: "", //用户家庭住址

livingStatus: true, //用户是否健在

dateOfDeath: "", //用户死亡日期

portraitUrl:" ", //用户头像信息

maritalStatus: false, //用户是否婚配

children: [], //用户孩子信息

mate:{ //用户配偶

name: "", //用户配偶姓名

birthday: "", //用户配偶出生日期

address: "", //用户配偶家庭住址

livingStatus: true, //用户配偶是否健在

dateOfDeath: "", //用户配偶死亡日期

portraitUrl:"" //用户配偶头像信息

}

};

**三、算法设计**

1 更新用户信息算法

遍历树图，找到当前操作的用户的父节点，先删掉原本的用户信息子节点，再遍历该父节点下的全部子节点，根据出生日期的先后顺序插入更新的用户信息子节点，完成更新用户信息。

2 增加孩子信息算法

遍历树图，找到要增加孩子信息的父节点，遍历该父节点下的全部子节点，根据出生日期的先后顺序插入新增的孩子信息子节点，完成增加孩子信息。

3 删除孩子信息算法

遍历树图，找到要删除孩子信息的父节点，遍历该父节点下的全部子节点，根据待删除孩子信息的名字找到该子节点下标并从父节点的子节点列表中移除，完成删除孩子信息。

4 删除配偶信息算法

遍历树图，找到要删除配偶信息的用户信息节点，删除用户信息下的配偶信息，完成删除配偶信息。

5 导出家族成员信息并下载算法

遍历树图，将每一个用户信息节点加入表中，如果该用户已经婚配，则将该用户的配偶信息也加入表中，并将用户信息中的配偶姓名填写上配偶名字，同时处理一些信息中的布尔值（将其转换成中文），最后再将表头与对应变量的key值一一对应清楚，完成导出家族成员信息算法。

6 按姓名搜索算法

遍历树图，根据搜索条件：姓名找到对应的用户信息节点，给它加上新类名表示选中，页面上以左右晃动提示用户搜索结果，如果没有搜索到则弹出对话框提示用户未找到，完成按姓名搜索算法。

7 按出生日期搜索算法

遍历树图，根据搜索条件：出生日期找到对应的用户信息节点，给它加上新类名表示选中，页面上以左右晃动提示用户搜索结果，如果没有搜索到则弹出对话框提示用户未找到，完成按出生日期搜索算法。

8 查询直系祖先后裔关系算法

遍历树图，根据搜索条件：两个名字，判断这两个人是否有一方为配偶，有则弹框提示用户这两个人不具有直系祖先后裔关系，否则提示用户这两个人具有直系祖先后裔关系，完成查询直系祖先后裔关系算法。

9 登录算法：区分用户权限

判断用户账户是否为超级管理员账户root，是则进入超级管理员树图界面，否则进入普通用户树图界面。

**四、详细设计**

/\*\*

\* @description: 更新用户信息

\* @param {Array} tree 当前循环的列表

\* @param { String } parent 当前选中准备进行操作的用户的父节点姓名

\* @param { String } current 当前选中准备进行操作的用户姓名

\* @param {Object} updateInformation 需要更新的用户信息

\*/

const updatePerson = (tree,parent,current,updateInformation) => {

let list = genealogy;

for (let node of tree) {

if (node.name === parent) {

let index = node.children.findIndex(child => child.name === current)

if(index !== -1){

node.children.splice(index,1)

for(let childIndex in node.children){

if(node.children[childIndex].birthday > updateInformation.birthday){

node.children.splice(childIndex,0,updateInformation)

setGenealogy(list)

setChange(!change)

return null

}

}

node.children.push(updateInformation)

setGenealogy(list)

setChange(!change)

return null

}

}

if (node.children) {

updatePerson(node.children,parent,current,updateInformation)

