

```

1
2 #####
3 #
4 #          グラフ関係
5 #
6 #####
7
8 N # 頂点数
9 lines = defaultdict(set)
10 lines[s].add((t,c)) # s->t コストc の辺 0-indexed に直す
11
12
13 # ベルマンフォード
14 # 最短経路 (負辺有り) and 負コスト回路検出
15 def BellmanFord(Start,Goal,lines,N):
16     Costs=[INF]*N
17     Costs[Start] = 0
18     upd8s = [True]*N
19     for i in range(2*N): #2N回ループ(負回路の検出までみる)
20         will_upd8s = [False]*N
21         upd8 = False
22         for s in range(N):
23             if not upd8s[s]: continue #前回更新してないので見ない
24             for t,c in lines[s]:
25                 if c + Costs[s] < Costs[t]:
26                     Costs[t] = Costs[s]+c
27                     upd8 = True
28                     will_upd8s[t] = True #更新した点だけ次に見る
29
30         if not upd8: #なにも更新しなかったら終わり
31             return Costs[Goal]
32
33         if i == N-1: #Nループ目のGoalのCostを記録
34             tmp = Costs[Goal]
35
36         upd8s = will_upd8s[:]
37
38     if tmp != Costs[Goal]:
39         return -INF
40     else:
41         return Costs[Goal]
42
43
44 # ダイクストラ
45 # 最短距離 (負辺無し)
46 def Dijkstra(s,lines,N):
47     weight = [INF]*N
48     weight[s] = 0
49
50     def search (s,w_0,q,weight): #s->t
51         for line in list(lines[s]):
52             t = line[0]
53             w = w_0 + line[1]
54             if weight[t] > w:
55                 heapq.heappush(q, [w,t])
56                 weight[t] = w
57
58     q = [[0,s]]
59     heapq.heapify(q)
60     while q:
61         w,n = heapq.heappop(q)
62         search(n,w,q,weight)
63
64     return weight
65
66
67
68
69
70
71
72

```

```

73
74 # Ford_Fulkerson
75 # 最小カット最大フローやつ
76 N,E = inpl()
77 Start = 0
78 Goal = N-1
79 ans = 0
80
81 lines = defaultdict(set)
82 cost = [[0]*N for i in range(N)]
83 for i in range(E):
84     a,b,c = inpl()
85     if c != 0:
86         lines[a].add(b)
87         cost[a][b] += c
88
89 def Ford_Fulkerson(s): #sからFord-Fulkerson
90     global lines
91     global cost
92     global ans
93
94     queue = deque() #BFS用のdeque
95     queue.append([s,INF])
96     ed = [True]*N #到達済み
97     ed[s] = False
98     route = [0 for i in range(N)] #ルート
99     route[s] = -1
100
101     #BFS
102     while queue:
103         s,flow = queue.pop()
104         for t in lines[s]: #s->t
105             if ed[t]:
106                 flow = min(cost[s][t],flow) #flow = min(直前のflow,line容量)
107                 route[t] = s
108                 queue.append([t,flow])
109                 ed[t] = False
110                 if t == Goal: #ゴール到達
111                     ans += flow
112                     break
113             else:
114                 continue
115         break
116     else:
117         return False
118
119     #ラインの更新
120     t = Goal
121     s = route[t]
122     while s != -1:
123
124         #s->tのコスト減少, ゼロになるなら辺を削除
125         cost[s][t] -= flow
126         if cost[s][t] == 0:
127             lines[s].remove(t)
128
129         #t->s(逆順)のコスト増加, 元がゼロなら辺を作成
130         if cost[t][s] == 0:
131             lines[t].add(s)
132         cost[t][s] += flow
133
134         t = s
135         s = route[t]
136
137     return True
138
139 while True:
140     if Ford_Fulkerson(Start):
141         continue
142     else:
143         break
144
145
146

```

```
147
148
149 # ワーシャルフロイド
150 # 全点間 最短距離
151 N,M = inpl()
152 cost = [[INF for i in range(N)] for j in range(N)]
153
154 for _ in range(M):
155     a,b,c = inpl()
156     a,b = a-1,b-1
157     cost[a][b] = c
158     cost[b][a] = c
159
160 for k in range(N):
161     for i in range(N):
162         for j in range(N):
163             cost[i][j]=min(cost[i][j],cost[i][k]+cost[k][j])
164
165
166
167 # クラスカル
168 # 最小全域木
169 class UnionFind:
170     # 貼る
171
172 N,M = inpl()
173 UF = UnionFind(N)
174 q = []
175 for _ in range(M):
176     a,b,w = inpl()
177     a,b = a-1,b-1
178     q.append([w,a,b])
179 q.sort()
180 weight = 0
181 for w,a,b in q:
182     if not UF.Check(a,b):
183         weight += w
184         UF.Unite(a,b)
185
```