Sublime Build

- Create code.cpp
- Tools -> build systems -> new build systems -> give code -> save as c++20
- Go to code.cpp and select the build system
- Alt + Shift + 3
- Save the 2nd window as 'inputf.in' and 3rd window as 'outputf.in'
- Goto inputf.in and go view -> groups -> max column:2
- CTRL + B to run, CTRL + Shift + B to compile and run.

Build System Code

```
{
  "cmd" : ["g++ -std=c++20 $file_name -o $file_base_name &&
  timeout 4s ./$file_base_name<inputf.in>outputf.in"],
  "selector" : "source.c",
  "shell": true,
  "working_dir" : "$file_path"
}
```

Precompile

```
On terminal run: find /usr/include -name "stdc++.h"
Go to the file path, open terminal there.
On terminal run: g++ -std=c++20 stdc++.h
```

Template

```
#include <bits/stdc++.h>
using namespace std;
#define int long long
#define ll long long
#define endl '\n'
#define sz(x)(int)(x).size()
#define all(x) x.begin(), x.end()
#define FAST(ios_base::sync_with_stdio(false),
cin.tie(nullptr));
ll power(ll x, ll y, ll m = 1e9 + 7) {
  ll ans = 1;
  x %= m;
  while (y) {
    if (y & 1) ans = (ans * x) % m;
    x = (x * x) % m;
    y >>= 1;
  }
  return ans;
}
void solve() {

```

```
}
signed main() {
  FAST;
  int TCS = 1;
  // cin >> TCS;
  for (int TC = 1; TC <= TCS; ++TC) {
    cout << "Case " << TC << ": ";
    solve();
  }
}
```

Debug Code

```
#ifndef ONLINE _ JUDGE
#define debug(args...)
cerr << "(" << #args << "):", _print(args);
#else
#define debug(args...)
#endif
template < typename A, typename B > ostream & operator <<
(ostream & os,
  const pair < A, B > & p) {
  return os << '(' << p.first << ", " << p.second << ')';
}
template < typename T_container, typename T = typename
enable_if < !is_same < T_container, string > ::value, typename
T_container::value_type > ::type > ostream & operator <<
(ostream & os,
  const T_container & v) {
  os << '{';
  string sep;
  for (const T & x: v) os << sep << x, sep = ", ";
  return os << '}';
}
void _print() {
  cerr << endl;
}
template < typename Head, typename...Tail > void _print(Head H,
Tail...T) {
  cerr << " " << H << ",";
  _print(T...);
}
```

Stress testing

- create correct.cpp, wrong.cpp, gen.cpp
- create test.sh or test.py
- open location on termina and
- run test.sh-> bash test.sh
- run test.sh-> python3 test.py

gen.cpp

```cpp
#include <bits/stdc++.h>
using namespace std;
int rnd(int a, int b) {
  return a + rand() % (b - a + 1);
}
int main() {
  int w = rnd(1, 100);
  cout << w << endl;
}
```

test.sh

```sh
set -e
g++ correct.cpp -o correct
g++ gen.cpp -o gen
g++ wrong.cpp -o wrong
for((i = 1; ; ++i)); do
    ./gen $i > input_file
    ./correct < input_file > wrongAnswer
    ./wrong < input_file > correctAnswer
    diff -Z wrongAnswer correctAnswer > /dev/null || break
    echo "Passed test: "  $i
done
echo "WA on the following test:"
cat input_file
echo "Your answer is:"
cat wrongAnswer
echo "Correct answer is:"
cat correctAnswer
```

test.py

```python
import os
import subprocess
import filecmp
subprocess.run(["g++", "correct.cpp", "-o", "correct"], check=True)
subprocess.run(["g++", "gen.cpp", "-o", "gen"], check=True)
subprocess.run(["g++", "wrong.cpp", "-o", "wrong"], check=True)
i = 1
while True:
    with open("input_file", "w") as input_file:
        subprocess.run(["./gen", str(i)], stdout=input_file, check=True)
    with open("input_file", "r") as input_file, open("correctAnswer", "w") as correct_output:
        subprocess.run(["./correct"], stdin=input_file, stdout=correct_output, check=True)
    with open("input_file", "r") as input_file, open("myAnswer", "w") as wrong_output:
        subprocess.run(["./wrong"], stdin=input_file, stdout=wrong_output, check=True)
    if not filecmp.cmp("myAnswer", "correctAnswer", shallow=False):
        print("\nWA on the following test:")
        with open("input_file", "r") as f:
            print(f.read())
```

```
        print("Your answer is:")
        with open("myAnswer", "r") as f:
            print(f.read())
        print("\nCorrect answer is:")
        with open("correctAnswer", "r") as f:
            print(f.read())
        break
    print(f"Passed test: {i}")
    i += 1
```

Basic STL Operations

```
vector<int> v;
// Operations                       // Time Complexity
v.push_back(x)                      // O(1)
v.pop_back()                        // O(1)
v.size()                            // O(1)
v.empty()                           // O(1)
v.clear()                           // O(n)
v.front()                           // O(1)
v.back()                            // O(1)
v.begin(), v.end()                  // O(1)
v.erase(iterator)                   // O(n)
v.erase(start_iter, end_iter)       // O(n)
v.insert(iterator, x)               // O(n)
v[i]                                // O(1)

list<int> l;
// Operations                       // Time Complexity
l.push_back(x)                      // O(1)
l.push_front(x)                     // O(1)
l.pop_back()                        // O(1)
l.pop_front()                       // O(1)
l.size()                            // O(1)
l.empty()                           // O(1)
l.clear()                           // O(n)
l.begin(), l.end()                  // O(1)
l.insert(iterator, x)               // O(1)
l.erase(iterator)                   // O(1)

// Max Heap
priority_queue<int> pq;
// Min Heap
priority_queue<int, vector<int>, greater<int>> pq;
// Operations                       // Time Complexity
pq.push(x)                          // O(log n)
pq.pop()                            // O(log n)
pq.top()                            // O(1)
pq.empty()                          // O(1)

set<int> s;
// Operations                       // Time Complexity
s.insert(x)                         // O(log n)
s.erase(x)                          // O(log n)
s.erase(iterator)                   // O(1)
s.find(x)                           // O(log n)
s.count(x)                          // O(log n)
s.empty()                           // O(1)
s.clear()                           // O(n)
```

```
s.lower_bound(x)                    // O(log n)
s.upper_bound(x)                    // O(log n)

map<string, int> m;
// Operations                       // Time Complexity
m[key] = value                      // O(log n)
m.insert({key, value})              // O(log n)
m.erase(key)                        // O(log n)
m.find(key)                         // O(log n)
m.count(key)                        // O(log n)
m.empty()                           // O(1)
m.clear()                           // O(n)
m.lower_bound(key)                  // O(log n)
m.upper_bound(key)                  // O(log n)

unordered_set<int> us;
// Operations              // Average      // Worst
us.insert(x)               // O(1)         // O(n)
us.erase(x)                // O(1)         // O(n)
us.find(x)                 // O(1)         // O(n)
us.count(x)                // O(1)         // O(n)

unordered_map<string, int> um;
// Operations              // Average      // Worst
um[key] = value            // O(1)         // O(n)
um.insert({key, value})    // O(1)         // O(n)
 um.erase(key)             // O(1)         // O(n)
um.find(key)               // O(1)         // O(n)
um.count(key)              // O(1)         // O(n)

stack<int> st;
// Operations                       // Time Complexity
st.push(x)                          // O(1)
st.pop()                            // O(1)
st.top()                            // O(1)
st.size()                           // O(1)
st.empty()                          // O(1)

queue<int> q;
// Operations                       // Time Complexity
q.push(x)                           // O(1)
q.pop()                             // O(1)
q.front()                           // O(1)
q.back()                            // O(1)
q.size()                            // O(1)
q.empty()                           // O(1)

deque<int> d;
// Operations                       // Time Complexity
d.push_back(x)                      // O(1)
d.push_front(x)                     // O(1)
d.pop_back()                        // O(1)
d.pop_front()                       // O(1)
d.front()                           // O(1)
d.back()                            // O(1)
d[i]                                // O(1)
d.size()                            // O(1)
d.empty()                           // O(1)

string s;
string s1 = "Hello";               // Direct initialization, O(n)
string s2(5, 'a');                 // Creates "aaaaa", O(n)
```

```cpp
string s3(s1);                    // Copy constructor, O(n)
// Access
s1.empty();                     // Checks if string is empty, O(1)
s1.at(2);                  // Same, but with bounds checking, O(1)
// Modification
s1 += "World";                    // Concatenation, O(n+m)
s1.append("World");        // Another way to concatenate, O(n+m)
s1.push_back('!');        // Add single character, O(1) amortized
s1.pop_back();                    // Remove last character, O(1)
// Substring and Slicing
s1.substr(0, 3);          // Substring, size 3 from index 0, O(k)
s1.substr(2);              // Substring from index 2 to end, O(n)
// Search
s1.find("Hello");        // Find substring, returns index, O(n*m)
s1.rfind("Hello");                // Find from right side, O(n*m)
s1.find_first_of("aeiou");        // Find first vowel, O(n*m)
s1.find_last_of("aeiou");         // Find last vowel, O(n*m)
// Comparison
s1.compare(s2);                   // Compare strings, O(n)
s1 == s2;                         // Equality comparison, O(n)
s1 < s2;                        // Lexicographic comparison, O(n)
// Insert and Erase
s1.insert(2, "abc");         // Insert substring at index 2, O(n)
s1.erase(2, 3);         // Erase 3 characters from index 2, O(n)
// Replace
s1.replace(2, 3, "xyz"); // Replace 3 chars from index 2, O(n)
// Conversion
stoi(s1);                         // String to integer, O(n)
to_string(42);                // Integer to string, O(log(number))
// Modify Case
transform(s1.begin(), s1.end(), s1.begin(), ::tolower);  // To lowercase, O(n)
transform(s1.begin(), s1.end(), s1.begin(), ::toupper);  // To uppercase, O(n)
// Trim
s1.erase(0, s1.find_first_not_of(" \t")); // Left trim, O(n)
s1.erase(s1.find_last_not_of(" \t") + 1); // Right trim, O(n)
// Copying
s2.assign(s1);              // Assign one string to another, O(n)
// Clear
s1.clear();                       // Make string empty, O(1)
```

