Contents

1 Geometry

1.1 Closest Pair

```
import sys
import collections

Point = collections.namedtuple('Point',['x','y'])
INF = 9999999999999999999
arr = []

def ykey(a):
    return (a.y,a.x)

def xkey(a):
    return (a.x,a.y)

def dist(a,b):
    return (a.x-b.x)*(a.x-b.x) + (a.y-b.y)*(a.y-b.y)

def solve(l,r):
    n = (r-l+1)
    if n == 1:
        return INF
    mid = (l+r)//2
    d = min(solve(l,mid), solve(mid+1,r))

    ml = mid
    while ml >= l and (arr[ml].x-arr[mid].x)**2 < d:
        ml -= 1
    mr = mid
    while mr <= r and (arr[mr].x-arr[mid].x)**2 < d:
        mr += 1

    midlist = arr[ml+1:mr]
    midlist.sort(key=ykey)

    for i in range(len(midlist)):
        for j in range(1,7,1):
            if i+j >= len(midlist):
                break
            d = min(d, dist(midlist[i],midlist[i+j]))

    return d

N = int(sys.stdin.readline())
for i in range(N):
    x,y = [int(x) for x in sys.stdin.readline().split()]
    p = Point(x,y)
    arr.append(p)

arr.sort(key=xkey)

print(solve(0,N-1))
```

1.2 Binary Search

```
import sys
from bisect import *

N = int(sys.stdin.readline().strip())
arr = [int(x) for x in sys.stdin.readline().strip().split()]
M = int(sys.stdin.readline().strip())
arr2 = [int(x) for x in sys.stdin.readline().strip().split()]

arr.sort()

#print(arr)
for x in arr2:
    p = bisect_left(arr,x)
    #print(x,p,0,end='')
    if p < len(arr) and arr[p] == x:
        print(1)
    else:
        print(0)
```

2 Data Structure

2.1 SegTree Class

```
import sys
import collections

class segTree:
    def __init__(self, x):
        self.n = 1
        while self.n < x:
            self.n = self.n * 2
        self.data = [0 for i in range(self.n*2+10)]

    def query(self,l,r,node,nodeL,nodeR):
        if r < nodeL or l > nodeR:
            return 0
        if l <= nodeL and nodeR <= r:
            return self.data[node]
        val1 = self.query(l,r,node*2,nodeL,(nodeL+nodeR)//2)
        val2 = self.query(l,r,node*2+1,(nodeL+nodeR)//2+1,nodeR)
        return val1 + val2

    def update(self,p,x):
        p = p + self.n - 1
        self.data[p] = x
        p = p//2
        while p > 0:
            self.data[p] = self.data[p*2] + self.data[p*2+1]
            p = p // 2


N,M,K = [int(x) for x in sys.stdin.readline().split() ]
tree = segTree(N)
for i in range(1,N+1,1):
    x = int(sys.stdin.readline())
    tree.update(i,x)

for i in range(M+K):
    a,b,c = [int(x) for x in sys.stdin.readline().split() ]
    if a == 1:
        tree.update(b,c)
    else:
        if b > c :
            swap(b,c)
        print(tree.query(b,c,1,1,tree.n))
```

2.2 BIT

```
import sys
import collections


N,M,K = [int(x) for x in sys.stdin.readline().split() ]
data = [0] * (N+20)

def update(p,x):
    original = query(p,p)
    diff = x - original
    while p <= N:
        data[p] = data[p] + diff
        p = p + (p&-p)

def query(l,r):
    res = 0
    l = l - 1
    while r > 0:
        res = res + data[r]
        r = r - (r&-r)
    while l > 0 :
        res = res - data[l]
        l = l - (l&-l)

    return res

for i in range(1,N+1,1):
    x = int(sys.stdin.readline())
    update(i,x)


for i in range(M+K):
    a,b,c = [int(x) for x in sys.stdin.readline().split() ]
    if a == 1:
        update(b,c)
    else:
        if b > c :
            swap(b,c)
        print(query(b,c))
```

3 String

3.1 KMP

```
import sys

def preprocess(p):
    n = len(p)
    fail = [0] * n
    j = 0
    for i in range(1,n,1):
        while j > 0 and p[i] != p[j] :
            j = fail[j-1]
        if p[i] == p[j]:
            j = j + 1
            fail[i] = j
        else:
            fail[i] = 0
```

```
    return fail

def KMP(s,p):
    n = len(s)
    m = len(p)
    j = 0
    ans = []
    fail = preprocess(p)
    for i in range(n):
        while j > 0 and s[i] != p[j]:
            j = fail[j-1]
        if s[i] == p[j]:
            if j == m-1:
                ans.append(i-m+1)
                j = fail[j]
            else:
                j = j + 1

    return ans

s = str(sys.stdin.readline().strip('\n'))
p = str(sys.stdin.readline().strip('\n'))

ans = KMP(s,p)

print(len(ans))
for x in ans:
    print(x+1,end='')
    print(" ",end='')
```

