Multiplicación de matrices

Dadas n matrices $M_1, M_2, \ldots, M_n$. La matriz M_i tiene n_i filas y m_i columnas, además se cumple que $m_i = n_{i+1}$. Queremos multiplicar las n matrices $M_1 \cdot M_2 \cdot \ \ldots \cdot M_n$.

Calcular el resultado de $A \cdot B$ donde A y B son matrices, tenemos que hacer $n_a \cdot m_a \cdot m_b$ operaciones, tened en cuenta que para multiplicar $A \cdot B$ necesitamos $m_a \cdot n_b$. Una vez multiplicamos $A \cdot B \cdot C$ tenemos $n_c = n_a$ y $m_c = m_b$.

Si podemos escoger el orden en el que multiplicamos $M_1 \cdot M_2 \cdot \ \ldots \cdot M_n$, ¿cuál es el mínimo número de operaciones que tenemos que hacer?

Dadas M_1, M_2, M_3, M_4, M_5 un orden podría ser: $((M_1 \cdot (M_2 \cdot M_3)) \cdot (M_4 \cdot M_5))$. Aquí multiplicamos:

$M_2 \cdot M_3 = M_{2,3}$ con $n_2 \cdot m_2 \cdot m_3$ operaciones.

$M_1 \cdot M_{2,3} = M_{1,2,3}$ con $n_1 \cdot m_1 \cdot m_3$ operaciones.

$M_4 \cdot M_5 = M_{4,5}$ con $n_4 \cdot m_4 \cdot m_5$ operaciones.

$M_{1,2,3} \cdot M_{4,5} = M_{1,2,3,4,5}$ con $n_1 \cdot m_3 \cdot m_5$ operaciones.

Manuales que pueden ayudar: Dinámica 1 y Dinámica 2.

Estrategia de la solución

Este es el único problema donde no se daba un enlace a una solución completa. Para solucionar este problema necesitábamos la técnica de programación dinámica. Este es un problema muy típico que puede ser encontrado con el nombre estándar de "*Matrix chain multiplication*".

Solución en Python

```
INF = 10**9
t = int(input())
for test in range(t):
    N = int(input())
    n = []
    m = []
    for i in range(N):
        words = input().split()
        n.append(int(words[0]))
        m.append(int(words[1]))
    dp = []
    for i in range(N):
        dp.append([])
        for j in range(N):
```

```
            dp[i].append(INF)
    for i in range(N):
        dp[i][i] = 0
    for k in range(2, N+1):
        for i in range(0, N-k+1):
            for j in range(i+1, i+k):
                v = dp[i][j-1] + dp[j][i+k-1] + n[i]*m[j-1]*m[i+k-1]
                if v < dp[i][i+k-1]:
                    dp[i][i+k-1] = v
    print(dp[0][N-1])
```

Autor de la solución: Izan Beltrán (Miembro del comité organizador)

Solución en C++

```
#include <iostream>
#include <vector>

using namespace std;

typedef vector<int> vi;
typedef vector<vi> vvi;
typedef pair<int, int> pi;
typedef vector<pi> vpi;

int main() {
  int t;
  cin >> t;
  while (t--) {
        int n;
        cin >> n;
        vpi M(n);
        for (int i = 0; i < n; ++i)
        cin >> M[i].first >> M[i].second;
        vvi D(n, vi(n,0));
        for (int l = 1; l < n; ++l) {
        for (int i = 0; i+l < n; ++i) {
        int j = i + l;
        D[i][j] = 1e9;
        for (int w = i +1; w <= j; ++w) {
        D[i][j] = min(D[i][j], D[i][w-1] + D[w][j] + M[i].first*M[w].first*M[j].second);
        }
        }
        }
        cout << D[0][n-1] << endl;
  }
}
```

Autor de la solución: Cesc Folch (Miembro del comité organizador)

Componentes fuertemente conexas

Dado un grafo dirigido, una componente fuertemente conexa es un conjunto de vértices tales que se puede ir de cualquier vértice del conjunto a otro con las aristas del grafo.

