Game Over Team Reference - Contents

NUCES Fast Karachi
Huzaifa Rashid, Abdullah Gohar, M. Shaheer Luqman

1. BitMask

1.1. Number of Simple Cycles.

```
/*
      task: Finding the number of simple cycles in a
                  directed graph G = <V, E>.

      complexity: O(2^n * n^2)

      notes: Let dp[msk][v] be the number of Hamiltonian
                  walks in the subgraph generated by vertices
                  in msk that begin in the lowest vertex in
                  msk and end in vertex v.
*/

#define BIT(n) (1 << n)
#define ONES(n) __builtin_popcount(n)

const int MAXN = 20;

int n, m, u, v, g[MAXN];
long long dp[BIT(MAXN)][MAXN], ans;

int main() {
  cin >> n >> m;

  for (int i = 0; i < m; ++i) {
    cin >> u >> v;
    g[u] |= BIT(v);
  }

  for (int i = 0; i < n; ++i)
    dp[BIT(i)][i] = 1;

  for (int msk = 1; msk < BIT(n); ++msk) {
    for (int i = 0; i < n; ++i) {
      if ((msk & BIT(i)) && !(msk & -msk & BIT(i))) {
        int tmsk = msk ^ BIT(i);

        for (int j = 0; tmsk && j < n; ++j)
          if (g[j] & BIT(i))
            dp[msk][i] += dp[tmsk][j];

        if (ONES(msk) > 2 && (g[i] & msk & -msk))
          ans += dp[msk][i];
      }
    }
  }
  cout << ans << endl;
  return 0;
}
```

1.2. Shortest Hamiltonian Walk.

```
/*
      task: Search for the shortest Hamiltonian walk.
                  Let the directed graph G = (V, E) have n
                  vertices, and each edge have weight d(i, j).
                  We want to find a Hamiltonian walk for which
                  the sum of weights of its edges is minimal.

      complexity: O(2^n * n^2)

      notes: Let dp[msk][v] be the length of the shortest
                  Hamiltonian walk on the subgraph generated by
                  vertices in msk that end in vertex v.
*/

#define MAXN 20
#define INF 0x1fffffff
#define BIT(n) (1 << n)

using namespace std;

int n, m, ans = INF, d[MAXN][MAXN], u, v, w, dp[1 << MAXN][MAXN];

int main() {
  cin >> n >> m;

  for (int i = 0; i < n; ++i) {
    for (int j = 0; j < n; ++j)
      d[i][j] = INF;
```

```
}

for (int i = 0; i < BIT(n); ++i) {
 for (int j = 0; j < n; ++j)
  dp[i][j] = INF;
}

for (int i = 0; i < m; ++i) {
 cin >> u >> v >> w;
 d[u][v] = w;
}

for (int i = 0; i < n; ++i)
 dp[1 << i][i] = 0;

for (int msk = 1; msk < (1 << n); ++msk) {
 for (int i = 0; i < n; ++i)
  if (msk & BIT(i)) {
   int tmsk = msk ^ BIT(i);

   for (int j = 0; tmsk && j < n; ++j)
    dp[msk][i] = min(dp[tmsk][j] + d[j][i], dp[msk][i]);
  }
}

for (int i = 0; i < n; ++i)
 ans = min(ans, dp[BIT(n) - 1][i]);

cout << ans << endl;
return 0;
}
```

2. Data Structures

2.1. Disjoint Set.

```
int N;
int parent[N], cont[N];

void initSet() {
  for (int i = 0; i < N; ++i) {
    parent[i] = i;
    cont[i] = 1;
  }
}

int SetOf(int x) { return (x == parent[x]) ? x : parent[x] = SetOf(parent[x]); }

void Merge(int x, int y) {
  x = SetOf(x);
  y = SetOf(y);

  if (x == y)
    return;

  if (cont[x] < cont[y])
    swap(x, y);

  parent[y] = x;
  cont[x] += cont[y];
}
```

2.2. Ordered Set.

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;

#define ordered_set tree<int, null_type,less<int>, rb_tree_tag,tree_order_statisti
```

2.3. Segment Tree Lazy Propagation.

```
/*
      In this example:
      update item[l...r] + val
      query sum(item[l...r])
*/

#define MaxN 1000
#define Left(x) ((x << 1) + 1)
#define Right(x) ((x << 1) + 2)

int st[4 * MaxN], lazy[4 * MaxN];

void push(int node, int nodeL, int nodeR) {
  int m = (nodeL + nodeR) / 2;

  lazy[Left(node)] += lazy[node];
  lazy[Right(node)] += lazy[node];
  st[Left(node)] += (m - nodeL + 1) * lazy[node];
  st[Right(node)] += (nodeR - m) * lazy[node];

  lazy[node] = 0;
}

void update(int node, int nodeL, int nodeR, int l, int r, int val) {
  if (l > nodeR || r < nodeL)
    return;
  if (nodeL >= l && nodeR <= r) {
    st[node] += (nodeR - nodeL + 1) * val;
    lazy[node] += val;
    return;
  }
  push(node, nodeL, nodeR);

  int m = (nodeL + nodeR) / 2;
  update(Left(node), nodeL, m, l, r, val);
```

```
  update(Right(node), m + 1, nodeR, l, r, val);
  st[node] = st[Left(node)] + st[Right(node)];
}

int query(int node, int nodeL, int nodeR, int l, int r) {
  if (l > nodeR || r < nodeL)
    return 0;
  if (nodeL >= l && nodeR <= r)
    return st[node];
  push(node, nodeL, nodeR);
  int m = (nodeL + nodeR) / 2;
  return query(Left(node), nodeL, m, l, r) +
       query(Right(node), m + 1, nodeR, l, r);
}
```

2.4. Segment Tree-1D Query.

