Template 1

Binary Search SQRT Function 1

Pair Sort 2

LCM 2

GCD 2

Power 2

Sieve 2

Prime Factorization 2

Integer to Binary 2

Binary to Integer 3

Base Conversion 3

Factorial 3

nCr 3

nPr 3

Fibonacci 3

Fibonacci Using DP 4

Maximum Subarray Sum 4

Generating Subset 4

BFS 4

DFS Topological Sort 4

Dijkstra Algorithm (Undirected-Weighted) 5

Dijkstra Algorithm (Directed-Weighted) 6

Cycle Detection (Undirected DFS) 6

Cycle Detection (Directed DFS) 7

Complexity 7

Bit Operations 8

Bit Manipulation 8

Number Theory 8

Modular Arithmatic 8

Combinatorics 9

Probability 9

Game Theory 9

Technique 9

Verdict 9

Geometry 10

**\*\*Template\*\***

#include<bits/stdc++.h>

using namespace std;

#define optimize() ios\_base::sync\_with\_stdio(0);

#define all\_op() cin.tie(0);cout.tie(0);

#define sp ' '

#define endl '\n'

const long double pi=acos(-1);

typedef long long int ll;

int main()

{

optimize();

all\_op();

ll \_=1;

cin >> \_;

while(\_--)

{

}

return 0;

}

**\*\*Binary Search sqrt Function\*\***

ll bs\_sqrt(ll x)

{

ll left = 0, right = 2000000123;

while (right > left)

{

ll mid = (left + right) / 2;

if (mid \* mid > x) right = mid;

else left = mid + 1;

}

return left - 1;

}

**\*\*Pair Sort\*\***

bool cmp(const pair<long long int,long long int>&p1,const pair<long long int,long long int>&p2)

{

if(p1.first<p2.first) return 1;

else if(p1.first==p2.first)

return (p1.second < p2.second);

return 0;

}

**\*\*LCM\*\***

ll lcm(ll a,ll b){ return a\*(b/\_\_gcd(a,b)); }

**\*\*GCD\*\***

int gcd(int a, int b)

{

if (b == 0)

return a;

return gcd(b, a % b);

}

**\*\*Power\*\***

ll power(ll a,ll n)//,ll m)

{

int res=1;

while(n)

{

if(n%2)

{

res=res\*a;//(res\*(a%m))%m;

}

n/=2;

a=a\*a;//(a\*(a%m))%m;

}

return res;

}

**\*\*Sieve of Eratosthenes\*\***

bool a[100000000];

void sieve()

{

int mx=1e6;

a[1]=1;

for(int i=2; i<=mx; i++)

{

if(a[i]==0)

{

for(int j=i+i; j<=mx; j+=i)

{

if(a[j]==0)

a[j]=i;

}

}

}

}

**\*\*Prime Factorization\*\***

bool a[100000000];

void prime(int n)

{

int mx=1e6;

for(int i=2;i<=mx;i++)

{

if(a[i]==0)

{

for(int j=i;j<=mx;j+=i)

{

if(a[j]==0)

a[j]=i;

}

}

}

while(n>1)

{

cout << a[n] << ' ';

n=n/a[n];

}

cout << endl;

}

**\*\*Integer to Binary\*\***

string bin(ll num)

{

string str="00000000000000000000000000000000";

ll i=0;

while(num)

{

if(num & 1) // 1

str[i]='1';

else // 0

str[i]='0';

num>>=1; // Right Shift by 1

i++;

}

//reverse(str.begin(),str.end());

return str;

}

**\*\*Binary to Integer\*\***

ll intob(string str)

{

string n = str;

ll val = 0;

ll temp = 1;

ll len = n.length();

for (ll i = len - 1; i >= 0; i--)

{

if (n[i] == '1')

{

val += temp;

}

temp = temp \* 2;

}

return val;

}

**\*\*Base Convertion\*\***

#include<bits/stdc++.h>

using namespace std;

void convert10tobaseN(int N, int base2)

