Contest

template.cpp

```cpp
// Kactl Template
#include <bits/stdc++.h>
using namespace std;

#define rep(i, a, b) for (int i = a; i < (b); ++i)
#define all(x) begin(x), end(x)
#define sz(x) (int)(x).size()
#define ln "\n"

typedef long long ll;
typedef pair<int, int> pii;
typedef vector<int> vint;

int main() {
  cin.tie(0)->sync_with_stdio(0);
  cin.exceptions(cin.failbit);

  int t;
  cin >> t;
  while (t--) {
  }
  return 0;
}
```

Graph

DSU.cpp

```cpp
struct DSU {
  vint e;
  DSU(int n) : e(n, -1) {}
  bool sameSet(int a, int b) { return find(a) == find(b); }
  int size(int x) { return -e[find(x)]; }
  int find(int x) { return e[x] < 0 ? x : e[x] = find(e[x]); }
  bool join(int a, int b) {
    a = find(a), b = find(b);
    if (a == b)
      return false;
    if (e[a] > e[b])
      swap(a, b);
    e[a] += e[b];
    e[b] = a;
    return true;
  }
};
```

Kruskal.cpp

```cpp
typedef pair<int, pair<int, int>> Edge;
    // {weight, {vertex1, vertex2}}

int kruskal(int n, vector<Edge> &edges) {
  DSU dsu(n);
  sort(edges.begin(), edges.end()); // sort by weight
  int totalWeight = 0;
  for (const auto &edge : edges) {
    int weight = edge.first;
    int u = edge.second.first;
    int v = edge.second.second;
    if (!dsu.sameSet(u, v)) {
      dsu.join(u, v);
      totalWeight += weight;
    }
  }
  return totalWeight;
}
```

MaxFlow.cpp

```cpp
#include <climits>
#include <cstdio>

struct flow_graph {
  int MAX_V, E, s, t, head, tail;
  int *cap, *to, *next, *last, *dist, *q, *now;

  flow_graph() {}

  flow_graph(int V, int MAX_E) {
    MAX_V = V;
    E = 0;
    cap = new int[2 * MAX_E], to = new int[2 * MAX_E],
    next = new int[2 * MAX_E];
    last = new int[MAX_V], q = new int[MAX_V], dist = new int[MAX_V],
    now = new int[MAX_V];
    fill(last, last + MAX_V, -1);
  }

  void clear() {
    fill(last, last + MAX_V, -1);
    E = 0;
  }

  void add_edge(int u, int v, int uv, int vu = 0) {
    to[E] = v, cap[E] = uv, next[E] = last[u];
    last[u] = E++;
    to[E] = u, cap[E] = vu, next[E] = last[v];
    last[v] = E++;
  }

  bool bfs() {
    fill(dist, dist + MAX_V, -1);
    head = tail = 0;

    q[tail] = t;
    ++tail;
    dist[t] = 0;

    while (head < tail) {
      int v = q[head];
      ++head;

      for (int e = last[v]; e != -1; e = next[e]) {
        if (cap[e ^ 1] > 0 && dist[to[e]] == -1) {
          q[tail] = to[e];
          ++tail;
          dist[to[e]] = dist[v] + 1;
        }
      }
    }

    return dist[s] != -1;
  }

  int dfs(int v, int f) {
    if (v == t)
      return f;

    for (int &e = now[v]; e != -1; e = next[e]) {
      if (cap[e] > 0 && dist[to[e]] == dist[v] - 1) {
        int ret = dfs(to[e], min(f, cap[e]));
        if (ret > 0) {
          cap[e] -= ret;
          cap[e ^ 1] += ret;
          return ret;
        }
      }
    }
    return 0;
  }

  long long max_flow(int source, int sink) {
    s = source;
    t = sink;
    long long f = 0;
```

```
    int x;

    while (bfs()) {
      for (int i = 0; i < MAX_V; ++i)
        now[i] = last[i];

      while (true) {
        x = dfs(s, INT_MAX);
        if (x == 0)
          break;
        f += x;
      }
    }

    return f;
  }
} G;

int main() {
  int V, E, u, v, c;
  scanf("%d %d", &V, &E);

  G = flow_graph(V, E);

  for (int i = 0; i < E; ++i) {
    scanf("%d %d %d", &u, &v, &c);
    G.add_edge(u - 1, v - 1, c, c);
  }

  printf("%lld\n", G.max_flow(0, V - 1));
  return 0;
}
```

