Experiment 5

BY YANGZHE PB17000025

1 道路规划

1.1 Description

沙漠上，新建了 N 座城市，用 $1, 2, 3...N$ 表示，城市与城市之间还没有道路，现在需要建设这些城市的道路网络，需要在城市间修建道路。施工队给出了 M 条道路的预计费用信息，每条道路的预计费用信息可以表示为 $U_i\,V_i\,W_i$（即如果要在 U_i 和 V_i 之间修建道路，预计费用为 Wi），道路是双向的。现从 M 条道路中选择一定数量的道路来修建，使得这些城市之间两两之间可达（可以通过其他城市间接到达），你需要求出达成上述条件的最少预算.

1.2 Input Description

第 1 行为 2 个整数 N、M, 表示城市数量和施工队给出的 M 条道路的预计费用信息

接下来的 M 行, 每行描述一条道路的预计费用信息, 形式为 $U_i\,V_i\,W_i$, 表示如果要在 U_i 和 V_i 之间修建道路, 预计费用为 W_i。

对于所有数据, 满足 $M \geqslant N-1, 1 \leqslant \mathrm{Ui} \leqslant N, 1 \leqslant \mathrm{Vi} \leqslant N, 1 \leqslant \mathrm{Wi} \leqslant 100$.

对于 100% 的数据, 满足一定存在一种方案, 使得任意两座城市都可以互相到达, 且一条道路仅会出现一次。

数据规模：

30%的数据,$2 \leqslant N \leqslant 100, 2 \leqslant M \leqslant 500$

60%的数据,$2 \leqslant N \leqslant 1000, 2 \leqslant M \leqslant 5000$

100%的数据,$2 \leqslant N \leqslant 10^5, 2 \leqslant M \leqslant 10^6$

1.3 Output Description

输出一个数字，表示使得所有城市连通的最少预算。

1.4 Solutions

要求N个城市两两间可达，且预算最小。也就是求该无向图的最小生成树(MST)。

这里使用Prim算法，伪代码如下：

```
MST-PRIM(G,w,r)
for each vertex u in G,V do
    u.key=INF
    u.pi=NIL
r.key=0//从r开始
Q=G.V
while Q != Empty do
    u=Extract-Min(Q)//选出离当前的生成树最近的顶点
    for each v in G.adj[u] do
```

```
9.          if v in Q and w(u,v)<v.key do//调整
10.             v.pi=u
11.             v.key=w(u,v)
```

其中Q是一个优先队列，实现方式有很多。这里直接使用C++的vector，在下一题的Dijkstra算法中使用了C++的priority_queue。

Q涉及到三种操作,Extract-Min,Find以及Decrease-key。

Find则通过一个布尔数组visited来实现,visFited=True表示在队列中。

Decrease-key通过辅助数组key来实现，所有的key都直接存到数组里，如果要修改，就直接修改key[v]即可,v是某个顶点。

Extract-Min是直接查找key[]中已经在队列里面(visited=True)元素的最大值。并把该元素的visited置为False，表示从队列中删除。

除了Q的三种操作，本题中的图使用邻接链表表示:

使用Adj保存每个顶点相邻的点，它的类型是std::vector<std::vector<int>>，使用W存储边的权重,W是一个简单的二维数组，由两个顶索引。

AC:https://202.38.86.171/status/699754d1b677b07a5aeb1cf4233926b6

1.5 Complexity Analysis

时间复杂

度:

1-5行需要$O(V)$，6-11行共执行$|V|$次，Extract-Min需要$O(V)$，8-11行执行$O(E)$次，其中Decrease-key和Find都是$O(1)$，所以总时间复杂度为$O(V+V^2+E)=O(V^2)=O(N^2)$

空间复杂度:

存储点之间的关系Adj需要$O(E)$的空间，而存储权重的W[][]需要$O(VE)$,key[]需要$O(V)$,visited[]需要$O(V)$。故共$O(E+VE+V+V)=O(VE)=O(NM)$。空间复杂度相对较高，会在下一题有改进。

2 逃离迷宫

2.1 Description

小明被困在一个迷宫之中，迷宫中共有 N 个点，标号分别为 $1,2,3...N$，且迷宫只有一个出口。N 个点之间互相有一些道路连通（单向），两个点之间可能有多条道路连通，但是并不存在一条两端都是同一个点的道路。小明希望知道从当前位置 S 去往出口 T 的最短距离是多少。如果不存在去往出口的道路，输出 -1。