{

//converting any base to base 10

int dec = 0, i = 0, rem,base1=10;

while (N!=0)

{

rem = N % 10;

N /= 10;

dec += rem \* pow(base1, i);

++i;

}

printf("%d\n",dec);

//converting base10 to any base

long long bin = 0;

rem=0;

i = 1;

int n=dec;

while (n!=0)

{

rem = n % base2;

n /= base2;

bin += rem \* i;

i \*= 10;

}

printf("%lld\n",bin);

return;

}

int main(void)

{

int number=16;

int numberBase=10;

int reqBase=2;

convert10tobaseN(number, reqBase);

return 0;

}

**\*\*Factorial\*\*** O(n)

ll factorial(ll y) {

ll i, fact = 1;

for (i = 2; i <= y; i++) {

fact = fact \* i;

}

return fact;

}

**\*\*nCr** \*\* O(r)

ll nCr(ll n, ll r) {

double res = 1;

for (ll i = 1; i <= r; ++i)

res = res \* (n - r + i) / i;

return (ll)(res + 0.01);

}

**\*\*nPr**\*\* O(r+r)

ll nPr(ll n, ll r)

{

return nCr(n,r)\*factorial(r);

}

**\*\*Fibonacci\*\***

ll fibo(ll n)

{

if(n==1)

return 0;

else if(n==2)

return 1;

return fibo(n-1)+fibo(n-2);

}

**\*\*Fibonacci --Using DP\*\***

ll fibo\_seq(ll number) {

std::vector<int> table(number + 1);

table[0] = 0;

table[1] = 1;

for (ll i = 2; i <= number; i++) {

table[i] = table[i - 1] + table[i - 2];

}

return table[number];

}

**\*\*Maximum Subarray Sum\*\* O(N)**

int best = 0, sum = 0;

for (int k = 0; k < n; k++)

{

sum = max(array[k],sum+array[k]);

best = max(best,sum);

}

cout << best << "\n";

**\*\*Generating Subsets\*\***

for (int b = 0; b < (1<<n); b++)

{

vector<int> subset;

for (int i = 0; i < n; i++)

{

if (b&(1<<i)) subset.push\_back(i);

}

}

**\*\*BFS\*\* O(N+E)**

void bfs(int x)

{

for(auto u:level) u=0;

level[x]=1;

queue<int>q;

q.push(x);

while(!q.empty())

{

int u=q.front();

q.pop();

for(auto v:adj[u])

{

if(level[v]==0)

{

level[v]=level[u]+1;

q.push(v);

}

}

}

}

**\*\*DFS\*\***

long long int const mx=2e4+123;

vector<long long int>adj[mx];

vector<long long int>freq(mx);

bool visited[mx];

long long int cnt,cnt1;

void dfs(ll u,ll color)

{

visited[u]=1;

if(color==1) cnt1++;

cnt++;

ll temp;

(color==1)?temp=2:temp=1;

for(auto v:adj[u])

{

if(!visited[v])

{

dfs(v,temp);

}

}

}

**\*\*DFS TOP. SORT\*\* O(N+E)**

#include<bits/stdc++.h>

using namespace std;

#define ll long long

#define vl vector<ll>

#define vvl vector<vector<ll>>

void dfs(int source, vvl &v, vl &vis, vl &sequence)

{

vis[source]=1;

//cout << source << ' '; // do not need in top-sort/Topological sort

for(auto &x: v[source]){

if(vis[x]==-1){

//cout << "-->"; // do not need in top-sort/Topological sort

dfs(x,v,vis,sequence);

}

}//completely visited

sequence.push\_back(source);//need in top-sort/Topological sort

}

int main(){

int n,e;

cin >> n >> e;

vvl v(n+1);

vl vis(n+1,-1);

vl sequence;

for(int i=0; i<e; i++){

ll x,y;

cin >> x >> y;

v[x].push\_back(y);

