COMILLA UNIVERSITY

CoU_Expecto_Patronum

NUMBER THEORY

Binary Exponentiation.cpp

```
// (a^b)%mod without overflow
#define long long ll
ll bin_expo(ll a, ll b, ll mod) {
   ll result = 1;
   while (b > 0) {
      if (b % 2 == 1) result = (result * a) % mod;
      a = (a * a) % mod;
      b /= 2;
   }
   return result % mod;
}
```

Fibonacchi in Logn.cpp

```
vector<vector<ll>> identityMatrix({
   {1, 0},
   {0, 1}
});
vector<vector<ll>> result({
   {0, 0},
   {0, 0}
});
void multiply(vector<vector<ll>> a, vector<vector<ll>>
b){
   for(auto &i : result){
      for(auto &j : i){
         j = 0;
      }
   }
   for(int i = 0; i < 2; i++){
      for(int j = 0; j < 2; j++){
         for(int k = 0; k < 2; k++){
            result[i][k] += a[i][j] * b[j][k];
            result[i][k] %= MOD;
         }
      }
   }
}
void matrixExpo(vector<vector<ll>> matrix, ll n){
   if(n == 0){
      result = identityMatrix;
      return;
   }
   matrixExpo(matrix, n / 2);
   multiply(result, result);
   if(n & 1){
      multiply(result, matrix);
   }
}
ll nthFibonacciNumber(ll n){
// n <= 1e18, F0 = 0, F1 = 1, F2 = 1
   if(n <= 1)
      return n;
   vector<vector<ll>> baseMatrix({
      {1, 1},
      {1, 0}
   });
   multiply(baseMatrix, identityMatrix);
   matrixExpo(baseMatrix, n - 1);
   return result[0][0];
}
```

Gcd.cpp

```
int gcd (int a, int b) {
   while (b) {
      a %= b;
      swap(a, b);
   }
   return a;
}
```

Miller Rabin Test.cpp

```
ull modmul(ull a, ull b, ull M) {
   ll ret = a * b - M * ull(1.L / M * a * b);
   return ret + M * (ret < 0) - M * (ret >= (ll)M);
}
ull modpow(ull b, ull e, ull mod) {
```

```
    ull ans = 1;
    for (; e; b = modmul(b, b, mod), e /= 2)
    if (e & 1) ans = modmul(ans, b, mod);
    return ans;
}
bool isPrime(ull n) {
    if (n < 2 || n % 6 % 4 != 1) return (n | 1) == 3;
    ull A[] = {2, 325, 9375, 28178, 450775, 9780504,
1795265022},
        s = __builtin_ctzll(n-1), d = n >> s;
    for (ull a : A) {
        ull p = modpow(a%n, d, n), i = s;
        while (p != 1 && p != n - 1 && a % n && i--)
            p = modmul(p, p, n);
        if (p != n-1 && i != s) return 0;
    }
    return 1;
}
```

Number of Divisor for test case.cpp

```
const ll N = 10000004;
bool mark[N];
vector<ll> prime;
void siv() {
    // Initialize the sieve
    for (ll i = 2; i < N; i++) {
        mark[i] = true;
    }
    for (ll i = 2; i * i < N; i++) {
        if (mark[i]) {
            for (ll j = i * i; j < N; j += i) {
                mark[j] = false;
            }
        }
    }
    // Store all primes
    for (ll i = 2; i < N; i++) {
        if (mark[i]) prime.push_back(i);
    }
}

void solve() {
    ll n; cin >> n;
    ll ans = 1;
    for (ll i = 0; i < prime.size(); i++) {
        ll p = prime[i];
        if (p * p * p > n) break;
        ll count = 1;
        while (n % p == 0) {
            n /= p;
            count++;
        }
        ans *= count;
    }
    if (n > 1) {
        ll sqrt_n = sqrt(n);
        if (sqrt_n * sqrt_n == n &&
binary_search(prime.begin(), prime.end(), sqrt_n))
ans *= 3;
        else if (binary_search(prime.begin(), prime.end(),
n))
                    ans *=2;
        else ans *= 4;
    }
    cout << ans << endl;
}
```

Number of divisor.cpp

```
long long numberOfDivisors(long long num) {
    long long total = 1;
    for (int i = 2; (long long)i * i <= num; i++) {
        if (num % i == 0) {
            int e = 0;
            do {
                e++;
                num /= i;
            } while (num % i == 0);
            total *= e + 1;
        }
    }
    if (num > 1) {
        total *= 2;
    }
    return total;   }
```

Sum of divisor.cpp

```
long long SumOfDivisors(long long num) {
    long long total = 1;
    for (int i = 2; (long long)i * i <= num; i++) {
        if (num % i == 0) {
            int e = 0;
            do {
                e++;
                num /= i;
            } while (num % i == 0);

            long long sum = 0, pow = 1;
            do {
                sum += pow;
                pow *= i;
            } while (e-- > 0);
            total *= sum;
        }
    }
    if (num > 1) total *= (1 + num);
    return total;
}
```

Phi function.cpp

```
const int N = 1e5 + 9;
int phi[N];
void totient() {
  for (int i = 1; i < N; i++) phi[i] = i;
  for (int i = 2; i < N; i++) {
    if (phi[i] == i) {
      for (int j = i; j < N; j += i) phi[j] -= phi[j] / i;
    }
  }
}
int phi(int n) {
    int result = n;
    for (int i = 2; i * i <= n; i++) {
        if (n % i == 0) {
            while (n % i == 0)
                n /= i;
            result -= result / i;
        }
    }
    if (n > 1)
        result -= result / n;
    return result;
}
```

Sieve of eratosthenes.cpp

```
const int N = 1e6 + 9;
vector<int> primes;
bool is_prime[N];
// use bitset<N> is_prime; to have O(N/64) memory complexity
void sieve_v0() {
  for (int i = 2; i < N; i++) {
    is_prime[i] = true;
  }
  for (int i = 2; i * i < N; i++) {
    if (is_prime[i]) {
      for (int j = i * i; j < N; j += i) {
        is_prime[j] = false;
      }
    }
  }
  for (int i = 2; i < N; i++) {
    if (is_prime[i]) {
      primes.push_back(i);
    }
  }
}
```

Sieve with smallest prime factors (spf):

```
int spf[N];
void sieve() {
  for (int i = 2; i < N; i++) {
    spf[i] = i;
  }
  for (int i = 2; i * i < N; i++) {
    if (spf[i] == i) {
      for (int j = i * i; j < N; j += i) {
```

```
      spf[j] = min(spf[j], i);
     }
    }
   }
  for (int i = 2; i < N; i++) {
   if (spf[i] == i) {
    primes.push_back(i);
   }
  }
}
```