Dado un grafo dirigido, ¿cuál es el mínimo número de componentes fuertemente conexas que necesitamos para que cada vértice esté como mínimo en una componente fuertemente conexa?

Manuales que pueden ser útiles: Grafos y DFS

Enlaces que pueden ayudar: CFC wikipedia y Algoritmo

Estrategia de la solución

Este problema tenía un enlace con el que fácilmente se podía encontrar una explicación sobre cómo programar la solución.

Solución en Python

```python
import sys
sys.setrecursionlimit(10000)

adj = []
dfs_num = []
dfs_low = []
in_stack = []
stack = []
dfs_num_counter = 0
scc_counter = 0

def tarjanSCC(u):
    global adj, dfs_num, dfs_low, in_stack, dfs_num_counter, scc_counter
    dfs_low[u] = dfs_num_counter
    dfs_num[u] = dfs_num_counter
    dfs_num_counter += 1
    stack.append(u)
    in_stack[u] = True
    for v in adj[u]:
        if dfs_num[v] == -1:
            tarjanSCC(v)
        if in_stack[v]:
            dfs_low[u] = min(dfs_low[u], dfs_low[v])
    if dfs_low[u] == dfs_num[u]:
        scc_counter += 1
        while True:
            v = stack.pop()
            in_stack[v] = False
            if u == v:
                break
```

```
t = int(input())
for tc in range(t):
    strings = input().split()
    n = int(strings[0])
    m = int(strings[1])
    adj = []
    for i in range(n):
        adj.append([])
    dfs_num = [-1] * n
    dfs_low = [-1] * n
    in_stack = [False] * n
    stack = []
    dfs_num_counter = 0
    scc_counter = 0
    for i in range(m):
        strings = input().split()
        a = int(strings[0])
        b = int(strings[1])
        adj[a].append(b)
    for i in range(n):
        if dfs_num[i] == -1:
            tarjanSCC(i)
    print(scc_counter)
```

Autor de la solución: Izan Beltrán (Miembro del comité organizador)

Solución en C++

```
#include <iostream>
#include <vector>

using namespace std;

typedef vector<int> vi;
typedef vector<vi> vvi;

vvi G, T;
vi V,A;

void dfs1(int x) {
  V[x] = 1;
  for (auto y : G[x]) {
    if (!V[y]) dfs1(y);
  }
  A.push_back(x);
}

void dfs2(int x) {
  V[x] = 0;
  for (auto y : T[x]) {
    if (V[y]) dfs2(y);
  }
}
```

```
int main() {
  int t;
  cin >> t;
  while (t--) {
    int n, m;
    cin >> n >> m;

    G = T = vvi(n);
    V = vi(n,0);
    A = vi(0);

    for (int i = 0; i < m; ++i) {
      int x, y;
      cin >> x >> y;
      G[x].push_back(y);
      T[y].push_back(x);
    }

    for (int i = 0; i < n; ++i) {
      if (!V[i]) dfs1(i);
    }
    int res = 0;
    for (int i = n-1; i >= 0; --i) {
      if (V[A[i]]) {
        ++res;
        dfs2(A[i]);
      }
    }

    cout << res << endl;
  }
}
```

Autor de la solución: Cesc Folch (Miembro del comité organizador)

Minimum spanning tree

Dado un grafo no dirigido con costes en las aristas, calculad el coste del árbol de expansión de coste mínimo.

Manuales que pueden ser útiles: MST, Grafos

Estrategia de la solución

Este problema tenía un enlace con el que fácilmente se podía encontrar una explicación sobre cómo programar la solución.

Solución en Python

```
parent = []

def find(u):
    global parent
    if parent[u] != u:
        parent[u] = find(parent[u])
    return parent[u]

def join(u, v):
    global parent
    u = find(u)
    v = find(v)
    if u != v:
        parent[v] = u
        return True
    return False

t = int(input())
for tc in range(t):
    strings = input().split()
    n = int(strings[0])
    m = int(strings[1])
    parent = []
    for i in range(n):
        parent.append(i)
    edges = []
    for i in range(m):
        strings = input().split()
        x = int(strings[0])
        y = int(strings[1])
        c = int(strings[2])
        edges.append((c, x, y))
    edges = sorted(edges)
    cost = 0
    for edge in edges:
        if join(edge[1], edge[2]):
            cost += edge[0]
    print(cost)
```