//v[y].push\_back(x);// do not need in top-sort/Topological sort

}

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

if(vis[i]==-1){

dfs(i,v,vis,sequence);

}

}

// DFS END

reverse(sequence.begin(),sequence.end());

for(auto &x:sequence)

cout << x << ' ';

}

**\*\*Dijkstra's Algorithm | Shortest Path In Undirected Weighted Graphs\*\***

// T.C. = O((N+E)log N)

// S.C. = 2\*O(N)

#include<bits/stdc++.h>

using namespace std;

#define ll long long

#define vl vector<ll>

int main()

{

ll n,e;

cin >> n >> e;

vector<pair<ll,ll>> adj[n+1];

for(ll i=0; i<e; i++)

{

ll x,y,w;

cin >> x >> y >> w;

adj[x].push\_back({y,w});

adj[y].push\_back({x,w});

}

ll source;

cin >> source;

// Dijkstra's Algorithm start

priority\_queue<pair<ll,ll>,vector<pair<ll,ll>>,greater<pair<ll,ll>>> pq;

vl dist\_arr(n+1,LONG\_MAX);

dist\_arr[source] = 0;

pq.push({0,source});

while(!pq.empty())

{

ll dist = pq.top().first;

ll prev = pq.top().second;

pq.pop();

for(auto &x:adj[prev])

{

ll next = x.first;

ll nextdist = x.second;

if(dist\_arr[next]>dist\_arr[prev]+nextdist)

{

dist\_arr[next]=dist\_arr[prev]+nextdist;

pq.push({dist\_arr[next],next});

}

}

}

cout << "the distances from source, " << source << ", are : \n";

for(ll i=1; i<=n; i++)

cout << dist\_arr[i] << ' ';

cout << "\n";

return 0;

}

**\*\*Shortest Path In Directed Weighted Graphs\*\***

#include<bits/stdc++.h>

using namespace std;

#define ll long long

#define vl vector<ll>

#define vvl vector<vector<ll>>

void topsort(ll source, vl &vis, stack<ll> &st, vector<pair<ll,ll>> adj[])

{

vis[source]=1;

for(auto x: adj[source])

{

if(!vis[x.first])

{

topsort(x.first,vis,st,adj);

}

}

st.push(source);

}

void shortestpath(ll src, ll n, vector<pair<ll,ll>> adj[])

{

vl vis(n,0);

stack<ll> st;

for(ll i=0; i<n; i++)

if(!vis[i])

topsort(i,vis,st,adj);

vl dist(n,1e9);

dist[src]=0;

while(!st.empty())

{

ll node = st.top();

st.pop();

if(dist[node]!=1e9)

{

for(auto &it:adj[node])

{

if(dist[node]+it.second < dist[it.first])

{

dist[it.first]=dist[node]+it.second;

}

}

}

}

for(ll i=0; i<n; i++)

dist[i]==1e9 ? cout << "INF":cout << dist[i] << " ";

}

int main()

{

ll n,e;

cin >> n >> e;

vector<pair<ll,ll>> adj[n];

for(ll i=0; i<e; i++)

{

ll x,y,w;

cin >> x >> y >> w;

adj[x].push\_back({y,w});

}

shortestpath(0,n,adj);

}

**\*\*Cycle Detection in Undirected Graph using DFS\*\***

#include<bits/stdc++.h>

using namespace std;

#define ll long long

#define vl vector<ll>

#define vvl vector<vector<ll>>

bool dfs(ll son, ll father, vvl &v, vl &par, vl &vis)

{

vis[son]=1;

par[son]=father;

for(auto &x: v[son])

{

if(vis[x]==-1)

dfs(x,son,v,par,vis);

else if(x!=par[son])

return true;

}

return false;

}

int main()

{

ll n,e;

cin >> n >> e;

vvl v(n+1);

vl vis(n+1,-1);

vl par(n+1);

for(ll i=0; i<e; i++)

