山东大学 计算机科学与技术 学院

数据结构与算法 课程实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| 学号：201705130113 | 姓名：黄瑞哲 | | 班级：计科17.3 |
| 实验题目：贪婪算法 | | | |
| 实验学时：2 | | 实验日期：2018.12.20 | |
| 实验目的：   1. 掌握最小生成树的Prim算法和Kruskal算法及其实现 | | | |
| 软件环境：  Visual Studio Community 2017 | | | |
| 1. 实验内容（题目内容，输入要求，输出要求） 2. 创建加权无向图类。设图没有重边和自环，存储结构分别使用邻接矩阵和邻接链表。提供必要的基本操作。 3. 键盘输入图中顶点的个数n和边的数目e，以三元组（i，j，w）形式依次输入图的每一条边或随机生成含e条边的图，其中（i，j，w）表示顶点i和顶点j之间拥有权值为w的边，建立图。 4. 对建立好的图，分别使用Prim算法和Kruskal算法求最小生成树，输出求得的最小生成树。 5. 数据结构与算法描述 （整体思路描述，所需要的数据结构与算法） 6. Prim算法：任意以一个点为开始，把这个初始点加入集合A中，从集合B中减去这个点。寻找与它相邻的点中路径最短的点，如后把这个点也加入集合A中,从集合B中减去这个点。更新未被访问的节点的dis[]值。重复上述过程。一直到所有的点都在A集合中结束。 7. Kruskal算法：将所有边储存下来，记录每条边关联的两个点的编号以及边权，按照边权从小到大排序。然后顺序遍历每条边，如果加入这条边不会产生环则将其加入生成树中，直至生成树中有n-1条边为止。 8. 测试结果（测试输入，测试输出，结果分析）      1. 分析与探讨（结果分析，若存在问题，探讨解决问题的途径）   Prim算法与Dijkstra算法极为相似，Dijkstra算法dis记录的是到源点的最短路，Prim中的dis记录的是到生成树的最短路。   1. 附录：实现源代码（本实验的全部源程序代码，程序风格清晰易理解，有充分的注释）   /\*main.cpp\*/  #include "pch.h"  #include <iostream>  using namespace std;  int main() {  int n, m;  cin >> n >> m;  adjacencyGraph<int> g(n);  for (auto i = 0; i < m; ++i) {  int u, v, w;  cin >> u >> v >> w;  g.add(u, v, w);  g.add(v, u, w);  }  cout << "Kruskal" << endl;  cout << g.kru() << endl;  cout << "Prim" << endl;  cout << g.prim(1) << endl;  return 0;  }  /\*  5 6  1 3 3  2 3 1  4 1 5  4 5 2  5 1 2  5 2 1  \*/    /\*pch.h\*/  #ifndef PCH\_H  #define PCH\_H  // TODO: 添加要在此处预编译的标头  #include <graph/adjacencyGrap.h>  #include <graph/linkedGraph.h>  #endif //PCH\_H  /\*adjacencyGrap.h\*/  #ifndef adjacencyGraph\_  #define adjacencyGraph\_  #include <cstring>  #include <queue/linkedQueue.h>  #include <iostream>  #include "ufset.h"  using namespace std;  template<typename T>  class adjacencyGraph {  public:  explicit adjacencyGraph(const int n, T MAX = 0x3fffffff) :n(n), MAX(MAX) {  e = new T\*[n + 1];  for (auto i = 0; i <= n; ++i) e[i] = new T[n + 1];  for (auto i = 0; i <= n; ++i)  for (auto j = 0; j <= n; ++j) {  if (i == j) e[i][j] = 0;  else e[i][j] = MAX;  }  g = new T\*[n + 1];  for (auto i = 0; i <= n; ++i) g[i] = new T[n + 1];  vis = new bool[n + 1];  for (auto i = 0; i <= n; ++i) vis[i] = false;  changed = true;  m = 0;  }  ~adjacencyGraph() {  for (auto i = 0; i <= n; ++i) delete[] e[i];  for (auto i = 0; i <= n; ++i) delete[] g[i];  delete[] e;  delete[] g;  delete[] vis;  };  void add(int u, int v, const T w) {  changed = true;  e[u][v] = w;  ++m;  }  void erase(int u, int v) {  changed = true;  e[u][v] = MAX;  --m;  }  void bfs(int s) {  linkedQueue<int> q;  vis[s] = true;  q.push(s);  while (!q.empty()) {  auto u = q.front(); q.pop();  cout << u << ' ';  for (auto i = 1; i <= n; ++i)  if (!vis[i] && e[u][i] < MAX) {  q.push(i);  vis[i] = true;  }  }  cout << endl;  }  void dfs(int u) {  cout << u << ' ';  vis[u] = true;  