重庆大学本科学生实验项目任务书

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 实验题目 | 图算法实践 | | | |
| 实验时间 |  | 实验地点 | |  |
| 实验性质 | □验证性 √设计性 □综合性 | | | |
| 实验目的  1. 掌握图的存储结构与基本操作  2. 训练使用经典的图算法，通过编程解决不同难度问题的实践能力 | | | | |
| 实验内容   1. 图的基本数据结构与基本操作，经典图算法 2. 在线编程解决2道不同难度的编程题，共40分   3． 在线编程时间（答题时长）150分钟，从开始答题时刻起算  4． 在线程序测评平台采用PTA (https://pintia.cn/)  5． 每人必须独立完成编程，可查阅教科书、PPT等资料，不得相互抄袭以及抄袭网上已有的程序  6． 实验课结束后，会对所有程序进行查重，如检测出有抄袭的程序，成绩计零分处理  注意：最后提交完整的实验报告，包括对每道题的算法思路、代码描述、复杂度分析等内容并回答思考题。 | | | | |
| 参考资料：   * Data Structures and Algorithm Analysis (C++ Version) Clifford A. Shaffer   + Introduction to Algorithms, 3rd Edition, MIT Press, T.H. Cormen, et al.   + 《数据结构（ C 语言版）》，严蔚敏，吴伟民编著，清华大学出版社 | | | | |
| 任务下达日期 年 月 日 | | | 完成日期 年 月 日 | |

说明：学院、专业、年级均填全称，如：计算机学院、计算机科学与技术、2021。

**《数据结构与算法》实验报告**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **年级、专业、班级** | |  | | | **姓名** |  |
| **实验题目** | 图算法实践 | | | | | |
| **实验时间** |  | | **实验地点** |  | | |
| **实验成绩** |  | | **实验性质** | **□验证性** √**设计性 □综合性** | | |
| 教师评价：  □算法/实验过程正确； □源程序/实验内容提交 □程序结构/实验步骤合理；  □实验结果正确； □语法、语义正确； □报告规范；  其他：  评价教师签名： | | | | | | |
| 实验目的  1. 掌握图的存储结构与基本操作  2. 训练使用经典的图算法，通过编程解决不同难度问题的实践能力 | | | | | | |
| 二、实验项目内容  注：每道题按下面的格式分别描述  实验课题1：  题目内容：  **7-1 生化危机**  人类正在经历一场生化危机，许多城市已经被病毒侵袭，这些城市中的人们为了避免感染病毒，计划开车逃往其他没有被病毒入侵的城市（安全城市）。有些城市之间有公路直达，有些没有。虽然他们知道哪些城市是安全的，但是不知道有没有一条安全路径能够到达安全城市（只有该路径上经过的所有城市都是安全的，该路径才是安全路径）。请你编写一个程序帮助他们判断。  **输入格式:**  输入第一行为三个正整数，分别表示所有城市个数m（m<=100）、安全城市个数n（m<=50）、公路个数k（k<=100）。随后一行给出n个安全城市的编号。随后k行，每一行给出两个整数，表示连接一条公路的两个城市编号。最后一行输入两个整数，分别表示当前所在城市s、目标城市d。每行整数之间都用空格分隔。  **输出格式:**  若目标城市已被病毒入侵（非安全城市），输出"The city i is not safe!"；若目标城市为安全城市且从当前所在城市能够经过一条安全路径到达目标城市，输出"The city can arrive safely！"；若目标城市为安全城市但是从当前所在城市没有一条安全路径到达目标城市，输出"The city can not arrive safely！"，i为目标城市编号。  解题思路：  如果目标城市未被病毒入侵，则从起点s开始对未被访问且未被病毒入侵的城市进行深度优先搜索。如果在搜索结束前能访问到目标城市，则该城市可以安全抵达，反之则无法安全抵达。  源代码（格式正确，对关键步骤加注释）：  #include<bits/stdc++.h>  using namespace std;  typedef pair<int,int> P;  typedef long long ll;  typedef unsigned long long ull;  const int N=1e5+10,inf=0x3f3f3f3f,mod=1e9+7;  int n,m,k,b,t;  vector<int>edge[100];//储存边  bool s[100],vis[100];//s[i]=true,vis[i]=true分别表示序号为i的城市未被病毒入侵/已经访问过了  bool dfs(int i)//深度优先搜索的返回值表示序号为i的城市与目标城市间是否至少有一条安全路径  {      vis[i]=true;//当前城市变更为已访问状态      if(i==t)//如果当前城市就是目标城市      return true;      for(auto y:edge[i])//对于与当前城市间有公路的所有城市      if(!vis[y]&&s[y]&&dfs(y))//如果该城市未被访问,未被病毒入侵且可以安全抵达目标城市      