**山东大学计算机科学与技术学院  
《数据结构与算法》课程设计报告**

|  |  |  |  |
| --- | --- | --- | --- |
| 学号：201705130120 | 姓名： 苑宗鹤 | | 班级： 17.1 |
| 上机学时：4 | | 日期： 2019.4.28 | |
| 课程设计题目：关键活动 | | | |
| 软件开发环境：win10 x64 vs2019 | | | |
| 报告内容：  1.需求描述  1.1 问题描述  残缺棋盘(defective chessboard)：是一个有 2k×2k个方格的棋盘，其中恰有一个方格残缺。对于任 意k，恰好存在22k种不同的残缺棋盘。 在残缺棋盘中，要求用三格板(triominoes)覆盖 残缺棋盘。在覆盖中，任意两个三格板不能重叠，任 意一个三格板不能覆盖残缺方格，但三格板必须覆盖 其他所有方格。  1.2 基本要求  输入棋盘大小和残缺方格的位置，输出覆盖后的棋 盘，输出棋盘时要着色，共享同一边界的覆盖应着不 同的颜色。棋盘是平面图，因此最多只需4种颜色，为 覆盖着色，要求设计贪婪着色启发式方法，以尽量使 用较少的颜色  1.3 输入说明  输入k,x,y  其中k为棋盘的2^k次幂大小 x,y为残缺坐标  1.4 输出说明  输出棋盘的涂色方案  2.设计  2.1 系统结构设计    2.2 设计思路  对一个有向无环图(Directed Acyclic Graph简称DAG)G进行拓扑排序，是将G中所有顶点排成一个线性序列，使得图中任意一对顶点u和v，若边(u,v)∈E(G)，则u在线性序列中出现在v之前。通常，这样的线性序列称为满足拓扑次序  对于DAG 只要按照拓扑序维护最短/最长路 即可在O(N)时间内算出结果  2.3 数据及数据类(型)定义  class goalPath {  int n, m; //任务交界点 任务数量  int indegree[105];//入度  int outdegree[105];//初读  int earlyFinish[105];//最早完成时间  int lastFinish[105];//最晚完成时间  vector<Point>G[105];//图G  vector<Point>G2[105];//反图  vector<Point>Edge;//边集  vector<Point>ans;//结果数组  queue<Point>que;//拓扑队列  };  2.4.算法设计及分析  在现代化管理中，人们常用有向图来描述和分析一项工程的计划和实施过程，一个工程常被分为多个小的子工程，这些子工程被称为活动（Activity)，在带权有向图中若以顶点表示事件，有向边表示活动，边上的权值表示该活动持续的时间，这样的图简称为AOE网。  拓扑排序是一种对非线性结构的有向图进行线性化的重要手段。在给定的有向图G中，若顶点序列Vi1，Vi2，Vi3，....,Vin，。满足下列条件：若在有向图G中从顶点Vi，到顶点Vj有一条路径，则在序列中顶点Vi必在顶点Vj之前，便称这个序列为一个拓扑序列。求一个有向图拓扑序列的过程称为拓扑排序。 　　拓扑排序的方法如下： 　　(1)从图中选择一个人度为O的顶点并输出； 　　(2)从图中删掉该顶点及其所有以该顶点为弧尾的弧。 　　反复执行这两个步骤，直到所有的顶点都被输出，输出的序列就是这个无环有向图的拓扑序列。  如果在带权有向图中，用顶点表示事件，用有向边表示活动，边上的权值表示活动的开销，则此带权有向图称为边活动网，简称AOE网。AOE网是一个有向无环图。AOE网是用来描述由许多交叉活动组成的复杂计划和工程的方法，比如某工程的AOE网。  在工程中用边表示活动，边上的权表示完成这项活动所需要的时间，顶点表示某项活动的开始，顶点1称为源点（或起点），表示整个工程开始，顶点2称为汇点（或终点），表示整个工程的结束。用AOE网来估算工程的最短工期（完成整个工程至少需要多少时间）以及哪些活动是影响工程进展的关键  3. 测试结果  输入:  7 8  1 2 4  1 3 3  2 4 5  3 4 3  4 5 1  4 6 6  5 7 5  6 7 2  输出    4. 分析与探讨  关键活动可以转化为求最长路问题 在最长路上的边均为关键活动  通过在入度为0 出度为0 的节点建立源点汇点 可以优化掉 枚举最长路起点的复杂度  5. 附录：实现源代码  **main.cpp]**  #include<iostream>  #include<vector>  #include<algorithm>  #include<queue>  #include"goalPath.h"  int main() {  int m, n;  cin >> n >> m;  goalPath gp(n, m);  gp.run();  }  **ExternSort.h**  #pragma once  #include<iostream>  #include<vector>  #include<algorithm>  #include<queue>  using namespace std;  struct Point {  int from, to, value, edgeid;  bool operator < (const Point& b)const {  if (from != b.from)return from < b.from;  else return edgeid > b.edgeid;  }  };  class goalPath {  public:  goalPath(int n\_in, int m\_in) {  n = n\_in, m = m\_in;  int x, y, z;  for (int a = 1; a <= m; ++a) {  cin >> x >> y >> z;  G[x].push\_back(Point{ x,y,z,a });  G2[y].push\_back(Point{ y,x,z,a });  Edge.push\_back(Point{ x,y,z,a });  indegree[y]++;  outdegree[x]++;  }  }  bool run() {  //添加源点汇点  for (int a = 1; a <= n; ++a) {  if (indegree[a] == 0) {  G[0].push\_back(Point{ 0,a,0,0 });  G2[a].push\_back(Point{ a,0,0,0 });  }  if (outdegree[a] == 0) {  G[a].push\_back(Point{ a,n + 1,0,0 });  G2[n + 1].push\_back(Point{ n + 1,a,0,0 });  }  }  int num = 0;  fill\_n(earlyFinish, n + 2, 0);  que.push(Point{ 0,0,0,0 });  while (!que.empty()) {  Point x = que.front();  que.pop();  num++;  for (auto& z : G[x.to]) {  earlyFinish[z.to] = max(earlyFinish[z.to], earlyFinish[x.to] + z.value);  if (--indegree[z.to] <= 0) {  que.push(z);  }  }  }  if (num < n + 2) {  cout << 0 << endl;  return false;  }  que.push(Point{ 0,n + 1,0,0 });  fill\_n(lastFinish, n + 2, 0x3f3f3f3f);  lastFinish[n + 1] = earlyFinish[n + 1];  while (!que.empty()) {  Point x = que.front();  que.pop();  for (auto& z : G2[x.to]) {  lastFinish[z.to] = min(lastFinish[z.to], lastFinish[x.to] - z.value);  if (--outdegree[z.to] <= 0) {  que.push(z);  }  }  }  for (auto x : Edge) {  if (lastFinish[x.to] - (earlyFinish[x.from] + x.value) == 0) {  ans.push\_back(x);  }  }  sort(ans.begin(), ans.end());  cout << lastFinish[n + 1] << endl;  for (auto x : ans) {  cout << x.from << "->" << x.to << endl;  }  cout << "最早完成" << endl;  for (int a = 1; a <= n; ++a) {  cout << earlyFinish[a]<<" ";  }  cout << endl << "最晚完成" << endl;  for (int a = 1; a <= n; ++a) {  cout << lastFinish[a] << " ";  }  return true;  }  private:  int n, m;  int indegree[105];  int outdegree[105];  int earlyFinish[105];  int lastFinish[105];  vector<Point>G[105];  vector<Point>G2[105];  vector<Point>Edge;  vector<Point>ans;  queue<Point>que;  }; | | | |