Dynamic Programming

0/1 Knapsack: complexity O(n*w)

```
int n, w; cin >> n >> w;
  int weight[n], val[n];
  for (int i = 0; i < n; i++) cin >> weight[i];
  for (int i = 0; i < n; i++) cin >> val[i];
  int dp[w + 1] = {};
  for (int i = 0; i < n; i++)
    for (int j = w; j >= weight[i]; j--)
      dp[j] = max(dp[j], dp[j - weight[i]] + val[i]);
  cout << dp[w] << "\n";
```

Coin Change: Time complexity: O(nx)

```
 int n, x; cin >> n >> x;
  int arr[n];
  for (int i = 0; i < n; i++) cin >> arr[i];
  //1) Find the number of distinct ways to sum up to x
  vector<int> dp(x + 1, 0);
  dp[0] = 1;
  for (int i = 1; i <= x; i++)
    for (int j : arr)
      if (i - j >= 0)
        dp[i] = (dp[i] + dp[i - j]) % MOD;
  cout << dp[x] << "\n";
  // 2) Find the number of distinct ordered ways to
sum up to x
  vector<int> dp(x + 1, 0);
  dp[0] = 1;
  for (int j : arr)
    for (int i = 1; i <= x; i++)
      if (i - j >= 0)
        dp[i] = (dp[i] + dp[i - j]) % MOD;
  cout << dp[x] << "\n";
  // 3) Find the minimum number of coins required to
sum up to x
  vector<int> dp(x + 1, INF);
  dp[0] = 0;
  for (int i = 1; i <= x; i++)
    for (int j : arr)
      if (i - j >= 0)
        dp[i] = min(dp[i], dp[i - j] + 1);
  cout << (dp[x] == INF ? -1 : dp[x]) << "\n";
```

DIGIT DP: O(log^2(n))

```
// Find the sum of the digits of the numbers between
a and b (0 <= a <= b <= 1e9)
vector<int> num;
ll dp[10][9 * 10][2];
ll memo(int pos, int sum, int flag) {
  if (pos == num.size()) return sum;
  if (dp[pos][sum][flag] != -1) return
dp[pos][sum][flag];

  ll res = 0;
  int lmt = (flag) ? 9 : num[pos];
  for (int i = 0; i <= lmt; i++) {
    int next_flag = (i < lmt) ? 1 : flag;
    res += memo(pos + 1, sum + i, next_flag);
  }
  return dp[pos][sum][flag] = res;
}
ll calc(int n) {
  num.clear();
  while (n) {
    num.push_back(n % 10);
    n /= 10;
  }
  reverse(num.begin(), num.end());
  memset(dp, -1, sizeof dp);
  return memo(0, 0, 0);
```

```
}
void solve() {
   int a, b; cin >> a >> b;
   if (a == -1 && b == -1) return;
   cout << calc(b) - calc(a - 1) << "\n";

}
```

FROG2

```
int no_of_stones, no_of_steps, nnn;
   cin >> no_of_stones >> no_of_steps;
   vector <int> height(no_of_stones + 1, 0);
   for(int i = 1; i <= no_of_stones; i++)
      cin >> height[i];
   const int oo = 1e9;
   vector <int> minimum_cost(no_of_stones + 1, oo);
   minimum_cost[1] = 0;
   for(int i = 2; i <= no_of_stones; i++)
   {
      for(int j = i - 1; j >= max(1, i - no_of_steps); j--)
      {
         minimum_cost[i] = min(minimum_cost[i],
minimum_cost[j] + abs(height[i] - height[j]));
      }
   }
   cout << minimum_cost[no_of_stones];
```

LCS: Time complexity: O(nm)

```
// Given 2 strings x and y of length n and m, find the
the longest common subsequence (LCS)
string x, y; cin >> x >> y;
int n = x.size(), m = y.size();
int dp[n + 1][m + 1] = {};
for (int i = 1; i <= n; i++) {
   for (int j = 1; j <= m; j++) {
      if (x[i - 1] == y[j - 1])
         dp[i][j] = max(dp[i][j], dp[i - 1][j - 1] + 1);
      else
         dp[i][j] = max({dp[i][j], dp[i - 1][j], dp[i][j - 1]});
   }
}
cout << dp[n][m];
```

LIS: Time complexity: O(nlogn)

```
// Find the longest increasing subsequence (LIS) in the
array of length n
int n; cin >> n;
vector<int> dp;
for (int i = 0; i < n; i++)
{
   int x;
   cin >> x;
   auto it = lower_bound(dp.begin(), dp.end(), x);
   if (it == dp.end())
      dp.push_back(x);
   else
      *it = x;
}
cout << dp.size() << "\n";
```

MIN cost to go 1,1 to x,y and Number of ways ways

```
int X, Y; cin>>X>>Y;
   vec<vi> Cost(X,
vector<int>(Y)),dp(x,y),Numway(x,y);
   for (int i = 0; i < X; ++i)
      for (int j = 0; j < Y; ++j)
         cin >> Cost[i][j];
   dp[0][0] = Cost[0][0];
   NumWays[0][0] = 1;
   for (int j = 1; j < Y; ++j) {
      dp[0][j] = dp[0][j - 1] + Cost[0][j];
      NumWays[0][j] = 1;
   }
   for (int i = 1; i < X; ++i) {
      dp[i][0] = dp[i - 1][0] + Cost[i][0];
      NumWays[i][0] = 1;
   }
```

```
    for (int i = 1; i < X; ++i) {
        for (int j = 1; j < Y; ++j) {
            dp[i][j] = min(dp[i - 1][j], dp[i][j - 1]) + Cost[i][j];
            NumWays[i][j] = NumWays[i - 1][j] +
NumWays[i][j - 1];
        }
    }
    cout << dp[X - 1][Y - 1] << endl;
    cout << NumWays[X - 1][Y - 1] << endl;
```

VACATION:

```
// given ai,bi,ci
//      ai,bi,ci
// find max
int main() {
    int n;
    cin >> n;
    int a[n], b[n], c[n];
    for (int i = 0; i < n; i++) cin >> a[i] >> b[i] >> c[i];
    int dp[n][3];
    dp[0][0] = a[0];
    dp[0][1] = b[0];
    dp[0][2] = c[0];
    for (int i = 1; i < n; i++) {
        dp[i][0] = a[i] + max(dp[i - 1][1], dp[i - 1][2]);
        dp[i][1] = b[i] + max(dp[i - 1][0], dp[i - 1][2]);
        dp[i][2] = c[i] + max(dp[i - 1][1], dp[i - 1][0]);
    }
    cout << max({dp[n - 1][0], dp[n - 1][1], dp[n - 1][2]});
    return 0;
}
```