2.2 Input Description

第一行为 4 个整数, $NMST$, 分别代表节点个数、道路条数、小明当前所处的位置的标号、出口标号

接下来 M 行, 每行表述一条道路, 表述为的形式, 表示一条从 U_i 到 V_i, 距离为 W_i 的单向边。

数据规模:

对于所有的数据, $1\leqslant U_i\leqslant N, 1\leqslant V_i\leqslant N, 1\leqslant W_i\leqslant 100$

30%的数据, $2\leqslant N\leqslant 100, 2\leqslant M\leqslant 500$

60%的数据, $2\leqslant N\leqslant 1000, 2\leqslant M\leqslant 20000$

100%的数据,$2 \leqslant N \leqslant 10^5, 2 \leqslant M \leqslant 10^6$

2.3 Output Description

如果小明能逃离迷宫，输出从他的位置到出口的最短距离，否则输出 '-1'.

2.4 Solutions

求从当前位置S到达T的最短距离，也就是求单源最短距离，我们直接使用Dijkstra算法。对于S到T没有路径的情况，因为我们设置的初始值为INF，直接判断处理后是否还是INF即可。

Dijkstra算法的伪代码如下：

```
DIJISTRA(G,w,s)
INITIAL-SINGLE-SOURCE(G,s)
S=Empty
Q=G.V//这里实现的时候做了一些调整,Q最开始只有s一个点
while Q!=Empty do
    u=Extract-Min(Q)
    S=S U {u}
    for each vertex v in G.adj[u] do
        RELAX(u,v,w)
```

这里同样涉及优先队列Q，涉及到三个操作，一个是`Extract-Min`，一个是`Decrease-Key`，还有一个是按照下标找到Q中对应的值。这里我们使用`C++`中的`priority_queue`。由于`C++`中的`priority_queue`并不支持按下标访问，并且仅支持push和pop两种操作，因此我们使用一些Trick。

首先Q的数据类型为`std::priority_queue<vertex,std::vector<vertex>,myoperator> Q;`

这里的`vertex`由下面定义：

```
class vertex
{
public:
        int d;
        int index;
        vertex(int d, int index) :d(d), index(index) {}
};
```

即Q中的每个元素都有下标和它对应的distance。

为了能够按下标索引，我们依然利用两个辅助数组`key`和`visited`，其中key中的值与Q中的d相同，而`visited`依然表示是否在队列中。

优先队列使用d进行排序，因此我们重载运算符，定义myoperator。

这样我们可以利用优先队列在$O(1)$的时间内找到最短的distance(直接取栈顶)，并用$O(\lg(V))$把它pop出来。然后再在$O(\lg(V))$的时间内完成`Decrease-Key`。这个操作通过直接push实现。因为我们每次只找最小值，所以如果一个点有两个d都在Q中，旧的d不会被访问到，这样变相地完成了`Decrease-Key`。

而`Extract-Min`可以直接通过修改`visited`完成，时间为$O(1)$。

通过这样修改，Relax就可以用`visited`和`key`来实现：

```
int v = (*it).node;
if (visited[v] &&((*it).weight +key[u]) < key[v])
{
```

```
        key[v] = ((*it).weight + key[u]);
        Q.push(vertex(key[v], v));
}
```

除了Q的三种操作, 我们这次依然使用邻接链表, 但与上一题不一样的是, 我们不再使用W[][]来保存权重,从上一题的空间复杂度分析可知,这样很浪费空间。我们定义了全新的数据结构:

```
typedef struct Entry {
        int node;
        double weight;

        bool operator == (const int e) const
        {
                return (node == e);
        }
        Entry(int n, int w) :node(n), weight(w) {}
} Entry;

std::vector<std::vector<Entry>> Graph(N);
```

Entry就是每个顶点的类型,它包含node表示编号,weight表示权重。

由于权重对应一条边两个点, 所以Graph是一个vector, 它的每个元素是一个Entry的vector, 表示该元素对应的邻接链表。这样,weight除了node的一个顶外,还有一个顶就是Graph[v]。

这样定义的数据结构充分利用了空间,使用$O(V^2)$的空间,远比上一题中的$O(VE)$好。

AC: https://202.38.86.171/status/b97d7273decc9490201ddcdac7a2a871

2.5 Complexity Analysis

时间复杂度:

4-8行执行$|V|$次,7-8行执行$|E|$次,Extract-Min需要$O(\lg(V))$,Relax需要$O(\lg(V))$

故时间复杂度为$O(V\lg(V)+E\lg(V))=O(E\lg(V))=O(M\lg(N))$

空间复杂度:

存储整个图需要$O(V^2)$, Q需要$O(V)$,visited和key需要$O(V)$, 故共$O(V^2+V+V)=O(V^2)=O(N^2)$.