```
/*
      In this example update is in a position and the query is
      the sum of interval. item[N], st[4*N]
*/
#define Left(x) ((x << 1) + 1)
#define Right(x) ((x << 1) + 2)
#define MaxN 1000

int item[MaxN];

void build(int *st, int node, int nodeL, int nodeR) {
  if (nodeL == nodeR) {
    st[node] = item[nodeL];
    return;
  }
  int m = (nodeL + nodeR) / 2;
  build(st, Left(node), nodeL, m);
  build(st, Right(node), m + 1, nodeR);
  st[node] = st[Left(node)] + st[Right(node)];
}

void update(int *st, int node, int nodeL, int nodeR, int pos, int val) {
  if (nodeL == nodeR) {
    st[node] = val;
    return;
  }
  int m = (nodeL + nodeR) / 2;
  if (pos <= m)
    update(st, Left(node), nodeL, m, pos, val);
  else
    update(st, Right(node), m + 1, nodeR, pos, val);
  st[node] = st[Left(node)] + st[Right(node)];
}

int query(int *st, int node, int nodeL, int nodeR, int l, int r) {
  if (nodeL == l && nodeR == r)
    return st[node];
  int m = (nodeL + nodeR) / 2;
  if (r <= m)
    return query(st, Left(node), nodeL, m, l, r);
  if (l > m)
    return query(st, Right(node), m + 1, nodeR, l, r);
  return query(st, Left(node), nodeL, m, l, m) +
       query(st, Right(node), m + 1, nodeR, m + 1, r);
}
```

3. Dynamic Programming

3.1. Boolean Para.

```
bool evaluate(bool b1, bool b2, char op) {
   if (op == '&') {
      return b1 & b2;
   }
   else if (op == '|') {
      return b1 | b2;
   }
   return b1 ^ b2;
}

// Function which returns the number of ways
// s[i:j] evaluates to req.
int countRecur(int i, int j, bool req, string &s) {

   // Base case:
   if (i == j) {
      return (req == (s[i] == 'T')) ? 1 : 0;
   }

   int ans = 0;
   for (int k = i + 1; k < j; k += 1) {

 int leftTrue = countRecur(i, k - 1, 1, s);
      int leftFalse = countRecur(i, k - 1, 0, s);

      // Count Ways in which right substring
      // evaluates to true and false.
      int rightTrue = countRecur(k + 1, j, 1, s);
      int rightFalse = countRecur(k + 1, j, 0, s);

      // Check if the combinations results
      // to req.
      if (evaluate(true, true, s[k]) == req) {
         ans += leftTrue * rightTrue;
      }
      if (evaluate(true, false, s[k]) == req) {
         ans += leftTrue * rightFalse;
      }
      if (evaluate(false, true, s[k]) == req) {
         ans += leftFalse * rightTrue;
      }
      if (evaluate(false, false, s[k]) == req) {
         ans += leftFalse * rightFalse;
      }
   }

   return ans;
}
int countWays(string s) {
    int n = s.length();
   return countRecur(0, n - 1, 1, s);
}
```

3.2. Dice Throw.

```
int noOfWays(int m, int n, int x) {
  // Base case: Valid combination if (n == 0 && x == 0)return 1;
   // Base case: Invalid combination
   if (n == 0 || x <= 0)
      return 0;
   int ans = 0;
   // Check for all values of m.
   for (int i = 1; i <= m; i++) {
      ans += noOfWays(m, n - 1, x - i);
   }

   return ans;
}
```

3.3. Edit Distance.

```
int dp[n + 1][m + 1];
```

```
memset(dp, 0, sizeof(dp));

for (int i = 0; i <= n; i++)
{
   dp[i][0] = i;
}
for (int j = 0; j <= m; j++)
{
   dp[0][j] = j;
}

for (int i = 1; i <= n; i++)
{
   for (int j = 1; j <= m; j++)
   {

      char x = a[i - 1];
      char y = b[j - 1];
      if (x == y)
      {
         dp[i][j] = dp[i - 1][j - 1];
      }
      else
      {
         // remove char from a
         dp[i][j] = dp[i - 1][j] + 1;

         // add char to a
         dp[i][j] = min(dp[i][j], dp[i][j - 1] +
                             1);

         // replace char
         dp[i][j] = min(dp[i][j], dp[i - 1][j - 1] + 1);
      }
   }
}
```

3.4. Knapsack.

```
def kProfit(W, N, wt, pr, dp):
   if N == 0 or W == 0:
      return 0
   if dp[N][W] is not None:
      return dp[N][W]
   if wt[N - 1] <= W:
      dp[N][W] = max(
         pr[N - 1] + kProfit(W - wt[N - 1], N - 1, wt, pr, dp),
         kProfit(W, N - 1, wt, pr, dp),
      )
      return dp[N][W]
   else:
      dp[N][W] = kProfit(W, N - 1, wt, pr, dp)
      return dp[N][W]
# define DP array
dp = [[None] * (W + 1) for _ in range(N + 1)]
maxProfit = kProfit(W, N, wt, pr, dp)
```

3.5. LCS.

```
for (int i = 0; i <= m; i++)
{
   for (int j = 0; j <= n; j++)
   {
      if (i == 0 || j == 0)
         L[i][j] = 0;

      else if (X[i - 1] == Y[j - 1])
         L[i][j] = L[i - 1][j - 1] + 1;

      else
         L[i][j] = max(L[i - 1][j], L[i][j - 1]);
   }
}

// L[m][n] contains length of LCS
// for X[0..n-1] and Y[0..m-1]
return L[m][n];
```

3.6. Longest Increasing Subsequence.

