**黄燮聪 3117000680 综合信息管理系统**

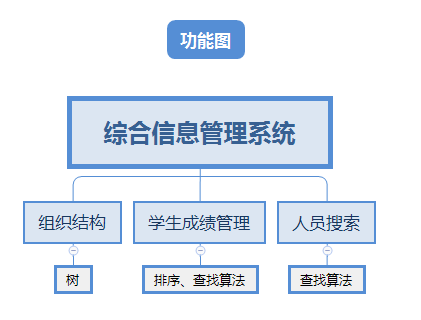
## 选题

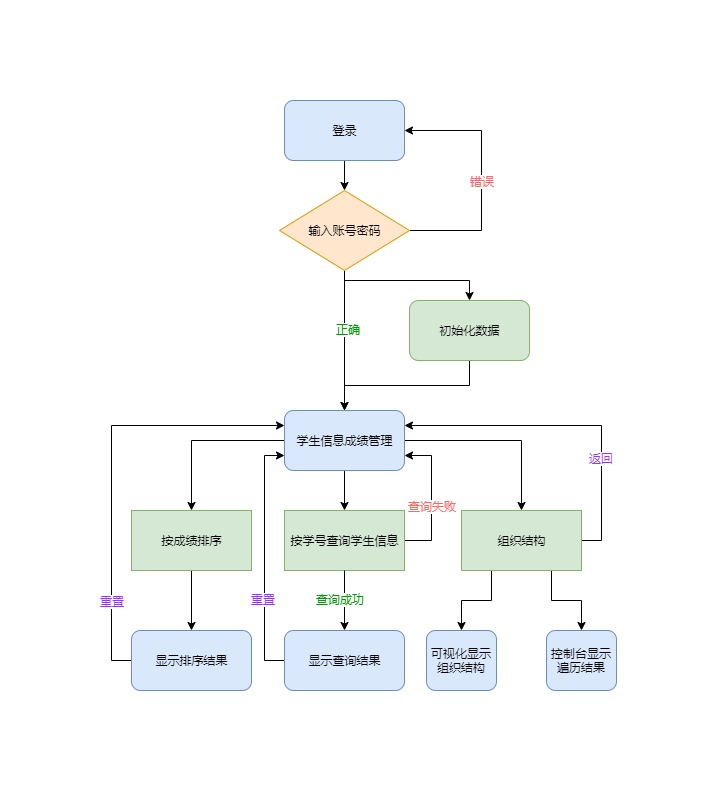
综合信息管理系统

## 问题阐述

当前仍有许多资料是用excel等方式进行管理，我将利用基本数据结构知识实现一个拥有排序、查询功能的综合信息管理系统

## 功能图与流程图





## 关键算法分析

组织结构：

1. **class** treeNode {
2. constructor(data, fatherNode){
3. **this**.data = data;  //树数据
4. **this**.fatherNode = fatherNode;  //父亲节点指针
5. **this**.sonNode = [];
6. //如果父节点不为空（即不是根节点）
7. **if**(**this**.fatherNode != **null**){
8. //在建立节点时，添加进父节点中的子节点数组
9. **this**.fatherNode.sonNode.push(**this**);
10. }
11. }
12. }

利用上述数据结构当作组织管理中所需树的结构，实现建树、遍历等操作

遍历寻找父亲节点与前序遍历寻找子节点

1. //遍历寻找父亲节点
2. preOrder(T){
3. //递归调用结束条件
4. **if**(T == **null**){
5. **return**;
6. }**else** {
7. console.log(T.data);  //访问节点T的数据域
8. **this**.preOrder(T.fatherNode);  //递归往上寻找父亲节点
9. }
10. }
12. //前序遍历寻找子节点
13. pre(T){
14. **if**(T == **null**){
15. **return**;
16. }**else**{
17. console.log(T.data);
18. **for**(let i **in** T.sonNode){
19. **this**.pre(T.sonNode[i]);  //前序遍历寻找子节点
20. }
21. }
22. }

学生信息管理模块：

**查找算法**使用**折半查找算法**，时间复杂度为**O(logn)**

1. binarySearch(a, key){
2. let low, mid, high;
3. low = 0;// 最小下标
4. high = a.length - 1;// 最大小标
5. **while** (low <= high) {
6. mid = Math.round((high + low) / 2);// 折半下标
7. **if** (key > a[mid]) {
8. low = mid + 1; // 关键字比折半值大，则最小下标值调成折半下标的下一位
9. } **else** **if** (key < a[mid]) {
10. high = mid - 1;// 关键字比折半值小，则最大下标值调成折半下标的前一位
11. } **else** {
12. **return** mid; // 当key == a[mid]，表示已经找到，返回下标
13. }
14. }
15. }

**排序算法**使用**快速排序算法**，这是对冒泡排序算法的一种改进，在数据量比较庞大时效果较好。在本实验中，通过利用快速排序算法分别对不同的成绩进行排序，有比较直观的显示效果。时间复杂度为最好为**O(nlog2n)**，最差时为**O(n2)**

采用**交换**法：

1. quickSort(arr, i, j) {
2. **if**(i < j) {
3. let left = i;
4. let right = j;
5. let pivot = arr[left];  //令第一个数为基准
6. **while**(i < j) {
7. **while**(arr[j] >= pivot && i < j) {  // 从后往前找比基准小的数
8. j--;
9. }
10. **if**(i < j) {
11. let temp = arr[i];
12. arr[i] = arr[j];
13. arr[j] = temp;
14. }
15. **while**(arr[i] <= pivot && i < j) {  // 从前往后找比基准大的数
16. i++;
17. }
18. **if**(i < j) {
19. let temp = arr[i];
20. arr[i] = arr[j];
21. arr[j] = temp;
22. }
23. }
24. **this**.quickSort(arr, left, i-1);
25. **this**.quickSort(arr, i+1, right);
26. }
27. }

采用**填坑**法：

1. quickSort(arr, i, j) {
2. **if**(i < j) {
3. let left = i;
4. let right = j;
5. let pivot = arr[left];  //令第一个数为基准
6. **while**(i < j) {
7. **while**(arr[j] >= pivot && i < j) {  // 从后往前找比基准小的数
8. j--;
9. }
10. **if**(i < j) {
11. arr[i++] = arr[j];
12. }
13. **while**(arr[i] <= pivot && i < j) {  // 从前往后找比基准大的数
14. i++;
15. }
16. **if**(i < j) {
17. arr[j--] = arr[i];
18. }
19. }
20. arr[i] = pivot;
21. **this**.quickSort(arr, left, i-1);
22. **this**.quickSort(arr, i+1, right);
23. }
24. }