}

}

}

/\*\*

\* @description: 增加孩子信息

\* @param {Array} tree 当前循环的列表

\* @param { String } parent 当前选中准备进行操作的用户的父节点姓名

\* @param {Object} newPerson 需要增加的孩子信息

\*/

const addPerson = (tree,parent,newPerson) => {

let list = genealogy;

for (const node of tree) {

if (node.name === parent) {

for(let childIndex in node.children){

if(node.children[childIndex].birthday > newPerson.birthday){

node.children.splice(childIndex,0,newPerson)

setGenealogy(list)

setChange(!change)

return null

}

}

node.children.push(newPerson)

setGenealogy(list)

setChange(!change)

return null

}

if (node.children) {

addPerson(node.children, parent,newPerson)

}

}

}

/\*\*

\* @description: 删除孩子信息

\* @param {Array} tree 当前循环的列表

\* @param {String} parent 当前选中准备进行操作的用户的父节点姓名

\* @param { String } deleteChildName 需要删除的孩子姓名

\*/

const deleteChild = (tree,parent,deleteChildName) => {

let list = genealogy;

for (const node of tree) {

if (node.name === parent) {

let index = node.children.findIndex(child => child.name === deleteChildName)

if(index !== -1){

node.children.splice(index,1)

setGenealogy(list)

setChange(!change)

}

return null

}

if (node.children) {

deleteChild(node.children, parent,deleteChildName)

}

}

}

/\*\*

\* @description: 删除配偶信息

\* @param {\*} tree 当前循环的列表

\* @param { String } deleteMateName 需要删除的配偶姓名

\*/

const deleteMate = (tree,deleteMateName) => {

let list = genealogy;

for (const node of tree) {

if (node.name === deleteMateName) {

delete node.mate

setGenealogy(list)

setChange(!change)

return null

}

if (node.children) {

deleteMate(node.children,deleteMateName)

}

}

}

/\*\*

\* @description: 导出家族成员信息并下载

\* @param {Function} genealogy 家族成员信息

\*/

function downloadExcel(genealogy) {

let option = {}

let data = []

const getInformation = (tree) => {

for (const node of tree) {

node.maritalStatus = node.maritalStatus ? "是" : "否"

node.livingStatus = node.livingStatus ? "是" : "否"

node.mateName = node.mate ? node.mate.name : ""

data.push(node)

if(node.mate){

node.mate.maritalStatus = node.mate.maritalStatus ? "是" : "否"

node.mate.livingStatus = node.mate.livingStatus ? "是" : "否"

node.mate.mateName = node.name

data.push(node.mate)

}

if (node.children && node.children.length) {

getInformation(node.children)

}

}

}

getInformation(genealogy)

if (data) {

option.fileName = "家族成员信息"

option.datas = [

{

sheetData: data,

sheetName: '家族成员信息表',

sheetFilter: ["name","birthday","maritalStatus","address","livingStatus","dateOfDeath","mateName"],

sheetHeader: ["名字","出生日期","是否婚配","家庭住址","是否健在","死亡日期","配偶姓名"]

}

]

}

const exportExcel = new ExportJsonExcel(option)

exportExcel.saveExcel()

}

/\*\*

\* @description: 按姓名搜索

\* @param {String} name 搜索条件名字

\*/

const searchByName = (name) => {

let nodeList = document.querySelectorAll(".information")

let node

for(let child of nodeList){

node = child.parentNode.parentNode

node.className = ""

if(child.innerText === name){

node.className = "searchNode"

setSearch("")

}

}

}

/\*\*

\* @description: 按照出生日期查询

\* @param {String} birthday 搜索条件出生日期

\*/

const searchByBirthday = (birthday) => {

let nodeList = document.querySelectorAll(".birthdayDate")

let node

for(let child of nodeList){

node = child.parentNode.parentNode.parentNode.parentNode.parentNode

node.className = ""

if(child.innerText === birthday){

node.className = "searchNode"

setSearch("")

}

}

}

/\*\*

\* @description: 查询直系祖先后裔关系

\* @param {String} name 搜索条件之一名字

\* @param {String} comparativeName 搜索条件之一名字

\*/

const searchRelationship = (name,comparativeName) => {

let nodeList = document.querySelectorAll(".information")

let result = "具有直系祖先后裔关系"

for(let child of nodeList){

if(child.innerText === name || child.innerText === comparativeName){

if(child.parentNode.classList.contains("mate")){

result = "不具有直系祖先后裔关系"

break ;

