**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：201705130113 | 姓名： 黄瑞哲 | | 班级：计科17.3 |
| 上机学时：4 | | 日期： 2019.4.17 | |
| 课程设计题目：基本数据结构及其应用之关键活动 | | | |
| 软件开发环境：  Microsoft Visual Studio 2019 | | | |
| 报告内容：  1.需求描述  1.1 问题描述  一个工程项目一般有一些子任务（活动）构成，子任务之间有的可以并行执行，有的则必须在完成了其它一些任务后才能执行，如果给出了完成每个任务需要的时间，则可以算出完成整个工程需要的最短时间。在这些子任务中，有些任务即使推迟几天，也不会影响全局的工期，但是有些任务必须 准时完成，否则整个工程项目就要因此延误，这种任务就叫“关键活动”。  1.2 基本要求  判断一个工程项目的任务调度是否可行，如果可行，则计算出完成整个工程需要的最短时间，输出所有的关键活动，各项活动的最早开始时间，不影响全局工期的最晚开始时间。  1.3 输入说明  第一行两个整数n，m分别表示任务个数和关系条数。  接下来一行有n个数表示每个任务需要的时间。  最后有m行，每行两个数u，v表示任务u要在任务v之前完成。  1.4 输出说明  输出关键活动编号和每个活动最早的开始时间和最晚开始时间。  2.设计  2.1 系统结构设计  使用标准输入输出。  使用图论算法解决问题。  2.2 设计思路  如果任务u需要在任务v之前完成，则建立一条从u到v的有向边。整个合法的任务调度形成一个有向无环图，如果图中存在环路则说明任务调度不合法。对这个DAG正向动态规划求最长路可得任务最早的开始时间，再反向动态规划可得最晚开始时间，如果一个节点的最早开始时间和最晚开始时间相等则说明该任务是关键任务。如果存在多个连通块则建立一个超级源点和超级汇点，超级源点向所有入度为0的点建一条有向边，所有出度为0的点向超级汇点建一条有向边后此图变成了一个连通图，按照上述算法即可求解。  2.3 数据及数据类(型)定义  使用邻接数组存储图的结构，定义start[u]表示到点u时的最早时间，end[u]表示到点u时的最晚时间。  2.4.算法设计及分析  （各模块算法及类内函数的算法伪码表示）    建立完整连通图    正向拓扑动态规划求解最早开始时间    反向拓扑动态规划求解最晚开始时间。  3. 测试结果    4. 分析与探讨  测试数据的任务调度图如下图所示，红色为关键路径。  5. 附录：实现源代码  #include <iostream>  #include <algorithm>  #include "linearList/arrayList.h"  #include "queue/linkedQueue.h"  template<typename T = int>  class Arranger {  public:  Arranger(int, const T\*, const arrayList<int>\*);  ~Arranger();  bool solve(arrayList<int>&, T\*, T\*) const;  private:  int n;  T\* t;  arrayList<int>\* e;  void dfs(int);  };  template<typename T>  Arranger<T>::Arranger(int n, const T\* p, const arrayList<int>\* mp) : n(n) {  t = new T[n + 2];  for (int i = 1; i <= n; ++i) t[i] = p[i];  e = new arrayList<int>[n + 2];  for (int i = 1; i <= n; ++i) e[i] = mp[i];  }  template<typename T>  Arranger<T>::~Arranger() {  delete[] t;  delete[] e;  }  template<typename T>  bool Arranger<T>::solve(arrayList<int> & keyAction, T \* start, T \* end) const {  int\* in = new int[n + 2];  t[0] = t[n + 1] = 0;  for (int i = 0; i <= n + 1; ++i) in[i] = start[i] = 0;  for (int i = 1; i <= n; ++i)  for (int j = 0; j < e[i].size(); ++j)  in[e[i][j]]++;  linkedQueue<int> q;  for (int i = 1; i <= n; ++i) {  if (!in[i]) {  e[0].push\_back(i);  in[i]++;  }  if (e[i].size() == 0) {  e[i].push\_back(n + 1);  in[n + 1]++;  }  }  q.push(0);  int cnt = 0;  while (!q.empty()) {  int u = q.front();  q.pop();  cnt++;  if (u == n + 1) break;  for (int i = 0; i < e[u].size(); ++i) {  int v = e[u][i];  start[v] = max(start[v], start[u] + t[u]);  if (--in[v] == 0) q.push(v);  }  }  cout << cnt << endl;  if (cnt != n + 2) return false;  arrayList<int> \* ee = new arrayList<int>[n + 2];  for (int i = 0; i <= n + 1; ++i)  for (int j = 0; j < e[i].size(); ++j)  ee[e[i][j]].push\_back(i);    for (int i = 0; i <= n + 1; ++i) in[i] = 0;  for (int i = 0; i <= n + 1; ++i) end[i] = 0x3f3f3f3f;  for (int i = 0; i <= n + 1; ++i) {  for (int j = 0; j < ee[i].size(); ++j)  in[ee[i][j]]++;  }  cnt = 0;  q.push(n + 1);  end[n + 1] = start[n + 1];  while (!q.empty()) {  int u = q.front();  q.pop();  cnt++;  if (u == 0) break;  for (int i = 0; i < ee[u].size(); ++i) {  int v = ee[u][i];  end[v] = min(end[v], end[u] - t[v]);  if (--in[v] == 0) q.push(v);  }  }  for (int i = 1; i <= n; ++i) if (start[i] == end[i]) keyAction.push\_back(i);  delete[] in;  delete[] ee;  return true;  }  int main() {  int n, m;  cin >> n >> m;  int\* t = new int[n + 2];  int\* start = new int[n + 2];  int\* end = new int[n + 2];  arrayList<int> key;  arrayList<int>\* e = new arrayList<int>[n + 2];  for (int i = 1; i <= n; ++i) cin >> t[i];  for (int i = 1; i <= m; ++i) {  int u, v;  cin >> u >> v;  e[u].push\_back(v);  }  Arranger<int> it(n, t, e);  if (!it.solve(key, start, end)) {  cout << "illegal arragement" << endl;  }  else {  cout << "key action: " << endl;  for (int i : key) {  cout << i << ' ';  }  cout << endl << "earliest and latest: " << endl;  for (int i = 1; i <= n; ++i) {  cout << start[i] << ' ' << end[i] << endl;  }  }  delete[] t;  delete[] start;  delete[] end;  delete[] e;  return 0;  }  /\*  6 7  1 2 3 4 5 6  4 3  4 6  3 6  3 2  6 1  6 5  1 2  7 8  4 5 3 6 3 2 7  1 2  1 3  2 4  3 4  4 5  4 6  5 7  6 7  \*/ | | | |