GRAPH/TREE
BELLMAN FORD

```
vector<int> bellman_ford(int V, vector<vector<int>>&
edges, int S) {
    vector<int> dist(V, 1e8);
    dist[S] = 0;
    for (int i = 0; i < V - 1; i++) {
        for (auto it : edges) {
            int u = it[0];
            int v = it[1];
            int wt = it[2];
            if (dist[u] != 1e8 && dist[u] + wt < dist[v]) {
                dist[v] = dist[u] + wt;
            }
        }
    }
    for (auto it : edges) {
        int u = it[0];
        int v = it[1];
        int wt = it[2];
        if (dist[u] != 1e8 && dist[u] + wt < dist[v]) {
            return {-1};
        }
    }
    return dist;
}
int main() {
    int V, E; cin>>V>>E;
    vector<vector<int>> edges(E, vector<int>(3));
    for (int i = 0; i < E; i++)
        cin >> edges[i][0] >> edges[i][1] >> edges[i][2];
    int S; cin >> S;
    vi dist = bellman_ford(V, edges, S);
    for (auto d : dist) cout << d << " ";
    cout << endl;
    return 0;
}
```

BRIDGES and Articulation POINT : Time complexity: O(n + m)

```
// Given an undirected graph, find all bridges and
articulation points
// Time complexity: O(n + m)
const int MAX_N = 1e5 + 1;
int n, m, dfsCounter;
int dfs_num[MAX_N], dfs_low[MAX_N],
visited[MAX_N];
vector<int> adj[MAX_N];
void dfs(int u, int p = -1) {
    dfs_num[u] = dfs_low[u] = dfsCounter++;
```

```
    visited[u] = 1;
    int num_child = 0;
    for (int v : adj[u]) {
        if (v == p) continue;
        // back edge
        if (visited[v]) dfs_low[u] = min(dfs_low[u],
dfs_num[v]);
        // tree edge
        else {
            dfs(v, u);
            dfs_low[u] = min(dfs_low[u], dfs_low[v]);
            num_child++;
            if (dfs_low[v] > dfs_num[u])
                cout << "Edge " << u << "-" << v << " is a
bridge\n";
            if (dfs_low[v] >= dfs_num[u] && p != -1)
                cout << "Node " << u << " is an articulation
point\n";
        }
    }
    // special case: the root node is an articulation point
if it has more than 1 child
    if (p == -1 && num_child > 1)
        cout << "Node " << u << " is an articulation
point\n";
}
void solve() {
    cin >> n >> m;
    for (int i = 0; i < m; i++) {
        int u, v; cin >> u >> v;
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
    memset(dfs_low, -1, sizeof dfs_low);
    memset(dfs_num, -1, sizeof dfs_num);
    for (int i = 1; i <= n; i++)
        if (!visited[i])
            dfs(i);
}
    /*
    Example input:
        12 16
        1 3
        3 5
        5 7
        .......
    Expected output:
        Edge 4-10 is a bridge
        Node 4 is an articulation point
        Edge 2-4 is a bridge
        ....
```

DIAMETER OF TREE

```
const int N = 2e5 + 9;
vector<int> g[N];
int farthest(int s, int n, vector<int> &d) {
  static const int inf = N;
  d.assign(n + 1, inf); d[s] = 0;
  vector<bool> vis(n + 1);
  queue<int> q; q.push(s);
  vis[s] = 1; int last = s;
  while (!q.empty()) {
    int u = q.front(); q.pop();
    for (int v: g[u]) {
      if (vis[v]) continue;
      d[v] = d[u] + 1;
      q.push(v); vis[v] = 1;
    }
    last = u;
  }
  return last;
}
int32_t main() {
  int n; cin >> n;
  for (int i = 1; i < n; i++) {
    int u, v; cin >> u >> v;
    g[u].push_back(v);
    g[v].push_back(u);
  }
  vector<int> dx, dy;
  int x = farthest(1, n, dx);
  int y = farthest(x, n, dx);
  farthest(y, n, dy);
  for (int i = 1; i <= n; i++) {
    cout << max(dx[i], dy[i]) << ' ';
```

```
 }
 cout << '\n'; return 0; }
```

DIJKSTRA

```
const int N = 3e5 + 9, mod = 998244353;
int n, m;
vector<pair<int, int>> g[N], r[N];
vector<long long> dijkstra(int s, int t, vector<int>
&cnt) {
  const long long inf = 1e18;
  priority_queue<pair<long long, int>,
vector<pair<long long, int>>, greater<pair<long long,
int>>> q;
  vector<long long> d(n + 1, inf);
  vector<bool> vis(n + 1, 0);
  q.push({0, s});
  d[s] = 0;
  cnt.resize(n + 1, 0); // number of shortest paths
  cnt[s] = 1;
  while(!q.empty()) {
    auto x = q.top();
    q.pop();
    int u = x.second;
    if(vis[u]) continue;
    vis[u] = 1;
    for(auto y: g[u]) {
      int v = y.first;
      long long w = y.second;
      if(d[u] + w < d[v]) {
        d[v] = d[u] + w;
        q.push({d[v], v});
        cnt[v] = cnt[u];
      } else if(d[u] + w == d[v]) cnt[v] = (cnt[v] + cnt[u]) %
mod;
    }
  }
  return d;
}
int u[N], v[N], w[N];
int32_t main() {
  int s, t;
  cin >> n >> m >> s >> t;
  for(int i = 1; i <= m; i++) {
    cin >> u[i] >> v[i] >> w[i];
    g[u[i]].push_back({v[i], w[i]});
    r[v[i]].push_back({u[i], w[i]});
  }
  vector<int> cnt1, cnt2;
  auto d1 = dijkstra(s, t, cnt1);
  auto d2 = dijkstra(t, s, cnt2);
  long long shortest_distance = d1[t];
  int number_of_ways = cnt1[t];
  cout << shortest_distance << '\n';
  cout << number_of_ways << '\n';
  return 0;
}
```