}

}

}

alert(result)

setSearch("")

}

/\*\*

\* @description: 登录：进行用户权限和管理员权限的区分

\*/

const login = () => {

if(state.account === "root"){

getPermissions("root")

}else{

getPermissions("user")

}

}

**五、测试数据及其结果分析**

图2 普通用户与超级管理员登录界面

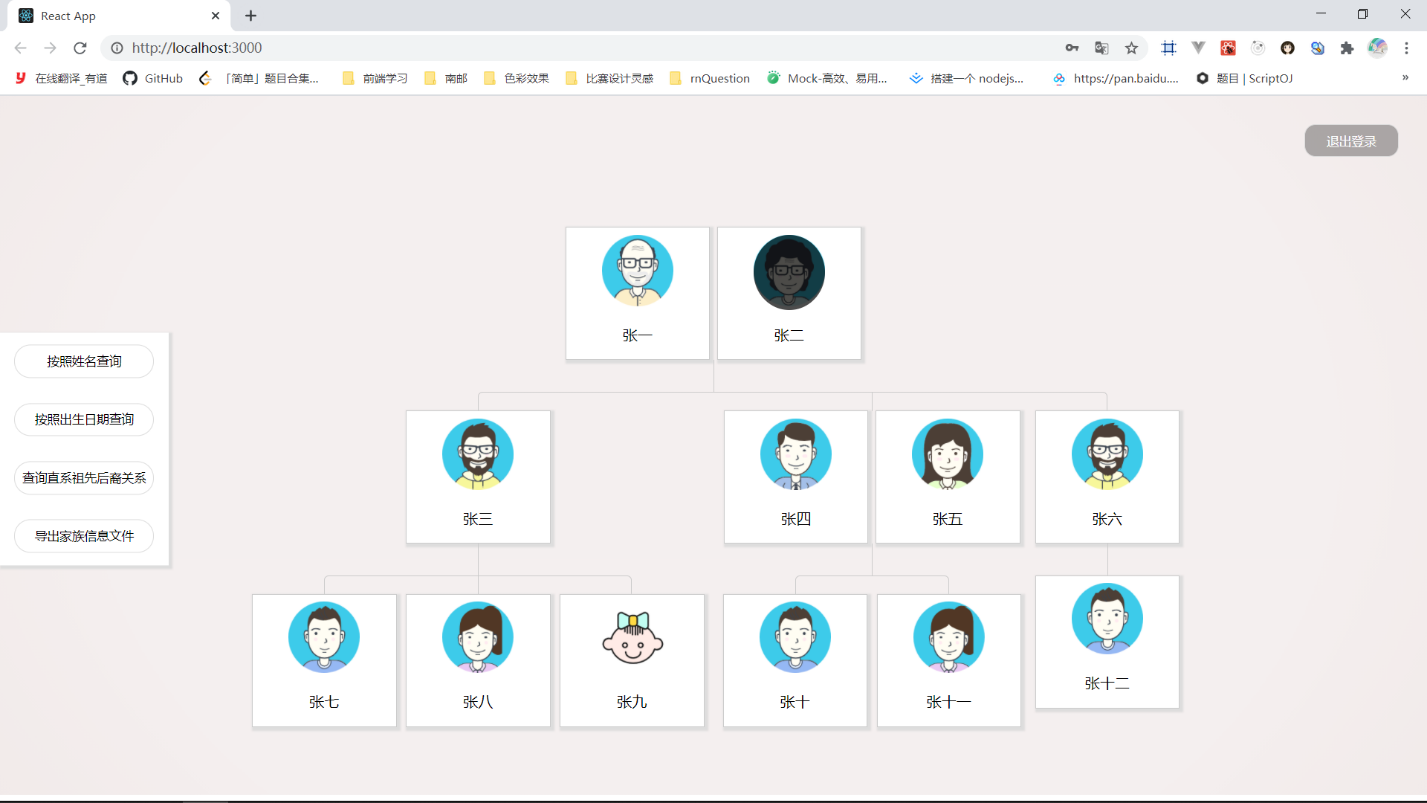


图3 普通用户与超级管理员树图展示家谱图界面