4 Graph

4.1 DFS(Tree Diameter)

```
import sys
import collections

sys.setrecursionlimit(15000)
Pi = collections.namedtuple('Pi',['Fi','Se'])

def dfs(x):
    vis[x] = 1
    mx = 0
    mxx = x
    for e in E[x]:
        if vis[e.Fi] == 1: continue
        res = dfs(e.Fi)
        if res.Se + e.Se > mx:
            mx = res.Se + e.Se
            mxx = res.Fi
    return Pi(mxx,mx)

N = int(sys.stdin.readline())
E = [[] for i in range(N+10) ]
vis = [0 for i in range(N+10)]
for i in range(N-1):
    a,b,c = [ int(x) for x in sys.stdin.readline().split() ]
    E[a].append(Pi(b,c))
    E[b].append(Pi(a,c))

res = dfs(1)
vis = [0 for i in range (N+10) ]
res = dfs(res.Fi)
print(res.Se)
```

4.2 Bipartite Matching

```
import sys
import collections


N,M = [int(x) for x in sys.stdin.readline().split()]
vis = [0 for i in range(N+10) ]
xy = [0 for i in range(N+10) ]
yx = [0 for i in range(M+10) ]
E = [ [] for i in range(N+2) ]

def dfs(x):
    vis[x] = True
    for e in E[x]:
        if yx[e] == 0 or ( vis[yx[e]] == False and dfs(yx[e]) == 1 ):
            yx[e] = x
            xy[x] = e
            return 1
    return 0


for i in range(1,N+1,1):
    arr = [int(x) for x in sys.stdin.readline().split()]
    n = len(arr)
    for j in range(1,n,1):
        E[i].append(arr[j])

ans = 0
for i in range(1,N+1,1):
    for j in range(N+10):
        vis[j] = 0
    ans = ans + dfs(i)

print(ans)
```

4.3 Dinic

```
import sys
import collections

class Edge:
    def __init__(self,to,inv,cap):
```

```
        self.to = to
        self.inv = inv
        self.cap = cap
        self.flow = 0
    def res(self):
        return self.cap - self.flow

n = 0
E = []
lev = []
work = []

def init(x):
    global E
    global lev
    global work
    n = x+5
    E = [ [] for i in range(n) ]
    lev = [0 for i in range(n) ]
    work = [0 for i in range(n) ]

def make_edge(from1,to,cap):
    forward = Edge(to,len(E[to]),cap)
    backward = Edge(from1,len(E[from1]),0)
    E[from1].append(forward)
    E[to].append(backward)

def bfs(source,sink):
    q = collections.deque()
    for i in range(len(lev)):
        lev[i] = -1
    lev[source] = 0
    q.append(source)
    while len(q) >= 1:
        cur = q.popleft()
        for e in E[cur]:
            if lev[e.to] == -1 and e.res() > 0:
                lev[e.to] = lev[cur] + 1
                q.append(e.to)
    return lev[sink] != -1

def dfs(cur,sink,flow):
    if cur == sink:
        return flow
    while work[cur] < len(E[cur]):
        i = work[cur]
        e = E[cur][i]
        if e.res() == 0 or lev[e.to] != lev[cur]+1:
            work[cur] = work[cur] + 1
            continue
        df = dfs(e.to,sink,min(flow, e.res() ) )
        if df > 0 :
            e.flow += df
            E[e.to][e.inv].flow -= df
            return df
        work[cur] = work[cur] + 1
    return 0

def solve(source, sink):
    ans = 0
    while bfs(source,sink) == True:
        for i in range(len(work)):
            work[i] = 0
        while True:
            flow = dfs(source,sink,99999999999)
            if flow == 0 :
                break
            ans = ans + flow
    return ans

N,M = [ int(x) for x in sys.stdin.readline().split() ]
init(N+M+5)

for i in range(1,N+1,1):
    make_edge(0,i,1)
for i in range(1,M+1,1):
    make_edge(N+i,N+M+1,1)

for i in range(1,N+1,1):
    arr = [ int(x) for x in sys.stdin.readline().split() ]
    for j in range(1,len(arr),1):
        make_edge(i,N+arr[j],1)

print(solve(0,N+M+1))
```

