Олимпиада «Ломоносов» по информатике
2024-2025 учебный год. Заключительный этап
Работа участника с id заявки 1361789, логином inf25f_102

Сводный итог по всем задачам в проверяющей системе

RunID	Time	Username	Prob	Lang	Result	Tests	Score
107	1:59:49	inf25f_102	2	python3	OK	28	100
102	1:56:27	inf25f_102	1	python3	Partial solution	11	28
98	1:49:03	inf25f_102	5	python3	OK	11	100
91	1:39:30	inf25f_102	4	python3	Partial solution	0	0
73	1:24:26	inf25f_102	3	python3	OK	23	100

328 технических баллов
94 итоговых балла

Посылка по задаче 1

```
[1] from collections import deque
[2]
[3] n1 = int(input())
[4] n2 = int(input())
[5]
[6] if n1 == n2:
[7]     print(0)
[8] else:
[9]     q = deque([(n1, 0)])
[10]    vis = {n1}
[11]    found = False
[12]
[13]    while q:
[14]        x, d = q.popleft()
[15]        nxt = x // 2 if x % 2 == 0 else 3 * x + 1
[16]        candidates = [nxt, 2 * x]
[17]        if (x - 1) % 3 == 0:
[18]            cand = (x - 1) // 3
[19]            if cand > 0 and cand % 2 == 1:
[20]                candidates.append(cand)
[21]        for y in candidates:
[22]            if y == n2:
[23]                print(d + 1)
[24]                found = True
[25]                break
[26]            if y not in vis:
[27]                vis.add(y)
[28]                q.append((y, d + 1))
[29]        if found:
[30]            break
[31]
[32]    if not found:
[33]        print(-1)
```

Посылка по задаче 2

```
[1] def parse_modified_number(s: str) -> int:
[2]     i = 0
[3]     result = 0
[4]     base = 26
[5]     while i < len(s):
[6]         ch = s[i]
[7]         if i + 1 < len(s) and s[i+1] in "^~_":
[8]             modifier = s[i+1]
[9]             base_val = ord(ch) - ord('a')
[10]            if modifier == '^':
[11]                digit = base_val + 26
[12]            elif modifier == '~':
[13]                digit = base_val + 52
[14]            elif modifier == '_':
[15]                digit = base_val + 78
[16]            i += 2
[17]        else:
[18]            digit = ord(ch) - ord('a')
[19]            i += 1
[20]        result = result * base + digit
[21]    return result
[22]
[23] n = int(input())
[24] numbers = [parse_modified_number(input().strip()) for _ in range(n)]
[25] sorted_numbers = sorted(numbers)
[26] diff_idx = []
[27] for i in range(n):
[28]     if numbers[i] != sorted_numbers[i]:
[29]         diff_idx.append(i + 1)
[30] print(diff_idx[0], diff_idx[1])
```

Посылка по задаче 3

```
[1] N, T_lim = map(int, input().split())
[2] pts = list(map(int, input().split()))
[3] G = [list(map(int, input().split())) for _ in range(N)]
[4] INF = 10**18
[5] d = [row[:] for row in G]
[6] for k in range(N):
[7]     for i in range(N):
[8]         for j in range(N):
[9]             if d[i][j] > d[i][k] + d[k][j]:
[10]                d[i][j] = d[i][k] + d[k][j]
[11] def tsp(mask):
[12]     size = 1 << N
[13]     dp = [[INF] * N for _ in range(size)]
[14]     dp[1][0] = 0
[15]     for state in range(size):
[16]         if state & mask != state:
[17]             continue
[18]         if not (state & 1):
[19]             continue
[20]         for j in range(N):
[21]             if (state >> j) & 1 and dp[state][j] < INF:
[22]                 for k in range(N):
[23]                     if (mask >> k) & 1 == 0:
[24]                         continue
[25]                     if (state >> k) & 1:
[26]                         continue
[27]                     ns = state | (1 << k)
[28]                     if dp[ns][k] > dp[state][j] + d[j][k]:
[29]                         dp[ns][k] = dp[state][j] + d[j][k]
[30]     best = INF
[31]     for j in range(N):
[32]         if (mask >> j) & 1:
[33]             best = min(best, dp[mask][j] + d[j][0])
[34]     return best
[35] best_score = -1
[36] candidates = []
[37] for mask in range(1, 1 << N):
[38]     if mask & 1 == 0:
[39]         continue
[40]     cost = tsp(mask)
[41]     if cost <= T_lim:
[42]         score = sum(pts[i] for i in range(N) if (mask >> i) & 1)
[43]         cnt = bin(mask).count("1")
[44]         if score > best_score:
[45]             best_score = score
[46]             candidates = [(mask, cnt)]
[47]         elif score == best_score:
[48]             candidates.append((mask, cnt))
[49] min_cnt = min(cnt for _, cnt in candidates)
[50] filtered = [mask for mask, cnt in candidates if cnt == min_cnt]
[51] def mask_to_tuple(mask):
[52]     return tuple(i + 1 for i in range(N) if (mask >> i) & 1)
[53] filtered.sort(key=mask_to_tuple)
[54] ans_mask = filtered[0]
[55] ans_list = [i + 1 for i in range(N) if (ans_mask >> i) & 1]
[56] print(len(ans_list))
[57] print(" ".join(map(str, ans_list)))
```

