《离散数学》课程实验报告

4 最小生成树

实验用例

如下图所示的赋权图表示某七个城市，预先计算出它们之间的一些直接通信道路造价（单位：万元），试给出一个设计方案，使得各城市之间既能够保持通信，又使得总造价最小，并计算其最小值。

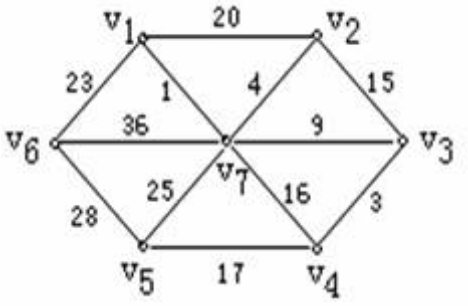

图1 七个城市赋权图

实验原理和方法

为了求解最小代价，使花费的总代价最小，这是数学中经典的求解最小(费用)生成树的算法。我们采用Kruskal算法，此算法的核心思想是：
(1)假设该图G是不连通的，对该图的边以权值非降序重新排列；
(2)对于排序表中的每条边，如果现在把它放入最小生成树T不会形成回路的话，则把它加入到T中；否则丢弃；
(3)输出最小生成树T的结果，得到想要的答案。

Kruskal算法是基于贪心的思想得到的。为实现Kruskal算法，我们用并查集来进行点集的合并。在计算机科学中，并查集是一种树型的数据结构，用于处理一些不交集（Disjoint Sets）的合并及查询问题。有一个联合-查找算法（union-find algorithm）定义了两个用于此数据结构的操作：Union 以及Find，Union为将两个集合合并，Find是对合并过后的集合进行查找。

C++语言源代码

```
#include<iostream>
#include<vector>
#include<map>
#include<algorithm>
using namespace std;

//存储边的结构，分为起点、终点以及边权
struct Edge {
  string from;
  string to;
  int distance;
  bool operator <(const Edge& rhs)const {
```

```
    return distance < rhs.distance;
  }
};

//并查集的实现
string unionFind(map<string, string> m, string x){
  return m[x] == x ? x : m[x] = unionFind(m, m[x]);
}

//判断是否能生成一个最小生成树
//标志为所有的结点都处在一个并查集之中
bool checkUnionFind(map<string, string>m) {
  map<string, string>::iterator it = m.begin();
  string ans = unionFind(m, it->first);
  for (it = m.begin();it != m.end();it++) {
    if (unionFind(m,it->second) != ans)
      return false;
  }
  return true;
}

//Kruskal算法
/*
算法的核心思想是：
(1)假设该图G是不连通的，对该图的边以权值非降序重新排列；
(2)对于排序表中的每条边，如果现在把它放入最小生成树T不会形成回路的话，则把它加入到T中；否则丢
弃；
(3)输出最小生成树T的结果，得到想要的答案。
*/
vector<Edge> Kruskal(vector<Edge> edge, map<string, string> m, vector<string>
vertex) {
  vector<Edge> ans;
  sort(edge.begin(), edge.end());
  for (int i = 0;i < edge.size();i++) {
    if (unionFind(m, edge[i].from) == unionFind(m, edge[i].to))
      continue;
    m[unionFind(m, edge[i].to)] = unionFind(m, edge[i].from);
    ans.push_back(edge[i]);
  }
  if (!checkUnionFind(m))
  {
    cout << "There is no minimum spanning tree in the graph." << endl;
    ans.clear();
  }
  return ans;
}

//输出当前最小生成树
void print(vector<Edge> MST) {
  if (MST.size() == 0) {
    cout << "The tree is empty!" << endl;
    return;
  }
  for (int i = 0;i < MST.size();i++) {
    cout << MST[i].from << "-<" << MST[i].distance << ">->" << MST[i].to << "
   ";
  }
  cout << endl;
```

```
  return;
}

//判断输入边是否合法
bool is_valid(string s, vector<string> ver) {
  for (int i = 0;i < ver.size();i++)
    if (s == ver[i]) return true;
  return false;
}

//测试函数
void solve() {
  cout << "**      Power grid cost simulation system      **" << endl;
  cout << "===============================================" << endl;
  cout << "**          A---Creating grid vertices         **" << endl;
  cout << "**             B---Add side of grid            **" << endl;
  cout << "**    C---Constructing minimum spanning tree   **" << endl;
  cout << "**        D---Show minimum spanning tree       **" << endl;
  cout << "**                   E---Exit                  **" << endl;
  cout << "===============================================" << endl;
  vector<Edge> min_spanning_tree;
  vector<Edge> edge;
  vector<string> vertices;
  map<string, string> union_find;
  while (true) {
    cout << "Please select an action:";
    char op;
    cin >> op;
    if (op == 'A') {
      int total;
      cout << "Please input the number of vertices:";
      cin >> total;
      if (cin.fail())
        throw invalid_argument("Wrong Input!");
      cout << "Please input the vertices:";
      for (int i = 0;i < total;i++) {
        string vertex;
        cin >> vertex;
        vertices.push_back(vertex);
      }
    }
    else if (op == 'B') {
      while (true) {
        cout << "Please input two vertices and edge(end with ? ? 0 ):";
        Edge e;
        cin >> e.from >> e.to >> e.distance;
        if (e.from == "?" && e.to == "?" && e.distance == 0)
          break;
        if (!is_valid(e.from, vertices) || !is_valid(e.to, vertices))
          throw invalid_argument("Wrong Input!");
        union_find[e.from] = e.from;
        union_find[e.to] = e.to;
        edge.push_back(e);
      }
    }
    else if (op == 'C') {
      //Kruskal
      min_spanning_tree = Kruskal(edge, union_find, vertices);
```

```
      if(min_spanning_tree.size()!=0)
        cout << "Minimum spanning tree generated!" << endl;
    }
    else if (op == 'D') {
      cout << "Vertices and edges in minimum spanning tree are:" << endl;
      print(min_spanning_tree);
    }
    else if (op == 'E') {
      break;
    }
    else {
      throw invalid_argument("Wrong Input");
    }
  }
  return;
}

int main(void) {
  solve();
  return 0;
}
```