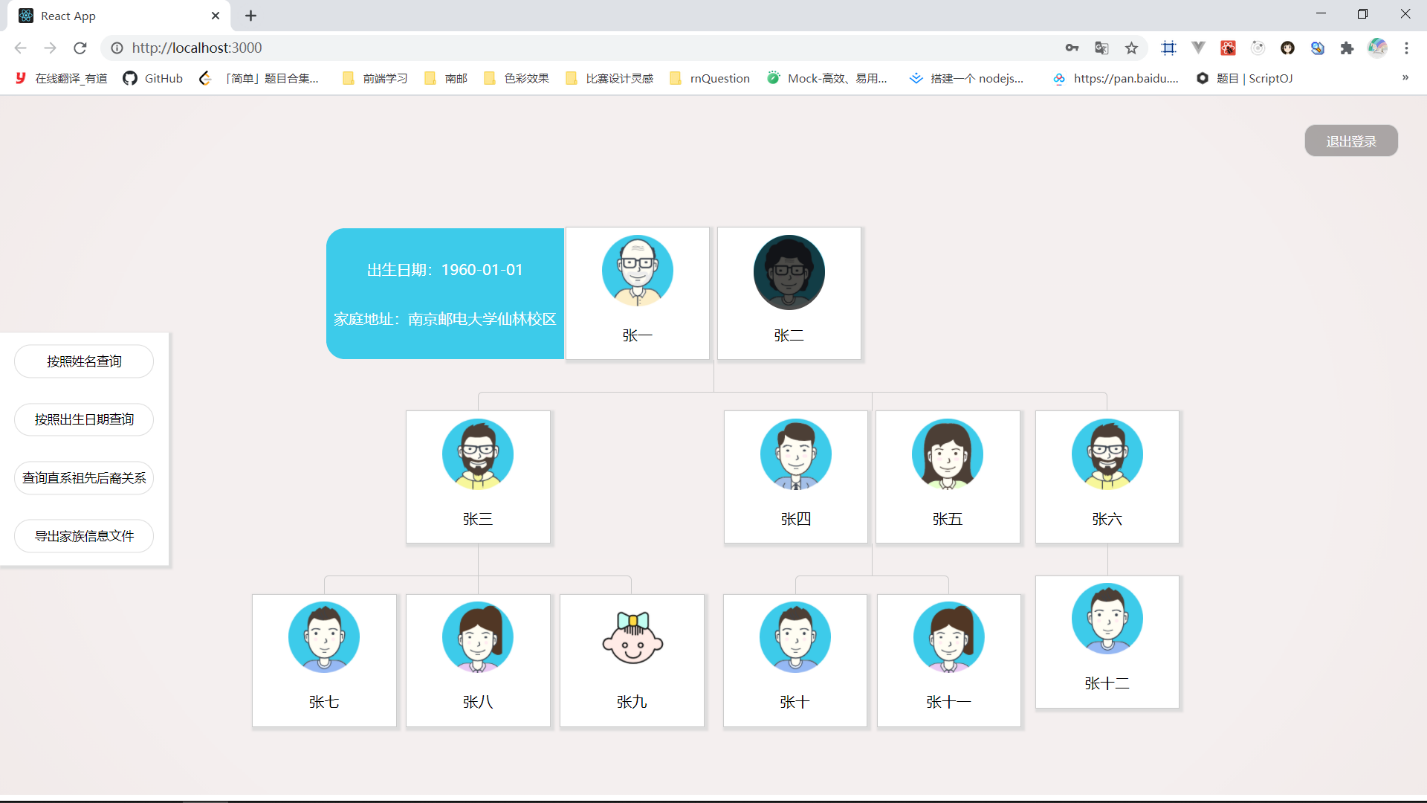


图4 普通用户与超级管理员树图节点展示用户信息界面

图5 按姓名查询家族成员查询成功界面（此处为动画，不便截图）

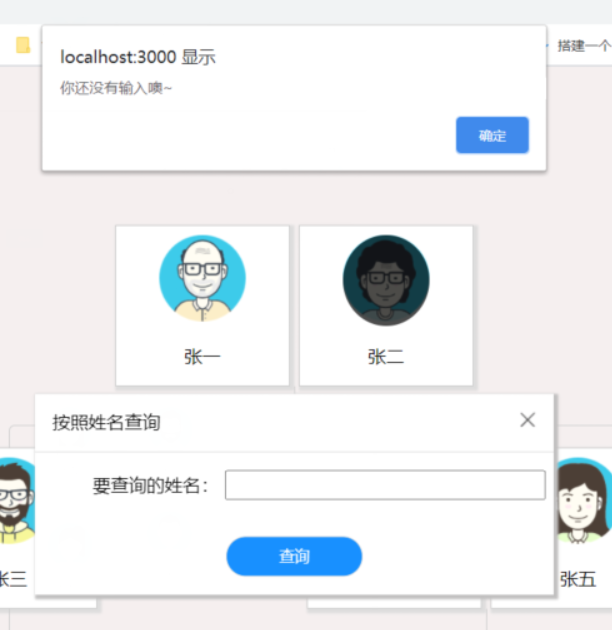
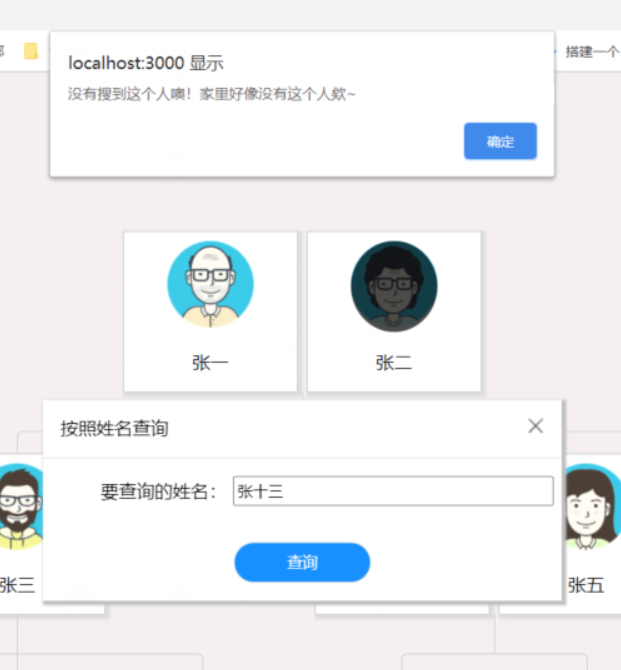
 