Autor de la solución: Izan Beltrán (Miembro del comité organizador)

Solución en C++

```cpp
#include <iostream>
#include <vector>
#include <queue>

using namespace std;

typedef pair<int, int> pi;
typedef pair<int, pi> ppi;
typedef vector<int> vi;

vi P;

int pare(int x) {
  if (P[x] == -1) return x;
  return P[x] = pare(P[x]);
}

int main() {
  int t;
  cin >> t;
  while (t--) {
        int n, m;
        cin >> n >> m;
        priority_queue<ppi> Q;
        while (m--) {
        ppi k;
        cin >> k.second.first >> k.second.second >> k.first;
        k.first = -k.first;
        Q.push(k);
        }
        int res = 0;
        P = vi(n, -1);
        while (!Q.empty()) {
        int x = Q.top().second.first;
        int y = Q.top().second.second;
        int c = -Q.top().first;
        Q.pop();
        if (pare(x) != pare(y)) {
        res += c;
        P[pare(x)] = pare(y);
        }
        }
        cout << res << endl;
  }
}
```

Autor de la solución: Cesc Folch (Miembro del comité organizador)

Aristas y vértices de corte

Dado un grafo conexo, una arista es de corte si al eliminar esta arista el grafo resultante no es conexo. Igualmente definimos un vértice de corte si al eliminarlo el grafo resultante no es conexo, cuando eliminamos un vértice también eliminamos todas sus aristas adyacentes.

Dado un grafo conexo, cuantas aristas y vértices de corte tiene?

Manuales que pueden ser útiles: Grafos y DFS

Enlaces que pueden ayudar: Arista de corte wikipedia, Vértice de corte wikipedia y Algoritmo

Estrategia de la solución

Este problema tenía un enlace con el que fácilmente se podía encontrar una explicación sobre cómo programar la solución.

Solución en Python

```python
import sys
sys.setrecursionlimit(10000)

adj = []
dfs_num = []
dfs_low = []
articulation_vertex = []
dfs_parent = []
dfs_num_counter = 0
root_children = 0
dfs_root = 0
bridge_counter = 0

def dfs(u):
    global adj, dfs_num, dfs_low, articulation_vertex, dfs_parent
    global dfs_num_counter, root_children, dfs_root, bridge_counter
    dfs_low[u] = dfs_num_counter
    dfs_num[u] = dfs_num_counter
    dfs_num_counter += 1
    for v in adj[u]:
        if dfs_num[v] == -1:
            dfs_parent[v] = u
            if u == dfs_root:
                root_children += 1
            dfs(v)
            if dfs_low[v] >= dfs_num[u]:
                articulation_vertex[u] = True
            if dfs_low[v] > dfs_num[u]:
                bridge_counter += 1
            dfs_low[u] = min(dfs_low[u], dfs_low[v])
        elif v != dfs_parent[u]:
```

```
            dfs_low[u] = min(dfs_low[u], dfs_num[v])

t = int(input())
for test in range(t):
    strings = input().split()
    n = int(strings[0])
    m = int(strings[1])
    adj = []
    for i in range(n):
        adj.append([])
    dfs_num = [-1] * n
    dfs_low = [-1] * n
    dfs_parent = [-1] * n
    articulation_vertex = [False] * n
    dfs_num_counter = 0
    bridge_counter = 0
    for i in range(m):
        strings = input().split()
        a = int(strings[0])
        b = int(strings[1])
        adj[a].append(b)
        adj[b].append(a)
    for i in range(n):
        if dfs_num[i] == -1:
            dfs_root = i
            root_children = 0
            dfs(i)
            articulation_vertex[i] = (root_children > 1)
    articulations = 0
    for i in range(n):
        if articulation_vertex[i]:
            articulations += 1
    print(articulations, bridge_counter)
```

Autor de la solución: Izan Beltrán (Miembro del comité organizador)