Trie.cpp

```
struct node {
  vector<node *> ch;
  bool isWord = false;
  int cnt = 0;

  node() { ch = vector<node *>(26,
       nullptr); }

  void insert(string &s, int idx) {
    cnt++;
    if (idx == sz(s)) {
      isWord = true;
      return;
    }
    int edge = s[idx] - 'a';
    if (!ch[edge])
      ch[edge] = new node();
    ch[edge]->insert(s, idx + 1);
  }

  bool find(string &s, int idx) {
    if (idx == sz(s))
      return isWord;
    int edge = s[idx] - 'a';
    if (!ch[edge])
      return false;
    return ch[edge]->find(s, idx + 1);
  }

  int count(string &s, int idx) {
    if (idx == sz(s))
      return cnt;
    int edge = s[idx] - 'a';
    if (!ch[edge])
      return 0;
    return ch[edge]->count(s, idx + 1);
  }
};
```

Math

BinaryExpo.cpp

```
ll b_exp(ll a, ll b) {
  if (b == 0 or a == 1)
    return 1;
  return ((ll)(b % 2 == 1 ? a : 1) *
       (ll)expo((a * a) % MOD, b / 2)) %
       MOD;
}
```

CombinationUnderMod.cpp

```
vlong fact(3e5);

ll MOD = 1e9 + 7;

fact[0] = 1;
rep(i, 1, sz(fact)) fact[i] = (i * fact[i
     - 1]) % MOD;

ll C(int n, int k) {
  if (k > n)
    return 0;
  return ((fact[n] * inverse_mod(fact[k])
       % MOD * inverse_mod(fact[n - k]))
       % MOD;
}

ll inv_mod(int n) { return b_exp(n, MOD -
     2); }
```

MatrixExpo.cpp

```
const ll MOD = 1e9 + 7;

vvlong multMat(vvlong const &a, vvlong
     const &b) {
  int n = sz(a);
  vvlong ans = vvint(n, vint(n));
  rep(i, 0, n) {
    rep(j, 0, n) {
      rep(k, 0, n) { ans[i][j] +=
           (a[i][k] * b[k][j]) % MOD; }
      ans[i][j] %= MOD;
    }
  }
  return ans;
}

vvlong matrixExpo(vvlong const &base, ll
     n) {
  if (n == 1) {
    return base;
  }

  vvlong matrix = matrixExpo(base, n / 2);
  if (n % 2 == 0) {
    return multMat(matrix, matrix);
  } else {
    return multMat(multMat(matrix,
         matrix), base);
  }
}
```

PrimeFactorization.cpp

```
set<int> pf(int n) {
  set<int> f;
  for (int i = 2; i * i <= n; i++)
    while (n % i == 0)
      f.insert(i), n /= i;
  if (n > 1)
    f.insert(n);
  return f;
}

vint factor(ll num) {
  vint facs;
  for (int div = 2; div * div < num;
       div++) {
    // OR loop through all primes
    while (num % div == 0) {
      num /= div;
      facs.pb(div);
    }
  }
```

```
  return facs;
}
```

String

InverseMod+RollingHash.cpp

```
ll compute_hash(const string &s) {
  int p = 31;
  ll hash_value = 0;
  ll p_pow = 1;
  for (char c : s) {
    hash_value = (hash_value + (c - 'a' +
         1) * p_pow) % MOD;
    p_pow = (p_pow * p) % MOD;
  }
  return hash_value;
}

int suffixCount(std::string S, int L) {
  string suffix = S.substr(sz(S) - L,
       sz(S));
  ll problem = compute_hash(suffix);

  string substring = S.substr(0, L);
  ll start = compute_hash(substring);

  ll count = 0;
  if (start == problem)
    count++;

  rep(i, L, sz(S)) {
    start -= (S[i - L] - 'a' + 1);
    start = start * inv_mod(31) % MOD;
    start += (S[i] - 'a' + 1) *
         (ll)pow(31, L - 1) % MOD;

    if (start == problem)
      count++;
  }
  return count;
}
```