

```

1
2 ##### いつもの #####
3
4 from collections import defaultdict, deque
5 import sys, heapq, bisect, math, itertools, string, queue
6 sys.setrecursionlimit(10**8)
7 INF = float('inf')
8 mod = 10**9+7
9 eps = 10**-7
10 def inpl(): return list(map(int, input().split()))
11 def inpl_str(): return list(input().split())
12
13
14
15
16 ##### bit 関係 #####
17
18 #bit shift
19 a << b, a >> b
20 > a = 101, b=2
21 > a<<b = 10100
22 > a<<b = 1
23
24 #bitwiseOR AND XOR
25 a|b , a&b , a^b
26 > a = 10010, b=00110
27 > a|b = 10110
28 > a&b = 00010
29 > a^b = 10100
30
31 #bit長
32 b = 10101
33 > b.bit_length() = 5
34
35 #一番最初のbitが立ってるものをとる
36 b&-b
37 > b = 10110
38 > b&-b = 10
39
40 #ビットマスク(特定の桁だけ)
41 a = 10100
42 > (a>>0) & 1 = 0
43 > (a>>1) & 1 = 0
44 > (a>>2) & 1 = 1
45 > (a>>3) & 1 = 0
46 > (a>>4) & 1 = 1
47
48 #next_combination (n桁でk箇所bitが立ってる物を全探索)
49 def next_com(bit):
50     x = bit & -bit
51     y = bit + x
52     return (((bit & ~y) // x) >> 1) | y
53 n,k = 5,3
54 bit = (1<<k)-1
55 ans = 0
56 while bit < (1<<n):
57     for i in range(n):
58         if (bit>>i) & 1:
59             # 処理
60     bit = next_com(bit)
61
62
63 #bitが立ってる数をカウント
64 def bitcount(bits):
65     bits = (bits & 0x55555555) + (bits >> 1 & 0x55555555)
66     bits = (bits & 0x33333333) + (bits >> 2 & 0x33333333)
67     bits = (bits & 0x0f0f0f0f) + (bits >> 4 & 0x0f0f0f0f)
68     bits = (bits & 0x00ff00ff) + (bits >> 8 & 0x00ff00ff)
69     return (bits & 0x0000ffff) + (bits >> 16 & 0x0000ffff)
70
71
72

```

```

73 ##### itertools #####
74 seq = ('a', 'b', 'c', 'd', 'e')
75
76 # 並べ方
77 list(itertools.permutations(seq))
78 >[('a', 'b', 'c', 'd', 'e'),
79   ('a', 'b', 'c', 'e', 'd'),
80   ('a', 'b', 'd', 'c', 'e'),
81   ('a', 'b', 'd', 'e', 'c'),
82   中略
83   ('e', 'd', 'c', 'a', 'b'),
84   ('e', 'd', 'c', 'b', 'a')]
85
86 # 何個かを選ぶ並べ方
87 list(itertools.permutations(seq, 3))
88 > [('a', 'b', 'c'),
89   ('a', 'b', 'd'),
90   ('a', 'b', 'e'),
91   ('a', 'c', 'b'),
92   中略
93   ('e', 'd', 'a'),
94   ('e', 'd', 'b'),
95   ('e', 'd', 'c')]
96
97 # 重複を許す順列 ([True,False]でやればbit全探索ができる)
98 list(itertools.product(A, repeat=3))
99 > [('a', 'a', 'a'),
100   ('a', 'a', 'b'),
101   ('a', 'a', 'c'),
102   ('a', 'b', 'a'),
103   ('a', 'b', 'b'), ...
104
105 # 組み合わせ
106 list(itertools.combinations(seq,5))
107 > [('a', 'b', 'c', 'd', 'e')]
108
109 # 何個かを選ぶ組み合わせ
110 list(itertools.combinations(seq,3))
111 [('a', 'b', 'c'),
112   ('a', 'b', 'd'),
113   ('a', 'b', 'e'),
114   ('a', 'c', 'd'),
115   ('a', 'c', 'e'), ...
116
117 # 重複を許す組み合わせ
118 list(itertools.combinations_with_replacement(A, 3))
119 [('a', 'a', 'a'),
120   ('a', 'a', 'b'),
121   ('a', 'a', 'c'),
122   ('a', 'b', 'b'),
123   ('a', 'b', 'c'), ...
124
125
126 ##### 編集距離 #####
127 S1 = 'yafo'
128 S2 = 'yahoo'
129 def levenshtein(s1,s2):
130     n, m = len(s1), len(s2)
131     dp = [[0]*(m+1) for _ in range(n + 1)]
132
133     for i in range(n+1):dp[i][0] = i
134     for j in range(m+1):dp[0][j] = j
135
136     for i in range(1, n + 1):
137         for j in range(1, m + 1):
138             if s1[i-1] == s2[j-1]:cost = 0
139             else:cost = 1
140             dp[i][j] = min(dp[i - 1][j] + 1,          # insertion
141                           dp[i][j - 1] + 1,          # deletion
142                           dp[i - 1][j - 1] + cost)    # replacement
143     return dp[n][m]
144 print(levenshtein(S1,S2))
145