```
const int oo = 99999999;

#define index_of(as, x) \
 distance(as.begin(), lower_bound(as.begin(), as.end(), x))

/*
   Tested: LISTA
   Contest 3 COCI 2006-2007
*/
vector<int> lis_fast(const vector<int> &a) {
 const int n = a.size();
 vector<int> A(n, oo), id(n);

 for (int i = 0; i < n; ++i) {
  id[i] = index_of(A, a[i]);
  A[id[i]] = a[i];
 }

 int m = *max_element(id.begin(), id.end());
 vector<int> b(m + 1);

 for (int i = n - 1; i >= 0; --i)
  if (id[i] == m)
   b[m--] = a[i];

 return b;
}
```

3.7. Longest Path.

```
int longestPath(int i, int j, vector<vector<int>> &matrix) {
   int ans = 1;

   vector<vector<int>> dir = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};

   // Check for all 4 directions
   for (auto d : dir) {
      int x = i + d[0];
      int y = j + d[1];

      // If new cells are valid and
      // increasing by 1.
      if (x >= 0 && x < matrix.size() && y >= 0 &&
         y < matrix[0].size() && matrix[x][y] == matrix[i][j] + 1) {
         ans = max(ans, 1 + longestPath(x, y, matrix));
      }
   }

   return ans;
}

int longestIncreasingPath(vector<vector<int>> &matrix) {
   int ans = 0;

   // Find length of longest path
   // from each cell i, j
   for (int i = 0; i < matrix.size(); i++) {
      for (int j = 0; j < matrix[0].size(); j++) {
         int val = longestPath(i, j, matrix);
         ans = max(ans, val);
      }
   }

   return ans;
}
```

3.8. Matrix Chain.

```
const int oo = 1 << 30;

int matrix_chain(const vector<int> &p) {
 int n = p.size() - 1;
 int dp[n + 1][n + 1];

 for (int i = 1; i <= n; ++i)
  dp[i][i] = 0;
```

```
 for (int len = 2; len <= n; ++len) {
  for (int i = 1, j = i + len - 1; j <= n; ++i, ++j) {
   dp[i][j] = oo;
   for (int k = i; k < j; ++k)
    dp[i][j] =
     min(dp[i][j], dp[i][k] + dp[k + 1][j] + p[i - 1] * p[k] * p[j]);
  }
 }

 return dp[1][n];
}
```

3.9. Minimum Partition.

```
int findMinDifference(vector<int> &arr, int n,
                int sumCalculated, int sumTotal)
{

   // Base case: if we've considered all elements
   if (n == 0)
   {
      return abs((sumTotal - sumCalculated) - sumCalculated);
   }

   // Include the current element in the subset
   int include = findMinDifference(arr, n - 1, sumCalculated + arr[n - 1], sumTotal);

   // Exclude the current element from the subset
   int exclude = findMinDifference(arr, n - 1, sumCalculated, sumTotal);
   return min(include, exclude);
}

// Function to get the minimum difference
int minDifference(vector<int> &arr)
{
   int sumTotal = 0;
   for (int num : arr)
   {
      sumTotal += num;
   }

   // Call recursive function to find
   // the minimum difference
   return findMinDifference(arr, arr.size(), 0, sumTotal);
}
```

3.10. Rod Cutting.

```
int cutRodRecur(int i, vector<int> &price) {
   if (i==0) return 0;
   int ans = 0;
   // Find maximum value for each cut.
   // Take value of rod of length j, and
   // recursively find value of rod of
   // length (i-j).
   for (int j=1; j<=i; j++) {
      ans = max(ans, price[j-1]+cutRodRecur(i-j, price)); return ans;
   }
}
```

3.11. Shortest Common Super Sequence.

```
int shortestCommonSupersequence(string &s1, string &s2) {
   return s1.size() + s2.size() - lcs(s1, s2);
}
```

4. Graphs

4.1. Bipartite Matching.

```
/*
      Tested: AIZU(judge.u-aizu.ac.jp) GRL_7_A
      Complexity: O(nm)
*/

struct graph {
 int L, R;
 vector<vector<int>> adj;

 graph(int L, int R) : L(L), R(R), adj(L + R) {}

 void add_edge(int u, int v) {
  adj[u].push_back(v + L);
  adj[v + L].push_back(u);
 }

 int maximum_matching() {
  vector<int> visited(L), mate(L + R, -1);
  function<bool(int)> augment = [&](int u) {
   if (visited[u])
    return false;
    visited[u] = true;
    for (int w : adj[u]) {
     int v = mate[w];
     if (v < 0 || augment(v)) {
      mate[u] = w;
      mate[w] = u;
      return true;
     }
    }
    return false;
   };
   int match = 0;
   for (int u = 0; u < L; ++u) {
    fill(visited.begin(), visited.end(), 0);
    if (augment(u))
     ++match;
   }
   return match;
  }
 };
```

4.2. Hopcroft Karp.

```
/*
      Tested: SPOJ MATCHING
      Complexity: O(m n^0.5)
*/

struct graph {
 int L, R;
 vector<vector<int>> adj;

 graph(int L, int R) : L(L), R(R), adj(L + R) {}

 void add_edge(int u, int v) {
  adj[u].push_back(v + L);
  adj[v + L].push_back(u);
 }

 int maximum_matching() {
   vector<int> level(L), mate(L + R, -1);

   function<bool(void)> levelize = [&]() {
    queue<int> Q;
    for (int u = 0; u < L; ++u) {
     level[u] = -1;
     if (mate[u] < 0) {
      level[u] = 0;
      Q.push(u);
     }
    }
    while (!Q.empty()) {
     int u = Q.front();
     Q.pop();
     for (int w : adj[u]) {
      int v = mate[w];
      if (v < 0)
```

```
        return true;
      if (level[v] < 0) {
        level[v] = level[u] + 1;
        Q.push(v);
      }
    }
  }
  return false;
};

function<bool(int)> augment = [&](int u) {
  for (int w : adj[u]) {
    int v = mate[w];
    if (v < 0 || (level[v] > level[u] && augment(v))) {
      mate[u] = w;
      mate[w] = u;
      return true;
    }
  }
  return false;
};
int match = 0;
while (levelize())
  for (int u = 0; u < L; ++u)
    if (mate[u] < 0 && augment(u))
      ++match;
return match;
}
};
```

4.3. Kruskal.