2.6 遇到的问题

1. 这个题最大的问题应该是时间, 这一题的数据量比较大。 如果直接用第一题的那种遍历Q的方式找min, 肯定会超时。 但是C++提供的优先队列不支持按照索引修改, 也不支持按照索引访问。 因此利用两个辅助数组和priority_queue联合解决是我的解决方法。

2. 另一个问题是空间, 第一题的数据结构直接用到这一题也会超时, 因为$O(VE)$还是太浪费空间了,这里参考了stackoverflow, 修改了邻接链表的写法。

3 货物运输

3.1 Description

在一个工厂货物运输系统中共有N个节点,编号为$1,2,3...N$,节点之间有传送带(单向)连接,每个传送带都有使用寿命,传送带的寿命为一个数字L,表示在传送完L个单位的货物后,传送带就会破损无法使用。现在需要从节点S向节点T传送货物,求在当前传输系统中,最多可以顺利传输多少单位的货物从节点S到节点T.

3.2 Input Description

第一行为 4 个整数, $N\,M\,S\,T$, 分别代表节点个数、传送带数目、起点标号 S、目标点标号 T 接下来 M 行, 每行表述一条传送带的信息, 表述为 $U_i\,V_i\,L_i$ 的形式, 表示一条从节点 U_i 到节点 V_i, 寿命为 L_i 的传送带。

数据规模:

对于所有的数据, $1 \leqslant U_i \leqslant N, 1 \leqslant V_i \leqslant N, 1 \leqslant L_i \leqslant 100$

40%的数据, $2 \leqslant N \leqslant 50, 2 \leqslant M \leqslant 500$

100%的数据, $2 \leqslant N \leqslant 500, 2 \leqslant M \leqslant 20,000$

3.3 Output Description

输出一个整数, 表示在当前传输系统中, 最多可以顺利传输多少单位的货物从节点 S 到节点 T。

3.4 Solutions

最大流问题，这里使用改进后的Ford-Fulkerson算法，即Edmonds-Karp算法。

伪代码如下:

```
Edmonds-Karp(G,s,t)
for each edge (u,v) in G.E do
    (u,v).f=0 //初始化流为全0
while BFS(Gf,s,t)!=Empty do //通过BFS判断残存网络中是否有增广路径
     cf(p)=min{cf(u,v):(u,v) is in p} //寻找增广路径中的最小边
     for each edge in u,v p do
         if(u,v) in E then
             (u,v).f=(u,v).f+cf(p)//(u,v)在E中,说明流还可以增加
         else
             (v,u).f=(v.u).f-cf(p)
            //(u,v)不在E中，说明已经不能增加，这是一条反向边，需减少
```

重要的是根据当前的流网络，构建出残存网络，然后用BFS寻找一条从S到T 的增广路径。

一条流网络里的边，如果流等于容量，那么在残存网络里会有一条反向边。如果流小于容量，则在残存网络中会有两条边，一条正向一条反向:

$$c_f(u,v) = \left\{ \begin{array}{ll} c(u,v) - f(u,v) & \text{if } (u,v) \in E \\ f(v,u) & \text{if } (v,u) \in E \\ 0 & \text{otherwise} \end{array} \right\}$$

在实现的时候，直接通过原网络中这条边的流是否为0，判断(u,v)是否属于E.

这样， 我们可以调用GetRes()获得原网络的残存网络， 然后把得到的G_f传入BFS， 寻找从S到T的的路径， 如果找到了， 利用$T.\pi$不断往回迭代， 找到S， 得到反向的p， 反着存储就能得到p。 然后根据p的最小边调整原来的流网络。

调整的规则是：如果原网络中有增广路径上的边(u,v)，说明这条边表示剩余的容量，我们给原网络加上这个量。如果原网络中不存在边(u,v),说明这条边是生成的反向边，表示已经有的流，我们应该给原网络减掉这个量。

这样一直调整， 直到BFS找不到一个从S到T的路径， 说明已经达到最大流。 我们可以在每次调整流网络的时候记录下来增加的量，这样加在一起就是最大流。

AC: https://202.38.86.171/status/f5ee24542c12f1441b7d5a98c5fd7dbe

3.5 Complexity Analysis

时间复杂度：

生成残存网络需要$O(E)$，BFS需要$O(E)$的时间执行完毕，而Edmonds-Karp总共要执行$O(VE)$次.

故总时间复杂度为$O(VE^2)=O(NM^2)$.