return true;//则当前城市可以安全抵达目标城市,剪枝      return false;//与当前城市间有公路的所有城市都不能安全抵达目标城市  }  int main()  {  scanf("%d%d%d",&n,&m,&k);  memset(s,0,sizeof(s));//初始化所有城市为已被病毒入侵  memset(vis,0,sizeof(vis));//初始化所有城市为未访问状态  for(int i=1;i<=m;i++)  {      int x;      scanf("%d",&x);      s[x]=true;//记录未被病毒入侵的城市  }  for(int i=1;i<=k;i++)  {      int x,y;      scanf("%d%d",&x,&y);      edge[x].push\_back(y);//记录边      edge[y].push\_back(x);  }  scanf("%d%d",&b,&t);//读入起点和终点  if(s[t])  printf("The city %d %s",t,dfs(b)?"can arrive safely!":"can not arrive safely!");  else printf("The city %d is not safe!",t);  return 0;  }  时间与空间复杂度分析：  时间复杂度：  最坏情况下，深度优先搜索需要访问图中的每一条边，因此时间复杂度为O（m）。  空间复杂度：  记录访问情况的数组复杂度为O（n），记录边的数组复杂度为O（m），总体的空间复杂度为O（m+n）。  实验课题2：  题目内容：  **7-2 大众情人**  人与人之间总有一点距离感。我们假定两个人之间的亲密程度跟他们之间的距离感成反比，并且距离感是单向的。例如小蓝对小红患了单相思，从小蓝的眼中看去，他和小红之间的距离为 1，只差一层窗户纸；但在小红的眼里，她和小蓝之间的距离为 108000，差了十万八千里…… 另外，我们进一步假定，距离感在认识的人之间是可传递的。例如小绿觉得自己跟小蓝之间的距离为 2，则即使小绿并不直接认识小红，我们也默认小绿早晚会认识小红，并且因为跟小蓝很亲近的关系，小绿会觉得自己跟小红之间的距离为 1+2=3。当然这带来一个问题，如果小绿本来也认识小红，或者他通过其他人也能认识小红，但通过不同渠道推导出来的距离感不一样，该怎么算呢？我们在这里做个简单定义，就将小绿对小红的距离感定义为所有推导出来的距离感的最小值。  一个人的异性缘不是由最喜欢他/她的那个异性决定的，而是由对他/她最无感的那个异性决定的。我们记一个人 i 在一个异性 j 眼中的距离感为 Dij​；将 i 的“异性缘”定义为 1/maxj∈S(i)​{Dij​}，其中 S(i) 是相对于 i 的所有异性的集合。那么“大众情人”就是异性缘最好（值最大）的那个人。  本题就请你从给定的一批人与人之间的距离感中分别找出两个性别中的“大众情人”。  **输入格式：**  输入在第一行中给出一个正整数 N（≤500），为总人数。于是我们默认所有人从 1 到 N 编号。  随后 N 行，第 i 行描述了编号为 i 的人与其他人的关系，格式为：  性别 K 朋友1:距离1 朋友2:距离2 …… 朋友K:距离K  其中 性别 是这个人的性别，F 表示女性，M 表示男性；K（<N 的非负整数）为这个人直接认识的朋友数；随后给出的是这 K 个朋友的编号、以及这个人对该朋友的距离感。距离感是不超过 106 的正整数。  题目保证给出的关系中一定两种性别的人都有，不会出现重复给出的关系，并且每个人的朋友中都不包含自己。  **输出格式：**  第一行给出自身为女性的“大众情人”的编号，第二行给出自身为男性的“大众情人”的编号。如果存在并列，则按编号递增的顺序输出所有。数字间以一个空格分隔，行首尾不得有多余空格。  解题思路（含递推方程）：  由“将小绿对小红的距离感定义为所有推导出来的距离感的最小值”可知，求“任意两个人之间的距离感”可以转化为多源最短路径问题，因此采用Floyd算法，定义f[i][j][k]为途径序号为1到k的点中的某些点，节点i到节点j的距离的最小值，递推方程为：  **f[k][i][j] = min(f[k-1][i][j],f[k-1][i][k]+f[k-1][k][j]);**  其中f[k-1][i][j]表示不经过序号为k的点，f[k-1][i][k]+f[k-1][k][j]表示途经序号为k的点。求出任意两人间的距离感后，对于某一男性，取所有女性对于他的距离感的最大值；对于所有男性，取其对应最大值的最小值，即为异性缘最好的人。女性同理。  源代码（格式正确，对关键步骤加注释）：  #include<bits/stdc++.h>  using namespace std;  typedef pair<int,int> P;  typedef long long ll;  typedef unsigned long long ull;  const int N=1e5+10,inf=0x3f3f3f3f,mod=1e9+7;  int n,f[501][501],d[501],m1=inf,m2=inf,cnt1=0,cnt2=0;  bool man[501];//man[i]=true,序号为i的人是男性，否则是女性  int main()  {  scanf("%d",&n);  memset(f,0x3f,sizeof(f));//初始化距离感为无穷大  for(int i=1;i<=n;i++)  