**Graph DS   
Illuminati**

n = int(input())  
  
d = [[] **for** \_ **in** range(n)]  
res = 0  
**for** i **in** range(n):  
 ref = list(map(int, input().split()))[i+1:]  
 **for** count, val **in** enumerate(ref):  
 **if** val == 1:  
 d[i].append(count+ i+1)  
**for** val **in** d:  
 **for** k **in** range(0,len(val)):  
 compare\_a = set(val[k+1:])  
 compare\_b = set(d[val[k]])  
 res += len(compare\_a & compare\_b)  
print(res)

**Buslines**

**import** sys  
a,b = map(int,input().split())  
max\_val = a \* (a-1) //2  
min\_val = a -1  
**if** b > max\_val **or** b < min\_val:  
 print(-1)  
 sys.exit()  
count = 0  
distinct = set()  
gap = 1  
res = []  
  
**while** count < b **and** gap <= a - 1:  
 **for** i **in** range(1, a + 1):  
 j = i + gap  
 **if** j > a **or** count >=b:  
 **break** sum = i + j  
 **if** sum **not in** distinct:  
 distinct.add(sum)  
 res.append((i,j))  
 count += 1  
 gap += 1  
**if** len(res) < b:  
 print(-1)  
**else**:  
 **for** item **in** res:  
 print(item[0],item[1])

**Graph Traversal**

**Sky Island – connected BFS**

**from** collections **import** deque  
  
N, M = map(int, input().split())  
AL = [[] **for** \_ **in** range(N)]  
**for** \_ **in** range(M):  
 a, b = map(**lambda** x: int(x) - 1, input().split())  
 AL[a].append(b)  
 AL[b].append(a)  
vis = [**False**] \* N  
q = deque([0])  
vis[0] = **True  
  
while** q:  
 u = q.popleft()  
 **for** v **in** AL[u]:  
 **if** vis[v]: **continue** vis[v] = **True** q.append(v)  
all\_connected = **True  
for** u **in** range(N):  
 **if not** vis[u]:  
 print(**'No'**)  
 all\_connected = **False  
 break  
if** all\_connected:  
 print(**"Yes"**)

**Conquest**

**from** heapq **import** heappush, heappop  
pq\_heapq = []  
n, m = map(int, input().split())  
AL = [[] **for** \_ **in** range(n)]  
weight = []  
**for** \_ **in** range(m):  
 u, v = map(int, input().split())  
 u,v = u-1, v-1  
 AL[u].append(v)  
 AL[v].append(u)  
  
**for** \_ **in** range(n):  
 weight.append(int(input()))  
conquered = {0}  
added = set()  
  
curr = weight[0]  
**for** val **in** AL[0]:  
 heappush(pq\_heapq,(weight[val], val))  
 added.add(val)  
  
**while** pq\_heapq:  
 tmp = heappop(pq\_heapq)  
 w, v = tmp[0], tmp[1]  
 **if** w >= curr:  
 **break  
 if** v **in** conquered:  
 **continue** conquered.add(v)  
 curr += w  
 **for** val **in** AL[v]:  
 **if** val **not in** conquered **and** val **not in** added:  
 heappush(pq\_heapq, (weight[val], val))  
  
print(curr)  
**SSSP**

**gezzer script**

**import** sys  
**from** collections **import** deque  
m, n, s = map(int, input().split())  
sys.setrecursionlimit(10000000)  
  
el = deque()  
vis = {s}  
  
q = deque([[s,0]])  
  
**for** \_ **in** range(n):  
 a,b = map(int, input().split())  
 el.append((a,b))  
  
  
**while** q:  
 checker = q.popleft()  
 s1, level = checker[0], checker[1]  
 **for** i,j **in** el:  
 res = (s1 \* i + j) % m  
 **if** res == 0:  
 print(level + 1)  
 sys.exit()  
 **if** res **in** vis:  
 **continue** vis.add(res)  
 q.append([res, level+ 1])  
  
print(-1)

**modulo solitaire**

**from** math **import** inf  
**from** heapq **import** heappush,heappop  
  
A, H = map(int, input().split())  
n,m = map(int, input().split())  
  