{

ll x,y;

cin >> x >> y;

v[x].push\_back(y);

v[y].push\_back(x);

}

for(ll i=1; i<=n; i++)

{

bool check = false;

if(vis[i]==-1)

{

check = dfs(i,-1,v,par,vis);

if(check)

{

cout << "Has cycle\n";

return 0;

}

}

}

cout << "No cycle\n";

}

**\*\*Cycle Detection in Directed Graph using DFS\*\***

#include<bits/stdc++.h>

using namespace std;

#define ll long long

#define vl vector<ll>

#define vvl vector<vector<ll>>

bool dfs(ll node, vvl adj, vl &vis, vl &dfsvis)

{

vis[node]=1;

dfsvis[node]=1;

for(auto &x:adj[node])

{

if(!vis[x])

{

if(dfs(x, adj, vis, dfsvis))

return true;

}

else if(dfsvis[x])

return true;

}

dfsvis[node]=0;

return false;

}

int main()

{

ll n,e;

cin >> n >> e;

vvl adj(n+1);

vl vis(n+1,0);

vl dfsvis(n+1,0);

for(ll i=0; i<e; i++)

{

ll x,y;

cin >> x >> y;

adj[x].push\_back(y);

}

for(ll i=1; i<=n; i++)

{

if(!vis[i])

{

if(dfs(i,adj,vis,dfsvis))

{

cout << "Has cycle\n";

return 0;

}

}

}

cout << "No cycle\n";

for(ll i=1; i<=n; i++)

{

cout << vis[i] << " " << dfsvis[i] << endl;

}

}

**\*\*COMPLEXITY\*\***

• The C++11 standard requires that the sort function works in O(n log n) time; the exact implementation depends on the compiler.

• SWAP 🡪 O(1) for 2 elements

O(n) for array of n elements

**\*\*PRIORITY QUEUE\*\***

Function Time

Q.top() O(1)

Q.push() O(log n)

Q.pop() O(log n)

Q.empty() O(1)

**\*\*MAP/SET\*\***

Function Time

M.find(x) O(log n)

M.insert(pair<int, int> (x, y) O(log n)

M.erase(x) O(log n)

M.empty( ) O(1)

M.size( ) O(1)

**\*\*STACK/ QUEUE\*\***

All operations O(1)

**\*\*VECTOR\*\***

Function Time

sort(v.begin( ), v.end( )) Theta(nlog(n))

reverse(v.begin( ), v.end( )) O(n)

v.push\_back(x) O(1)

v.pop\_back(x) O(1)

v.size() O(1)

v.clear() O(n)

v.erase() O(n)

**\*\*BIT OPERATIONS\*\***

• x<<k corresponds to multiplying x by 2k, and x>>k correspondsto dividing x by 2k rounded down to an integer.

• The kth bit of a number is one exactly when x & (1 k) is not zero.

• The formula x | (1<<k) sets the kth bit of x to one, the formula x &~(1<<k) sets the kth bit of x to zero, and the formula x ^ (1<<k) inverts thekth bit of x.

• The formula x & (x-1) sets the last one bit of x to zero, and the formula x & -x sets all the one bits to zero, except for the last one bit. The formula x | (x-1) inverts all the bits after the last one bit.

• A positive number x is a power of two exactly when x & (x-1)=0.

The g++ compiler provides the following functions for counting bits:

• \_\_builtin\_clz(x): the number of zeros at the beginning of the number

• \_\_builtin\_ctz(x): the number of zeros at the end of the number

• \_\_builtin\_popcount(x): the number of ones in the number

• \_\_builtin\_parity(x): the parity (even or odd) of the number of ones

**\*\*Bit Manipulation\*\***

1. To multiply by 2^x : S = S<<x

2. To divide by 2^x : S = S>>x

3. To set jth bit : S|=(1<<j)

4. To check jth bit : T = S &(1<<j) (If T=0 not set else set)

5. To turn off jth bit : S&=~(1<<j)

6. To flip jth bit : S^=(1<<j)

**\*\*Number Theory\*\***

1. To calculate sum of factors of a number, we can find the number of prime factors and their exponents. N = ae1 \* be2 \* ce3 …

Then sum = (1 + a + a^2….)(1 + b + b^2 .. )...