f[i][i]=0;//和自己的距离感是0  for(int i=1;i<=n;i++)  {      char chr;      int k;      cin>>chr>>k;      man[i]=(chr=='M')?true:false;      for(int j=1;j<=k;j++)      {          int x,y;          scanf("%d:%d",&x,&y);          f[i][x]=y;//序号为i的人眼中与序号为k的人的的距离感      }  }  for(int k=1;k<=n;k++)  for(int i=1;i<=n;i++)  for(int j=1;j<=n;j++)  f[i][j]=min(f[i][j],f[i][k]+f[k][j]);//对于多源最短路径采用Floyd算法计算任意两个人间的距离感  for(int i=1;i<=n;i++)  {      if(man[i])//如果是男性      {          for(int j=1;j<=n;j++)          if(!man[j])          d[i]=max(d[i],f[j][i]);//记录所有女性对序号为i的男性的距离感的最大值          m1=min(m1,d[i]);//所有女性对他距离感最大值最小的那个人就是异性缘最好的人,保留这个最大值      }      else//女性同理      {          for(int j=1;j<=n;j++)          if(man[j])          d[i]=max(d[i],f[j][i]);          m2=min(m2,d[i]);      }  }  for(int i=1;i<=n;i++)//按升序输出所有符合条件的女性的序号  if(!man[i]&&d[i]==m2)  {      if(cnt1)      putchar(' ');      cnt1++;      printf("%d",i);  }  putchar('\n');  for(int i=1;i<=n;i++)//男性同理  if(man[i]&&d[i]==m1)  {      if(cnt2)      putchar(' ');      cnt2++;      printf("%d",i);  }  return 0;  }  时间与空间复杂度分析：  时间复杂度：  Floyd算法的时间复杂度为O（n^3），程序的其他部分时间复杂度不超过O（n^2），因此总的时间复杂度为O（n^3）。  空间复杂度：  空间压缩后的Floyd算法复杂度为O（n^2），程序的其他部分空间复杂度不超过O（n），因此总的时间复杂度为O（n^2）。 | | | | | | |
| 三、思考题  对于无边权重的无向图，可以用广度优先遍历(BFS)算法求从起点到其它所有结点的最短路径，并且使用的FIFO队列的长度为O(|V|),即结点的数量。而有权重的图，使用Dijsktra算法求最短路径，通常需要使用长度为O(|E|)的优先队列。造成这一区别的主要原因是什么？如果要让Dijsktra算法也只使用O(|V|)长度的优先队列，该如何处理？  答：  设d[i]为序号为i的点距离起点s的距离。对于无边权重的无向图，从起点开始进行广度优先遍历，对于先加入队列的点x和后加入队列的点y，一定有d[x]<=d[y]。因此，对于已经加入队列的节点，其最短路径不再被刷新，即**FIFO队列维护的是最短路径已经确定的节点**；同时，在对与当前出队列节点有边相连的节点进行遍历的过程中，**不需要重复加入**已经加入队列的节点，即**每个节点入且仅入队一次**，因此FIFO队列的长度为O(|V|)。  而对于Dijkstra算法中所使用的**优先队列**（最小堆），其堆顶**维护的是当前距离起点s距离最短的节点及其距离**。当使用堆顶弹出的节点x尝试刷新仍在堆中的节点（**其最短路径未确定**）的距离时，需要同时调整其位置以满足最小堆的要求。但是，一般而言，堆（如STL中提供的priority\_queue容器）不提供查找、修改或删除特定节点的功能。因此，在实际操作中，一般选择将节点及其刷新后的距离**重新加入堆中而不删除原有的节点**。由最小堆的性质，在堆中重复出现的节点（序号相同，距离大小不同，以结构体形式储存，根据距离大小确定其在堆中的位置），先弹出的一定是距离小的。通过添加visited数组记录每次弹出的节点，如果之前已经弹出过同序号的节点，则放弃使用该节点刷新其他节点的距离，可以保证算法的正确性。在最坏情况下，**每次访问某一条边，都可能进行一次距离刷新，向堆中加入一个新的结构体**。因此，优先队列的长度为O(|E|)。  显然，如果想要将优先队列的长度压缩至O(|V|)，一种可行的办法是为优先队列实现查找功能，避免在优先队列中重复加入节点序号相同的结构体。设图中一共有n个节点，m条边。在优先队列中进行查找的时间复杂度为O（n）。因此，算法整体的时间复杂度从O（(n+m)logn）增加至O（nlogn+mn）。而不使用优先队列优化的算法时间复杂度为O（n²+m），一般而言，后者的效率反而更好。因此在代码实现的过程中，如果要同时保障算法时间复杂度为O（(n+m)logn），空间复杂度为O（|V|）这两个前提，结合Dijkstra算法中动态维护最小值和查找特定值并进行删改的需要，相比一般的堆结构，二叉搜索树及其优化（如STL中的set容器）可能是更好的选择。 | | | | | | |