AL = [[] **for** \_ **in** range(n)]  
dist = [inf] \* n  
dist[0] = -H  
**for** \_ **in** range(m):  
 u,v,a,h = map(int, input().split())  
 u, v = u-1, v -1  
 AL[u].append((v,a,h))  
pq = [(-H,0)]  
  
w = 0  
**while** pq:  
 H, loc = heappop(pq)  
 **if** H > dist[loc]: **continue  
 for** i, ea, eh **in** AL[loc]:  
 num\_of\_attack = eh // A - 1 **if** eh % A == 0 **else** eh // A  
 w = num\_of\_attack \* ea  
 **if** dist[loc] + w >= dist[i]: **continue** dist[i] = dist[loc] + w  
 heappush(pq, (dist[i], i))  
**if** dist[n-1] >= 0 :  
 print(**"Oh no"**)  
**else**:  
 print(-dist[n-1])

**Single Shortest Path**

**from** heapq **import** heappush, heappop  
**from** math **import** inf  
**while True**:  
 n, m, q,s = map(int, input().split())  
 **if** n + m + q + s == 0:  
 **break** AL = [[] **for** \_ **in** range(n)]  
 **for** \_ **in** range(m):  
 u, v, w = map(int,input().split())  
 AL[u].append((v,w))  
 dist = [inf] \* n  
 dist[s] = 0  
  
 pq = [(0,s)]  
  
 **while** pq:  
 d, u = heappop(pq)  
 **if** d > dist[u]: **continue  
 for** v, w **in** AL[u]:  
 **if** dist[u] + w >= dist[v]: **continue** dist[v] = dist[u] + w  
 heappush(pq, (dist[v],v))  
 **for** \_ **in** range(q):  
 t = int(input())  
 print(**"Impossible" if** dist[t] == inf **else** dist[t])  
 print()

**Cross Country**

**from** math **import** inf  
**from** heapq **import** heappush,heappop  
N, S, T = map(int, input().split())  
D = [list(map(int, input().split())) **for** \_ **in** range(N)]  
  
dist = [inf] \* N  
dist[S] = 0  
pq = [(dist[S],S)]  
  
**while** pq:  
 d, u = heappop(pq)  
 **if** D[u][v] == 0: **continue  
 for** v **in** range(N):  
 **if** d > dist[u]: **continue  
 if** dist[u] + D[u][v] >= dist[v]: **continue** dist[v] = dist[u] + D[u][v]  
 heappush(pq, (dist[v],v))  
print(dist[T])

#brexit

**from** collections **import** deque  
  
C,P,X,L = map(int, input().split())  
X, L = X-1, L-1  
  
AL = [[] **for** \_ **in** range(C)]  
  
deg = [0] \* C  
  
**for** \_ **in** range(P):  
 A, B = map(**lambda** x: int(x) -1 , input().split())  
 AL[A].append(B)  
 deg[B] += 1  
 AL[B].append(A)  
 deg[A] += 1  
  
ori\_deg = [deg[i] **for** i **in** range(C)]  
  
q = deque([L])  
leave = [**False**] \* C  
**while** q:  
 u= q.popleft()  
 leave[u] = **True  
 for** v **in** AL[u]:  
 **if** leave[v]: **continue** deg[v] -= 1  
 **if** 2 \* deg[v] <= ori\_deg[v]:  
 q.append(v)  
print(**"stay" if not** leave[X] **else 'leave'**)

*#shopping list*n,m = map(int, input().split())  
res = set(input().split())  
**for** \_ **in** range(1,n):  
 res = res & set(input().split())  
print(len(res))  
**for** item **in** sorted(list(res)):  
 print(item)  
  