图6 按姓名查询家族成员查询失败界面

图7 按出生日期查询家族成员查询成功界面（此处为动画，不便截图）

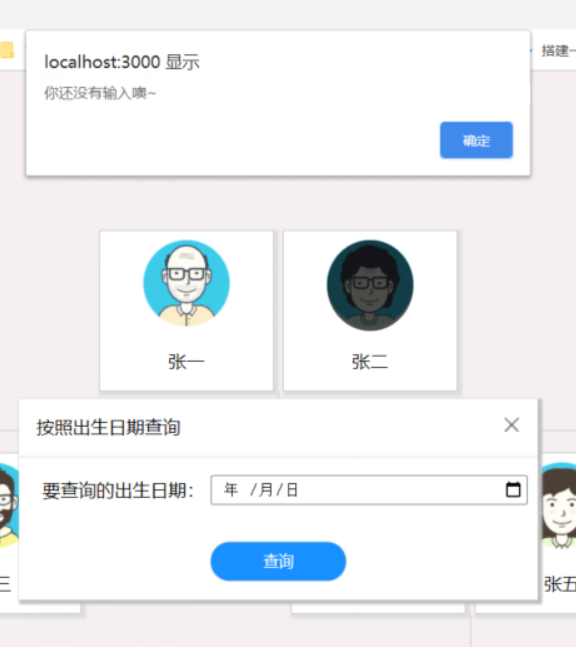
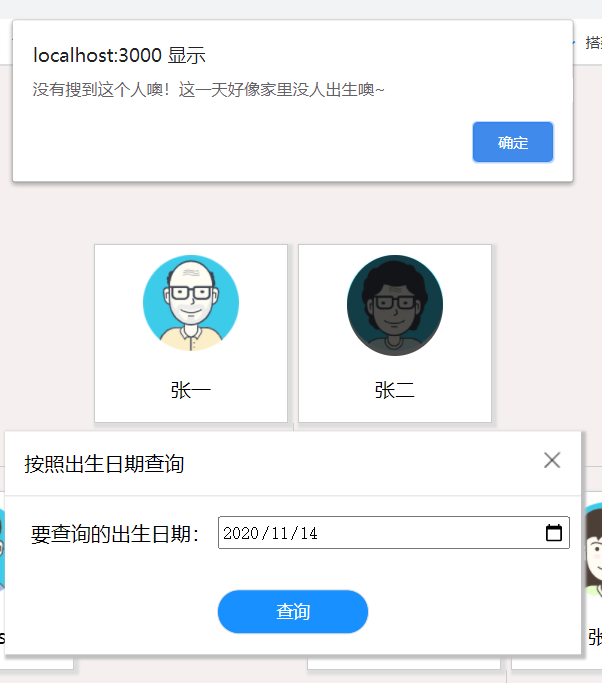
 

图8 按出生日期查询家族成员查询失败界面

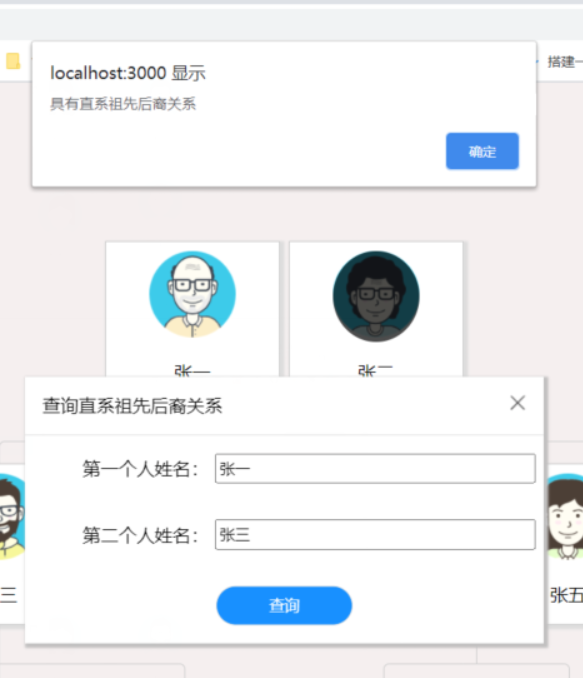
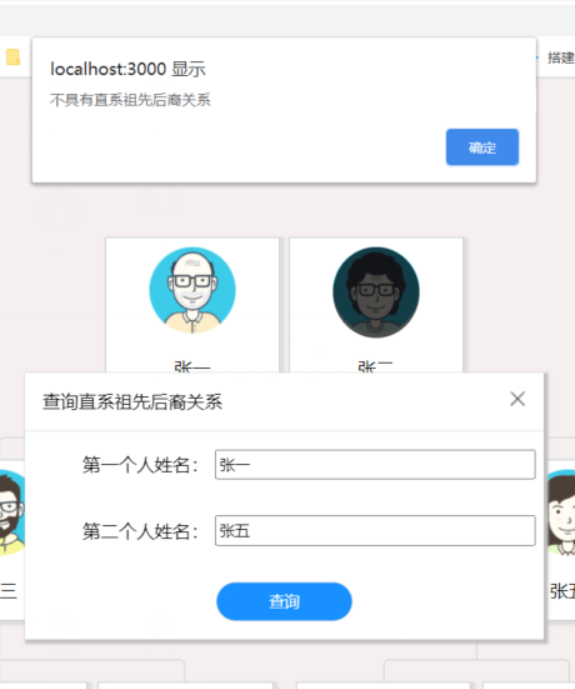
 