```
struct Edge {
 int src, dst, weight;
 Edge(int a, int b, int c) : src(a), dst(b), weight(c) {}
};

const int MaxN = 10000;

vector<Edge> mst;
vector<Edge> edge;

bool cmp(Edge x, Edge y) { return x.weight < y.weight; }

int cost = 0;
void Kruskal() {
 mst.clear();
 initDisjointSet();

 sort(ALL(edge), cmp);

 for (int i = 0; i < (int)edge.size(); ++i) {
  int u = edge[i].src;
  int v = edge[i].dst;
  if (SetOf(u) != SetOf(v)) {
   cost += edge[i].weight;
   Merge(u, v);
  }
 }
}
```

4.4. Min Cost Max Flow.

```
/*
    Minimum Cost Flow (Tomizawa, Edmonds-Karp)

    Complexity: O(F m log n), where F is the amount of maximum flow

    Tested: Codeforces [http://codeforces.com/problemset/problem/717/G]
*/

template <typename flow_type, typename cost_type> struct min_cost_max_flow {
 struct edge {
  size_t src, dst, rev;
  flow_type flow, cap;
  cost_type cost;
 };

 int n;
 vector<vector<edge>> adj;

 min_cost_max_flow(int n) : n(n), adj(n), potential(n), dist(n), back(n) {}
```

```
  void add_edge(size_t src, size_t dst, flow_type cap, cost_type cost) {
    adj[src].push_back({src, dst, adj[dst].size(), 0, cap, cost});
    if (src == dst)
      adj[src].back().rev++;
    adj[dst].push_back({dst, src, adj[src].size() - 1, 0, 0, -cost});
  }

  vector<cost_type> potential;

  inline cost_type rcost(const edge &e) {
    return e.cost + potential[e.src] - potential[e.dst];
  }

  void bellman_ford(int source) {
    for (int k = 0; k < n; ++k)
      for (int u = 0; u < n; ++u)
        for (edge &e : adj[u])
          if (e.cap > 0 && rcost(e) < 0)
            potential[e.dst] += rcost(e);
  }

  const cost_type oo = numeric_limits<cost_type>::max();

  vector<cost_type> dist;
  vector<edge *> back;

  cost_type dijkstra(int source, int sink) {
    fill(dist.begin(), dist.end(), oo);

    typedef pair<cost_type, int> node;
    priority_queue<node, vector<node>, greater<node>> pq;

    for (pq.push({dist[source] = 0, source}); !pq.empty();) {
      node p = pq.top();
      pq.pop();

      if (dist[p.second] < p.first)
        continue;
      if (p.second == sink)
        break;

      for (edge &e : adj[p.second])
        if (e.flow < e.cap && dist[e.dst] > dist[e.src] + rcost(e)) {
          back[e.dst] = &e;
          pq.push({dist[e.dst] = dist[e.src] + rcost(e), e.dst});
        }
    }

    return dist[sink];
  }

  pair<flow_type, cost_type> max_flow(int source, int sink) {
    flow_type flow = 0;
    cost_type cost = 0;

    for (int u = 0; u < n; ++u)
      for (edge &e : adj[u])
        e.flow = 0;

    potential.assign(n, 0);
    dist.assign(n, 0);
    back.assign(n, nullptr);

    bellman_ford(source); // remove negative costs

    while (dijkstra(source, sink) < oo) {
      for (int u = 0; u < n; ++u)
        if (dist[u] < dist[sink])
          potential[u] += dist[u] - dist[sink];

      flow_type f = numeric_limits<flow_type>::max();

      for (edge *e = back[sink]; e; e = back[e->src])
        f = min(f, e->cap - e->flow);
      for (edge *e = back[sink]; e; e = back[e->src])
        e->flow += f, adj[e->dst][e->rev].flow -= f;

      flow += f;
      cost += f * (potential[sink] - potential[source]);
    }
    return {flow, cost};
  }
};
```

4.5. Prim.

```
const int MaxN = 10000;
```

```cpp
int n, m;
typedef pair<int, pii> par;
priority_queue<par, vector<par>, greater<par>> pq;
vi taken;
vector<pii> g[MaxN];
int mstCost;
vector<pii> mstEdge;

void process(int u) {
 taken[u] = 1;
 for (int i = 0; i < (int)g[u].size(); ++i) {
  pii v = g[u][i];
  if (!taken[v.S])
   pq.push(par(v.F, pii(u, v.S)));
 }
}

void Prim(int s) {
 taken.assign(n, 0);
 pq = priority_queue<par, vector<par>, greater<par>>();
 process(s);
 mstCost = 0;

 while (!pq.empty()) {
  par top = pq.top();
  pq.pop();
  pii node = top.S;

  int w = top.F;
  int u = node.F;
  int v = node.S;

  if (!taken[v]) {
   mstCost += w;
   mstEdge.pb(pii(u, v));
   process(v);
  }
 }
}
```

4.6. Satisfiability Two SAT.