DSU

```
struct DSU {
  vector<int> par, rnk, sz;
  int c;
  DSU(int n) : par(n + 1), rnk(n + 1, 0), sz(n + 1, 1), c(n) {
    for (int i = 1; i <= n; ++i) par[i] = i;
  }
  int find(int i) {
    return (par[i] == i ? i : (par[i] = find(par[i])));
  }
  bool same(int i, int j) {
    return find(i) == find(j);
  }
  int get_size(int i) {
    return sz[find(i)];
  }
  int count() {
    return c;    //connected components
  }
  int merge(int i, int j) {
    if ((i = find(i)) == (j = find(j))) return -1;
    else --c;
```

```
    if (rnk[i] > rnk[j]) swap(i, j);
    par[i] = j;
    sz[j] += sz[i];
    if (rnk[i] == rnk[j]) rnk[j]++;
    return j;
  }
};
```

FLOYD WARSHALL

```
#include<bits/stdc++.h>
using namespace std;
const int N = 105;
int d[N][N];
int main() {
  int n = 10;
  for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= n; j++) {
      if (i != j) {
        d[i][j] = 1e9;
      }
    }
  }
  for (int k = 1; k <= n; ++k) {
    for (int i = 1; i <= n; ++i) {
      for (int j = 1; j <= n; ++j) {
        d[i][j] = min(d[i][j], d[i][k] + d[k][j]);
      }
    }
  }
  return 0;
}
```

TOPOLOGICAL SORT

```
const int N = 1e5 + 9;
vector<int> g[N];
bool vis[N];
vector<int> ord;
void dfs(int u) {
  vis[u] = true;
  for (auto v: g[u]) {
    if (!vis[v]) {
      dfs(v);
    }
  }
  ord.push_back(u);
}
reverse(ord.begin(), ord.end());
```

DATA STRUCTURE

KADANE

```
int ans = LONG_LONG_MIN, sum = 0;
for (int i = 0; i < n; i++)
{
  sum = max(v[i], sum + v[i]);
  ans = max(ans, sum);
}
ans = max(ans, sum);
```

KMP : Time complexity: O(n + m)

```
// Given a string s (with length n) and a pattern p (with length m), find all occurrence of p in s
// f[i] = length of the longest proper prefix of the substring s[0...i] which is also a suffix of this substring
vector<int> prefix_func(string s) {
  int n = s.size();
  vector<int> f(n);
  for (int i = 1; i < n; i++) {
    int j = f[i - 1];
    while (j && s[i] != s[j]) j = f[j - 1];
    f[i] = j + (s[i] == s[j]);
  }
  return f;
}
int cnt_occ(string s, string t) {
  string ts = t + "#" + s;
  int n = t.size(), m = s.size(), nm = ts.size();
  auto f = prefix_func(ts);
```

```
  int res = 0;
  for (int i = n + 1; i < nm; i++) res += (f[i] == n);
  return res;
}
void solve() {
  string s, t; cin >> s >> t;
  cout << cnt_occ(s, t) << "\n";
}
```

SEGMENT TREE POINT UPDATE RANGE QUERY

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

const int N = 3e5 + 9;
int a[N];
struct ST {
  int t[4 * N];
  static const int inf = 1e9;
  ST() {
    memset(t, 0, sizeof t);
  }
  void build(int n, int b, int e) {
    if (b == e) {
      t[n] = a[b];
      return;
    }
    int mid = (b + e) >> 1, l = n << 1, r = l | 1;
    build(l, b, mid);
    build(r, mid + 1, e);
    t[n] = max(t[l], t[r]);
  }
  void upd(int n, int b, int e, int i, int x) {
    if (b > i || e < i) return;
    if (b == e && b == i) {
      t[n] = x;
      // to increase t[n]+=x
      return;
    }
    int mid = (b + e) >> 1, l = n << 1, r = l | 1;
    upd(l, b, mid, i, x);
    upd(r, mid + 1, e, i, x);
    t[n] = max(t[l], t[r]);
  }
  int query(int n, int b, int e, int i, int j) {
    if (b > j || e < i) return -inf;
    if (b >= i && e <= j) return t[n];
    int mid = (b + e) >> 1, l = n << 1, r = l | 1;
    int L = query(l, b, mid, i, j);
    int R = query(r, mid + 1, e, i, j);
    return max(L, R);
  }
} t;
```

SEGMENT TREE RANGE UPDATE RANGE QUERY

```
const int N = 5e5 + 9;
int a[N];
struct ST {
 #define lc (n << 1)
 #define rc ((n << 1) | 1)
 long long t[4 * N], lazy[4 * N];
 ST() {
  memset(t, 0, sizeof t);
  memset(lazy, 0, sizeof lazy);
 }
 inline void push(int n, int b, int e) {
  if (lazy[n] == 0) return;
  t[n] = t[n] + lazy[n] * (e - b + 1);
  if (b != e) {
   lazy[lc] = lazy[lc] + lazy[n];
   lazy[rc] = lazy[rc] + lazy[n];
  }
  lazy[n] = 0;
 }
 inline long long combine(long long a,long long b) {
  return a + b;
 }
 inline void pull(int n) {
  t[n] = t[lc] + t[rc];
 }
 void build(int n, int b, int e) {
  lazy[n] = 0;
  if (b == e) {
```

```
      t[n] = a[b];
      return;
    }
    int mid = (b + e) >> 1;
    build(lc, b, mid);
    build(rc, mid + 1, e);
    pull(n);
  }
  void upd(int n, int b, int e, int i, int j, long long v) {
    push(n, b, e);
    if (j < b || e < i) return;
    if (i <= b && e <= j) {
      lazy[n] = v; //set lazy
      push(n, b, e);
      return;
    }
    int mid = (b + e) >> 1;
    upd(lc, b, mid, i, j, v);
    upd(rc, mid + 1, e, i, j, v);
    pull(n);
  }
  long long query(int n, int b, int e, int i, int j) {
    push(n, b, e);
    if (i > e || b > j) return 0; //return null
    if (i <= b && e <= j) return t[n];
    int mid = (b + e) >> 1;
    return combine(query(lc, b, mid, i, j), query(rc, mid
+ 1, e, i, j));
  }
};
```

SPARSE TABLE:

```
const int N = 1e5 + 9;
int t[N][18], a[N];
void build(int n) {
  for(int i = 1; i <= n; ++i) t[i][0] = a[i];
  for(int k = 1; k < 18; ++k) {
    for(int i = 1; i + (1 << k) - 1 <= n; ++i) {
      t[i][k] = min(t[i][k - 1], t[i + (1 << (k - 1))][k - 1]);
    }
  }
}
int query(int l, int r) {
  int k = 31 - __builtin_clz(r - l + 1);
  return min(t[l][k], t[r - (1 << k) + 1][k]);
}
build(n);
```

TERNARY SEARCH:

```
int lo=0,hi=1e12,ans=LONG_LONG_MAX;
while (hi-lo>4)
{
    int m1 = (hi+lo)>>1LL;
    int m2 = (hi+lo)/2 + 1;

    int a1 = koro(v, m1);
    int a2 = koro(v, m2);
    if(a1>a2) lo = m1;
    else hi = m2;
    // for max
    if(a1>a2) hi = m1;
    else lo = m2;
}
```

TERNARY SEARCH ON DOUBLE

```
double find_max_volume(double S) {
    double left = 0.0, right = sqrt(S);
    double epsilon = 1e-7;
    while (right - left > epsilon) {
        double mid1 = left + (right - left) / 3.0;
        double mid2 = right - (right - left) / 3.0;
        double volume1 = calculate_volume(mid1, S);
        double volume2 = calculate_volume(mid2, S);
        if (volume1 > volume2) {
            right = mid2;
        } else {
            left = mid1;
        }
    }
    return calculate_volume((left + right) / 2.0, S);
```

```
}
cout << fixed << setprecision(4) <<
max_volume << endl;
```

Segment tree point update min val and freq

```
const int N = 1e5+5;
int arr[N];
struct node {
   int ele,freq;
};
node tr[4*N];
node merge(node left, node right) {
   node notun;
   int m = min(left.ele, right.ele);
   int f = 0;
   if(m==left.ele) f+=left.freq;
   if(m==right.ele) f+=right.freq;
   notun.ele=m;
   notun.freq=f;
   return notun;
}
void build(int idx, int b, int e) {
   if(b==e) {
      tr[idx].ele=arr[b],
      tr[idx].freq=1;
      return;
   }
   int left = idx*2+1, right = idx*2+2;
   int mid = (b+e)>>1LL;
   build(left, b, mid);
   build(right, mid+1, e);
   tr[idx]=merge(tr[left], tr[right]);
}
node query(int idx, int b, int e, int l, int r) {
   // ! overlap
   if(e<l || r<b) {
      node notun;
      notun.ele=inf;
      notun.freq=0;
      return notun;
   }
   // full overlap
   if(b>=l && e<=r) return tr[idx];
   int mid = (b+e)>>1LL;
   return merge(
      query(2*idx+1, b, mid, l, r),
      query(2*idx+2, mid+1, e, l, r)
   );
}
void update(int idx, int b, int e, int pos, int val){
   if(b==e) {
      tr[idx].ele=val;
      tr[idx].freq=1;
      return;
   }
   int mid = (b+e)>>1LL;
   if(pos<=mid) update(2*idx+1, b, mid, pos, val);
   else update(2*idx+2, mid+1, e, pos, val);
   tr[idx]=merge(tr[2*idx+1], tr[2*idx+2]);
}
update(0,0,n-1,pos,val);
node koto = query(0,0,n-1,l,r);
cout<<koto.ele<<" "<<koto.freq<<endl;
```

NUMBER OF RIGHT BRACKET SEQUENCE

```
string s;
const int N = 1e6+2;
struct node {
   int open,close,full;
};
node tree[4*N];
node merge(node l, node r) {
   node notun;
   notun.full = l.full+r.full+min(l.open, r.close);
   notun.open = l.open+r.open-min(l.open, r.close);
   notun.close = l.close+r.close-min(l.open, r.close);
   return notun;
}
node query(int ind, int b, int e, int i, int j) {
   if(j<b || e<i) {
```

```
    node ans;
    ans.open=0,ans.close=0,ans.full=0;
    return ans;
  }
  if(b>=i && e<=j) return tree[ind];
  int mid = (b+e)>>1;
  return merge(
    query(2*ind+1, b, mid, i, j),
    query(2*ind+2, mid+1, e, i, j)
  );
}
void build(int ind, int b, int e) {
  if(b==e) {
    if(s[b]=='(')
tree[ind].open=1,tree[ind].close=0,tree[ind].full=0;
    else
tree[ind].open=0,tree[ind].close=1,tree[ind].full=0;
    return;
  }
  int left = ind*2+1;
  int right = ind*2+2;
  int mid = (b+e)>>1;
  build(left, b,mid);
  build(right, mid+1,e);
  tree[ind]=merge(tree[left],tree[right]);
}
```

Largest subarray having sum less than equal to k

```
int subarray_start = 0;
  int subarray_end = 0;
  int subarray_sum = 0;
  int max_len = -1;
  for (int i : s) {
    subarray_sum += i;
    subarray_end++;
    while (subarray_sum > k) {
      subarray_sum -= s[subarray_start];
      subarray_start++;
    }
    max_len = max(max_len, subarray_end -
subarray_start);
  }
  return max_len;
```

Longest sub-array having sum k

```
 unordered_map<int, int> sum_index_map;
  int maxLen = 0;
  int prefix_sum = 0;
  for (int i = 0; i < N; ++i) {
    prefix_sum += A[i];
    if (prefix_sum == K) {
      maxLen = i + 1;
    }
    else if (sum_index_map.find(prefix_sum - K) !=
sum_index_map.end()) {
      maxLen = max(maxLen, i -
sum_index_map[prefix_sum - K]);
    }

    if (sum_index_map.find(prefix_sum) ==
sum_index_map.end()) {
      sum_index_map[prefix_sum] = i;
    }
  }
  return maxLen;
```

Longest subarray lenght such that sum is divisible by k

```
int n,k; cin>>n>>k;
  vector<int> v(n);
  for (int i = 0; i < n; i++) cin>>v[i];
  int ans = -1,csum=0
  map<int,int> mp;
  for (int i = 0; i < n; i++)
  {
    csum += v[i];
    int rem = csum%k;
    if(rem==0)
      ans = max(ans, i+1);
    else if(mp.find(rem)!=mp.end())
      ans = max(ans, i-mp[rem]);
```

```
    else mp[rem]=i;
  }
  if(ans==-1) cout<<ans<<endl;
  else cout<<n-ans<<endl;
```

The number of trailing zeros in the factorial n!