图7 查询家族成员直系祖先后裔关系查询成功与失败界面

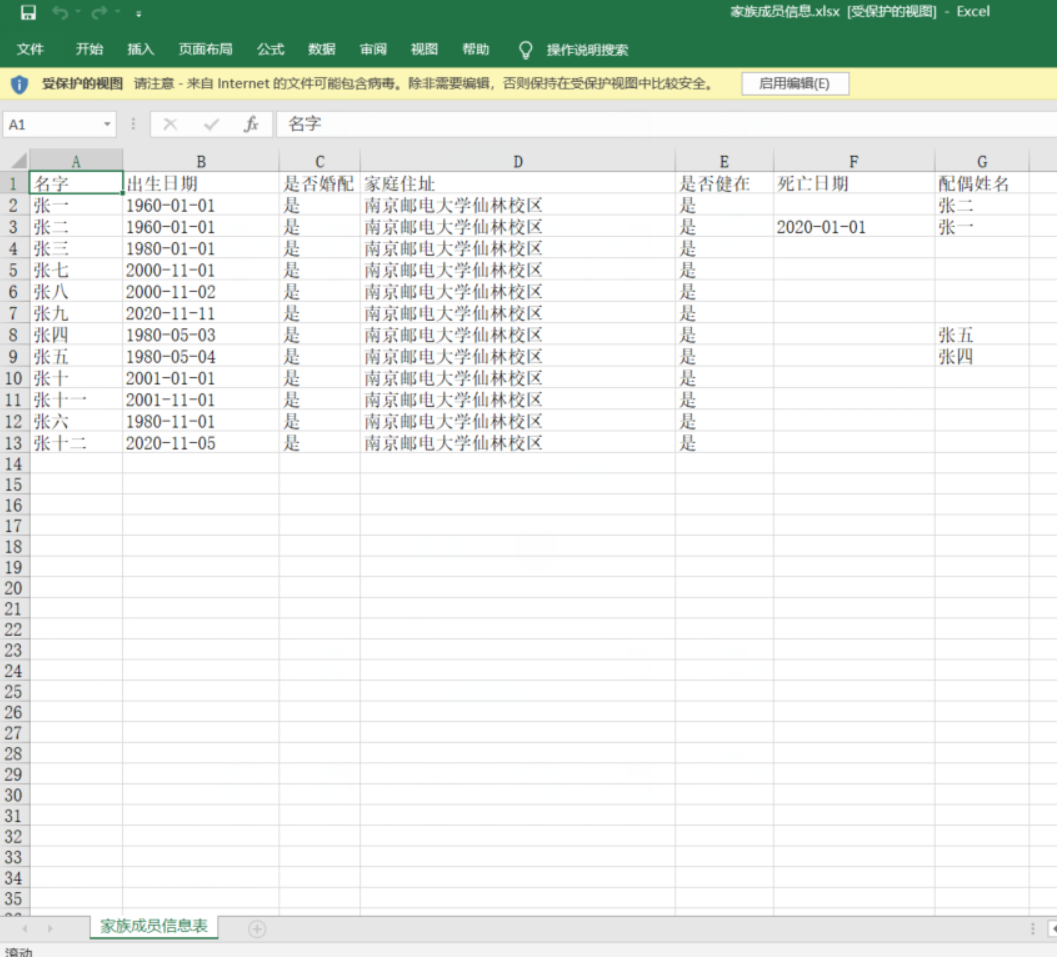
 

图8 导出家族成员信息文件并下载与.xlsx文件内容界面

图9 增加孩子信息前后与提示当天生日的家族成员界面（因为测试时mock数据填的正好为今天）

图10 删除孩子信息前后界面

图11 删除配偶信息前后界面

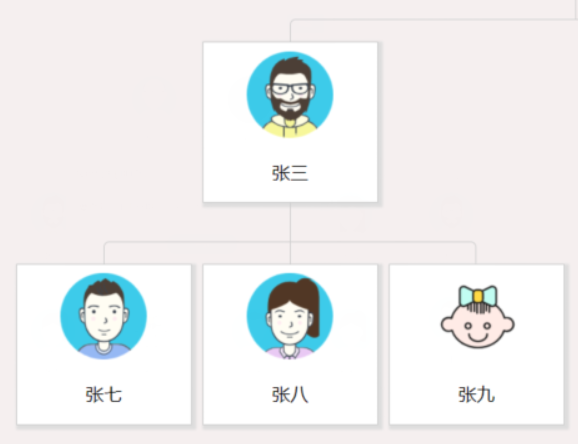
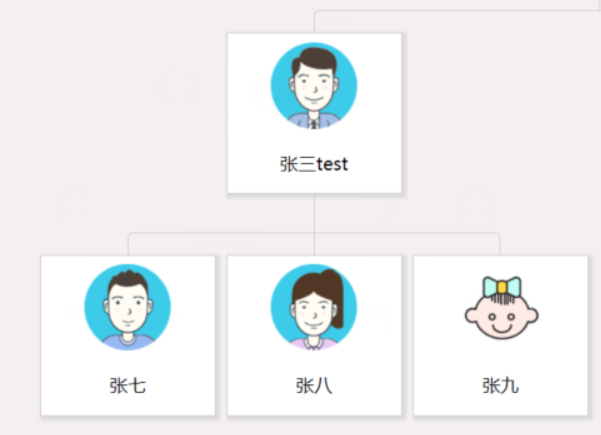
  

图11 修改用户信息前后界面

**六、算法设计和程序调试过程中的问题**

**问题1**：在设置数据结构的时候，我是让用户信息里包含配偶信息的，但是我在后续编写操作用户信息和配偶信息的时候，是用两个reducer来进行管理的，这样就导致一个问题：只有加一个点击事件才可以将配偶信息和用户信息整合，但是那边的页面展示有两个按钮会显得非常突兀，理应只有一个按钮才对。

**解决方法**：改成只用一个reducer来管理数据，dispatch的事件不一样就可以直接对两个信息进行操作了。

**问题2**：在取今天的日期来判断是否有成员生日时，感觉功能函数没写错，但是一直没有实现效果。