```

```

1 #####
2 #
3 #           データ構造
4 #
5 #####
6
7 class UnionFind:
8     def __init__(self,N): # 頂点数 N
9         self.table = [i for i in range(N)] # 親 table[x] == x で根
10        self.rank  = [1 for i in range(N)] # 木の長さ
11        self.size  = [1 for i in range(N)] # 集合のサイズ
12
13    def Find(self,x): #xの根を返す
14        if self.table[x] == x:
15            return x
16        else:
17            self.table[x] = self.Find(self.table[x]) #親の更新
18            self.size[x] = self.size[self.table[x]]
19            return table[x]
20
21    def Unite(self,x,y,w): #xとyをdiff(x,y)=W で繋げる
22        w = w - self.weight(y) + self.weight(x)
23        x,y = self.Find(x), self.Find(y)
24        sx,sy = self.Size(x), self.Size(y)
25        if x == y: return
26        if self.rank[x] > self.rank[y]:
27            self.table[y] = x
28            self.size[x] = sx + sy
29        else:
30            self.table[x] = y
31            self.size[y] = sx + sy
32            if self.rank[x] == self.rank[y]:
33                self.rank[y] += 1
34
35    def Check(self,x,y):
36        return self.Find(x) == self.Find(y)
37
38    def Size(self,x):
39        return self.size[self.Find(x)]
40
41
42
43 class BinaryIndexedTree():
44     def __init__(self,N):
45         self.N = N
46         self.bit = [0]*(self.N+1)
47
48     def add(self,a,w):
49         x = a
50         while x <= self.N:
51             self.bit[x] += w
52             x += x & -x
53
54     def sum(self,a):
55         tmp = 0
56         x = a
57         while x > 0:
58             tmp += self.bit[x]
59             x -= x & -x
60         return tmp
61
62
63
64 class SegmentTree:
65     def __init__(self,N,d):
66         self.NN = 1
67         while self.NN < N:
68             self.NN *= 2
69         self.SegTree = [d]*(self.NN*2-1)
70
71     def update(self,i,x): #iの値をxに更新
72         i += self.NN - 1

```

```

73     self.SegTree[i] = x
74     while i>0:
75         i = (i-1)//2
76         self.SegTree[i] = self.process(self.SegTree[i*2+1],self.SegTree[i*2+2])
77
78 def query(self,a,b,k=0,l=0,r=None): #[A,B]の値, 呼ぶときはquery(a,b)
79     if r == None: r = self.NN
80     if r <= a or b <= l: #完全に含まない
81         return INF
82     elif a <= l and r <= b : #完全に含む
83         return self.SegTree[k]
84     else: #交差する
85         vl = self.query(a,b,k*2+1,l,(l+r)//2)
86         vr = self.query(a,b,k*2+2,(l+r)//2,r)
87         return(self.process(vl,vr))
88
89 def process(self,x,y): #x,yが子の時, 親に返る値
90     return min(x,y)
91
92
93
94 class PotentialUnionFind:
95     def __init__(self,N): # 頂点数 N
96         self.table = [i for i in range(N)] # 親 table[x] == x で根
97         self.rank = [1 for i in range(N)] # 木の長さ
98         self.size = [1 for i in range(N)] # 集合のサイズ
99         self.diffweight = [0 for i in range(N)]
100
101 def Find(self,x): #xの根を返す
102     if self.table[x] == x:
103         return x
104     else:
105         root = self.Find(self.table[x]) #親の更新
106         self.size[x] = self.size[self.table[x]]
107         self.diffweight[x] += self.diffweight[self.table[x]]
108         self.table[x] = root
109         return root
110
111 def Unite(self,x,y,w): #xとyをDiff(x,y)=W で繋げる
112     w = w - self.Weight(y) + self.Weight(x)
113     x,y = self.Find(x), self.Find(y)
114     sx,sy = self.Size(x), self.Size(y)
115     if x == y: return
116     if self.rank[x] > self.rank[y]:
117         self.table[y] = x
118         self.size[x] = sx + sy
119         self.diffweight[y] = w
120     else:
121         self.table[x] = y
122         self.size[y] = sx + sy
123         self.diffweight[x] = -w
124         if self.rank[x] == self.rank[y]:
125             self.rank[y] += 1
126
127 def Check(self,x,y):
128     return self.Find(x) == self.Find(y)
129
130 def Size(self,x):
131     return self.size[self.Find(x)]
132
133 def Weight(self,x): # 重さ(根からの距離)
134     self.Find(x)
135     return self.diffweight[x]
136
137 def Diff(self,x,y): # 繋がってる二点間距離
138     return self.Weight(y) - self.Weight(x)
139

```

```

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

```