Ordered set

```cpp
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
typedef tree<int, null_type, less<int>, rb_tree_tag,
tree_order_statistics_node_update> ordered_set;
```

128bit integer

```cpp
// int128 bit for numbers larger than 1e18. Will support numbers till 1e36
// Typedef to ell -> extra long long
typedef __int128 ell;
```

```cpp
// For printing
std::ostream & operator << (std::ostream & dest, __int128_t value) {
  std::ostream::sentry s(dest);
  if (s) {
    __uint128_t tmp = value < 0 ? -value : value;
    char buffer[128];
    char * d = std::end(buffer);
    do {
      --d;* d = "0123456789" [tmp % 10];
      tmp /= 10;
    } while (tmp != 0);
    if (value < 0) {
      --d;* d = '-';
    }
    int len = std::end(buffer) - d;
    if (dest.rdbuf() -> sputn(d, len) != len) {
      dest.setstate(std::ios_base::badbit);
    }
  }
  return dest;
}
// For reading _int128 to_read = read()
__int128 read() {
  __int128 x = 0, f = 1;
  char ch = getchar();
  while (ch < '0' || ch > '9') {
    if (ch == '-') f = -1;
    ch = getchar();
  }
  while (ch >= '0' && ch <= '9') {
    x = x * 10 + ch - '0';
    ch = getchar();
  }
  return x * f;
}
// For debugging
void _print(ell t) {
  cerr << t;
}
```

Bitset

```cpp
const int SIZE = 8; // Define bitset size
bitset < SIZE > b; // Initialize bitset with all bits set to 0
cout << "Initial bitset: " << b << endl;
// set(): Set a bit at a given index
b.set(2);
b.set(5);
cout << "After set(2) and set(5): " << b << endl;
// reset(): Reset a bit at a given index (set to 0)
b.reset(2);
cout << "After reset(2): " << b << endl;
// flip(): Flip a bit at a given index
b.flip(5);
cout << "After flip(5): " << b << endl;
// count(): Count number of set bits
cout << "Count of set bits: " << b.count() << endl;
// test(): Check if a bit is set at an index
```

```
cout << "Is bit at index 3 set? " << (b.test(3) ? "Yes" : "No")
<< endl;
// any(): Check if any bit is set
cout << "Is any bit set? " << (b.any() ? "Yes" : "No") << endl;
// none(): Check if no bit is set
cout << "Are all bits 0? " << (b.none() ? "Yes" : "No") <<
endl;
// all(): Check if all bits are set
b.set(); // Setting all bits to 1
cout << "After setting all bits: " << b << endl;
cout << "Are all bits set? " << (b.all() ? "Yes" : "No") <<
endl;
// size(): Get size of bitset
cout << "Size of bitset: " << b.size() << endl;
// to_string(): Convert bitset to string
cout << "Bitset as string: " << b.to_string() << endl;
// to_ulong(): Convert bitset to unsigned long
cout << "Bitset as unsigned long: " << b.to_ulong() << endl;
// to_ullong(): Convert bitset to unsigned long long
cout << "Bitset as unsigned long long: " << b.to_ullong() <<
endl;
```

Bit manipulation and operation

```
int num = 42; // Example number (00101010 in binary)
cout << "Number: " << num << " (Binary: " << bitset<8>(num) <<
")\n";
int pos = 3; // Bit position (0-based index)
cout << "Set bit at " << pos << ": " << bitset<8>(num | (1 <<
pos)) << endl;
cout << "Clear bit at " << pos << ": " << bitset<8>(num & ~(1
<< pos)) << endl;
cout << "Toggle bit at " << pos << ": " << bitset<8>(num ^ (1
<< pos)) << endl;
cout << "Is bit " << pos << " set? " << ((num & (1 << pos)) ?
"Yes" : "No") << endl;
cout << "Set bits: " << __builtin_popcount(num) << endl;
cout << "First set bit pos (1-based): " << __builtin_ffs(num)
<< endl;
cout << "Highest set bit pos (0-based): " << (31 -
__builtin_clz(num)) << endl;
cout << "Is power of two? " << ((num && !(num & (num - 1))) ?
"Yes" : "No") << endl;
cout << num << " is " << (num & 1 ? "Odd" : "Even") << endl;
cout << num << " * 2 = " << (num << 1) << ", " << num << " / 2
= " << (num >> 1) << endl;
int a = 5, b = 9;
cout << "Swap: a = " << a << ", b = " << b;
a ^= b; b ^= a; a ^= b;
cout << " → a = " << a << ", b = " << b << endl;
// Number System Conversions
cout << "Binary: " << bitset<8>(num) << ", Octal: " << oct <<
num << ", Hex: " << hex << uppercase << num << endl;
// Convert Binary, Octal, Hexadecimal to Decimal
cout << "Binary to Decimal: " << stoi("101010", 0, 2) << ", "
     << "Octal to Decimal: " << stoi("52", 0, 8) << ", "
     << "Hex to Decimal: " << stoi("2A", 0, 16) << endl;
```

Bit operation basics

```
// __builtin_popcountll(n); -> counts number of set bits
// check bit (n & (1LL << ith))
// n set bit number -> (1LL << n) - 1
// set bit -> [ n | (1LL << ith) ];
// unset bit -> [ n & (~(1LL << ith)) ];
// to uppercase -> [ 'a' ^ 32 ];
// flip the kth bit -> X = X ^ (1LL << k);
// Use 1LL when shifting bits : (1LL << x)
Some properties of bitwise operations:
 a|b = a^b + a&b
 a^(a&b) = (a|b)^b
 b^(a&b) = (a|b)^a
 (a&b)^(a|b) = a^b
Addition:
 a+b = a|b + a&b
 a+b = a^b + 2(a&b)
Subtraction:
 a-b = (a^(a&b))-((a|b)^a)
 a-b = ((a|b)^b)-((a|b)^a)
 a-b = (a^(a&b))-(b^(a&b))
 a-b = ((a|b)^b)-(b^(a&b))
bool isPowerof2(int n){ return (!(n & (n - 1)) && n); }
int toggleBit(int n, int ith) { return n ^ (1 << ith);}
int numOfDigits(int n) { return floor(log10(n) + 1); }
```

Modular Arithmetic

```
(a+b) mod m = ((a mod m)+(b mod m)) mod m
(a-b) mod m = ((a mod m)-(b mod m)) mod m
(a*b) mod m = ((a mod m)*(b mod m)) mod m
(a/b) mod m = ((a mod m)*(b^-1 mod m)) mod m
```

Binary Exponentiation

```
/* Inverse : (n^-1)%m = (n^m-2)%m
Ex: Inverse of x would be,
BinaryExponentiation(x, MOD-2);
For normal constrain just divide by x(number) */
int BinaryExponentiation(int x, int y)
{
  int res = 1;
  while(y > 0){
    if(y & 1) res *= x; // MOD
    y >>= 1; // -> y /= 2;
    x *= x; // MOD
  } // MOD for larger numbers
  return res;
}
```

Binary search

```
int low=0, high=1e18, ans=-1;
while(low<=high)
{
```

```
    int mid = low + (high-low)/2;
    if(v[mid]==target)
    {
        ans = mid;
        break;
    }
    else if(v[mid]<target) low = mid+1;
    else high = mid-1;
}
cout<<ans<<endl;
```

Binary Search Lambda

```
auto check = [&](int mid) -> bool {
        return false;
    };
int lo = 0, hi = n - 1, ans, target;
while(lo <= hi){
    int mid = lo + (hi - lo) / 2;
    if(check(mid)){
        ans = mid;
        hi = mid - 1;
    } else lo = mid + 1;
}
```

Ternary search

```
int ternarySearch(int arr[], int l, int r, int key)
{
    while (r >= l) {
        int mid1 = l + (r - l) / 3;
        int mid2 = r - (r - l) / 3;
        if (arr[mid1] == key) {
            return mid1;
        }
        if (arr[mid2] == key) {
            return mid2;
        }
        if (key < arr[mid1]) {
            r = mid1 - 1;
        }
        else if (key > arr[mid2]) {
            l = mid2 + 1;
        }
        else {
            l = mid1 + 1;
            r = mid2 - 1;
        }
    }
    return -1;
}
```