```
ll n; cin>>n;
int five=0; int ache = 5;
  while (ache<=n) {
    five+=n/ache; ache*=5;
  }
  cout<<five<<endl;
```

Number of subbarray having sum x

```
int n,k; cin>>n>>k;
  vl v(n); inv(v);
  map<int,int> mp;
  int csum = 0;
  int ans = 0;
  for (int i = 0; i < n; i++)
  {
    csum+=v[i];
    if(mp.find(csum-k)!=mp.end()) {
      ans+=mp[csum-k];
    }
    mp[csum]++;
  }
  ans += mp[k];
  cout<<ans<<endl;
```

Number of subarray sum divisible by x

```
int n;  cin>>n; vl v(n);
  inv(v);
  map<int,int> mp;
  int sum = 0;
  int ans = 0;
  for (int i = 0; i < n; i++)
  {
    sum+=v[i];
    int rm = (sum%n+n)%n;

    if(rm==0) ans++;
    mp[rm]++;
  }
  for(auto &it: mp) ans+=(it.S*(it.S-1))/2;
  cout<<ans<<endl;
```

NUMBER OF INVERSION

```
// Inversion: i < j and A[i] > A[j]
// find the number of inversions of A.
int n;cin >> n;
int a[n + 1];
for (int i = 1; i <= n; i++)
  cin >> a[i];
o_set<int> se;
long long ans = 0;
for (int i = n; i >= 1; i--)
  ans += se.order_of_key(a[i]);
  se.insert(a[i]);
cout << ans << '\n';
```

ORDERED SET

```
#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
using namespace std;
template <typename T> using o_set = tree<T,
null_type, less<T>, rb_tree_tag,
tree_order_statistics_node_update>;
template <typename T, typename R> using o_map =
tree<T, R, less<T>, rb_tree_tag,
tree_order_statistics_node_update>;
int main() {
  int i, j, k, n, m;
  o_set<int>se;
  se.insert(1);
  se.insert(2);
```

```
  cout << *se.find_by_order(0) << endl; ///k th element
  cout << se.order_of_key(2) << endl; ///number of elements less than k
  o_map<int, int>mp;
  mp.insert({1, 10});
  mp.insert({2, 20});
  cout << mp.find_by_order(0)->second << endl; ///k th element
  cout << mp.order_of_key(2) << endl; ///number of first elements less than k
  return 0;
}
```

THEOREM

```
a^b^c
b = power(b,c, mod-1);
cout<<power(a,b, mod)<<endl;

(1/a)%m = pow(a,m-2,m)
```

Chicken McNugget Theorem / Postage Stamp Problem / Frobenius Coin Problem

For any two relatively prime positive integers m,n.
The greatest integer that cannot be written in the form am + bn is (mn-m-n).
There are exactly (m - 1)(n - 1)/2 positive integers which cannot be expressed in the form am + bn

```
ios_base::sync_with_stdio(false); cin.tie(NULL); cout.tie(NULL)
```

PYTHON

```
import math
t = int(input())  # Number of test cases
 for i in range(1, t + 1):
   a, b = map(int, input().split())
   print(a + b)  # Sum
print(a - b)  # Difference
product = a * b  # Store the product of a and b
print(product)  # Product
print(a // b)  # Quotient (integer division)
print(a % b)  # Remainder
g = math.gcd(a, b)  # GCD
print(g)  # Print GCD
print(product // g)  # LCM
print(math.sqrt(a))  # Square root of a
print(math.sqrt(b))  # Square root of b
a, b, c, d = map(int, input().split())
if a * d == b * c:
   print("Equal")
else:
   print("Not Equal")
```

1.1 Primality Test

```
bool prime (int  N) {
    if (N < 2)
        return false;
    if (N <= 3)
        return true;
    if (N % 2 == 0)
        return false;

    for (int i = 3; i * i <= N; i += 2) {
        if (N % i == 0)
            return false;
    }
    return true;
}
```

1.2 Miller-Rabin Primality Test

Description: Deterministic Miller-Rabin primality test. Guaranteed to work for numbers up to 7·10e18; for larger numbers, use Python and extend A randomly.
Time: 7 times the complexity of a^b mod c.

```
typedef unsigned long long ull;
#define ll long long int

ull modmul(ull a, ull b, ull M) {
    ll ret = a * b - M * ull(1.L / M * a * b);
    return ret + M * (ret < 0) - M * (ret >= (ll)M);
}

ull modpow(ull b, ull e, ull mod) {
    ull ans = 1;
    for (; e; b = modmul(b, b, mod), e /= 2)
        if (e & 1) ans = modmul(ans, b, mod);
    return ans;
}

bool isPrime(ull n) {
    if (n < 2 || n % 6 % 4 != 1) return (n | 1) == 3;
    ull A[] = {2, 325, 9375, 28178, 450775, 9780504, 1795265022},
          s = __builtin_ctzll(n-1), d = n >> s;
    for (ull a : A) {
        ull p = modpow(a%n, d, n), i = s;
        while (p != 1 && p != n - 1 && a % n && i--)
            p = modmul(p, p, n);
        if (p != n-1 && i != s) return 0;
    }
    return 1;
}
```

1.3 Sieve of Eratosthenes

Description: Normal sieve for generating all primes numbers up to 1e8.

```
const int N = 100000000;
bool is_prime[N+1];
vector<int>prime;
void sieve ( ) {
    memset (is_prime, true,sizeof(is_prime));
    for (int i = 3; i * i <= N; i += 2) {
        if (is_prime[i]) {
            for (int j = 2*i; j <= N; j += i)
                is_prime[j] = false;
        }
    }
    prime.push_back(2);
    for (int i = 3; i <= N; i += 2) {
        if (is_prime[i])
            prime.push_back(i);
    }
}
```

1.4 Prime Factorization

Description: For first one, Generating Sieve of Eratosthenes.

```
map<int, int>mp;                    // Complexity: O(log(log(N))
void prime_factor (int  N) {
    int i = 0, pf = prime [0];
    while (pf * pf <= N) {
        while (N % pf == 0) {
            N /= pf;
            mp[pf]++;
        }
        pf = prime[i++];
    }
    if (N > 1)
        mp[N]++;
}
```

```
map<int,int>mp;                     // Complexity: O(Sqrt(N))
void prime_factor (int  N) {
    for (int i = 2; i * i <= N; i++) {
        if (N % i == 0) {
            while (N % i == 0) {
                N /= i;
                mp[i]++;
            }
        }
    }
    if (N > 1)
        mp[N]++;
}
```

1.5 Divisors

Description: Complexity: O(Sqrt(N))

```
set<int>st;
void Divisors (int  N) {
    for (int i = 1; i*i <= N; i++) {
        if (N % i == 0) {
            st.insert(i);    st.insert(N / i);
        }
    }
}
```

1.6 Number of Divisors

Description: Complexity: O(Sqrt(N))

```
int number_of_divisors (int  N) {
    int total = 1;
    for (int i = 2; i*i <= N; i++) {
        if (N % i == 0) {
            int e = 0;
            while (N % i == 0) {
                e++;   N /= i;
            }
            total *= e + 1;
        }
    }
    if (N > 1)   total *= 2;
    return total;
}
```

Description: For multiple test cases. (CSES-Counting Divisors)
At first, Generating Sieve of Eratosthenes up to 10^6.