*#flying safely***for** \_ **in** range(int(input())):  
 n, m = map(int,input().split())  
 **for** \_ **in** range(m):  
 input()  
 print(n-1)  
*#where my internet connected***from** collections **import** deque  
N, M = map(int, input().split())  
AL = [[] **for** \_ **in** range(N)]  
**for** \_ **in** range(M):  
 a, b = map(**lambda** x: int(x) - 1, input().split())  
 AL[a].append(b)  
 AL[b].append(a)  
vis = [**False**] \* N  
q = deque([0])  
vis[0] = **True  
  
while** q:  
 u = q.popleft()  
 **for** v **in** AL[u]:  
 **if** vis[v]: **continue** vis[v] = **True** q.append(v)  
all\_connected = **True  
for** u **in** range(N):  
 **if not** vis[u]:  
 print(u + 1)  
 all\_connected = **False  
if** all\_connected:  
 print(**"Connected"**)  
*# popularity contest -degree*N, M = map(int, input().split())  
degree = [0] \* (N+1)  
**for** \_ **in** range(M):  
 a, b = map(int, input().split())  
 degree[a] += 1  
 degree[b] += 1  
**for** i **in** range(1, N+1):  
 print(degree[i] -1 , end = **' '**)  
*# reachable road***def** dfs(u):  
 vis[u] = **True  
 for** v **in** AL[u]:  
 **if** vis[v]: **continue** dfs(v)  
**for** \_ **in** range(int(input())) :  
 V = int(input())  
 E = int(input())  
 AL = [[] **for** \_ **in** range(V)]  
 **for** \_ **in** range(E):  
 u,v = map(int, input().split())  
 AL[u].append(v)  
 AL[v].append(u)  
  
 numCC = 0  
 vis = [**False**] \* V  
 **for** u **in** range(V):  
 **if not** vis[u]:  
 numCC += 1  
 dfs(u)  
 print(numCC -1)  
*#amoeba*dr = [0, 1, 1, 1, 0, -1, -1, -1] *# E/SE/S/SW/W/NW/N/NE*dc = [1, 1, 0, -1, -1, -1, 0, 1]  
  
  
**def** dfs(r, c):  
 grid[r][c] = **'.'  
 for** d **in** range(8):  
 nr, nc = r + dr[d], c + dc[d]  
 **if** nr < 0 **or** nr >= m **or** nc < 0 **or** nc >= n: **continue  
 if** grid[nr][nc] != **'#'**: **continue** dfs(nr, nc)  
  
  
m, n = map(int, input().split())  
grid = [list(input()) **for** \_ **in** range(m)]  
numCC = 0  
**for** row **in** range(m):  
 **for** col **in** range(n):  
 **if** grid[row][col] == **'#'**:  
 numCC += 1  
 dfs(row, col)  
print(numCC)

*#successful zoom*n = int(input())  
lst = [int(i) **for** i **in** input().split()]  
  
**def** checking(n, lst):  
 maxj = n // 2  
 j = 1  
 **while** j <= maxj:  
 check = **True  
 for** i **in** range(j -1,n - j, j):  
 **if** lst[i + j] <= lst[i]:  
 check = **False  
 break  
 if** check == **True**:  
 **return** j  
 j += 1  
  
 **return 'ABORT!'**print(checking(n, lst))  
  
*#fluortanten*n=int(input())  
lst = [int(i) **for** i **in** input().split()]  
sum = 0  
res = []  
**for** i **in** range(n):  
 inp = lst[i]  
 **if** inp != 0:  
 res.append(inp)  
 sum += inp \* len(res)  
largest = 0  
curr = 0  
**for** i **in** range(n-2,-1,-1):  
 curr += res[i]  
 **if** curr > largest:  
 largest = curr  
print(sum + largest)  
  
*#card sorting***import** bisect  
**def** merge\_sort(lst):  
 n = len(lst)  
 **if** n == 1:  
 **return** lst  
  
 mid = n//2  
 l = lst[:mid]  
 r = lst[mid:]  
 merge\_sort(l)  
 merge\_sort(r)  
  
 a = b = c = 0  
 **while** a < len(l) **and** b < len(r):  
 **if** l[a] <= r[b]:  
 lst[c] = l[a]  
 a += 1  
 **else**:  
 lst[c] = r[b]  
 b += 1  
 c += 1  
 **while** a < len(l):  
 lst[c] = l[a]  
 c += 1  
 a += 1  
 **while** b < len(r):  
 lst[c] = r[b]  
 c += 1  
 b += 1  
  
n = int(input())  
inp = list(map(int,input().split()))  
merge\_sort(inp)  
  
q = int(input())  
**for** \_ **in** range(q):  
 l, h = map(int, input().split())  
 print(bisect.bisect\_right(inp,h)-bisect.bisect\_left(inp,l))  
  