Prime Check

```
bool prime_check(int n) {
    if(n<2) return false;
    else if(n <4) return true;
    else if(!(n&1)) return false;
```

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

Sieve

```
const int N = 10^7 +10;
vector<bool>isPrime(N,1);
int main()
{
    isPrime[0]=isPrime[1]=0;
    for(int i=2; i<N; ++i)
    {
        if(isPrime[i])
            for(int j=2*i; j<N; j+=i)
                isPrime[j]=0;
    }
}
```

```
const int N = 10^7 +10;
vector<bool>isPrime(N,1);
vector<int>lp(N,0), hp(N,0);
int main()
{
    isPrime[0]=isPrime[1]=0;
    for(int i=2; i<N; ++i)
    {
        if(isPrime[i])
        {
            lp[i]=hp[i]=i;
            for(int j=2*i; j<N; j+=i)
            {
                isPrime[j]=0;
                hp[j]=i;
                if(lp[j]==0) lp[j]=i;
            }
        }
    }
    int num; cin>>num;
    unordered_map<int,int>prime_factors;
    while(num>1)
    {
        int prime_factor = hp[num];
        while(num % prime_factor == 0)
        {
            num /= prime_factor;
            prime_factors[prime_factor]++;
        }
    }
    for(auto factor:prime_factors)
        cout<<factor.first<<" "<<factor.second<<endl;
}
```

Combinatorics

```
vector<int> fact(101, 0), factinv(101, 0);
int md = 998244353;
int genfact(int n) {if(!n)return 1;return fact[n]?fact[n]:fact[n]=n*genfact(n-1)%md;}
int genfactinv(int n){if(!n)return 1;return factinv[n]?factinv[n]:factinv[n]=power(genfact(n),md-2);}
int ncrmod(int n, int r) {return (genfact(n) * genfactinv(r) % md) * genfactinv(n - r) % md; }
int nprmod(int n, int r) {return genfact(n) * genfactinv(n - r) % md; }
```

Sparse Table

```
const int MAX_N = 100000;
const int LOG = 17;
int st[MAX_N][LOG + 1];
int logTable[MAX_N + 1];
int n;
int gcd(int a, int b) {
    return b == 0 ? a : gcd(b, a % b);
}
void buildSparseTable(vector<int>& arr) {
    n = arr.size();
    logTable[1] = 0;
    for (int i = 2; i <= n; i++) {
        logTable[i] = logTable[i / 2] + 1;
    }
    for (int i = 0; i < n; i++) {
        st[i][0] = arr[i];
    }
    for (int j = 1; (1 << j) <= n; j++) {
        for (int i = 0; i + (1 << j) <= n; i++) {
            st[i][j] = gcd(st[i][j - 1], st[i + (1 << (j - 1))][j - 1]);
        }
    }
}
int queryGCD(int L, int R) {
    int j = logTable[R - L + 1];
    return gcd(st[L][j], st[R - (1 << j) + 1][j]);
}
```

DFS

```
const int N = 1e5+10;
vector<int>g[N];
int vis[N];
void dfs(int vertex)
{
    vis[vertex] = true;
    for(int child : g[vertex])
    {
        if(vis[child])  continue;
        dfs(child);
```

```
    }
}
int main()
{
    int n,m;
    cin>>n>>m;
    for(int i=0; i<m; ++i)
    {
        int v1,v2;
        cin>>v1>>v2;
        g[v1].push_back(v2);
        g[v2].push_back(v1);
    }
}
```

Tree height and depth

```
const int N=1e5+10;
vector<int>g[N];
int depth[N], height[N];
void dfs(int vertex, int parent)
{
    for(int child : g[vertex])
    {
        if(child==parent)  continue;
        depth[child]=depth[vertex]+1;
        dfs(child,vertex);
        height[vertex]=max(height[vertex], height[child]+1);
    }
}
int main()
{
    int n; cin>>n;
    for(int i=0; i<n-1; ++i)
    {
        int x,y;
        cin>>x>>y;
        g[x].push_back(y);
        g[y].push_back(x);
    }
    dfs(1,0);
    for(int i=1; i<=n; ++i)
        cout<<depth[i]<<" "<<height[i]<<endl;
}
```

Diameter of a Tree

- With any root find max depth node.
- With that node as root find max depth which will be the diameter

```
void dfs(int v, int parent)
{
    for(int child : g[v])
    {
        if(child==parent)
            continue;
        depth[child]=depth[v]+1;
```

```
        dfs(child,v);
    }
}
int main()
{
    // Take input
    dfs(1,-1);
    int mx_depth=-1,mx_depth_node;
    for(int i=1; i<=n; ++i)
    {
        if(mx_depth<depth[i])
        {
            mx_depth=depth[i];
            mx_depth_node=i;
        }
        depth[i]=0;
    }
    dfs(mx_depth_node,-1);
    mx_depth=-1;
    for(int i=1; i<=n; ++i)
    {
        if(mx_depth<depth[i])
        {
            mx_depth=depth[i];
        }
    }
    cout<<mx_depth<<endl;
}
```

Loop Check

```
#include<bits/stdc++.h>
using namespace std;
#define ll   long long
const int N=1e5+10;
vector<int>g[N];
bool vis[N];
bool dfs(int vertex,int parent)
{
    vis[vertex]=1;
    bool loop = false;
    for(int child : g[vertex])
    {
        if(vis[child] && child == parent) continue;
        if(vis[child])   return true;
        loop |= dfs(child,vertex);
    }
    return loop;
}
int main()
{
    int n,e; cin>>n>>e;
    for(int i=0; i<e; ++i)
    {
        int x,y;
        cin>>x>>y;
        g[x].push_back(y);
        g[y].push_back(x);
    }
    bool loop = false;
```

```
    for(int i=1; i<=n; ++i)
    {
        if(vis[i])  continue;
        if(dfs(i,0))
        {
            loop = true;
            break;
        }
    }
    cout<<loop<<endl;
}
```

BFS

```
const int N=1e5+10;
vector<int>g[N];
int vis[N];
void bfs(int source)
{
    queue<int>q;
    q.push(source);
    vis[source]=1;
    while(!q.empty())
    {
        int vertex=q.front();
        q.pop();
        cout<<vertex<<endl;
        for(int child:g[vertex])
        {
            if(!vis[child])
            {
                q.push(child);
                vis[child]=1;
            }
        }
    }
}
int main()
{
    int n; cin>>n;
    for(int i=0; i<n-1; ++i)
    {
        int x,y; cin>>x>>y;
        g[x].push_back(y);
        g[y].push_back(x);
    }
    bfs(1);
}
```

0/1 BFS

```
const int N = 1e5+10;
const int INF = 1e9+10;
vector<pair<int,int>>g[N];
vector<int>level(N,INF);
int n,m;
int bfs()
{
    deque<int>q;
```

```
    q.push_back(1);
    level[1]=0;
    while(!q.empty())
    {
        int vertex = q.front();
        q.pop_front();

        for(auto child : g[vertex])
        {
            int v=child.first;
            int wt=child.second;
            if(level[vertex]+wt<level[v])
            {
                level[v]=level[vertex]+wt;
                if(wt==1)
                    q.push_back(v);
                else
                    q.push_front(v);
            }
        }
    }
    if(level[n]==INF) return -1;
    return level[n];
}
int main()
{
    cin>>n>>m;
    for(int i=0; i<m; ++i)
    {
        int x,y;
        cin>>x>>y;
        if(x==y)
            continue;
        g[x].push_back({y,0});
        g[y].push_back({x,1});
    }
    cout<<bfs()<<endl;
}
```

Multi Source BFS

```
const int N = 1e3+10;
const int MAX = 1e9+10;

int val[N][N];
int vis[N][N];
int level[N][N];
int n,m;

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

bool valid(int i, int j)
{
    return i>=0 && j>=0 && i<n && j<m;
}

int bfs()
{
    int mx=0;
```

```
    for(int i=0; i<n; ++i)
    {
        for(int j=0; j<m; ++j)
            mx=max(mx,val[i][j]);
    }
    queue<pair<int,int>> q;
    for(int i=0; i<n; ++i)
    {
        for(int j=0; j<m; ++j)
        {
            if(mx==val[i][j])
            {
                q.push({i,j});
                level[i][j]=0;
                vis[i][j]=1;
            }
        }
    }
    int ans = 0;
    while(!q.empty())
    {
        auto v =  q.front();
        int vx = v.first;
        int vy = v.second;
        q.pop();
        for(auto move:moves)
        {
            int x = move.first + vx;
            int y = move.second + vy;
            if(valid(x,y) && !vis[x][y])
            {
                q.push({x,y});
                level[x][y]=level[vx][vy]+1;
                vis[x][y]=1;
                ans = max(ans,level[x][y]);
            }
        }
    }
    return ans;
}

void reset()
{
    for(int i=0; i<n; ++i)
    {
        for(int j=0; j<m; ++j)
        {
            vis[i][j] = 0;
            level[i][j] = MAX;
        }
    }
}

void solve()
{
    cin>>n>>m;
    reset();
    for(int i=0; i<n; ++i)
    {
        for(int j=0; j<m; ++j)
        {
```

```
            cin >> val[i][j];
        }
    }
    cout<<bfs()<<endl;
}

int main()
{
    int TC;
    cin >> TC;
    while (TC--)
    {
        cin>>n>>m;
        reset();
        for(int i=0; i<n; ++i)
        {
            for(int j=0; j<m; ++j)
            {
                cin >> val[i][j];
            }
        }
        cout<<bfs()<<endl;
    }
}
```