Number of factors=(a+1)\*(b+1)...

2.Every even integer greater than 2 can be expressed as the sum of 2 primes.

3. For rootn prime method, check for 2, 3 then:

for (i=5; i\*i<=n; i=i+6) n%i and n%(i+2)

4. Number of divisors will be prime only if N=p^x where p is prime.

5. Kth prime factor= store smallest factor in seive and repeatedly divide with it to get the answer.

6. fib(n+m)=fib(n)fib(m+1)+fib(n-1)fib(m)

7. A number is Fibonacci if and only if one or both of (5\*n2 + 4) or (5\*n2 – 4) is a perfect square

8. every positive Every positive integer can be written uniquely as a sum of distinct non-neighbouring Fibonacci numbers.

9. Matrix multiplication

mul[i][j] += a[i][k]\*b[k][j];

10. Root n under mod p exists only if

n^((p-1)/2) % p = 1

11.divisibility by 4: last 2 digits divisible by 4

12.divisibility by 8: last 3 digits divisible by 8

13. Divisibility by 3,9: sum of digs divisible by 3,9

14. Divisibility by 11: alternate (+ve,-ve) digit sum is divisible by 11

15. Divisibility by 12: divisible by 3 and 4

16. Divisibility by 13: alternating sum in blocks of 3 (L to R) div 13

17. Integral solution of ax+by=c exists if gcd(a,b) divides c

**\*\*Modular arithmetic\*\***

(a+b)%n = (a%n +b%n)%n;

(a\*b)%n = (a%n \* b%n)%n;

(a-b)%n = (a%n - b%n)%n;

(a^b)%n = ((a%n)^b)%n;

**\*\*Combinatorics\*\***

**\*\*All Subset\*\***

ll n; cin>>n; vl ar(n);

for(int i = 0;i<n;i++) { cin>>ar[i]; }

int szz = pow(2,n);

for(int mask = 0;mask<szz;mask++)

{

vl tmp;

for(int i = 0;i<n;i++){

if(mask&(1<<i))

{

tmp.pb(ar[i]);

}

}

for(auto x: tmp) cout<<x<<" ";

cout<<endl;

**\*\*Subset Sum\*\***

ll n,target; cin>>n>>target;

vl ar(n);

for(int i = 0;i<n;i++) { cin>>ar[i]; }

ll cnt =0;

int szz = pow(2,n);

for(int mask = 0;mask<szz;mask++){

vl tmp;

for(int i = 0;i<n;i++){

if(mask&(1<<i)){

tmp.pb(ar[i]);

}

}

ll sum =0;

for(auto x: tmp) sum+=x;

if(sum==target)

{

for(auto x : tmp) cout<<x<<" ";

cout<<endl;

cnt++;

}

}

cout<<cnt<<endl;

**\*\*Probability\*\***

P(A∩B) = P(A) + P(B) - P(A∪B)

Probability of A if B has happened:

P(A|B) = P(A∩B) / P(B)

expected value is the sum of: [(each of the possible outcomes) × (the probability of the outcome occurring)].

Var(X) = E(X^2) – m^2

**\*\*Game Theory\*\***

=> nim-sum = 2 player will play optimally. Player will do operations in every move. Whoever can’t make a operation, will loss.

1. If nim-sum is non-zero, player starting first wins.

2. Mex: smallest non-negative number not present in a set.

3. Grundy=0 means game lost.

4. Grundy=mex of all possible next states.

5. Sprague-Grundy theorem:

If a game consists of sub games (nim with multiple piles)

Calculate grundy number of each sub game (each pile)

Take xor of all grundy numbers:

If non-zero, player starting first wins.