其实如果按照实现计算的话，应该是$O(V^3 E + VE^2) = O(VE(V^2 + E))$，因为实现中生成残存网络用了$O(V^2)$。

空间复杂度：

存储流网络需要一个表示容量的C，一个表示流量的F。

保存残存网络需要一个表示边的R。

上面的每一个都用二维数组存储，所以需要$O(3N^2)=O(N^2)$

表示增广路径p需要$O(N)$，执行BFS需要一个$O(N)$的队列。

故总空间复杂度为$O\left(N^2+N+N\right)=O(N^2)$

4 图中最大的集合

4.1 Description

在一张有向图 G 中，你需要找出节点数最多的一个节点集合 S，使得 S 中的任意两个节点 A、B 至少满足"A 可达 B"或者"B 可达 A"中的一个。如果 A 到 B 有连边，B 到 C 有连边，那么我们认为 C 对 A 是可达的，即 A 可达 C。

4.2 Input Description

第一行为 2 个整数，$N\,M$，分别代表节点个数、边的个数接下来 M 行，每行一条边的信息，表述为 $U_i\,V_i$ 的形式，表示一条从节点 U_i 到节点 V_i 的边。

数据规模：

对于所有的数据，$1\leqslant U_i\leqslant N, 1\leqslant V_i\leqslant N$

40%的数据，$2\leqslant N\leqslant 50, 0\leqslant M\leqslant 500$

100%的数据，$2\leqslant N\leqslant 5000, 0\leqslant M\leqslant 10^5$

4.3 Output Description

输出一个数字，表示满足上述条件的最大的集合包含的节点的个数。

4.4 Solutions

这个题要求最大的可达集合，根据可达的定义和传递性，其实就是求这个图中的最长路径。

更贴切的说，是找到一条经过节点最多的路径（因为可能有环，路径可能一直增长）。

我们对于有向无环图，可以用如下算法计算最长路径：

```
LONGEST-PATH(G)
```

```
Input:DAG G=(V,E)
Output: the cost of longest path
Topologically sort G
for each vertex v in V in topological order:
    do dist(v)=max{dist(u)+w(u,v)| (u,v) in E}
return max{dist(v)| v in V}
```

但是本题中可能是有环的（比如样例），所以我们要找到那些强连通分量(SCC)，把他们缩成点，并且经过这些点时，会增加该SCC中所含有的元素的个数。

所以我们先找到SCC：

```
STRONGLY-CONNECTED-COMPONENTS(G)
1. call DFS(S) to compute the finishing time u.f for each vertex u
2. compute the converse of G,which is GT
3. call DFS(GT) and in the main loop of DFS, use the order of decreasing u.f
4. output the vertices of each tree in DFS
```

其实这里第3步就是拓扑序。

在找到SCC后，我们记录下每个SCC的数量scc_n。当统计结束后，共有N个SCC。

这个时候，我们就可以建立一个节点数N的新图，每个顶点就是原来的SCC，并且经过该顶点，就会增加scc_n个cost。

新图的边就是其他SCC与该SCC的连线，为了更加快速的判断，我在DFS中加入了一个辅助数组parent[]

每一次在循环里执行DFS的时候，就给DFS传入一个数字count，然后所有迭代的节点，parent都是count，直到这一次DFS结束，循环重新走到下一个DFS，count自增。

于是，一个SCC的元素都有共同的parent，这样我们判断边在不在新图里时，只需要判断边的两个端点是不是具有相同的parent即可（其实这个相当于并查集）。

然后我们构建完新图后，新图是一个有向无环图，我们直接跑最开始那个算法即可。唯一不同的是，正常使用最长路径算法时，dist[]都初始化为0，而我们这里因为SCC自带cost，所以dist[n]=scc_n，然后直接输出最长路径就是我们要的答案。

总结一下思路：SCC缩点→构建新图→找到新图中的最长路径。

AC:https://202.38.86.171/status/5b9d4580cbc0ee2ed55a048542550401

4.5 Complexity Analysis

时间复杂度：

最开始为了获得拓扑序，跑了一遍DFS，并把序列反转，时间$O(E+V)$。

然后求G的转置，需要$O(E)$，在G^T中跑DFS,$O(V+E)$。

构建新图，需要遍历每个边,$O(E)$，获得新图的拓扑序$O(V+E)$。

找最长路径$O(V+E)$。

故总的时间复杂度为$O(V+E)+O(E)=O(V+E)=O(N+M)$

空间复杂度：

首先存储G原图需要$O(V^2)$,存储G^T需要$O(V^2)$,存储新图需要$O(V^2)$。

parent、Order、ReverseOrder、SCC、visited、dist都需要$O(V)$

故总的空间复杂度为$O(V^2)=O(N^2)$