```cpp
/*
     Two-Sat

     Complexity: O(n)

     Tested: POI (Gates)
*/

struct satisfiability_twosat {
 int n;
 vector<vector<int>> imp;

 satisfiability_twosat(int n) : n(n), imp(2 * n) {}

 void add_edge(int u, int v) { imp[u].push_back(v); }

 int neg(int u) { return (n << 1) - u - 1; }

 void implication(int u, int v) {
  add_edge(u, v);
  add_edge(neg(v), neg(u));
 }

 vector<bool> solve() {
  int size = 2 * n;
  vector<int> S, B, I(size);

  function<void(int)> dfs = [&](int u) {
   B.push_back(I[u] = S.size());
   S.push_back(u);

   for (int v : imp[u])
    if (!I[v])
     dfs(v);
    else
     while (I[v] < B.back())
      B.pop_back();

   if (I[u] == B.back())
    for (B.pop_back(), ++size; I[u] < S.size(); S.pop_back())
     I[S.back()] = size;
  };

  for (int u = 0; u < 2 * n; ++u)
   if (!I[u])
    dfs(u);
```

```
  vector<bool> values(n);

  for (int u = 0; u < n; ++u)
   if (I[u] == I[neg(u)])
    return {};
   else
    values[u] = I[u] < I[neg(u)];

  return values;
 }
};
```

4.7. Strongly Connected Components.

```
const int MaxN = 10000;

struct edge {

 int src, dst, w;
 edge(int a, int b, int c) : src(a), dst(b), w(c) {}
};

typedef vector<edge> Graph;
int n, m;
Graph g[MaxN];
Graph gt[MaxN];
int order[MaxN], mk[MaxN];
int scc[MaxN];
int vcount[MaxN];
int cur;
int cur_scc;

void dfs(int u) {
 mk[u] = true;
 for (int i = 0; i < (int)g[u].size(); ++i) {
  int v = g[u][i].dst;
  if (!mk[v])
   dfs(v);
 }
 order[n - 1 - cur++] = u;
}

void dfs_rev(int u) {
 scc[u] = cur_scc;
 ++vcount[cur_scc];
 mk[u] = true;
 for (int i = 0; i < (int)gt[u].size(); ++i) {
  int v = gt[u][i].dst;
  if (!mk[v])
   dfs_rev(v);
 }
}

void make_scc() {
 cur = 0;
 memset(mk, 0, sizeof(mk));
 for (int i = 0; i < n; ++i)
  if (!mk[i])
   dfs(i);

 cur_scc = 0;
 memset(mk, 0, sizeof(mk));

 for (int i = 0; i < n; ++i) {
  int v = order[i];
  if (!mk[v]) {
   dfs_rev(v);
   ++cur_scc;
  }
 }
}

void init() {
 for (int i = 0; i < n; ++i) {
  g[i].clear();
  gt[i].clear();
  vcount[i] = 0;
 }
}
```

4.8. Reduce Graph.

```
//============================================================================
// Name : Reduce.cpp
// Author : Ivan Galban Smith
// Version :
// Copyright : Your copyright notice
// Description : Hello World in C++, Ansi-style
//============================================================================

#include <bits/stdc++.h>

using namespace std;

/////////////////////////////////////////////////
typedef complex<double> P;
typedef vector<P> Pol;
typedef long long Int;
typedef pair<int, int> pii;
typedef vector<int> vi;
typedef vector<vi> Graph;
/////////////////////////////////////////////////
#define REP(i, n) for (int i = 0; i < (int)n; ++i)
#define FOR(i, n) for (int i = 1; i <= (int)n; ++i)
#define ITR(c) __typeof((c).begin())
#define foreach(i, c) for (ITR(c) i = (c).begin(); i != (c).end(); ++i)
#define ALL(c) (c).begin(), (c).end()
#define DB(x) cout << #x << "␣=␣" << x << endl

#define X(c) real(c)
#define Y(c) imag(c)
#define endl '\n'
#define F first
#define S second
#define pb push_back
#define mp make_pair
#define BIT(n) (1 << n)
/////////////////////////////////////////////////
const double EPS = 1e-15;
const int oo = (1 << 30);
const double PI = M_PI;
const int MOD = 1000000000 + 7;
/////////////////////////////////////////////////

const int MaxN = 1000;

struct Edge {
  int src, dst, wt;
  Edge(int a, int b, int c) : src(a), dst(b), wt(c) {}
};

int n, m;

vector<Edge> g[MaxN];
vector<Edge> gt[MaxN];
vector<Edge> gr[MaxN];

int cur, cur_scc;

int mk[MaxN];
int order[MaxN];

int scc[MaxN];
int vcountSCC[MaxN];

void dfs(int u) {
  mk[u] = true;
  REP(i, g[u].size()) {
    int v = g[u][i].dst;
    if (!mk[v])
      dfs(v);
  }
  order[n - 1 - cur++] = u;
}

void dfs_rev(int u) {
  scc[u] = cur_scc;
  ++vcountSCC[cur_scc];
  mk[u] = true;

  REP(i, gt[u].size()) {
    int v = gt[u][i].dst;
    if (!mk[v])
      dfs_rev(v);
  }
}

void make_scc() {
  cur = 0;
  memset(mk, 0, sizeof(mk));
  REP(i, n)
  if (!mk[i])
```

```
    dfs(i);

  cur_scc = 0;
  memset(mk, 0, sizeof(mk));

  REP(i, n) {
    int v = order[i];
    if (!mk[v]) {
      dfs_rev(v);
      ++cur_scc;
    }
  }
}

void build_reduce_graph() {
  make_scc();
  REP(i, n)
  REP(j, g[i].size())
  if (scc[i] != scc[g[i][j].dst])
    gr[scc[i]].pb(Edge(scc[i], scc[g[i][j].dst], g[i][j].wt));
}

int main() {
  // ios_base::sync_with_stdio(false);
  // cin.tie(0);

  cin >> n >> m;
  REP(i, m) {
    int a, b, c;
    cin >> a >> b >> c;
    g[a].pb(Edge(a, b, c));
    gt[b].pb(Edge(b, a, c));
  }

  build_reduce_graph();

  cout << "V␣" << cur_scc << "\nEdges:\n";
  REP(u, cur_scc)
  REP(i, gr[u].size()) {
    Edge e = gr[u][i];
    cout << e.src << "␣" << e.dst << "␣" << e.wt << endl;
    ;
  }

  return 0;
}
```

5. Matrix

5.1. Gauss.