Path Between Two Nodes

```
vector<vector<int>>adj;
vector<int>parent;
vector<bool>vis;
int x, y;
void BFS(int initial) {
    queue<int>q;
    q.push(initial);
    while(!q.empty()) {
        int curr = q.front();
        vis[curr] = true;
        if(curr == y) return;
        for(auto i: adj[curr]) {
            if(!vis[i]) {
                q.push(i);
                parent[i] = curr;
            }
        }
        q.pop();
    }
}
void backtrack(vector<int>&v, int last) {
    while(last != -1) {
        v.pb(last);
        last = parent[last];
    }
    reverse(v.begin(), v.end());
}
void Path_Between_Two_Nodes(int x, int y) {
    // adj.resize(n+1, vector<int>()); -> in solve()
    BFS(x);
    vector<int>ans;
    backtrack(ans, y);
    for(int i = 0; i < size(ans); i++) cout<<ans[i]<<" ";
```

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

Dijkstra

```
const int N = 1e5+10;
const int INF = 1e9+10;
vector<pair<int,int>> g[N];
void dijkstra(int v, int n)
{
    vector<int>vis(N,0);
    vector<int>dis(N,INF);
    set<pair<int,int>>st;
    st.insert({0,v});
    dis[v]=0;
    while(st.size())
    {
        auto node = *st.begin();
        int v =  node.second;
        int d = node.first;
        st.erase(st.begin());
        if(vis[v])
            continue;
        vis[v] = 1;
        for(auto child : g[v])
        {
            int child_v =  child.first;
            int wt =  child.second;
            if(d + wt < dis[child_v])
            {
                dis[child_v] = d + wt;
                st.insert({dis[child_v],child_v});
            }
        }
    }
}
int main()
{
    int n,m;
    cin>>n>>m;
    for(int i=0; i<m; ++i)
    {
        int x,y,wt;
        cin>>x>>y>>wt;
        g[x].push_back({y,wt});
    }
    int v;
    cin>>v;
    dijkstra(v,n);
}
```

Floyd Warshall

```
const int N=510;
const int INF=1e9+10;
int dist[N][N];
int main()
{
    for(int i=0; i<N; ++i)
    {
```

```
        for(int j=0; j<N; ++j)
        {
            if(i==j) dist[i][j]=0;
            else dist[i][j]=INF;
        }
    }
    int n,m; cin>>n>>m;
    for(int i=0; i<m; ++i)
    {
        int x,y,wt; cin>>x>>y>>wt;
        dist[x][y]=wt;
    }
    for(int k=1; k<=n; ++k)
    {
        for(int i=0; i<=n; ++i)
        {
            for(int j=0; j<=n; ++j)
                dist[i][j]=min(dist[i][j],dist[i][k]+dist[k][j]);
        }
    }
}
```

Bellman ford

```
void bellmanFord(vector<vector<int>>& edges, int V, int E, int
src) {
    vector<int> dist(V, INT_MAX);
    dist[src] = 0;
    for (int i = 1; i < V; i++) {
        for (auto& edge : edges) {
            int u = edge[0], v = edge[1], w = edge[2];
            if (dist[u] != INT_MAX && dist[u] + w < dist[v]) {
                dist[v] = dist[u] + w;
            }
        }
    }
    for (auto& edge : edges) {
        int u = edge[0], v = edge[1], w = edge[2];
        if (dist[u] != INT_MAX && dist[u] + w < dist[v]) {
            cout << "Negative cycle exists" << endl;
            return;
        }
    }
    cout << "Vertex\tDistance from source" << endl;
    for (int i = 0; i < V; i++) {
        cout << i << "\t"
             << (dist[i] == INT_MAX ? "INF" :
             to_string(dist[i])) << endl;
    }
}
```

DSU

```
class DSU
{
    private:
    vector<int> par;
```

```
    vector<int> size;
    public:
    DSU(int n)
    {
        par = vector<int>(n);
        iota(par.begin(), par.end(), 0);
        size = vector<int>(n, 1);
    }
    int find(int u)
    {
        if(par[u] != par[par[u]])
            par[u] = find(par[par[u]]);
        return par[u];
    }
    bool connected(int u, int v)
    {
        u = find(u);
        v = find(v);
        if(u == v)
            return true;
        return false;
    }
    bool join(int u, int v)
    {
        u = find(u);
        v = find(v);
        if(u == v)
            return false;
        if(size[u] <= size[v])
        {
            size[v] += size[u];
            par[u] = v;
        }
        else
        {
            size[u] += size[v];
            par[v] = u;
        }
        return true;
    }
};
void solve()
{
    int n, m;
    cin >> n >> m;
    DSU dsu(n);
    for(int i = 0; i < n; ++i)
    {
        int v, u; cin >> v >> u;
        dsu.join(v, u);
    }
}
```

Strongly Connected Components

```
const int N = 100;  // Adjust as needed for the maximum number
of nodes
vector<int> g[N];   // Adjacency list representation of the
graph
// Variables for Tarjan's Algorithm
```

```
vector<int> low, disc;
vector<bool> inStack;
stack<int> s;
int timeCounter = 0;  // Used to assign discovery times
// Function to perform DFS and find SCCs
void tarjanSCC(int node, vector<vector<int>>&
stronglyConnectedComponents) {
  // Initialize discovery time and low value
  disc[node] = low[node] = ++timeCounter;
  s.push(node);
  inStack[node] = true;
  // Explore all adjacent nodes
  for (int neighbor : g[node]) {
    if (disc[neighbor] == -1) {  // If neighbor hasn't been
    visited
      tarjanSCC(neighbor, stronglyConnectedComponents);
      low[node] = min(low[node], low[neighbor]);   // Update
      low value
    } else if (inStack[neighbor]) {                // If
    neighbor is in stack, it's part of the current SCC
      low[node] = min(low[node], disc[neighbor]);  // Update
      low value
    }
  }
  // If the node is a root node, pop the stack and generate an
  SCC
  if (low[node] == disc[node]) {
    vector<int> component;
    while (s.top() != node) {
      component.push_back(s.top());
      inStack[s.top()] = false;  // Mark as not in stack
      s.pop();
    }
    component.push_back(s.top());  // Add the root node
    inStack[s.top()] = false;      // Mark as not in stack
    s.pop();
    stronglyConnectedComponents.push_back(component);  // Add
    the component to the list
  }
}
int main() {
  int edges, u, v;
  cout << "Enter number of edges: ";
  cin >> edges;
  // Initialize graph and variables
  for (int i = 0; i < edges; i++) {
    cout << "Enter edge (u v): ";
    cin >> u >> v;
    g[u].push_back(v);  // Add edge from u to v
  }
  // Prepare data structures
  low.assign(N, -1);                                  // Low
  values
  disc.assign(N, -1);                                 //
  Discovery times
  inStack.assign(N, false);                           // To track
  nodes in the stack
  vector<vector<int>> stronglyConnectedComponents;  // To store
  all SCCs
```

```
  // Call Tarjan's algorithm for each node
  for (int i = 1; i <= N; ++i) {                // Adjust the range based on the number of nodes
    if (disc[i] == -1 && g[i].size() != 0) {  // If the node hasn't been visited
      tarjanSCC(i, stronglyConnectedComponents);
    }
  }
  // Output the strongly connected components
  cout << "Strongly Connected Components:" << endl;
  for (const auto& component : stronglyConnectedComponents) {
    cout << "Component: ";
    for (int node : component) {
      cout << node << " ";  // Print each node in the component
    }
    cout << endl;  // New line for each component
  }
  return 0;
}
```

Segment tree

```
class SegmentTree
{
vector<int>segment;
public:
 SegmentTree(int sz){
  segment.resize(4*sz + 1);
 }
 // Build Segement Tree
 void build(int ind , int low , int high  , vector<int> &v ){
  if(low == high){
   segment[ind] = v[low];
   return;
  }
  int mid = (high+low)/2;
  build(ind*2 + 1 , low , mid,v);
  build(ind*2 + 2 , mid+1 , high ,v);
  segment[ind] = min(segment[ind*2+1] , segment[ind*2+2]);
 }
 //Query
 int query(int ind, int low, int high , int target_low , int target_high, vector<int> &v){
  if(low >  target_high || high < target_low) return INT_MAX;
  if(low>=target_low && high<=target_high) return segment[ind];
  int mid = (low+high)/2;
  int ans1 = query(ind*2+1,low,mid,target_low,target_high,v);
  int ans2 =
  query(ind*2+2,mid+1,high,target_low,target_high,v);
  return min(ans1,ans2);
 }
 // Update Value
 void update(int ind , int low, int high , int target_ind , int val , vector<int>&v){
  if(low == high){
   segment[ind] = val;
   return;
  }
```

```
  int mid = (low+high)/2;
  if(target_ind<=mid) update(ind*2 + 1 , low , mid , target_ind , val , v);
  else update(ind*2 + 2 , mid+1 , high, target_ind, val  , v);
  segment[ind] = min(segment[ind*2+1] , segment[ind*2+2]);
 }
};
int main()
{
 int n; cin >> n;
 vector<int>v(n);
 for(auto &x : v) cin >> x;
 SegmentTree sg(n);
 sg.build(0,0,n-1,v);
 for(int i=0;i<4*n;i++){
  cout << sg.segment[i] << " ";
 }
 cout << endl;
 int q; cin >> q;
 while(q--){
  int type ; cin >> type;
  if(type==1){
   int l,r; cin >> l >> r;
   cout << sg.query(0,0,n-1,l,r,v) << endl;
  }
  else{
   int ind,val; cin >> ind >> val;
   sg.update(0,0,n-1,ind,val,v);
  }
 }
cout << endl;
```

