**《程序设计进阶与实践》实验报告**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 姓名 | 舒英特 | | 学号 | PB21111704 | 日期 | 21.05.14 |
| 实验名称 | 随机分组问题的设计与实现 | | | | | |
| 实验环境：实验运行在ThinkPad X1 Carbon的个人电脑上，作为HTML文档运行在Microsoft Edge上。  CPU：Intel(R) Core(TM) i7-5500U CPU @ 2.40GHz 2.39GHz  RAM：8GB DDR3  OS：64位Windows 10 专业版  IDE：Visual Studio 2019  语言：html/javascript/css | | | | | | |
| 1. **问题分析与求解思路**   随机分组问题要求对一个人群进行随机分组，并且满足一定的条件。具体说来，结对编程的伙伴提出的要求是：  1.设这个班有n个人，将其编号0、1、2……n-1，最后分组结果输出用编号表示。  2.尽可能多分六人组，使得小组的个数最少，这样需要的助教数量也能减少。  3.实现令某两个人一定分在同一组，某两人一定不在同一组，两种加起来的需求数量不超过10个。  4.要求使用图形化界面，使界面更直观易懂，并且也美观。  一种较为理想的图形化界面的实现方式是HTML网页。因此，我们需要从github上寻找一些美观的页面。经查询，<https://github.com/hibbb/randomgroup>给出了一个初步的实现，但是其功能欠缺，没有“某两个人分在同一组”或者“某两个人部分在同一组”的功能，需要增添这些功能。    左侧是结对伙伴提出的要求。结合HTML/CSS语言的特点，我对需求的图形化界面作出一定的更改，但保持易用性不变。  **二、核心代码说明**  本项目的源文件结构如图所示：    其中算法的实现主要在于index.js当中。首先我们需要对读入的数据进行一定的处理。考虑到需要指定某两人分在一组，我们需要使用并查集的数据结构记录哪些人一定分在了一组。  var findroot = function(x) {          return unionSet[x][0] == x ? x : unionSet[x][0] = findroot(unionSet[x][0]);      };      var merge = function(x, y) {          x = findroot(x);          y = findroot(y);          unionSet[x][0] = y;          unionSet[y][1] += unionSet[x][1];          unionSet[x][1] = 0;      };  随后，我们将已有的“小组”随机合并，但合并不能超过小组人数的限制，合并之后也不能违反“两人不能在一队”的规定，直到不能再合并为止。      while(flag) {          flag = false;          for(var i = 0; i < 1000; ++i) {              var x = Math.floor((Math.random()\*names.length)), y = Math.floor((Math.random()\*names.length));              x = findroot(names[x]), y = findroot(names[y]);              if(x == y || unionSet[x][1] + unionSet[y][1] > number) {                  continue;              }              var old = clone(unionSet);              merge(x, y);              if(conflict()) {                  unionSet = clone(old);              } else {                  flag = true;                  break;              }          }      }  但是这样做有两个问题：1、小组的分配不能够做到十分均匀；2、不能保证小组数量的最小化。例如20个人的小组，要求小组人数不超过6。如果在某次合并之后形成4-4-4-4-4的结构，无法继续合并，显然会导致小组的数量增加，不符合要求。小组的人数也有可能形成6-6-6-2的结构，不是十分均匀。  在复杂的限制条件下，难以找到一个准确高效的算法满足“最小”、“均匀”的性质，但是仍然可以找到一些弥补的措施。1、多次重复上述过程，取最优的解。2、尝试在合理的区间内减少小组成员的上限，从而使分配更加均匀。3、采用特殊的方法，对于没有限制或者限制少的输入应用更好的方法。      var try10 = () => {          for(var i = 0; i < 10; i++) {              var new\_result = solve(input);              if(evaluate(new\_result) > evaluate(result)) {                  result = clone(new\_result);              }          }      }      try10();      number = Math.ceil(input["names"].length / Math.ceil(input["names"].length / number));      try10();    //限制每个组的人数，有利于更均匀的分配小组  function attempt\_solve()  function steady\_solve()  var old = clone(unionSet);  if(!attempt\_solve()) {      unionSet = clone(old);      steady\_solve();  }  **三、测试、运行与分析**  该部分由结对伙伴完成。  **四、总结**  有些问题看似简单，实则蕴藏着深刻的算法和逻辑。随机分组问题需要不断地优化算法和界面，才能达到良好的效果。 | | | | | | |
|  | | | | | | |
| 另附（源代码文件名称） | | index.html,index.css,index.js | | | | |

（不超过4页）