```
/*
[TESTED COJ 2536 05/11/2014]
*/
const int MAXN = 110;
const int oo = (1 << 30);
const double EPS = 1e-6;

double a[MAXN][MAXN];
double ans[MAXN];

int n; // ecuations
int m; // variables

void init(int _n, int _m) {
 n = _n;
 m = _m;
 memset(a, 0, sizeof a);
 memset(ans, 0, sizeof ans);
}

int solve() {
 vector<int> where(m, -1);
 for (int col = 0, row = 0; col < m && row < n; ++col) {
  int sel = row;
  for (int i = row; i < n; ++i)
   if (abs(a[i][col]) > abs(a[sel][col]))
    sel = i;

  if (abs(a[sel][col]) < EPS)
   continue;
  for (int i = col; i <= m; ++i)
   swap(a[sel][i], a[row][i]);

  where[col] = row;

  for (int i = 0; i < n; ++i) {
   if (i != row) {
    double c = a[i][col] / a[row][col];
    for (int j = col; j <= m; ++j)
     a[i][j] -= a[row][j] * c;
   }
  }
  ++row;
 }

 for (int i = 0; i < m; ++i)
  if (where[i] != -1)
   ans[i] = a[where[i]][m] / a[where[i]][i];

 for (int i = 0; i < n; ++i) {
  double sum = 0;
  for (int j = 0; j < m; ++j)
   sum += ans[j] * a[i][j];
  if (abs(sum - a[i][m]) > EPS)
   return 0;
 }
 for (int i = 0; i < m; ++i)
  if (where[i] == -1)
   return oo;
 return 1;
}
```

6. Number Theory

6.1. Binomial Coefficient.

```
/*
   CALCULA COMBINATORIA DE n en k
   USANDO EL TRIANGULO DE PASCAL
*/
#include <cstdio>
#include <iostream>

#define MAX 10000

using namespace std;

int C[MAX][MAX];

void Pascal(int level) {
  for (int n = 0; n <= level; ++n) {
    C[n][0] = C[n][n] = 1;
    for (int k = 1; k < n; ++k)
      C[n][k] = C[n - 1][k] + C[n - 1][k - 1];
  }
}

int main() {
  int n, k;
  cin >> n >> k;
  Pascal(n);
  cout << C[n][k];

  return 0;
}
```

6.2. ALL Number Theory.

```
/*
      Binary Multiplication
      [Tested Timus 1141,1204]**
*/
Int mod_mult(Int a, Int b, Int mod)
{
   Int x = 0;
   while (b)
   {
      if (b & 1)
         x = (x + a) % mod;
      a = (a << 1) % mod;
      b >>= 1;
   }
   return x;
}

Int mod_pow(Int a, Int n, Int mod)
{
   Int x = 1;
   while (n)
   {
      if (n & 1)
         x = mod_mult(x, a, mod);
      a = mod_mult(a, a, mod);
      n >>= 1;
   }
   return x;
}

int gcd(int a, int b, int &x, int &y)
{
   if (b == 0)
   {
      x = 1;
      y = 0;
      return a;
   }
   int r = gcd(b, a % b, y, x);
   y -= a / b * x;
   return r;
}

int inverse(int a, int m)
{
   int x, y;
```

```
   if (gcd(a, m, x, y) != 1)
      return 0;
   return (x % m + m) % m;
}

int discrete_log(Int a, Int b, Int m)
{
   map<Int, Int> hash;
   Int n = phi(m), k = sqrt(n);

   for (Int i = 0, t = 1; i < k; i++)
   {
      hash[t] = i;
      t = (t * a) % m;
   }
   Int c = mod_pow(a, n - k, m);
   for (Int i = 0; i * k < n; i++)
   {
      if (hash.find(b) != hash.end())
         return (i * k + hash[b]) % n;

      b = (b * c) % m;
   }
   return -1;
}

/*
      Solves a*x = b (mod p)
      [Tested CodeChef Quadratic Equations]
*/
long solve_linear(long a, long b, int p) { return (b * inverse(a, p)) % p; }

/*
      Solve x=ai(mod mi)
      For any i and j, (mi,mj)|ai-aj.
      Return x0 in [0,[M]).
      M = m1m2..mn
      All solutions are x=x0+t[M].
*/
int linear_con(int a[], int m[], int n)
{
   int u = a[0], v = m[0], p, q, r, t;
   for (int i = 1; i < n; i++)
   {
      r = gcd(v, m[i], p, q);
      t = v;
      v = v / r * m[i];
      u = ((a[i] - u) / r * p * t + u) % v;
   }
   if (u < 0)
      u += v;
   return u;
}

/*
      Solve x = ai(mod mi)
      For any i and j, (mi,mj)==1.
      Returns x0 in [0,M).
      M = m1m2..mn
      All solutions are x=x0 + tM.
*/
int chinese(int a[], int m[], int n)
{
   int s = 1, t, ans = 0, p, q;
   for (int i = 0; i < n; i++)
      s *= m[i];
   for (int i = 0; i < n; i++)
   {
      t = s / m[i];
      gcd(t, m[i], p, q);
      ans = (ans + t * p * a[i]) % s;
   }
   if (ans < 0)
      ans += s;
   return ans;
}

/*
Kth discrete roots of a (mod n)
x^k = a (n)
When (k, phi(n)) = 1
[Tested Timus 1141]**
*/
int discrete_root(int k, int a, int n)
{
   int _phi = phi(n);
   int s = (int)inverse(k, _phi);
   return (int)mod_pow(a, s, n);
}