```
#define all(v) v.begin(),v.end()
int number_of_divisors (int  n) {
    int ans = 1;
    for (int i = 0; i < prime.size(); i++) {
        int p = prime[i];
        if (p * p * p > n)
            break;
        int count = 1;
        while (n % p == 0) {
            n /= p; count++;
        }
        ans *= (count);
    }
    if (binary_search(all(prime), n)) ans *= 2;
    else if (binary_search(all(prime), sqrt(n))) ans *= 3;
    else if (n > 1) ans *= 4;
    return ans;
}
```

1.7 Sum of Divisors

Description: Complexity: O(Sqrt(N))

```
ll SumOfDivisors (ll num) {
    ll total = 1;
    for (ll i = 2; i * i <= num; i++) {
        if (num % i == 0) {
            ll e = 0;
            do {
                e++;
                num /= i;
            } while (num % i == 0);

            ll sum = 0, pow = 1;
            do {
                sum += pow;
                pow *= i;
            } while (e-- > 0);
            total *= sum;
        }
    }
    if (num > 1)
        total *= (1 + num);
    return total;
}
```

Description: Calculate the sum of divisors of 1 to N

```
const ll MOD = 1000000007;
ll sigma (ll x) {
    return ((x % MOD) * ((x + 1) % MOD) / 2) % MOD;
}
ll SumOfDivisors (ll N) {
    ll sum = 0;
    for (ll l = 1; l <= N; l++) {
        ll r = N / (N / l);
        sum += (N / l) % MOD * (sigma(r) - sigma(l-1) + MOD) % MOD;
        sum = (sum + MOD) % MOD;
        l = r;
    }
    return sum;
}
```

Modular Arithmetic

(a + b) % M = ((a % M) + (b % M)) % M
(a - b) % M = ((a % M) - (b % M) + M) % M
(a * b) % M = ((a % M) * (b % M)) % M
(a / b) % M = ((a % M) * (b^{-1} % M)) % M

Binary Exponentiation

Description: Calculate the values a^b modulo 10^9+7.
Complexity: O(log(b))

```
const ll MOD = 1000000007;
ll BinExpo (ll a, ll b) {
    ll ans = 1;
    while (b) {
        if (b & 1) ans = (ans * a) % MOD;
        a = (a * a) % MOD;
        b >>= 1;
    }
    return ans;
}
```

Large Exponentiation

Description: Calculate the values a^{b^c} modulo 10^9+7.
Complexity: O(log(b))

```
const ll MOD = 1000000007;
ll BinExpo (ll a, ll b, ll MOD) {
    ll ans = 1;
    while (b) {
        if (b & 1) ans = (ans * a) % MOD;
        a = (a * a) % MOD;
        b >>= 1;
    }
    return ans;
}
```

- BinExpo(a, BinExpo (b, c, MOD-1), MOD)

Binary Multiplication

Description: Calculate the values a*b modulo 10^9+7.
Complexity: O(log(b))

```
const ll MOD = 1000000007;
ll BinMultiply(ll a, ll b) {
    ll ans = 1;
    while (b) {
        if (b & 1) ans = (ans + a) % MOD;
        a = (a + a) % MOD;
        b >>= 1;
    }
    return ans;
}
```

Modular Multiplicative Inverse

Description: Calculate the values a^{-1} modulo 10^9+7.

```
const ll MOD = 1000000007;
ll ModInverse(ll a) {
    ll b = MOD – 2;
    ll ans = 1;
    while (b) {
        if (b & 1) ans = (ans * a) % MOD;
        a = (a * a) % MOD;
        b >>= 1;
    }
    return ans;
}
```

Binomial Coefficients

Description: Calculate the values $\binom{n}{r}$ modulo 10^9+7 for N test cases.
Where $1 \leq N \leq n \leq r \leq 10^6$

```
const ll N = 1000001;
const ll MOD = 1000000007;
ll fact[N];
void Factorial () {
    fact [0] = 1;
    for (ll i = 1; i <= N; i++)
        fact[i] = (fact[i-1] * i) % MOD;
}
ll ModInverse(ll a) {
    ll b = MOD - 2;
    ll ans = 1;
    while (b) {
        if (b & 1) ans = (ans * a) % MOD;
        a = (a * a) % MOD;
        b >>= 1;
    }
    return ans;
}
ll nCr (ll n, ll r) {
    ll ans = fact[n];
    ll den = (fact[n-r] * fact[r]) % MOD;
    ans = (ans * ModInverse(den)) % MOD;
    return ans;
}
```

Fibonacci Numbers

Description: Find the Fibonacci numbers till 10^{18}.

```
const ll MOD = 1000000007;
map<ll, ll> F;
ll f (ll n) {
   F [0] = F [1] = 1;
   if (F. count(n)) return F[n];
   ll k=n/2;
   if (n%2==0)
         return F[n] = (f(k)*f(k) + f(k-1) * f(k-1)) % MOD;
   else
        return F[n] = (f(k) * f(k+1) + f(k-1) * f(k)) % MOD;
}
ll Fibo (ll n) {
   return (n==0? 0: f(n-1));
}
```

Kadane's Algorithm

Description: Find the maximum subarray sum.
Complexity: O(N)

```
int KadanesAlgo(int a[ ], int N) {
   int MaxSum = INT_MIN;
   int CurrentSum = 0;

   for (int i=0; i<N; i++) {
      CurrentSum += a[i];
      MaxSum = max (MaxSum, CurrentSum);
      if (CurrentSum < 0)
         CurrentSum = 0;
   }
   return MaxSum;
}
```

Counting Subarrays

Description: Count the number of subarrays having sum x.

```
ll N, x, sum, cnt, a;
map<ll, ll> freq;
ll SubArray () {
   cin >> N >> x;
   freq[0] = 1;
   for (ll i = 1; i <= N; i++) {
      cin >> a;
      sum += a;
      cnt += freq[sum-x];
      freq[sum]++;
   }
   return cnt;
}
```

Description: Count the number of subarrays where the sum of values is divisible by x.