Посылка по задаче 4

```
[1]  import sys
[2]
[3]  def parse_atom(s, i):
[4]      r = s[i]
[5]      i += 1
[6]      if i < len(s) and s[i].islower():
[7]          while i < len(s) and s[i].islower():
[8]              r += s[i]
[9]              i += 1
[10]     return r, i
[11]
[12] def parse_num(s, i):
[13]     if i < len(s) and s[i].isdigit():
[14]         start = i
[15]         while i < len(s) and s[i].isdigit():
[16]             i += 1
[17]         return int(s[start:i]), i
[18]     return 1, i
[19]
[20] def parse_group(s, i):
[21]     stack = []
[22]     while i < len(s):
[23]         c = s[i]
[24]         if c in '([':
[25]             i, subres = parse_bracket(s, i)
[26]             stack.append(subres)
[27]         elif c in ')]':
[28]             break
[29]         elif c.isalpha():
[30]             atom, i = parse_atom(s, i)
[31]             num, i = parse_num(s, i)
[32]             stack.append({atom: num})
[33]         else:
[34]             break
[35]     res = {}
[36]     for dic in stack:
[37]         for k, v in dic.items():
[38]             res[k] = res.get(k, 0) + v
[39]     return i, res
[40]
[41] def parse_bracket(s, i):
[42]     open_char = s[i]
[43]     close_char = ')' if open_char == '(' else ']'
[44]     i += 1
[45]     i, inside = parse_group(s, i)
[46]     if i < len(s) and s[i] == close_char:
[47]         i += 1
[48]     mul, i = parse_num(s, i)
[49]     for k in inside:
[50]         inside[k] *= mul
[51]     return i, inside
[52]
[53] def parse_molecule(mol):
[54]     i = 0
[55]     res = {}
[56]     while i < len(mol):
[57]         if mol[i] in '([':
[58]             i, dic = parse_bracket(mol, i)
[59]             for k, v in dic.items():
[60]                 res[k] = res.get(k, 0) + v
[61]         else:
[62]             if mol[i].isalpha():
[63]                 atom, i = parse_atom(mol, i)
[64]                 num, i = parse_num(mol, i)
[65]                 res[atom] = res.get(atom, 0) + num
[66]             else:
[67]                 i += 1
[68]     return res
[69]
[70] def parse_side(side):
[71]     parts = side.split('+')
[72]     order = []
[73]     seen = set()
[74]     for p in parts:
[75]         p = p.strip()
[76]         k = 0
```

```
        while k < len(p) and p[k].isdigit():
            k += 1
        coef_str = p[:k]
        if coef_str == '':
            coef = 1
        else:
            coef = int(coef_str)
        mol_str = p[k:]
        mol_str = mol_str.strip()
        if mol_str not in seen:
            seen.add(mol_str)
            order.append(mol_str)
    return order

def build_compositions(molecules):
    comps = []
    for m in molecules:
        comps.append(parse_molecule(m))
    return comps

def all_atoms(left_comps, right_comps):
    s = set()
    for c in left_comps:
        for a in c:
            s.add(a)
    for c in right_comps:
        for a in c:
            s.add(a)
    return sorted(s)

def add_missing_atoms(left_mols, left_comps, right_mols, right_comps):
    at_left = set()
    for c in left_comps:
        for a in c:
            at_left.add(a)
    at_right = set()
    for c in right_comps:
        for a in c:
            at_right.add(a)
    all_at = at_left.union(at_right)
    need_left = sorted(list(all_at - at_left))
    need_right = sorted(list(all_at - at_right))
    for a in need_left:
        left_mols.append(a)
        left_comps.append({a:1})
    for a in need_right:
        right_mols.append(a)
        right_comps.append({a:1})

def build_matrix(left_comps, right_comps):
    atoms = []
    at_map = {}
    idx = 0
    for c in left_comps+right_comps:
        for a in c:
            if a not in at_map:
                at_map[a] = idx
                atoms.append(a)
                idx += 1
    n = len(atoms)
    m = len(left_comps) + len(right_comps)
    mat = [[0]*m for _ in range(n)]
    for j,c in enumerate(left_comps):
        for a,v in c.items():
            i = at_map[a]
            mat[i][j] = v
    for j,c in enumerate(right_comps, start=len(left_comps)):
        for a,v in c.items():
            i = at_map[a]
            mat[i][j] = -v
    return mat, atoms

def gcd(a, b):
    while b:
        a, b = b, a % b
    return abs(a)
```

```
[153]
[154] def lcm(a, b):
[155]     return a*b//gcd(a,b)
[156]
