对于最小费用最大流问题，首先给出问题定义：

给定一个网格，每条边除了有容量限制，还有单位流量费用。

当的流量为时，需要花费的费用。

满足斜对称性，即。

该网络图中总花费最小的最大流则为最小费用最大流，即在最大化的前提下最小化。

此问题我们可以用PD算法（即原始对偶算法）在内解决。

PD的总体思路为首先利用SPFA算法（队列优化的最短路算法）为每个节点设置一个 用于将所有节点的权值转为非负数使得后续增广路计算时可以运用算法解决。

由于算法的正确性，我们可以类比地证明设置了势能的新网络上最短路与原网络一一对应。

每次求得增广边后增广图形态会产生变化，因此需要更新各个节点的势能。

查阅资料[1]得，设增广后源点到第个节点的最短距离记为，只需给对应的加上即可。

由于对于任意次增广后的残量网络，有,即新增的边权也是非负的。

接下来的思路与算法相同，即每次寻找单位费用最小的增广路进行增广，直到图上不存在增广路为止。

**此处思路与教材所讲贪心法类似，但进行了时间复杂度上的优化。**

由于我们已经对网络图的权值增加了，因此可以用算法求解每一次的最小增广路。具体实现如下：

|  |
| --- |
| class PD  {  private:  class edge  {  public:  int v, f, c, next;  edge(int \_v,int \_f,int \_c,int \_next)  {  v = \_v;  f = \_f;  c = \_c;  next = \_next;  }  edge() { }  } ;  void vecset(int value,vector<int> &arr)  {  for(int i = 0;i < arr.size();i ++) arr[i] = value;  return;  }  class node  {  public:  int v, e;  } ;  class mypair  {  public:  int dis, id;  bool operator<(const mypair &a) const { return dis > a.dis; }  mypair(int d, int x)  {  dis = d;  id = x;  }  };  std::vector<int> head;  std::vector<int> dis;  std::vector<int> vis;  std::vector<int> h;  std::vector<edge> e;  std::vector<node> p;  int n, m, s, t, cnt = 1, maxf, minc;  public:PD(int \_n,int \_m,int \_s,int \_t)  {  n = \_n;  m = \_m;  s = \_s;  t = \_t;  maxf = 0;  minc = 0;  head.resize(n+2);  dis.resize(n+2);  vis.resize(n+2);  e.resize(2);  h.resize(n+2);  p.resize(m+2);  }  Public: void addedge(int u, int v, int f, int c)  {  e.push\_back(edge(v,f,c,head[u]));  head[u] = e.size()-1;  e.push\_back(edge(u,0,-c,head[v]));  head[v] = e.size()-1;  }  private:bool dijkstra()  {  std::priority\_queue<mypair> q;  vecset(INF,dis);  vecset(0,vis);  dis[s] = 0;  q.push(mypair(0, s));  while (!q.empty())  {  int u = q.top().id;  q.pop();  if (vis[u])  continue;  vis[u] = 1;  for (int i = head[u]; i; i = e[i].next)  {  int v = e[i].v, nc = e[i].c + h[u] - h[v];  if (e[i].f && dis[v] > dis[u] + nc)  {  dis[v] = dis[u] + nc;  p[v].v = u;  p[v].e = i;  if (!vis[v])  q.push(mypair(dis[v], v));  }  }  }  return dis[t] != INF;  }  private:void spfa()  {  std::queue<int> q;  vecset(63,h);  h[s] = 0, vis[s] = 1;  q.push(s);  while (!q.empty())  {  int u = q.front();  q.pop();  vis[u] = 0;  for (int i = head[u]; i; i = e[i].next)  {  int v = e[i].v;  if (e[i].f && h[v] > h[u] + e[i].c)  {  h[v] = h[u] + e[i].c;  if (!vis[v])  {  vis[v] = 1;  q.push(v);  }  }  }  }  }  private:int pd()  {  spfa();  while (dijkstra())  {  int minf = INF;  for (int i = 1; i <= n; i++)  h[i] += dis[i];  for (int i = t; i != s; i = p[i].v)  minf = min(minf, e[p[i].e].f);  for (int i = t; i != s; i = p[i].v)  {  e[p[i].e].f -= minf;  e[p[i].e ^ 1].f += minf;  }  maxf += minf;  minc += minf \* h[t];  }  return 0;  }  public: void printAns()  {  pd()  std::cout << maxf << " " << minc << "\n";  }  }; |

需要的STL库：

这段代码实现了一个名为的类，其中成员函数列表如下：

|  |  |  |
| --- | --- | --- |
| 成员名称 | 访问控制 | 用途 |
|  |  | 输出答案 |
|  |  | 在图上添加权边 |
|  |  | 用于运行算法 |
|  |  | 用于计算新增加的权值 |
|  |  | 利用求得最小费用增广边 |
|  |  | 构造函数，用于初始化类 |

此类的用法为：

首先调用构造函数，其中分别为节点数量、边数量、源点坐标、汇点坐标。

随后循环调用方法添加边，其中分别为边的出发点、目标点、最大流与每单位流费用。

完成所有边的添加以后，内部就完成了整个网络的构建。

随后便可以调用输出答案。

答案为一行两个数，与，代表最大流与对应的最小费用。

对于例2，记分别为点，构造源点与汇点，输入数据：

|  |
| --- |
| 8 15 1 8  1 2 1 0  1 3 1 0  1 4 1 0  2 5 1 3  2 6 1 1  2 7 1 2  3 5 1 5  3 6 1 10  3 7 1 5  4 5 1 26  4 6 1 28  4 7 1 2  5 8 1 0  6 8 1 0  7 8 1 0 |

答案为

|  |
| --- |
| 3 8 |

即最大流为3，最小总费用为8

对于其他节点数为十万量级的一般情况，代码已经通平台的正确性与性能测试。

参考文献：

[1] OI Wiki Team. SSP 算法[J/OL]．OIwiki，GitHub Repository，18(2)： https://github.com/OI-wiki/OI-wiki．