Bipartite Graph

```
vector<vector<int>>adj;
vector<int>color;
bool dfs(int curr, int col) {
    color[curr] = col;
    for(auto i: adj[curr]) {
        if(color[i] == -1) {
            if(!dfs(i, col^1)) { // if false for next vertex
                return false;
            }
        } else {
            if(color[curr] == color[i]) return false;
        }
    }
    return true;
}
void Bipartite_Graph_Coloring() {
    // adj.resize(n+1, vector<int>()); -> in solve
    // color.assign(n+1, -1);-> in solve
    if(dfs(1, 0)) cout<<"Bipartite\n";
    else cout<<"Not Bipartite\n";
}
```

Longest Increasing Sequence

```
vector<int> a(N);
int dp[N];
int lis(int i)
{
    if(dp[i] != -1)     return dp[i];
    int ans=1;
    for(int j=0; j<i; ++j)
    {
        if(a[i]>a[j]) ans = max(ans, lis(j)+1);
    }
    return dp[i] = ans;
}
int main()
{
    memset(dp,-1,sizeof(dp));
    int n; cin>>n;
    for(int i=0; i<n; ++i) cin>>a[i];
    int ans=0;
    for(int i=0; i<n; ++i) ans = max(ans, lis(i));
    cout<<ans;
}
```

Longest common subsequence

```
int longestCommonSubsequence(String text1, String text2) {
  int length1 = text1.length();
  int length2 = text2.length();
  int[][] dp = new int[length1 + 1][length2 + 1];
  // Build the dp array from the bottom up
  for (int i = 1; i <= length1; ++i) {
    for (int j = 1; j <= length2; ++j) {
      // If characters match, take diagonal value and add 1
      if (text1.charAt(i - 1) == text2.charAt(j - 1)) {
        dp[i][j] = dp[i - 1][j - 1] + 1;
      } else {
        dp[i][j] = Math.max(dp[i - 1][j], dp[i][j - 1]);
      }
    }
  }
  return dp[length1][length2];
}
```

Z hash

```
vector<ll> z_function(const string& s) {
  ll n = s.length();
  vector<ll> z(n);
  for (ll i = 1, l = 0, r = 0; i < n; i++) {
    if (i <= r) {
      z[i] = min(z[i-l], r-i+1);
    }
    while (i + z[i] < n and s[z[i]] == s[i+z[i]]) {
      ++z[i];
    }
    if (i + z[i] - 1 > r) {
```

```
        l = i, r = i + z[i] - 1;
      }
    }
    return z;
}
```

String Hashing

$hash(s) = s[0] + (s[1] * p) + (s[i] * p^2) + ... + (s[n-1] * p^n - 1) hash(s[i...j]) = (hash(s[0...j]) - hash(0...i-1))/(p^i)$;

- used double hashing.
- base = 31 and mod = 1e9+7 works if string's value = s[i] - 'a' + 1; (1 - 26)
- call precal_pow() from main()
- if getting WA even after 2 Hashes:

  use unpopular primes (!1e9 + 7)

  use 3 hashes (might give TLE as

  use other algoś
- some base = 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241, 251, 257, 263, 269, 271, 277, 281, 283, 293, 307, 311, 313, 317, 331, 337, 347, 349, 353, 359, 367, 373, 379, 383, 389, 397, 401, 409, 419, 421, 431, 433, 439, 443, 449, 457, 461, 463, 467, 479, 487, 491, 499;
- use int instead of long long and cast to ll with 1LL when doing any multiplication
- using Long Long Int could give TLE, because ll takes 2x more memory then int and some calculations (multiplication, modulo) are very expensive.

```
int N = 1e6 + 9;
const int MOD1 = 127657753, MOD2 = 987654319;
const int base1 = 137, base2 = 277;
vector<pair<int, int>>pw(N), inv_pw(N);

int BE(int x, int y, int mod){ // O(logn)
    int res = 1;
    x %= mod;
    if(x < 0) x += mod;
    while(y > 0){
        if(y & 1) res = (1LL * res * x) % mod; // MOD
        y >>= 1; // -> y /= 2;
        x = (1LL * x * x) % mod; // MOD
    } // MOD for larger numbers
    return res;
}

void precal_pow() { // O(N)
```

```
    pw[0] = {1, 1};
    inv_pw[0] = {1, 1};
    int inv_base1 = BE(base1, MOD1 - 2, MOD1);
    int inv_base2 = BE(base2, MOD2 - 2, MOD2);
    for(int i = 1; i < N; i++) {
        pw[i].first = (1LL * pw[i - 1].first * base1) % MOD1;
        pw[i].second = (1LL * pw[i - 1].second * base2) % MOD2;
        inv_pw[i].first = (1LL * inv_pw[i - 1].first *
        inv_base1) % MOD1;
        inv_pw[i].second = (1LL * inv_pw[i - 1].second *
        inv_base2) % MOD2;
    }
}

struct Hashing {
    int n;
    string s; // 0 - indexed
    vector<pair<int, int>> hs; // 1 - indexed
    Hashing() {}
    Hashing(string _s) { // O(n)
        n = size(_s);
        s = _s;
        hs.emplace_back(0, 0);
        for(int i = 0; i < n; i++) {
            pair<int, int> p;
            p.first = (hs[i].first + (1LL * pw[i].first * s[i])
            % MOD1) % MOD1;
            p.second = (hs[i].second + (1LL * pw[i].second *
            s[i]) % MOD2) % MOD2;
            hs.push_back(p);
        }
    }
    pair<int, int> get_hash(int l, int r) { // 1 - indexed
    // O(1)
        assert(1 <= l && l <= r && r <= n); // will cause RE
        pair<int, int> ans;
        ans.first = ((hs[r].first - hs[l - 1].first + MOD1) *
        1LL * inv_pw[l - 1].first) % MOD1;
        ans.second = ((hs[r].second - hs[l - 1].second + MOD2)
        * 1LL * inv_pw[l - 1].second) % MOD2;
        return ans;
    }
    pair<int, int> get_hash() { // gets full string hash
    // O(1)
        return get_hash(1, n);
    }
};
```

Inverse Hashing and Palindrome

```
    // Same as String Hashing, only change in hashing struct
    struct Hashing {
    int n;
    string s; // 0 - indexed
    vector<pair<int, int>> hs, rhs; // 1 - indexed
    Hashing() {}
    Hashing(string &_s) {
        n = size(_s);
```

```
        s = _s;
        hs.emplace_back(0, 0);
        rhs.emplace_back(0, 0);
        for(int i = 0; i < n; i++) {
            pair<int, int> p, rp;
            p.first = (hs[i].first + (1LL * pw[i].first * s[i])
            % MOD1) % MOD1;
            p.second = (hs[i].second + (1LL * pw[i].second *
            s[i]) % MOD2) % MOD2;
            hs.push_back(p);

            rp.first = (rhs[i].first + (1LL * pw[i].first * s[n
            - i - 1]) % MOD1) % MOD1;
            rp.second = (rhs[i].second + (1LL * pw[i].second *
            s[n - i - 1]) % MOD2) % MOD2;
            rhs.push_back(rp);
        }
    }
    pair<int, int> get_hash(int l, int r) { // 1 - indexed
        assert(1 <= l && l <= r && r <= n); // will cause RE
        pair<int, int> ans;
        ans.first = ((hs[r].first - hs[l - 1].first + MOD1) *
        1LL * inv_pw[l - 1].first) % MOD1;
        ans.second = ((hs[r].second - hs[l - 1].second + MOD2)
        * 1LL * inv_pw[l - 1].second) % MOD2;
        return ans;
    }
    pair<int, int> get_rev_hash(int _l, int _r) { // 1 -
    indexed
        int l = n - _r + 1;
        int r = n - _l + 1;
        assert(1 <= l && l <= r && r <= n); // will cause RE
        pair<int, int> ans;
        ans.first = ((rhs[r].first - rhs[l - 1].first + MOD1) *
        1LL * inv_pw[l - 1].first) % MOD1;
        ans.second = ((rhs[r].second - rhs[l - 1].second +
        MOD2) * 1LL * inv_pw[l - 1].second) % MOD2;
        return ans;
    }
    pair<int, int> get_hash() { // gets full string hash
        return get_hash(1, n);
    }
    bool is_palindrome(int l, int r) {
        return (get_hash(l, r) == get_rev_hash(l, r));
    }
};
```