[157] def mat_rref_fraction(mat):
[158]     import math
[159]     r = len(mat)
[160]     c = len(mat[0]) if mat else 0
[161]     M = [[(mat[i][j],1) for j in range(c)] for i in range(r)]
[162]     row = 0
[163]     for col in range(c):
[164]         pivot = -1
[165]         for i in range(row, r):
[166]             if M[i][col] != (0,1):
[167]                 pivot = i
[168]                 break
[169]         if pivot < 0:
[170]             continue
[171]         if pivot != row:
[172]             M[row], M[pivot] = M[pivot], M[row]
[173]         pv = M[row][col]
[174]         M[row] = [ frac_div(x, pv) for x in M[row] ]
[175]         for i in range(r):
[176]             if i != row:
[177]                 factor = M[i][col]
[178]                 if factor != (0,1):
[179]                     sub = [ frac_sub(M[i][k], frac_mul(factor, M[row][k])) for k in range(c) ]
[180]                     M[i] = sub
[181]         row += 1
[182]         if row == r:
[183]             break
[184]     return M
[185]
[186] def frac_add(a, b):
[187]     return (a[0]*b[1]+b[0]*a[1], a[1]*b[1])
[188]
[189] def frac_sub(a, b):
[190]     return (a[0]*b[1]-b[0]*a[1], a[1]*b[1])
[191]
[192] def frac_mul(a, b):
[193]     return (a[0]*b[0], a[1]*b[1])
[194]
[195] def frac_div(a, b):
[196]     return (a[0]*b[1], a[1]*b[0])
[197]
[198] def frac_simplify(a):
[199]     g = gcd(a[0], a[1])
[200]     num = a[0]//g
[201]     den = a[1]//g
[202]     if den<0:
[203]         num=-num
[204]         den=-den
[205]     return (num, den)
[206]
[207] def get_nullspace_one_solution(mat):
[208]     import math
[209]     M = mat_rref_fraction(mat)
[210]     r = len(M)
[211]     c = len(M[0]) if r>0 else 0
[212]     pivot_positions = []
[213]     rowpivot = 0
[214]     for i in range(r):
[215]         pivot_col = -1
[216]         for j in range(c):
[217]             if M[i][j] != (0,1):
[218]                 pivot_col = j
[219]                 break
[220]         if pivot_col>=0:
[221]             pivot_positions.append((i, pivot_col))
[222]     pivot_cols = [pc for (rp, pc) in pivot_positions]
[223]     free_cols = [j for j in range(c) if j not in pivot_cols]
[224]     if len(free_cols)==0:
[225]         return None
[226]     sol = [(0,1)]*c
[227]     fc = free_cols[0]
[228]     sol[fc] = (1,1)
```

```
    for (i, pc) in pivot_positions:
        s = (0,1)
        for j in free_cols:
            val = M[i][j]
            if val != (0,1) and j==fc:
                s = frac_sub(s, val)
        sol[pc] = s
    dens = []
    for x in sol:
        if x[0]!=0:
            dens.append(x[1])
    if not dens:
        return None
    from functools import reduce
    dd = 1
    for d in dens:
        dd = lcm(dd, d)
    i_sol = [ (x[0]*dd)//x[1] for x in sol ]
    if all(v==0 for v in i_sol):
        return None
    if all(v<=0 for v in i_sol):
        i_sol = [-v for v in i_sol]
    if not any(v>0 for v in i_sol):
        return None
    return i_sol

def gcd_list(lst):
    g = 0
    for x in lst:
        g = gcd(g, x)
    return g

for line in sys.stdin:
    line=line.strip()
    if not line:
        continue
    left,right = line.split('=')
    left_mols = parse_side(left)
    right_mols = parse_side(right)
    left_comps = build_compositions(left_mols)
    right_comps = build_compositions(right_mols)
    add_missing_atoms(left_mols, left_comps, right_mols, right_comps)
    mat, atoms = build_matrix(left_comps, right_comps)
    sol = get_nullspace_one_solution(mat)
    if not sol:
        sol = [1]*(len(left_comps)+len(right_comps))
    mn = min(sol)
    if mn<=0:
        shift = 1 - mn
        sol = [s+shift for s in sol]
    if any(s<=0 for s in sol):
        sol = [1]*(len(left_comps)+len(right_comps))
    g = gcd_list(sol)
    if g!=0:
        sol = [s//g for s in sol]
    left_coefs = sol[:len(left_comps)]
    right_coefs = sol[len(left_comps):]
    out_left = []
    for cf, mol in zip(left_coefs, left_mols):
        if cf==1:
            out_left.append(mol)
        else:
            out_left.append(str(cf)+mol)
    out_right = []
    for cf, mol in zip(right_coefs, right_mols):
        if cf==1:
            out_right.append(mol)
        else:
            out_right.append(str(cf)+mol)
    print('+'.join(out_left)+'='+'+'.join(out_right))
```