*# path cross*p, n = map(int, input().split())  
lst = []  
  
**for** \_ **in** range(n):  
 obj = [int(i) **for** i **in** input().split()]  
 lst.append(obj)  
  
merge\_sort\_key(lst, 3)  
  
out = set()  
  
**for** i **in** range(n - 1):  
 timing = lst[i][3]  
 j = i + 1  
 **while** j < min(i + 5, n) **and** lst[j][3] <= timing + 10:  
 res\_val = new\_res(lst[i][0], lst[j][0])  
 **if** lst[i][0] != lst[j][0] **and** res\_val **not in** out:  
 **if** get\_distance([lst[i][1], lst[i][2]], [lst[j][1], lst[j][2]]):  
 out.add(res\_val)  
 j += 1  
  
  
print(len(out))  
**if** len(out) != 0:  
 res = merge\_sort\_key(list(out), **None**)  
 **for** item **in** res:  
 print(item[0], item[1])  
*#bracket***from** collections **import** deque  
**def** match\_bracket(n, pattern):  
 open = {**"{"**, **"["**, **"("**}  
 close = {**"}"**, **"]"**, **")"**}  
 match = {**"}"**: **"{"**, **"]"**: **"["**, **")"**: **"("**}  
 invalid = **"Invalid"** valid = **"Valid"** res = []  
 pattern = deque(pattern)  
 **for** item **in** pattern:  
 **if** item **in** open:  
 res.append(item)  
 **elif** item **in** close:  
 matching = match[item]  
 **if not** res:  
 **return** invalid  
 **if** res.pop() != matching:  
 **return** invalid  
 **return** valid **if** res == [] **else** invalid  
  
  
n = int(input())  
p = input()  
print(match\_bracket(n, p))  
  
*#cogaline***class** Vertex:  
 **def** \_\_init\_\_(self, data):  
 self.item = data  
 self.next = **None** self.prev = **None** self.partner = **None  
  
  
class** DLL:  
 **def** \_\_init\_\_(self):  
 self.\_item = **None** self.\_head = **None** self.\_tail = **None  
  
 def** append\_pair(self,a,b):  
 a = Vertex(a)  
 b = Vertex(b)  
 a.partner = b  
 b.partner = a  
 self.append(a)  
 self.append(b)  
  
 **def** append(self, vtx):  
 *#vtx = Vertex(v)* **if** self.\_head **is None**:  
 self.\_head = vtx  
 self.\_tail = self.\_head  
 self.\_item = self.\_head  
 **else**:  
 self.\_tail.next = vtx  
 vtx.prev = self.\_tail  
 vtx.next = **None** self.\_tail = vtx  
  
 **def** traverse\_f(self):  
 **if** self.\_item **is not None and** self.\_item.prev **is not None**:  
 self.\_item = self.\_item.prev  
  
 **def** traverse\_b(self):  
 **if** self.\_item **is not None and** self.\_item.next **is not None**:  
 self.\_item = self.\_item.next  
  
 **def** remove\_curr\_node(self):  
 tmp = self.\_item  
 flag = **True  
 if** tmp == self.\_head:  
 self.\_head = tmp.next  
 self.\_head.prev = **None** self.\_item = self.\_head  
 **elif** self.\_item != self.\_tail:  
 tmp.prev.next = tmp.next  
 tmp.next.prev = tmp.prev  
 self.\_item = tmp.next  
 **else**:  
 self.\_item = self.\_head  
 flag = **False  
 return** tmp, flag  
  