**解决方法**：这是因为JavaScript取date转string的时候，日期格式不是标准格式，它的month和day如果小于10的时候不会显示0X，而是只显示X。所以编写函数dateFormat更改日期格式为标准格式。

**问题3**：导出家族信息文件（.xlsx）的时候，没有显示配偶的信息。

**解决方法**：在遍历树图查找的时候，如果用户已经婚配，就将用户配偶信息也添加进来。

**问题4**：导出家族信息文件（.xlsx）的时候，表头显示的是我的变量名，一些状态也直接显示的boolean值，不太符合常理

**解决方法**：进行表头和状态值的重新赋值处理

**七、课程设计总结**

这是一次非常有意思的程序设计周，之前在学数据结构的时候，关于树的实现我只会使用教材使用的语言，这一次因为题目要求用图形化来实现，于是我就想做一个web界面来展示题目要求的家谱图，提升用户体验感，但是这就得使用JavaScript语言来实现对树图的操作。实现过程有一些困难，但好在大多数问题在互联网上都能搜索得到，少数问题也是因为自己的粗心和对题目要求的理解不清楚而造成的。比如这一次写了很多递归，但常常因为自己的粗心，没有把退出递归的条件写好，然后超出范围进而频繁报错。

这一次设计过程中让我更加清楚地意识到自己应该多重视代码规范，多写注释，并且应该提前有一个整体规划，而不是随心所欲，只顾实现功能，而不注重组件化，要学会函数式编程。