Посылка по задаче 5

```
[1] M, N, K = map(int, input().split())
[2] children = [[] for _ in range(N+1)]
[3] parent = [0] * (N+1)
[4] depth = [0] * (N+1)
[5] mutation = [0] * (N+1)
[6] for _ in range(N-1):
[7]     s, d, b = map(int, input().split())
[8]     children[s].append((d, b))
[9]     parent[d] = s
[10] q = [1]
[11] depth[1] = 0
[12] while q:
[13]     u = q.pop()
[14]     for v, b in children[u]:
[15]         depth[v] = depth[u] + 1
[16]         mutation[v] = b
[17]         q.append(v)
[18] maxK = (N).bit_length()
[19] up = [[0] * (N+1) for _ in range(maxK)]
[20] for v in range(1, N+1):
[21]     up[0][v] = parent[v]
[22] for k in range(1, maxK):
[23]     for v in range(1, N+1):
[24]         up[k][v] = up[k-1][up[k-1][v]]
[25] def lca(u, v):
[26]     if depth[u] < depth[v]:
[27]         u, v = v, u
[28]     d = depth[u] - depth[v]
[29]     k = 0
[30]     while d:
[31]         if d & 1:
[32]             u = up[k][u]
[33]         d //= 2
[34]         k += 1
[35]     if u == v:
[36]         return u
[37]     for k in range(maxK-1, -1, -1):
[38]         if up[k][u] != up[k][v]:
[39]             u = up[k][u]
[40]             v = up[k][v]
[41]     return parent[u]
[42] results = []
[43] for _ in range(K):
[44]     a, b = map(int, input().split())
[45]     L = lca(a, b)
[46]     temp = []
[47]     u = a
[48]     while u != L:
[49]         temp.append(mutation[u])
[50]         u = parent[u]
[51]     v = b
[52]     while v != L:
[53]         temp.append(mutation[v])
[54]         v = parent[v]
[55]     if not temp:
[56]         results.append(M)
[57]     else:
[58]         arr = sorted(set(temp))
[59]         best = arr[0]
[60]         if M - 1 - arr[-1] > best:
[61]             best = M - 1 - arr[-1]
[62]         for i in range(len(arr)-1):
[63]             gap = arr[i+1] - arr[i] - 1
[64]             if gap > best:
[65]                 best = gap
[66]         results.append(best)
[67] for res in results:
[68]     print(res)
[69]
```