 **def** traverse\_r(self):  
 **if** self.\_item **is None**:  
 self.\_item = self.\_head  
 node, flag = self.remove\_curr\_node()  
 **if** flag:  
 self.append(node)  
  
 **def** traverse\_c(self):  
 **if** self.\_item **is None**:  
 self.\_item = self.\_head  
 **if** self.\_item.prev **is not None and** self.\_item.prev == self.\_item.partner:  
 self.\_item = self.\_item.next **if** self.\_item **is not** self.\_tail **else** self.\_head  
 **else**:  
 curr\_ele, flag = self.remove\_curr\_node()  
 **if** curr\_ele.partner == self.\_tail:  
 self.append(curr\_ele)  
 **else**:  
 **if not** flag:  
 self.\_tail = self.\_tail.prev  
 self.\_tail.next = **None** curr\_ele.next = curr\_ele.partner.next  
 curr\_ele.next.prev = curr\_ele  
 curr\_ele.prev = curr\_ele.partner  
 curr\_ele.partner.next = curr\_ele  
  
 **def** traverse\_print(self):  
 current\_ele = self.\_head  
 **while** current\_ele:  
 print(current\_ele.item)  
 current\_ele = current\_ele.next  
  
 **def** get\_curr(self):  
 **if** self.\_item **is None**:  
 self.\_item = self.\_head  
 **return** self.\_item  
  
  
n, instr = map(int, input().split())  
l = DLL()  
**for** \_ **in** range(n):  
 a, b = input().split()  
 l.append\_pair(a,b)  
instr\_set = list(input())  
**for** item **in** instr\_set:  
 **if** item == **'P'**:  
 print(l.get\_curr().partner.item)  
 **if** item == **'F'**:  
 l.traverse\_f()  
 **if** item == **'B'**:  
 l.traverse\_b()  
 **if** item == **'C'**:  
 l.traverse\_c()  
 **if** item == **'R'**:  
 l.traverse\_r()  
  
print()  
l.traverse\_print()  
  
*#array smoothen***from** heapq **import** heappop,heappush  
**from** collections **import** Counter  
**import** sys  
n, k = map(int, input().split())  
arr = sorted([int(i) **for** i **in** input().split()])  
freq = dict(Counter(arr))  
pq\_ls = sorted((v, k) **for** k, v **in** freq.items())  
pq = []  
**if** n == k:  
 print(0)  
 sys.exit()  
**if** k == 0:  
 print(pq\_ls.pop()[0])  
 sys.exit()  
tmp = pq\_ls.pop()  
**if** tmp[1] == 1:  
 print(1)  
**else**:  
 heappush(pq, (-tmp[0], tmp[1]))  
  
checker = ()  
**while** k > 0:  
 tmp\_head = pq[0][0]  
 **if not** pq\_ls **and** tmp\_head == -1:  
 print(1)  
 sys.exit()  
 **if not** checker **and** pq\_ls:  
 checker = pq\_ls.pop()  
 **if** checker **and** -checker[0] <= tmp\_head:  
 heappush(pq,(-checker[0] + 1, checker[1]))  
 checker = ()  
 **else**:  
 addi = heappop(pq)  
 heappush(pq, (addi[0] + 1, addi[1]))  
 k -= 1  
  
print(-pq[0][0])  
  
*#convoy***from** heapq **import** heapify, heappop, heappush  
n, k = map(int, input().split())  
ls = []  
**for** \_ **in** range(n):  
 ls.append(int(input()))  
ls.sort()  
pq = []  
d = {}  
  
**for** i **in** range(min(n,k)):  
 heappush(pq,(ls[i],i))  
  
  
people\_left, result = n, 0  
**while** people\_left > 0:  
 tmp = heappop(pq)  
 result = tmp[0]  
 car = tmp[1]  
 **if** d.get(car,0) == 0:  
 people\_left -= 5  
 **else**:  
 people\_left -= 4  
 d[car] = d.get(car, 0) + 1  
 heappush(pq, (ls[car] + d[car] \* 2 \* ls[car], car))  
  
print(result)  
*#kapoleb***from** collections **import** defaultdict  
l, k, s = map(int, input().split())  
d = defaultdict(**lambda**: [])  
  