/*
Tonelli Shank's algorithm
Solves x^2=a (mod p)
[Tested CodeChef Quadratic Equations, Timus 1132]
```

```
Warning: Precompute primes to avoid TLE
*/
int solve_quadratic(int a, int p)
{
   if (a == 0)
      return 0;
   if (p == 2)
      return a;
   if (mod_pow(a, (p - 1) / 2, p) != 1)
      return -1;

   int phi = p - 1;
   int n = 0, k = 0;

   while (phi % 2 == 0)
   {
      phi /= 2;
      n++;
   }

   k = phi;
   int q = 0;

   for (int j = 2; j < p; j++)
      if (mod_pow(j, (p - 1) / 2, p) == p - 1)
      {
         q = j;
         break;
      }

   int t = mod_pow(a, (k + 1) / 2, p);
   int r = mod_pow(a, k, p);

   while (r != 1)
   {
      int i = 0, v = 1;
      while (mod_pow(r, v, p) != 1)
      {
         v *= 2;
         i++;
      }

      int e = mod_pow(2, n - i - 1, p);
      int u = mod_pow(q, k * e, p);

      t = (t * u) % p;
      r = (r * u * u) % p;
   }

   return t;
}

/*
Solves a*x^2 + b*x + c = 0 (mod p)
[Tested CodeChef Quadratic Equations]
*/
set<Int> solve_quadratic(Int a, Int b, Int c, int p)
{
   set<Int> ans;
   if (c == 0)
      ans.insert(0L);
   if (a == 0)
      ans.insert(solve_linear((p - b) % p, c, p));
   else if (p == 2 && (a + b + c) % 2 == 0)
      ans.insert(1L);
   else
   {
      Int r = ((b * b) % p - (4 * a * c) % p + p) % p;
      Int x = solve_quadratic(r, p);
      if (x == -1)
         return ans;
      Int w = solve_linear((2 * a) % p, (x - b + p) % p, p);
      ans.insert(w);
      w = solve_linear((2 * a) % p, (p - x - b + p) % p, p);
      ans.insert(w);
   }
   return ans;
}

/*
Primitive roots
[Tested Timus 1268]
Warning: Precompute primes to avoid TLE
Only: m = 1, p^k , n = 2p^k (p prime > 2),
      m = 2, m = 4
*/
int primitive_root(int m, int p[])
{
   if (m == 1)
      return 0;
   if (m == 2)
      return 1;
   if (m == 4)
      return 3;
```

```
int t = m;
if ((t & 1) == 0)
   t >>= 1;

for (int i = 0; p[i] * p[i] <= t; ++i)
{
   if (t % p[i])
      continue;
   do
      t /= p[i];
   while (t % p[i] == 0);
   if (t > 1 || p[i] == 2)
      return 0;
}

int f[100];
int x = phi(m), y = x, n = 0;

for (int i = 0; p[i] * p[i] <= y; ++i)
{
   if (y % p[i])
      continue;
   do
      y /= p[i];
   while (y % p[i] == 0);
   f[n++] = p[i];
}

if (y > 1)
   f[n++] = y;

for (int i = 1; i < m; ++i)
{
   if (__gcd(i, m) > 1)
      continue;
   bool flag = true;

   for (int j = 0; j < n; ++j)
      if (mod_pow(i, x / f[j], m) == 1)
      {
         flag = false;
         break;
      }

   if (flag)
      return i;
   }
   return 0;
}

typedef long long ll;

ll divisor_sigma(ll n)
{
   ll sigma = 0, d = 1;
   for (; d * d < n; ++d)
      if (n % d == 0)
         sigma += d + n / d;
   if (d * d == n)
      sigma += d;
   return sigma;
}

// sigma(n) for all n in [lo, hi)
vector<ll> divisor_sigma(ll lo, ll hi)
{
   vector<ll> ps = primes(sqrt(hi) + 1);
   vector<ll> res(hi - lo), sigma(hi - lo, 1);
   iota(res.begin(), res.end(), lo);
   for (ll p : ps)
      for (ll k = ((lo + (p - 1)) / p) * p; k < hi; k += p)
      {
         ll b = 1;
         while (res[k - lo] > 1 && res[k - lo] % p == 0)
         {
            res[k - lo] /= p;
            b = 1 + b * p;
         }
         sigma[k - lo] *= b;
      }
   for (ll k = lo; k < hi; ++k)
      if (res[k - lo] > 1)
         sigma[k - lo] *= (1 + res[k - lo]);
   return sigma; // sigma[k-lo] = sigma(k)
}

typedef long long ll;

ll mobius_mu(ll n)
{
   if (n == 0)
      return 0;
   ll mu = 1;
```

```
    for (ll x = 2; x * x <= n; ++x)
        if (n % x == 0)
        {
            mu = -mu;
            n /= x;
            if (n % x == 0)
                return 0;
        }
    return n > 1 ? -mu : mu;
}

// phi(n) for all n in [lo, hi)
vector<ll> mobius_mu(ll lo, ll hi)
{
    vector<ll> ps = primes(sqrt(hi) + 1);
    vector<ll> res(hi - lo), mu(hi - lo, 1);
    iota(res.begin(), res.end(), lo);
    for (ll p : ps)
        for (ll k = ((lo + (p - 1)) / p) * p; k < hi; k += p)
        {
            mu[k - lo] = -mu[k - lo];
            if (res[k - lo] % p == 0)
            {
                res[k - lo] /= p;
                if (res[k - lo] % p == 0)
                {
                    mu[k - lo] = 0;
                    res[k - lo] = 1;
                }
            }
        }
    for (ll k = lo; k < hi; ++k)
        if (res[k - lo] > 1)
            mu[k - lo] = -mu[k - lo];
    return mu; // mu[k-lo] = mu(k)
}
```

7. Numeric Methods

7.1. Fast Fourier Transform.