Solución en C++

```
#include <iostream>
#include <vector>

using namespace std;

typedef vector<int> vi;
typedef vector<vi> vvi;

vvi G;
vi num, minnum;

int cont;
int vres, ares;

void dfs(int x, int p = -1) {
  num[x] = minnum[x] = cont++;
```

```
  int fill = 0;
  int k = 0;
  for (int y : G[x]) {
       if (num[y] == -1) {
       ++fill;
       dfs(y, x);
       if (minnum[y] >= num[y]) ++ares;
       minnum[x] = min(minnum[x], minnum[y]);
       if (minnum[y] >= num[x]) k = 1;
       }
       else if (y != p) {
       minnum[x] = min(minnum[x], num[y]);
       }
  }
  if (p != -1) vres += k;
  else if (p == -1 and fill > 1) ++vres;
}

int main() {
  int t;
  cin >> t;
  while (t--) {
       int n, m;
       cin >>  n >> m;
       G = vvi(n);
       num = minnum = vi(n,-1);
       cont = 0;
       vres = 0;
       ares = 0;
       for (int i = 0; i < m; ++i) {
       int x, y;
       cin >> x >> y;
       G[x].push_back(y);
       G[y].push_back(x);
       }
       dfs(0);
       cout << vres << ' ' << ares << endl;
  }
}
```

Autor de la solución: Cesc Folch (Miembro del comité organizador)

Máximo del segmento

Dados n enteros $a_1, a_2, \ldots, a_n$ queremos hacer q operaciones. Hay dos tipos de operaciones:

Tipo 1: Dados tres enteros l, r, x con $1 \leq l \leq r \leq n$ sumamos x a todos los enteros a_i con $l \leq i \leq r$.

Tipo 2: Dados dos enteros l, r con $1 \leq l \leq r \leq n$ queremos saber el valor de $max(a_1, a_{l+1},\ldots,a_{r-1},a_r)$.

En este problema hay dos subcasos:

50 puntos: Para todas las operaciones de tipo 1 tenemos $l = r$.

50 puntos: Sin restricciones extra.

Manuales que pueden ser útiles: AS1, AS2 y AS3

Estrategia de la solución

Este problema tenía un enlace con el que fácilmente se podía encontrar una explicación sobre cómo programar la solución.

Solución en C++

```
#include <iostream>
#include <vector>
using namespace std;

typedef long long ll;
typedef vector<ll> vi;

vi T, lazy;
const ll INF = 1e18;

void actu(int node, bool cond) {
  T[node] += lazy[node];
  if (cond) {
        lazy[2*node] += lazy[node];
        lazy[2*node+1] += lazy[node];
  }
  lazy[node] = 0;
}

void suma(int node, int a, int b, int x, int y, ll val) {
  actu(node, a != b);
  if (b < x or y < a) return;

  if (x <= a and b <= y) {
        T[node] += val;
        if (a != b) {
```

```
        lazy[2*node] += val;
        lazy[2*node+1] += val;
        }
        return;
  }

  int mig = (a+b)/2;
  suma(2*node, a, mig, x, y, val);
  suma(2*node+1, 1+mig, b, x, y, val);

  T[node] = max(T[2*node], T[2*node+1]);
}

ll maxim(int node, int a, int b, int x, int y) {
  actu(node, a != b);
  if (b < x or y < a) return -INF;

  if (x <= a and b <= y) return T[node];

  int mig = (a+b)/2;
  return max(
        maxim(2*node, a, mig, x, y),
        maxim(2*node+1, 1+mig, b, x, y));
}

int main() {
  int t;
  cin >> t;
  while (t--) {
        int n, q;
        cin >> n >> q;
        T = lazy = vi(4*n, 0);
        for (int i = 1; i <= n; ++i) {
        ll k;
        cin >> k;
        suma(1, 1, n, i, i, k);
        }
        while (q--) {
        int k, l, r;
        cin >> k >> l >> r;
        if (k == 1) {
        ll x;
        cin >> x;
        suma(1, 1, n, l, r, x);
        }
        else {
        cout << maxim(1, 1, n, l, r) << endl;
        }
        }
  }
}
```

Autor de la solución: Cesc Folch (Miembro del comité organizador)