**\*\*Techniques\*\***

1. For counting problems, try counting number of incorrect ways instead of correct ways.

2. Try solving the problem backwards

3. Analyse complexity carefully

4. Reduce the problem to some standard problem

5. Add m when doing modular arithmetic.

**\*\*Pre-Submit\*\***

1. Write a few sample testcases.
2. If sample is not enough. Are time limits close? If so, generate max cases.
3. Is the Memory usage fine?
4. Could anything Overflow?
5. Make sure to submit the right file.

**\*\*Wrong Answer\*\***

1. Print your solution. Print Debug output, as well.
2. Are you clearing all data structures between test cases?
3. Can your algorithm handle the whole range of input?
4. Read the full program statement again.
5. Do you handle all corner cases correctly?
6. Any uninitialized variables?
7. Any overflows? Comparison of char and int?
8. Confusing N and M, I and J, etc.?
9. Are you sure your algorithm works?
10. What special cases have you not thought of?
11. Are you sure the STL functions you use work as you think?
12. Add some Assertions, maybe resubmit.
13. Create some test cases to run your algorithm on.
14. Go through the algorithm for a simple case.
15. Explain your algorithm to a teammate.
16. Ask the teammate to look at your code.
17. Go for a small walk, e.g., go to the toilet.
18. Is your Output format is correct? (Including Whitespace)
19. Rewrite your solution from the start or let a teammate do it.

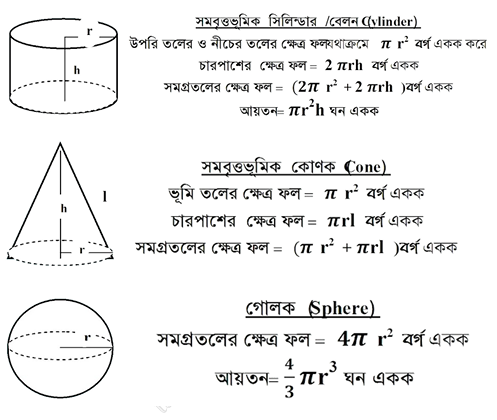
**\*\*Memory Limit Exceeded\*\***

1. What is the max amount of memory your algorithm should need?
2. Are you clearing all data structures between test cases?

**\*\*Runtime Error\*\***

1. Have you tested all corner cases locally?
2. Any uninitialized variables?
3. Are you reading or writing outside the range of any vector?
4. Any assertions that might fall?
5. Any possible division by 0? (mod 0 for example)
6. Any possible Infinite Recursion?
7. Invalidated pointers or iterators?
8. Are you using too much memory?

**\*\*Time Limit Exceeded\*\***

1. Do you have any possible infinite loops?
2. What is the complexity of your algorithm?
3. Are you copying a lot of unnecessary data?
4. How big is the input & output? (consider scanf)
5. Avoid vector, map (use arrays/ unordered\_maps)
6. What do your teammates think about your algorithm?

**\*\*Geometry\*\***

1.Area of a regular polygon(equal sides):

area= ((s\*s)\*n) / 4tan(180/n)

2. Angle between (m1, b1) and (m2, b2):

arctan ((m2 − m1) / (m1 · m2 + 1))

3. Triangle: Area = a · b · sin γ / 2

• Area = | x1 · y2 + x2 · y3 + x3 · y1 − y1 · x2 − y2 · x3 − y3 · x1 | / 2

• Heron’s formula:

Let s = (a + b + c) / 2; then Area = s⋅(s − a)⋅(s − b)⋅(s − c)

4. Circle: (x − xc)^2+ (y − yc)^2= r^2

5.Polygon area (vertex cordinates):

| x1 · y2 + x2 · y3 + ... + xn · y1 − y1 · x2 − y2 · x3 − ... − yn · x1 | / 2