```
typedef complex<double> base;

// y[i] = A(w^(dir*i)),
// w = exp(2pi/N) is N-th complex principal root of unity,
// A(x) = a[0] + a[1] x + ... + a[n-1] x^{n-1},
// * N must be a power of 2,
long double PI = 2 * acos(0.0L);

void fft(vector<base> &a, bool invert) {
  int n = (int)a.size();

  for (int i = 1, j = 0; i < n; ++i) {
    int bit = n >> 1;
    for (; j >= bit; bit >>= 1)
      j -= bit;
    j += bit;
    if (i < j)
      swap(a[i], a[j]);
  }
  for (int len = 2; len <= n; len <<= 1) {
    double ang = 2 * PI / len * (invert ? -1 : 1);
    base wlen(cos(ang), sin(ang));

    for (int i = 0; i < n; i += len) {
      base w(1);
      for (int j = 0; j < len / 2; ++j) {
        base u = a[i + j], v = a[i + j + len / 2] * w;
        a[i + j] = u + v;
        a[i + j + len / 2] = u - v;
        w *= wlen;
      }
    }
  }
  if (invert)
    for (int i = 0; i < n; ++i)
      a[i] /= n;
}

void convolve(const vector<int> &a, const vector<int> &b, vector<int> &res) {
  vector<base> fa(a.begin(), a.end()), fb(b.begin(), b.end());
  size_t n = 1;
  while (n < max(a.size(), b.size()))
    n <<= 1;
  n <<= 1;
  fa.resize(n), fb.resize(n);
  fft(fa, false), fft(fb, false);
  for (size_t i = 0; i < n; ++i)
    fa[i] *= fb[i];
  fft(fa, true);
  res.resize(n);
  for (size_t i = 0; i < n; ++i)
    res[i] = int(fa[i].real() + 0.5);
}

void print(vector<int> a) {
  cout << a.size() << endl;
  for (int i = 0; i < (int)a.size(); ++i)
    cout << a[i] << "␣";
  cout << endl;
}
```

8. String

8.1. Knuth-Morris-Pratt.

```
// pi[1...m]
vector<int> buildFail(string p) {
  int m = p.size();
  vector<int> pi(m + 1, 0);

  int j = pi[0] = -1;
  for (int i = 1; i <= m; ++i) {
    while (j >= 0 && p[j] != p[i - 1])
      j = pi[j];
    pi[i] = ++j;
  }
```

```
  return pi;
}
// KMP Cuenta la cantidad de veces que aparece una
// sub-cadena (p) en la cadena (t)
int match(string t, string p, vector<int> &pi) {
  int n = t.size(), m = p.size();
  int count = 0;

  for (int i = 0, k = 0; i < n; ++i) {
    while (k >= 0 && p[k] != t[i])
      k = pi[k];
    if (++k >= m) {
      ++count;
      k = pi[k];
    }
  }
  return count;
}
```

8.2. **Longest Palindrome Substring.**

```
// Transform S into T.
// For example, S = "abba", T = "^#a#b#b#a#$".
// ^ and $ signs are sentinels appended to each end to avoid bounds checking
string preProcess(string s) {
  int n = s.length();
  if (n == 0)
    return "^$";
  string ret = "^";
  for (int i = 0; i < n; i++)
    ret += "#" + s.substr(i, 1);

  ret += "#$";
  return ret;
}

// Time: O(n)
string longestPalindrome(string s) {
  string T = preProcess(s);
  int n = T.length();
  int *P = new int[n];
  int C = 0, R = 0;

  for (int i = 1; i < n - 1; i++) {
    int i_mirror = 2 * C - i; // equals to i' = C - (i-C)

    P[i] = (R > i) ? min(R - i, P[i_mirror]) : 0;

    // Attempt to expand palindrome centered at i
    while (T[i + 1 + P[i]] == T[i - 1 - P[i]])
      P[i]++;

    // If palindrome centered at i expand past R,
    // adjust center based on expanded palindrome.
    if (i + P[i] > R) {
      C = i;
      R = i + P[i];
    }
  }

  // Find the maximum element in P.
  int maxLen = 0;
  int centerIndex = 0;
  for (int i = 1; i < n - 1; i++) {
    if (P[i] > maxLen) {
      maxLen = P[i];
      centerIndex = i;
    }
  }

  delete[] P;

  return s.substr((centerIndex - 1 - maxLen) / 2, maxLen);
}
```

8.3. **Z Function.**

```
// Z[i] is the length of the longest substring
// starting from S[i] which is also a prefix of S.
vector<int> z_function(string s) {
  int n = (int)s.length();
  vector<int> z(n);
```

```
  for (int i = 1, l = 0, r = 0; i < n; ++i) {
    if (i <= r)
      z[i] = min(r - i + 1, z[i - l]);
    while (i + z[i] < n && s[z[i]] == s[i + z[i]])
      ++z[i];
    if (i + z[i] - 1 > r)
      l = i, r = i + z[i] - 1;
  }
  return z;
}

// suff[i] = length of the longest common suffix of s and s[0..i]
vector<int> suffixes(const string &s) {
  int n = s.length();

  vector<int> suff(n, n);

  for (int i = n - 2, g = n - 1, f; i >= 0; --i) {
    if (i > g && suff[i + n - 1 - f] != i - g)
      suff[i] = min(suff[i + n - 1 - f], i - g);
    else {
      for (g = min(g, f = i); g >= 0 && s[g] == s[g + n - 1 - f]; --g)
        ;
      suff[i] = f - g;
    }
  }

  return suff;
}
```