for (auto i = 1; i <= n; ++i)  if (!vis[i] && e[u][i] < MAX)  dfs(i);  }  int connectedComponent(bool method = false) {  for (auto i = 1; i <= n; ++i) vis[i] = false;  auto tot = 0;  for (auto i = 1; i <= n; ++i) {  if (vis[i]) continue;  ++tot;  if (method) {  this->dfs(i);  cout << endl;  }  else this->bfs(i);  }  return tot;  }  int dis(int u, int v) {  if (changed) floyd();  return g[u][v] >= MAX ? -1 : g[u][v];  }  int kru() const {  struct edge {  int u, v;  T w;  bool operator<(const edge& x) const { return w < x.w; }  }\*ee;  int cnt = 0;  ee = new edge[m];  for (int i = 1; i <= n; ++i)  for (int j = 1; j <= n; ++j)  if (i != j && e[i][j] < MAX)  ee[cnt++] = { i, j, e[i][j] };  std::sort(ee, ee + cnt);  ufset f(n);  int tot = 0;  int sum = 0;  for (int i = 0; i < cnt; ++i) {  int u = ee[i].u;  int v = ee[i].v;  if (f.unite(u, v)) {  sum += ee[i].w;  cout << u << "<->" << v << " " << ee[i].w << endl;  if (++tot >= n) break;  }  }  delete[] ee;  return sum;  }  int prim(int s) const {  T\* dis = new T[n + 1];  int\* path = new int[n + 1];  for (auto i = 1; i <= n; ++i) dis[i] = (T)0x6f6f6f6f;  for (auto i = 1; i <= n; ++i) vis[i] = false;  for (auto i = 1; i <= n; ++i) {  if (e[s][i] < MAX && i != s) {  dis[i] = e[s][i];  path[i] = s;  }  }  vis[s] = true;  int sum = 0;  for (int i = 1; i < n; ++i) {  int MIN = MAX;  int u = -1;  for (int j = 1; j <= n; ++j) {  if (!vis[j] && dis[j] < MIN) {  MIN = dis[j];  u = j;  }  }  if (u == -1) return -1;  vis[u] = true;  cout << path[u] << "<->" << u << " " << dis[u] << endl;  sum += dis[u];  for (auto v = 1; v <= n; ++v) {  if (!vis[v] && dis[v] > e[u][v]) {  dis[v] = e[u][v];  path[v] = u;  }  }  }  delete[] dis;  delete[] path;  return sum;  }  protected:  T\*\* e;  T\*\* g;  bool\* vis;  bool changed;  int n;  int m;  T MAX;  void floyd() {  for (auto i = 1; i <= n; ++i)  for (auto j = 1; j <= n; ++j)  g[i][j] = e[i][j];  for (auto k = 1; k <= n; ++k)  for (auto i = 1; i <= n; ++i)  for (auto j = 1; j <= n; ++j)  g[i][j] = (g[i][j] < g[i][k] + g[k][j] ? g[i][j] : g[i][k] + g[k][j]);  changed = false;  }  };  #endif //adjacencyGrap\_  /\*linkedGraph.h\*/#ifndef linkedGraph\_  #define linkedGraph\_  #include <linearList/chain.h>  #include <queue/linkedQueue.h>  #include <tree/Heap.h>  #include <algorithm>  #include <iostream>  #include "ufset.h"  using namespace std;  template<typename T>  class linkedGraph {  typedef pair<int, T> p;  protected:  chain<p>\* e;  bool\* vis;  int n;  int m;  public:  explicit linkedGraph(const int n) :n(n) {  e = new chain<p>[n + 1];  vis = new bool[n + 1];  m = 0;  }  ~linkedGraph() {  delete[] e;  delete[] vis;  }  void add(int u, int v, const T& w) {  e[u].push\_back(make\_pair(v, w));  ++m;  }  void erase(int u, int v) {  auto i = 0;  for (auto it = e[u].begin(); it != e[u].end() && it->first != v; ++it, ++i);  e[u].erase(i);  --m;  }  void bfs(int s) {  linkedQueue<int> q;  vis[s] = true;  q.push(s);  while (!q.empty()) {  auto u = q.front(); q.pop();  cout << u << ' ';  for (auto it = e[u].begin(); it != e[u].end(); ++it) {  auto v = it->first;  if (!vis[v]) {  q.push(v);  vis[v] = true;  }  }  }  cout << endl;  }  void dfs(int u) {  cout << u << ' ';  vis[u] = true;  for (auto it = e[u].begin(); it != e[u].end(); ++it) {  auto v = it->first;  if (!vis[v])  dfs(v);  };  }  int connectedComponent(bool method = false) {  for (auto i = 1; i <= n; ++i) vis[i] = false;  auto tot = 0;  for (auto i = 1; i <= n; ++i) {  if (vis[i]) continue;  ++tot;  if (method) {  this->dfs(i);  cout << endl;  }  else this->bfs(i);  }  return tot;  }  int dis(int s, int t) const {  T\* dis = new T[n + 1];  for (auto i = 1; i <= n; ++i) dis[i] = 0x6f6f6f6f;  struct dh {  int u;  T d;  bool operator<(const dh& x) const { return d < x.d; }  };  Heap<dh, less<>> q;  dis[s] = 0;  q.push({ s, 0 });  while (!q.empty()) {  dh uu = q.top(); q.pop();  int u = uu.u;  if (dis[u] < uu.d) continue;  for (auto it = e[u].begin(); it != e[u].end(); ++it) {  int v = it->first;  T& w = it->second;  if (dis[v] > dis[u] + w) {  dis[v] = dis[u] + w;  q.push({ v, dis[v] });  }  }  }  auto ans = dis[t];  delete[] dis;  return ans;  }  int kru() const {  struct edge {  int u, v;  T w;  bool operator<(const edge& x) const { return w < x.w; }  }\*ee;  int cnt = 0;  ee = new edge[m];  for (int i = 1; i <= n; ++i)  for (auto it = e[i].begin(); it != e[i].end(); ++it)  ee[cnt++] = { i, it->first, it->second };  std::sort(ee, ee + cnt);  ufset f(n);  int tot = 0;  int sum = 0;  for (int i = 0; i < cnt; ++i) {  int u = ee[i].u;  int v = ee[i].v;  if (f.unite(u, v)) {  sum += ee[i].w;  cout << u << "<->" << v << " " << ee[i].w << endl;  if (++tot >= n) break;  }  }  delete[] ee;  return tot == n - 1 ? sum : -1;  }  int prim(int s) const {  T\* dis = new T[n + 1];  int\* path = new int[n + 1];  for (auto i = 1; i <= n; ++i) dis[i] = (T)0x6f6f6f6f;  for (auto i = 1; i <= n; ++i) vis[i] = false;  for (auto it = e[s].begin(); it != e[s].end(); ++it) {  int v = it->first;  T w = it->second;  dis[v] = w;  path[v] = s;  }  vis[s] = true;  int sum = 0;  for (int i = 1; i < n; ++i) {  int MIN = 0x6f6f6f6f;  int u = -1;  for (int j = 1; j <= n; ++j) {  if (!vis[j] && dis[j] < MIN) {  MIN = dis[j];  u = j;  }  }  if (u == -1) return -1;  vis[u] = true;  cout << path[u] << "<->" << u << " " << dis[u] << endl;  sum += dis[u];  for (auto it = e[u].begin(); it != e[u].end(); ++it) {  int v = it->first;  T w = it->second;  if (!vis[v] && dis[v] > w) {  dis[v] = w;  path[v] = u;  }  }  }  delete[] dis;  delete[] path;  return sum;  }  };  #endif //linkedGraph\_ | | | |