4.4 MCMF

```
import sys
import collections

E = []; pv = []; pe = []; dist = []; inq = [];
INF = 99999999999999

class Edge:
    def __init__(self,to,inv,cap,flow,cost):
        self.to = to
        self.inv = inv
        self.cap = cap
        self.flow = flow
        self.cost = cost
    def res(self):
        return self.cap - self.flow

def make_edge(from1,to,cap,cost):
    forward = Edge(to,len(E[to]),cap,0,cost);
    backward = Edge(from1,len(E[from1]),0,0,-cost)
    E[from1].append(forward)
    E[to].append(backward)

def init(x):
    global E,inv,pv,pe,dist,inq
```

```
    E = [[] for i in range(x+10)]
    pv = [0 for i in range(x+10)]
    pe = [0 for i in range(x+10)]
    dist = [0 for i in range(x+10)]
    inq = [0 for i in range(x+10)]

def solve(source,sink):
    global E,inv,pv,pe,dist,inq
    ans = 0
    totalflow = 0
    totalcost = 0
    while True:
        dist = [INF] * len(dist)
        inq = [0] * len(inq)

        q = collections.deque()
        q.append(source)
        inq[source] = 1
        dist[source] = 0

        # SPFA
        while len(q) >= 1:
            cur = q.popleft()
            inq[cur] = 0
            for i in range(len(E[cur])):
                e = E[cur][i]
                if e.res() > 0 and dist[e.to] > dist[cur] + e.cost:
                    dist[e.to] = dist[cur] + e.cost
                    pv[e.to] = cur
                    pe[e.to] = i
                    if inq[e.to] == 0:
                        inq[e.to] = 1
                        q.append(e.to)

        if dist[sink] == INF:
            break

        # Add this limit when we don't require maxflow
        #if dist[sink] > 0: break

        mnflow = INF
        v = sink
        while v != source:
            mnflow = min(mnflow, E[pv[v]][pe[v]].res())
            v = pv[v]

        v = sink
        while v != source:
            tmp = E[pv[v]][pe[v]].inv
            E[pv[v]][pe[v]].flow += mnflow
            E[v][tmp].flow -= mnflow
            v = pv[v]

        totalflow += mnflow
        totalcost += dist[sink] * mnflow

    return(totalflow,totalcost)


N,M = [int(x) for x in sys.stdin.readline().split()]

init(N+M+10)

arr =  [int(x) for x in sys.stdin.readline().split()]
for i in range(1,N+1,1):
    make_edge(M+i,N+M+1,arr[i-1],0)

arr =  [int(x) for x in sys.stdin.readline().split()]
for i in range(1,M+1,1):
    make_edge(0,i,arr[i-1],0)

for i in range(1,M+1,1):
    arr = [int(x) for x in sys.stdin.readline().split()]
    for j in range(len(arr)):
        make_edge(i,M+j+1,INF,arr[j])


res = solve(0,N+M+1)
print(res[1])
```

4.5 LCA

```
import sys

sys.setrecursionlimit(150000)

N = int(sys.stdin.readline())
E = [[]for i in range(N+5)]
lev = [0 for i in range(N+5)]
pa = [ [-1 for i in range(N+5)] for j in range(18) ]

def dfs(x,p,l):
    pa[0][x] = p
    lev[x] = l
    for e in E[x]:
        if e == p: continue
        dfs(e,x,l+1)

def build_lca():
    for k in range(1,18):
        for n in range(1,N+1,1):
            if pa[k-1][n] == -1:
                pa[k][n] = -1
            else:
                pa[k][n] = pa[k-1][ pa[k-1][n] ]

def lca(a,b):
    if lev[a] < lev[b]:
        a,b = b,a

    diff = lev[a] - lev[b]
```

```
    for k in range(18):
        if ( diff & (1<<k) ) > 0:
            a = pa[k][a]

    if a == b :
        return a

    for k in range(17,-1,-1):
        if pa[k][a] != pa[k][b]:
            a = pa[k][a]
            b = pa[k][b]

    return pa[0][a]

for i in range(N-1):
    a,b = [int(x) for x in sys.stdin.readline().split()]
    E[a].append(b)
    E[b].append(a)

dfs(1,-1,1)
build_lca()

M = int(sys.stdin.readline())
for i in range(M):
    a,b =  [int(x) for x in sys.stdin.readline().split()]
    print(lca(a,b))
```

4.6 Dijkstra

```
import sys
from heapq import *
inf = 9989999999989



N,M = [int(x) for x in sys.stdin.readline().split()]
E = [[] for x in range(N+10)]

start = int(sys.stdin.readline())
for i in range(M):
    a,b,c = [int(x) for x in sys.stdin.readline().split()]
    forward = (b,c)
    E[a].append(forward)


dist = [inf]*20050
dist[start] = 0
pq = []
heappush(pq,(0,start))


while len(pq) > 0 :
    cur = heappop(pq)
    if cur[0] > dist[cur[1]]:
        continue
    for e in E[cur[1]]:
        if dist[e[0]] > cur[0] + e[1]:
            dist[e[0]] = cur[0] + e[1]
            heappush(pq,(dist[e[0]],e[0]))

for i in range(1,N+1,1):
    if dist[i] == inf:
        print("INF")
    else:
        print(dist[i])
```

5 Miscellaneous

5.1 Permutations / Combinations

```
import sys
import itertools

arr = [1,2,3,4]

for x in itertools.permutations(arr):
    print(x)

for x in itertools.permutations(arr,2):
    print(x)

for x in itertools.combinations(arr,2):
    print(x)
```