Manachers Algorithm

```
/* notes:
1. modifies string by adding '#' bef&after every char;
2. p[i] = size of palindrome that exist in original string;
here i is the center of that palindrome of modified string;
3. palindrome at ith(modified string's) position is:
s.substr((index/2) - (pal_size/2), pal_size);
*/
struct Manachers { // O(n)
  vector < int > p;
```

```
  string s;
  Manachers(string & _s) {
    s = "#"; // build new string
    for (int i = 0; i < size(_s); i++) {
      s.push_back(_s[i]);
      s.push_back('#');
    }
    int n = size(s);
    p.assign(n, 0);
    int l = 0, r = 0; //curr largest pal range
    for (int i = 0; i < n; i++) {
      int mirror = l + (r - i);
      if (i < r) p[i] = min(r - i, p[mirror]);
      while (i + p[i] + 1 < n && i - p[i] - 1 >= 0 && s[i +
      p[i] + 1] == s[i - p[i] - 1]) {
        p[i]++;
      }
      if (i + p[i] > r) { // if new largest pal found
        l = i - p[i];
        r = i + p[i];
      }
    }
  }
  bool is_palindrome(int l, int r) { // O(1)
    l = 2 * l + 1;
    r = 2 * r + 1;
    int center = (l + r) / 2; // of modified string
    return p[center] >= (r - l + 1) / 2;
  }
};
```

Mathematical series

```

```

Matrix exponentiation

```
Matrix multiply(const Matrix &A, const Matrix &B) {
    int n = A.size(); Matrix result(n, vector<long long>(n,
    0));
    for (int i = 0; i < n; ++i) { for (int j = 0; j < n; ++j) {
    for (int k = 0; k < n; ++k) {
    result[i][j] = (result[i][j] + A[i][k] * B[k][j]) % mod; }
    } } return result;}
Matrix matrixExpo(Matrix A, long long power) { int n =
A.size(); Matrix result(n, vector<long long>(n, 0));
    for (int i = 0; i < n; ++i) result[i][i] = 1; while (power
    > 0) { if (power % 2 == 1) {
    result = multiply(result, A); } A = multiply(A, A); power
    /= 2; } return result; }
```

Rectangle	A = l × w = b • h
	l= length; w = width; b = b
Parallelogram	A = b • h
Triangle	A = $\frac{1}{2}$ • b• h
Trapezoid	A = $\frac{1}{2}$ h (b1 + b2)
Regular Polygon	A = $\frac{1}{2}$ • a • p
	a = apothem; p = perimeter
Circle (circumference)	C = 2r = d
	r = radius; d = diameter
Circle (area)	A = r2
Rectangular Solid	#
(also called right rectangular prism)	SA formula assumes a "close
Cube	#
[special case of rectangular solid with all edges equal)	SA formula assumes a "close
Cylinder	#
	SA formula assumes a "close
Cone	#
	SA formula assumes a "close
Sphere	#
Right Prism	Vright prism = B • h; SA =
(rectangular or triangular)	B = area of the base; h = he
Pyramid	#
[assuming all of the faces (not the base) are the same]	B = area of the base; h = he

Digit DP

```cpp
//Digit DP Template: Count numbers  N with K non-zero digits
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
string num;
ll dp[20][2][20]; // pos, tight, count_nonzero
// Digit DP function
ll count(int pos, bool tight, int cnt_nonzero) {
    if (cnt_nonzero < 0) return 0;
    if (pos == num.size()) return cnt_nonzero == 0;
    if (dp[pos][tight][cnt_nonzero] != -1)
        return dp[pos][tight][cnt_nonzero];
    int limit = tight ? num[pos] - '0' : 9;
    ll res = 0;
    for (int digit = 0; digit <= limit; ++digit) {
        bool next_tight = tight && (digit == limit);
        int new_cnt = cnt_nonzero - (digit != 0 ? 1 : 0);
        res += count(pos + 1, next_tight, new_cnt);
    }
    return dp[pos][tight][cnt_nonzero] = res;
}
ll solve(string n, int k) {
    num = n;
    memset(dp, -1, sizeof(dp));
    return count(0, 1, k);
}
int main() {
    string N;
    int K;
    cin >> N >> K;
    cout << solve(N, K) << '\n';
```

```
}
```

Articulation Points and Bridges

```
// Tarjan's Algorithm for finding Articulation Points and
Bridges in an undirected graph
const int N = 1e5 + 5;
vector<int> adj[N];
bool visited[N];
int tin[N], low[N], timer;
set<int> articulation_points;
vector<pair<int, int>> bridges;
void dfs(int u, int parent = -1) {
    visited[u] = true;
    tin[u] = low[u] = timer++;
    int children = 0;
    for (int v : adj[u]) {
        if (v == parent) continue;
        if (visited[v]) {
            // Back edge
            low[u] = min(low[u], tin[v]);
        } else {
            dfs(v, u);
            low[u] = min(low[u], low[v]);
            // Bridge condition
            if (low[v] > tin[u]) {
                bridges.emplace_back(u, v);
            }
            // Articulation point condition
            if (low[v] >= tin[u] && parent != -1) {
                articulation_points.insert(u);
            }
            ++children;
        }
    }
    // Special case for root node
    if (parent == -1 && children > 1) {
        articulation_points.insert(u);
    }
}
// Driver function
void find_cutpoints_and_bridges(int n) {
    timer = 0;
    articulation_points.clear();
    bridges.clear();
    fill(visited, visited + n + 1, false);
    for (int i = 1; i <= n; ++i) {
        if (!visited[i]) {
            dfs(i);
        }
    }
}
// Example usage:
// 1-based node indexing
// for (int i = 0; i < m; ++i) {
//     int u, v; cin >> u >> v;
//     adj[u].push_back(v);
//     adj[v].push_back(u);
// }
```

```
// find_cutpoints_and_bridges(n);
// articulation_points -> set of articulation points
// bridges -> vector of bridge edges
```

Bitmask DP

```
// Bitmask DP: Travelling Salesman Problem (TSP)
// Time complexity: O(n^2 * 2^n)
const int INF = 1e9;
int n; // number of cities
int cost[20][20]; // cost[i][j]: cost to go from city i to city j
int dp[1 << 20][20]; // dp[mask][u]: min cost to reach mask ending at u
int tsp(int mask, int u) {
    if (mask == (1 << n) - 1) {
        return cost[u][0]; // return to starting city
    }
    if (dp[mask][u] != -1) return dp[mask][u];

    int ans = INF;
    for (int v = 0; v < n; ++v) {
        if (!(mask & (1 << v))) {
            ans = min(ans, cost[u][v] + tsp(mask | (1 << v), v));
        }
    }
    return dp[mask][u] = ans;
}
// Example usage:
// int main() {
//     cin >> n;
//     for (int i = 0; i < n; ++i)
//         for (int j = 0; j < n; ++j)
//             cin >> cost[i][j];
//
//     memset(dp, -1, sizeof(dp));
//     cout << tsp(1, 0) << '\n'; // start from city 0 with mask = 1 (only city 0 visited)
// }
```

Extended Euclidean Algorithm (for modular inverse & linear Diophantine eq.)

```
// Extended Euclidean Algorithm
// Solves: a * x + b * y = gcd(a, b)
// If gcd(a, b) = 1, x is the modular inverse of a mod b
// Returns gcd(a, b), and sets x, y such that: a*x + b*y = gcd(a, b)
long long extended_gcd(long long a, long long b, long long &x, long long &y) {
    if (b == 0) {
        x = 1;
        y = 0;
        return a;
    }
```

```
    long long x1, y1;
    long long gcd = extended_gcd(b, a % b, x1, y1);
    x = y1;
    y = x1 - (a / b) * y1;
    return gcd;
}
// Computes modular inverse of a under modulo m (i.e., a^(-1) mod m)
// Returns -1 if inverse doesn't exist
long long mod_inverse(long long a, long long m) {
    long long x, y;
    long long g = extended_gcd(a, m, x, y);
    if (g != 1) return -1; // Inverse doesn't exist
    x = (x % m + m) % m;
    return x;
}
// Example usage:
// long long x, y;
// long long g = extended_gcd(30, 20, x, y); // x, y will satisfy 30x + 20y = gcd
// long long inv = mod_inverse(3, 11); // inv = 4 because 3*4 1 mod 11
```

Convex Hull Trick

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
// Line: y = m * x + b
struct Line {
    ll m, b;
    mutable function<const Line*()> succ;
    bool operator<(const Line& rhs) const {
        if (!rhs.succ) return false;
        const Line* s = succ();
        if (!s) return false;
        return (b - s->b) * (rhs.m - m) < (m - s->m) * (rhs.b - b);
    }
};
// Convex Hull Trick for min query (slopes must be added in decreasing order)
struct CHT {
    deque<Line> dq;
    bool bad(Line a, Line b, Line c) {
        // Determines if line b is unnecessary between a and c
        return (b.b - a.b) * (c.m - b.m) >= (b.b - c.b) * (a.m - b.m);
    }
    void add_line(ll m, ll b) {
        Line new_line = {m, b};
        while (dq.size() >= 2 && bad(dq[dq.size() - 2], dq.back(), new_line)) {
            dq.pop_back();
        }
        dq.push_back(new_line);
    }
    ll query(ll x) {
```