**for** \_ **in** range(l):  
 inp = input()  
 i, t = int(inp.split()[0]), inp.split()[1]  
 t = t.split(**'.'**)  
 t = int(t[0]) \* 60 + int(t[1])  
 d[i].append(t)  
d\_new = {key:sum(value) **for** (key,value) **in** d.items() **if** len(value) == k}  
**for** w **in** sorted(d\_new.items(), key=**lambda** x: (x[1],x[0])):  
 print(w[0])  
*#quickscope***from** collections **import** defaultdict  
n = int(input())  
pointer\_dic = defaultdict(**lambda**: [])  
  
big\_d = [{}]  
scope\_ind = 0  
res = []  
  
**def** declare(inp):  
 **if** inp[1] **in** big\_d[-1]:  
 **return 'MULTIPLE DECLARATION'  
 else**:  
 pointer\_dic[inp[1]].append(scope\_ind)  
 big\_d[-1][inp[1]] = inp[2]  
  
  
**def** typeof(inp):  
 **if** inp[1] **in** pointer\_dic :  
 **return** big\_d[pointer\_dic[inp[1]][-1]][inp[1]]  
 **else**:  
 **return 'UNDECLARED'  
  
def** close():  
 rem = big\_d.pop()  
 **for** item **in** rem:  
 pointer\_dic[item].pop()  
 **if not** pointer\_dic[item]:  
 **del** pointer\_dic[item]  
  
  
  
**for** \_ **in** range(n):  
 inp = input().split()  
 **if** inp[0] == **'DECLARE'**:  
 tmp = declare(inp)  
 **if** tmp == **'MULTIPLE DECLARATION'**:  
 res.append(tmp)  
 print()  
 **break  
 elif** inp[0] == **'TYPEOF'**:  
 res.append(typeof(inp))  
 **elif** inp[0] == **'{'**:  
 scope\_ind = len(big\_d)  
 big\_d.append({})  
 **else**:  
 scope\_ind -= 1  
 close()  
**for** item **in** res:  
 print(item)  
  
*#no thanks*n = int(input())  
deck = sorted(list(map(int, input().split())))  
score = deck[0]  
*#mjehuric*n = 5  
a = list(map(int, input().split()))  
  
**for** i **in** range(n):  
 **for** j **in** range(n -1):  
 **if** a[j] > a[j +1]:  
 a[j], a[j +1] = a[j+1], a[j]  
 print(\*a)  
**for** i **in** range(n-1):  
 **if** deck[i] +1 != deck[i+1]:  
 score += deck[i+1]  
print(score)  
*#height***for** \_ **in** range(int(input())):  
 line = list(map(int, input().split()))  
 k, h = line[0], line[1:]  
 ans = 0  
 n = 20  
 **for** i **in** range(1,n):  
 x = h[i]  
 j = i - 1  
 **while** (j >= 0) **and** (h[j] > x):  
 h[j+1] = h[j]  
 j -= 1  
 ans += 1  
 h[j+1] = x  
 print(k,ans)  
*#sort of sorting***while True**:  
 n = int(input())  
 **if** n == 0: **break** names = [input() **for** \_ **in** range(n)]  
 print()  
 print(\*sorted(names, key = **lambda** x: x[:2]), sep =**"\n"**)  
*#knight***import** sys  
**from** heapq **import** heappop, heappush, heapify  
k, n = map(int, input().split())  
  
  
data = [list(map(int,input().split())) **for** \_ **in** range(n+ k -1)]  
karl\_strength = data[0][1]  
karl\_year = data[0][0]  
data.sort()  
max\_pq = [-data[i][1] **for** i **in** range(k)]  
heapify(max\_pq)  
  
**for** year **in** range(2011, 2011 + n):  
 **if** year > 2011:  
 heappush(max\_pq, - data[year -2011 + k -1][1])  
 **if** -max\_pq[0] == karl\_strength:  
 print(year)  
 sys.exit()  
 *#break* heappop(max\_pq)  
print(**"Unknown"**)