

```

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

```

```

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

```

```

147
148
149 # クラスカル
150 # 最小全域木
151 class UnionFind:
152     # 貼る
153
154 N,M = inpl()
155 UF = UnionFind(N)
156 q = []
157 for _ in range(M):
158     a,b,w = inpl()
159     a,b = a-1,b-1
160     q.append([w,a,b])
161 q.sort()
162 weight = 0
163 for w,a,b in q:
164     if not UF.Check(a,b):
165         weight += w
166         UF.Unite(a,b)
167
168
169
170 #####
171 #         数学系
172 #####
173
174
175 # Combination
176 class Combination:
177     def __init__(self,N):
178         self.fac = [1]*(N+1)
179         for i in range(1,N+1):
180             self.fac[i] = (self.fac[i-1]*i)%mod
181         self.invmod = [1]*(N+1)
182         self.invmod[N] = pow(self.fac[N],mod-2,mod)
183         for i in range(N,0,-1):
184             self.invmod[i-1] = (self.invmod[i]*i)%mod
185
186     def calc(self,n,k):#nCk
187         return self.fac[n]*self.invmod[k]%mod *self.invmod[n-k] %mod
188
189
190 #最大公約数
191 def gcd(a,b):
192     while b:
193         a,b = b, a%b
194     return a
195
196 #最小公倍数
197 def lcm(a,b):
198     return a*b // gcd(a,b)
199
200
201 # なんか早い素数判定
202 def is_prime(x):
203     if x < 2: return False # 2未満に素数はない
204     if x == 2 or x == 3 or x == 5: return True # 2,3,5は素数
205     if x % 2 == 0 or x % 3 == 0 or x % 5 == 0: return False # 2,3,5の倍数は合成数
206
207     # 疑似素数で割る
208     prime = 7
209     step = 4
210     while prime <= math.sqrt(x):
211         if x % prime == 0: return False
212         prime += step
213         step = 6 - step
214
215     return True
216

```