```
        while (dq.size() >= 2 && dq[0].m * x + dq[0].b >=
        dq[1].m * x + dq[1].b) {
            dq.pop_front();
        }
        return dq[0].m * x + dq[0].b;
    }
};
// Example usage: dp[i] = min(dp[j] + a[i] * b[j]) with a[i]
increasing
int main() {
    int n;
    cin >> n;
    vector<ll> a(n), b(n), dp(n);
    for (int i = 0; i < n; ++i) cin >> a[i]; // a[i] increasing
    for (int i = 0; i < n; ++i) cin >> b[i]; // b[i] = slope of
    line
    CHT cht;
    dp[0] = 0;
    cht.add_line(b[0], dp[0]);
    for (int i = 1; i < n; ++i) {
        dp[i] = cht.query(a[i]);
        cht.add_line(b[i], dp[i]);
    }
    cout << "Minimum value dp[n-1] = " << dp[n - 1] << '\n';
}
//The version shown is for decreasing slopes (b[j] increasing),
using min queries. You can adapt this for max queries,
increasing slopes, or even a fully dynamic version using
multiset and __int128.
```

Knapsack Variants (0/1, Unbounded, Bounded)

```
    // 0/1 Knapsack
    //Each item can be taken at most once.
    //Maximize total value without exceeding capacity.
#include <bits/stdc++.h>
using namespace std;
int main() {
    int n, W;
    cin >> n >> W;
    vector<int> wt(n), val(n);
    for (int i = 0; i < n; i++) cin >> wt[i] >> val[i];
    vector<int> dp(W + 1, 0);
    for (int i = 0; i < n; i++) {
        for (int w = W; w >= wt[i]; w--) {
            dp[w] = max(dp[w], dp[w - wt[i]] + val[i]);
        }
    }
    cout << dp[W] << '\n';
}
    //2. Unbounded Knapsack
    //You can take unlimited copies of each item.
    //Maximize total value without exceeding capacity.
#include <bits/stdc++.h>
using namespace std;
int main() {
    int n, W;
    cin >> n >> W;
    vector<int> wt(n), val(n);
```

```
    for (int i = 0; i < n; i++) cin >> wt[i] >> val[i];
    vector<int> dp(W + 1, 0);
    for (int i = 0; i < n; i++) {
        for (int w = wt[i]; w <= W; w++) {
            dp[w] = max(dp[w], dp[w - wt[i]] + val[i]);
        }
    }
    cout << dp[W] << '\n';
}
    //3. Bounded Knapsack
    //Each item has a limited quantity cnt[i].
    //Maximize total value without exceeding capacity.
#include <bits/stdc++.h>
using namespace std;
int main() {
    int n, W;
    cin >> n >> W;
    vector<int> wt(n), val(n), cnt(n);
    for (int i = 0; i < n; i++) cin >> wt[i] >> val[i] >>
    cnt[i];
    vector<int> dp(W + 1, 0);
    for (int i = 0; i < n; i++) {
        // Binary decomposition optimization
        int k = 1, c = cnt[i];
        while (k <= c) {
            int weight = wt[i] * k;
            int value = val[i] * k;
            for (int w = W; w >= weight; w--) {
                dp[w] = max(dp[w], dp[w - weight] + value);
            }
            c -= k;
            k <<= 1;
        }
        if (c > 0) {
            int weight = wt[i] * c;
            int value = val[i] * c;
            for (int w = W; w >= weight; w--) {
                dp[w] = max(dp[w], dp[w - weight] + value);
            }
        }
    }
    cout << dp[W] << '\n';
}
```

Meet in the Middle

```
    //Problem: Given an array and a target sum S, determine if
    any subset sums exactly to S.
    //Approach:
    //Split array into two halves.
    //Generate all subset sums of each half.
    //For each sum in the first half, binary search if S - sum
    exists in the second half.
#include <bits/stdc++.h>
using namespace std;
int main() {
    int n;
    long long S;
    cin >> n >> S;
```

```
    vector<long long> arr(n);
    for (int i = 0; i < n; i++) cin >> arr[i];
    // Split array into two halves
    int n1 = n / 2;
    int n2 = n - n1;
    vector<long long> leftSums, rightSums;
    // Generate all subset sums of left half
    for (int mask = 0; mask < (1 << n1); mask++) {
        long long sum = 0;
        for (int i = 0; i < n1; i++) {
            if (mask & (1 << i)) sum += arr[i];
        }
        leftSums.push_back(sum);
    }
    // Generate all subset sums of right half
    for (int mask = 0; mask < (1 << n2); mask++) {
        long long sum = 0;
        for (int i = 0; i < n2; i++) {
            if (mask & (1 << i)) sum += arr[n1 + i];
        }
        rightSums.push_back(sum);
    }
    sort(rightSums.begin(), rightSums.end());
    // Check if exists leftSum + rightSum = S
    bool found = false;
    for (auto &x : leftSums) {
        long long need = S - x;
        if (binary_search(rightSums.begin(), rightSums.end(),
        need)) {
            found = true;
            break;
        }
    }
    cout << (found ? "YES\n" : "NO\n");
}
```

Euler's Totient Function

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
// Computes (n): number of integers in [1, n] that are coprime
to n
ll phi_single(ll n) {
    ll result = n;
    for (ll p = 2; p * p <= n; ++p) {
        if (n % p == 0) {
            while (n % p == 0)
                n /= p;
            result -= result / p;
        }
    }
    if (n > 1)
        result -= result / n;
    return result;
}
// Computes (1), (2), ..., (n) using sieve in O(n log log n)
const int MAXN = 1e6 + 5;
int phi[MAXN];
```

```cpp
void compute_totients_up_to_n(int n) {
    for (int i = 0; i <= n; ++i) phi[i] = i;
    for (int i = 2; i <= n; ++i) {
        if (phi[i] == i) { // i is prime
            for (int j = i; j <= n; j += i) {
                phi[j] -= phi[j] / i;
            }
        }
    }
}
int main() {
    // Example 1: Single number (n)
    ll n;
    cout << "Enter a number n: ";
    cin >> n;
    cout << "phi(" << n << ") = " << phi_single(n) << "\n\n";
    // Example 2: Compute (1) to (n) using sieve
    int limit;
    cout << "Enter limit to compute phi(1..limit): ";
    cin >> limit;
    compute_totients_up_to_n(limit);
    cout << "Euler's Totient values from 1 to " << limit <<
    ":\n";
    for (int i = 1; i <= limit; ++i) {
        cout << "phi(" << i << ") = " << phi[i] << '\n';
    }
}
```

Miller-Rabin Primality Test

```cpp
// Probabilistic (but deterministic for small enough values
with fixed bases)
using u64 = uint64_t;
using u128 = __uint128_t;
// Fast modular multiplication (to avoid overflow)
u64 mod_mul(u64 a, u64 b, u64 mod) {
    return (u128)a * b % mod;
}
// Fast modular exponentiation
u64 mod_pow(u64 base, u64 exp, u64 mod) {
    u64 result = 1;
    base %= mod;
    while (exp > 0) {
        if (exp & 1) result = mod_mul(result, base, mod);
        base = mod_mul(base, base, mod);
        exp >>= 1;
    }
    return result;
}
// Miller-Rabin test for a single base
bool check_composite(u64 n, u64 a, u64 d, int s) {
    u64 x = mod_pow(a, d, n);
    if (x == 1 || x == n - 1) return false;
    for (int r = 1; r < s; ++r) {
        x = mod_mul(x, x, n);
        if (x == n - 1) return false;
    }
    return true; // definitely composite
}
```

```cpp
// Miller-Rabin primality test
bool is_prime(u64 n) {
    if (n < 2) return false;
    if (n == 2 || n == 3) return true;
    if (n % 2 == 0) return false;
    // Write n-1 as d * 2^s
    u64 d = n - 1;
    int s = 0;
    while ((d & 1) == 0) {
        d >>= 1;
        ++s;
    }
    // Deterministic bases for 64-bit integers
    for (u64 a : {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37})
    {
        if (a >= n) break;
        if (check_composite(n, a, d, s)) return false;
    }
    return true;
}
// Example usage:
// bool prime = is_prime(1e18 + 3); // fast check for large number
```

Euler Tour (Tree Flattening)

```cpp
#include <bits/stdc++.h>
using namespace std;
const int N = 1e5 + 5;
vector<int> tree[N];
int in[N], out[N], flat_tree[2 * N], timer = 0;
void dfs(int u, int p) {
    in[u] = timer;
    flat_tree[timer++] = u;
    for (int v : tree[u]) {
        if (v != p) {
            dfs(v, u);
        }
    }
    out[u] = timer;
    flat_tree[timer++] = u;
}
int main() {
    int n;
    cin >> n;
    // Input tree with n nodes (1-indexed or 0-indexed)
    for (int i = 1; i <= n - 1; ++i) {
        int u, v;
        cin >> u >> v;
        tree[u].push_back(v);
        tree[v].push_back(u);
    }
    dfs(1, -1); // assuming 1 as root
    cout << "In and Out Time:\n";
    for (int i = 1; i <= n; ++i) {
        cout << "Node " << i << ": in = " << in[i] << ", out = " << out[i] << '\n';
    }
    cout << "\nEuler Tour (Flattened Tree):\n";
    for (int i = 0; i < timer; ++i) {
```

```
        cout << flat_tree[i] << ' ';
    }
    cout << '\n';
}
```