实验结果

输入所求图的顶点数和边数，并依次输入边两端的顶点编号及边上对应的权值后，求得的最小耗费是：23+1+4+9+3+17=57（万元），如下图所示。

```
**      Power grid cost simulation system      **
==================================================
**          A---Creating grid vertices          **
**             B---Add side of grid             **
**    C---Constructing minimum spanning tree    **
**        D---Show minimum spanning tree        **
**                   E---Exit                   **
==================================================
Please select an action:A
Please input the number of vertices:7
Please input the vertices:1 2 3 4 5 6 7
Please select an action:B
Please input two vertices and edge(end with ? ? 0 ):1 2 20
Please input two vertices and edge(end with ? ? 0 ):2 3 15
Please input two vertices and edge(end with ? ? 0 ):3 4 3
Please input two vertices and edge(end with ? ? 0 ):4 5 17
Please input two vertices and edge(end with ? ? 0 ):5 6 28
Please input two vertices and edge(end with ? ? 0 ):6 1 23
Please input two vertices and edge(end with ? ? 0 ):1 7 1
Please input two vertices and edge(end with ? ? 0 ):2 7 4
Please input two vertices and edge(end with ? ? 0 ):3 7 9
Please input two vertices and edge(end with ? ? 0 ):4 7 16
Please input two vertices and edge(end with ? ? 0 ):5 7 25
Please input two vertices and edge(end with ? ? 0 ):6 7 36
Please input two vertices and edge(end with ? ? 0 ):? ? 0
Please select an action:C
Minimum spanning tree generated!
Please select an action:D
Vertices and edges in minimum spanning tree are:
1-<1>->7      3-<3>->4      2-<4>->7      3-<9>->7      4-<17>->5      6-<23>->1
Please select an action:E
```

图2 求最小生成树的图例