```
ll N, x, pre, cnt, a;
map<ll, ll> freq;
ll SubArray () {
   cin >> N >> x;
   freq[0] = 1;
   for (ll i = 1; i <= N; i++) {
      cin >> a;
      pre = ((pre+a) % x + x) % x;
      cnt += freq[pre];
      freq[pre]++;
   }
   return cnt;
}
```

AND (&): any 0 => 0
OR (|): any 1 => 1
X-OR (^): same => 0, different=> 0
Left Shift: n<<i,   n*2^(i)
Right Shift: n>>i,   n/2^(i)
2^n = 1<<n

Binary Search

```
while(l<=r) {
   mid = (l+r) / 2;
   if (a[mid] == value)
      return mid;
   if(a[mid] < value)
      l = mid + 1;
   else
      r = mid - 1;
}
```

- **binary_search** (v.begin(), v.end(), value)); //Return 0/1

auto it = **lower_bound**(v.begin(), v.end(), value) – v.begin()
auto *it = **upper_bound**(v.begin(), v.end(), value) - v.begin()
//it => index, *it => value

If a number n has an odd divisor, then it has an odd prime divisor.
If a number has no odd divisors, then it must be a power of two.

- Check Power of two: n&(n-1) == 0

Only perfect square number has odd number of divisors.

Vector

sort(v.begin(), v.end()); //**Increasing**
sort(v.begin(), v.end(), greater<int>()); //**Decreasing**
sort(v.rbegin(), v.rend()); //**Reverse**
rotate(v.begin(), v.begin()+1, v.end()) // **Rotate**

- swap(v[i], v[i+1]

int **UniqueValue** = unique(v.begin(), v.end()) - v.begin();
int **MaxValue** = max_element(v.begin(), v.end()) - v.begin();
int **MinValue** = min_element(v.begin(), v.end()) - v.begin();
cout << MaxValue << endl; //**Index**
cout << *MaxValue << endl; //**Value**
v.pop_back();
v.back(); //**LastValue**
v.erase(v.begin()); //O(n)
Check sorted or not:
if (is_sorted(v.begin(), v.end()))
   cout << "execute" << endl;

String

getline (cin, s);
s.push_back();
s.pop_back();
s.back(); //LastValue
s.erase(v.begin() + i); //i=>index
Subtring
string str = s.substring (i, j);
//i=>Strating index, j=>Ending index
Check Subtring:
if (a.substring(0, n) == s)
   cout << "Yes" << endl;
if(s.find(str)!=string::npos) //Return 0 or 1
   cout << "Yes" << endl;
2D String
string s [n];
for (int i =0; i<n; i++) cin >> s[i];
Accessing Index:

```
for (int i =0; i<n; i++) {
   for (int i =0; i<n; i++) {
      if (s[i][j] == "-") cout << "-" << endl;
   }
}
```

int x = stoi(s); // **String to Number**
for (auto i: s) x + = x*10 + (i – 48);
string s = to_string(x) //**Number to String**

Adjacency List

Description: n=>node, m=>edges

```
const int mx = 10e5+123;
vector<int>adj[mx];
cin >> n >> m;
for (i=0; i<m; i++) {
    int u,v;
    cin >> u >> v;
    adj[u].push_back(v);
    adj[v].push_back(u);
}
```

```
const int mx = 1e5+123;
vector<pair<int,int>> adj[mx];
cin >> n >> m;
for(i=1; i<=m; i++) {
    int u, v, w;
    cin >> u >> v >> w;
    adj[u].push_back({v,w});
    adj[v].push_back({u,w});
}
```

Adjacency Matrix

Description: n=>node, m=>edges

```
const int mx = 1002;
char adjMat[mx][mx];
cin >> n >> m;
for (int i=1; i<=n; i++) {
    for (int j=1; j<=m; j++)
        cin >> adjMat[i][j];
}
```

Depth First Search (DFS)

Description: n=>node, m=>edges

```
const int mx = 1002;
bool vis[max];
vector<int>adj[mx];
void dfs (int u) {
    vis[u] = 1;
    for (auto v: adj[u]) {
        if (vis[v] == 0)
            dfs(v);
    }
}
```

Find Connected Components

Description: dfs is required.

```
int CC (int n) {
    int cnt = 0;
    for (int i = 1; i <= n; i++) {
        if(vis[i]) continue;
        dfs(i);
        cnt++;
    }
    return cnt;
}
```

Description: To store connected components.

```
vector<vector<int>>cc;
vector<int>current_cc;
void dfs (int u) {
    vis[u] = 1;
    current_cc.push_back(u);
    for (auto v: adj[u]) {
        if (vis[v] == 0)
            dfs(v);
    }
}
int CC(int n) {
    for (int i = 1; i <= n; i++) {
        if(vis[i]) continue;
        current_cc.clear();
        dfs(i);
        cc.push_back(current_cc);
    }
    return cnt;
}
```

Cycle Detection

```
bool is_cycle(int node, int par){
        vis[node] = true;
        for (auto& new_node : adj[node]){
            if (!vis[new_node]){
                if (is_cycle(new_node, node))
                    return true;
            }
            else if (new_node != par) return true;
        }
        return false;
    }
```

Breath First Search (BFS)

Description: Distance vector provides the level of all nodes.

```
const int mx = 1e5+123;
int vis[mx],dis[mx];
vector<int>v[mx];
void bfs(int node) {
    queue<int>q;
    q.push(node);
    vis[node]=1; dis[node]=0;
    while (!q.empty()) {
        int a=q.front();
        q.pop();
        for (int child: v[a]) {
            if (vis[child]==0) {
                dis[child]=dis[a]+1;
                vis[child]=1;
                q.push(child);
            }
        }
    }
}
```

Dijkstra

Description:

```
const ll infLL = 9000000000000000000;
const int mx = 1e5+123;
vector<pair<int,int>> adj[mx];
ll dis[mx];
void dijkstra (int s, int n) {
    for (int i = 0; i <= n; i++) dis[i] = infLL;
    dis[s] = 0;
priority_queue<pair<ll,ll>,vector<pair<ll,ll>>,greater<pair<ll,ll>>>pq;
    pq.push ( { 0, s } );
    while (!pq.empty() ) {
        int u = pq.top().second;
        ll curD = pq.top().first;
        pq.pop();
        if (dis[u] < curD ) continue;
        for (auto p : adj[u] ) {
            int v = p.first;
            ll w = p.second;
            if (curD + w  < dis[v]) {
                dis[v] = curD + w;
                pq.push ( { dis[v], v } );
            }
        }
    }
}
```