Fenwick Tree (Binary Indexed Tree)

```
//Fenwick Tree (BIT) Implementation for 1-based indexing
#include <bits/stdc++.h>
using namespace std;
struct FenwickTree {
    int n;
    vector<long long> bit;
    FenwickTree(int size) {
        n = size;
        bit.assign(n + 1, 0);
    }
    // Update index idx by value delta (add delta to A[idx])
    void update(int idx, long long delta) {
        while (idx <= n) {
            bit[idx] += delta;
            idx += idx & (-idx);
        }
    }
    // Query prefix sum from 1 to idx
    long long query(int idx) {
        long long sum = 0;
        while (idx > 0) {
            sum += bit[idx];
            idx -= idx & (-idx);
        }
        return sum;
    }
    // Query sum in range [l, r]
    long long range_query(int l, int r) {
        return query(r) - query(l - 1);
    }
};
int main() {
    int n, q;
    cin >> n >> q;
    FenwickTree fenw(n);
    // Input initial array and build Fenwick Tree
    for (int i = 1; i <= n; i++) {
        int x; cin >> x;
        fenw.update(i, x);
    }
    while (q--) {
        int t; cin >> t;
        if (t == 1) {
            // Update operation: add val to A[idx]
            int idx; long long val;
            cin >> idx >> val;
            fenw.update(idx, val);
        } else if (t == 2) {
            // Query sum from l to r
            int l, r;
            cin >> l >> r;
            cout << fenw.range_query(l, r) << '\n';
        }
```

```
    }
}
```

Game Theory Basics (Nim Game, Grundy Numbers)

```
    //Nim Game
    //You have several piles of stones.
    // Players alternate removing any number of stones from one pile.
    // The player who removes the last stone wins.
    // The Nim value (xor of pile sizes) decides the winner:
    // If XOR = 0 → second player wins (if both play optimally)
    // Else → first player wins
#include <bits/stdc++.h>
using namespace std;
int main() {
    int n; // number of piles
    cin >> n;
    int nim_sum = 0;
    for (int i = 0; i < n; i++) {
        int pile;
        cin >> pile;
        nim_sum ^= pile;
    }
    if (nim_sum != 0)
        cout << "First player wins\n";
    else
        cout << "Second player wins\n";
}
    //Grundy Numbers (Sprague-Grundy Theorem)
    //Every impartial game position can be assigned a Grundy number (or nimber).
    //Grundy number = minimum excluded value (mex) of the Grundy numbers of next possible states.
    //The XOR of Grundy numbers of independent game components determines the winner.
    //Example: Calculate Grundy numbers for a game where from a pile of size n, you can remove either 1 or 2 stones.
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 1000;
int grundy[MAXN + 1];
// mex function: minimum excluded value
int mex(const set<int>& s) {
    int m = 0;
    while (s.count(m)) m++;
    return m;
}
void compute_grundy(int n) {
    grundy[0] = 0; // no moves => losing state
    for (int i = 1; i <= n; i++) {
        set<int> next_states;
        if (i - 1 >= 0) next_states.insert(grundy[i - 1]);
        if (i - 2 >= 0) next_states.insert(grundy[i - 2]);
        grundy[i] = mex(next_states);
    }
}
int main() {
```

```
    int n;
    cin >> n;
    compute_grundy(n);
    cout << "Grundy numbers for piles 0 to " << n << ":\n";
    for (int i = 0; i <= n; i++) {
        cout << "Pile size " << i << ": Grundy = " << grundy[i]
        << '\n';
    }
}
```

Pollard's Rho Factorization

```
//Absolutely! Pollard's Rho is a fast probabilistic algorithm
for integer //factorization, especially useful for factoring
large numbers (up to //~10^18) efficiently when trial division
is too slow.
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
// Modular multiplication to avoid overflow
ll modmul(ll a, ll b, ll mod) {
    ll result = 0;
    a %= mod;
    while (b) {
        if (b & 1) {
            result = (result + a) % mod;
        }
        a = (a * 2) % mod;
        b >>= 1;
    }
    return result;
}
// Modular exponentiation
ll modpow(ll base, ll exp, ll mod) {
    ll result = 1 % mod;
    base %= mod;
    while (exp > 0) {
        if (exp & 1) result = modmul(result, base, mod);
        base = modmul(base, base, mod);
        exp >>= 1;
    }
    return result;
}
// Miller-Rabin primality test
bool miller_rabin(ll n) {
    if (n < 2) return false;
    int r = 0;
    ll d = n - 1;
    while ((d & 1) == 0) {
        d >>= 1;
        r++;
    }
    // Test bases
    vector<ll> bases = {2, 3, 5, 7, 11, 13, 17, 19, 23};
    for (ll a : bases) {
        if (a >= n) break;
        ll x = modpow(a, d, n);
        if (x == 1 || x == n - 1) continue;

        bool composite = true;
```

```
        for (int i = 1; i < r; i++) {
            x = modmul(x, x, n);
            if (x == n - 1) {
                composite = false;
                break;
            }
        }
        if (composite) return false;
    }
    return true;
}
// Pollard's Rho algorithm
ll pollard_rho(ll n) {
    if (n % 2 == 0) return 2;
    if (miller_rabin(n)) return n;
    mt19937_64
    rng(chrono::steady_clock::now().time_since_epoch().count());
    ll x = rng() % (n - 2) + 2;
    ll y = x;
    ll c = rng() % (n - 1) + 1;
    ll d = 1;
    auto f = [&](ll val) {
        return (modmul(val, val, n) + c) % n;
    };
    while (d == 1) {
        x = f(x);
        y = f(f(y));
        d = gcd(abs(x - y), n);
        if (d == n) return pollard_rho(n);
    }
    if (miller_rabin(d)) return d;
    else return pollard_rho(d);
}
// Factorization function returning prime factors (not
necessarily sorted)
void factorize(ll n, vector<ll> &factors) {
    if (n == 1) return;
    if (miller_rabin(n)) {
        factors.push_back(n);
        return;
    }
    ll divisor = pollard_rho(n);
    factorize(divisor, factors);
    factorize(n / divisor, factors);
}
int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    ll n;
    cin >> n;
    vector<ll> factors;
    factorize(n, factors);
    sort(factors.begin(), factors.end());
    cout << "Factors:\n";
    for (ll f : factors) cout << f << " ";
    cout << "\n";
}
```

Trie with Bitwise & String operations

```cpp
//String Trie (for lowercase alphabets a-z)
#include <bits/stdc++.h>
using namespace std;
struct TrieNode {
    TrieNode* children[26] = {};
    int count = 0;  // how many strings pass here (for prefix count)
    bool isEnd = false;
};
struct Trie {
    TrieNode* root;
    Trie() { root = new TrieNode(); }
    void insert(const string& s) {
        TrieNode* node = root;
        for (char c : s) {
            int idx = c - 'a';
            if (!node->children[idx]) node->children[idx] = new TrieNode();
            node = node->children[idx];
            node->count++;
        }
        node->isEnd = true;
    }
    bool search(const string& s) {
        TrieNode* node = root;
        for (char c : s) {
            int idx = c - 'a';
            if (!node->children[idx]) return false;
            node = node->children[idx];
        }
        return node->isEnd;
    }
    int prefixCount(const string& prefix) {
        TrieNode* node = root;
        for (char c : prefix) {
            int idx = c - 'a';
            if (!node->children[idx]) return 0;
            node = node->children[idx];
        }
        return node->count;
    }
};
```

General Tips

- Do as little division as possible.
- `float` (least precise) < `double` < `long double` (most precise).
- Instead of `a == b`, use `fabs(a - b) < epsilon` (e.g., 10^{-9}) when comparing floating-point numbers.
- Check for integer overflow.
- Using a fixed number of iterations in binary search (e.g., 100) is often more reliable than epsilon-based termination (e.g., absolute or relative error of 10^{-6}).

- Assigning `lo` and `hi` as constant numbers (e.g., 10^9) may result in wrong answers; calculate bounds based on problem constraints for floating-point precision in binary search.
- Check for out-of-bounds errors in arrays (mostly results in RTE).
- Any possibility of negative index?
- Outer loop and inner loop shouldn't have the same iterator variable.
- To pass tricky test cases, sometimes shuffling the array helps:
  - `mt19937 rng(chrono::steady_clock::now().time_since_epoch().coun` // Generates seed.
  - `shuffle(nums.begin(), nums.end(), rng);` // $O(n)$ complexity.
- `sin`, `cos`, `asin` (`sin`$^{-1}$), `acos`, etc., return radians; multiply by $(180/\pi)$ to convert to degrees.
- If $a \bmod k = x$, one of the following holds:
  - $a \bmod 2k = x$
  - $a \bmod 2k = x + k$
  - If $a \bmod k = b \bmod k$, then $(a - b) \bmod k = 0$
- Heavy-Light Trick: If string length $> \sqrt{\texttt{max\_string\_length}}$, use hashing; else, use a trie.
- Send debug log to a file:
  - `ofstream fcerr("log.txt");` // Place before debug template.
  - Replace all `cerr` with `fcerr`.
  - `fcerr.close();` // Close the file before returning 0.

Sorting a Vector of Pairs

```
bool cmp(pair<int, int> x, pair<int, int> y) {
    if (x.first == y.first) {
        return x.second < y.second;
    }
    return x.first < y.first;
}
```