Template: SMUCT_BitClowns

Binary Search

```
int binarySearch(vector<int> &arr)
{
    int n = arr.size();
    int start = 0, end = n - 1;
    int target = 2;

    while (start <= end)
    {
        int mid = start + (end - start) /
2;
        if (arr[mid] == target)
            return mid;
        else if ((arr[mid] > target &&
arr[start] > arr[end]) || (arr[mid] <
target && arr[start] < arr[end]))
            start = mid + 1;
        else
            end = mid - 1;
    }

    return -1;
}
```

Binary search double

```
    while (r - l > 1e-7)
    {
        double mid = l + (r - l) / 2;

        if (ok(mid))
        {
            ans = mid;
            l = mid;
        }
        else
        {
            r = mid;
        }
    }
```

Ternary Search

```
int turnerySearch(int arr[], int l, int r,
int key)
{
    while(l <= r)
    {
        int m1 = l + (r - l) / 3;
        int m2 = r - (r - l) / 3;

        if(arr[m1] == key)
            return m1;
        if(arr[m2] == key)
            return m2;
        if (key < arr[m1])
            r = m1 - 1;
        else if (key > arr[m2])
            l = m2 + 1;
        else
        {
            l = m1 + 1;
            r = m2 - 1;
        }
    }
    return -1;
}
```

Sliding Window Template

```
int l = 0, r = 0, n = nums.size(), answer
= 0;

for (; r < n; ++r)
{
    // operations
    for (; windowInvalid(); ++l)
    {
        // codes
    }
}
return answer;
```

Prime Factorization

```
void prime_factorization(int n)
{
    vector<int> prime_factors;
    while (n % 2 == 0)
        prime_factors.push_back(2), n /=
2;

    for (int i = 3; i * i <= n; i += 2)
        while (n % i == 0)
            prime_factors.push_back(i), n
/= i;

    if (n > 2)
        prime_factors.push_back(n);

}
```

Extended Euclid

```
int gcd(int a, int b, int &x, int &y)
{
    if (b == 0)
    {
        x = 1, y = 0;
        return a;
    }
    int x1, y1;
    int d = gcd(b, a % b, x1, y1);
    x = y1, y = x1 - y1 * (a / b);
    return d;
}
```

nCr

```
#define nCr(n, r) ((r == 0 || r == n) ? 1
: nCr(n - 1, r - 1) + nCr(n - 1, r))
```

```
int n, r;
void ans()
    long double sum = 1;
    for (int i = 1; i <= r; ++i)
    {
        sum = sum * (n - r + i) / i;
    }
    cout << sum << endl;
```

Euler's Totient Function

```
int phi(int n)
{
    int result = 1;
    for (int i = 2; i < n; i++)
        if (__gcd(i, n) == 1)
            result++;
    return result;
}

int main()
{
    int n;
    for (n = 1; n <= 10; n++)
        cout << "phi(" << n << ") = " <<
phi(n) << endl;
    return 0;
}
```

Formula implementation

C++ Mathematical Constants

pi	M_PI	3.14159265358979323846
pi/2	M_PI_2	1.57079632679489661923
pi/4	M_PI_4	0.78539816339744830961
1/pi	M_1_PI	0.31830988618379067153
2/pi	M_2_PI	0.63661977236758134307
2/sqrt(pi)	M_2_SQRTPI	1.12837916709551257390
sqrt(2)	M_SQRT2	1.41421356237309504880
1/sqrt(2)	M_SQRT1_2	0.70710678118
e	M_E	2.71828182845904523536

log_2(e)	M_LOG2E	1.4426950408 8896340736
log_10(e)	M_LOG10E	0.4342944819 03251827651
log_e(2)	M_LN2	0.6931471805 59945309417
log_e(10)	M_LN10	2.3025850929 940456840

```
int PI = M_PI;
cos(60.0 * PI / 180.0);
sin(60.0 * PI / 180.0);
tan(45.0 * PI / 180.0);
acos(0.5) * 180.0 / PI;
asin(0.5) * 180.0 / PI;
atan(1.0) * 180.0 / PI;
pow(2, 3);
sqrt(49);
ceil(3.8);
floor(2.3);
fmod(5.3, 2);
trunc(2.3);
round(4.6);
remainder(18.5, 4.2);
fmax(100.0, 1.0);
fmin(100.0, 1.0);
fdim(2.0, 1.0);
fabs(3.1416);
abs(3.1416);
log(5);
exp(5.0);
log10(5);
```

Cirlce Area

```
#define circleArea(r) M_PI *r *r
```

Perimeter of a Triangle

```
cin >> a >> b >> c;
double perimeter = a + b + c;
```

Volume of a Cylinder

```
double volumeCylinder = M_PI * radius *
radius * height;
```

Pythagorean theorem

```
c = sqrt(a * a + b * b);
```

Rectangle area

```
#define rectangleArea(l, w) l *w
```

Area of a Triangle (Heron's Formula):

```
#define triangleArea(a, b, c) sqrt(s * (s
- a) * (s - b) * (s - c))
```

Factorial Function:

```
#define factorial(n) ((n)?(n)*
factorial((n) - 1) : 1)
```

Least Common Multiple (LCM):

```
#define lcm(a,b) ((a)/__gcd((a),(b)) *
(b))
```

Modular operations

```
ll inv(ll i) {if (i == 1) return 1; return
(mod - ((mod / i) * inv(mod % i)) % mod) %
mod;}
ll mod_mul(ll a, ll b) {a = a % mod; b = b
% mod; return (((a * b) % mod) + mod) %
mod;}

ll mod_add(ll a, ll b) {a = a % mod; b = b
% mod; return (((a + b) % mod) + mod) %
mod;}

ll gcd(ll a, ll b) { if (b == 0) return a;
return gcd(b, a % b);}

ll ceil_div(ll a, ll b) {return a % b == 0
? a / b : a / b + 1;}

ll pwr(ll a, ll b) {a %= mod; ll res = 1;
while (b > 0) {if (b & 1) res = res * a %
mod; a = a * a % mod; b >>= 1;} return
res;}
```

Modulo

```
const int mod = 1e9 + 7;
(a + b) % n = ((a % n) + (b % n)) % n
(a - b) % n = ((a % n) - (b % n)) % n
(a x b) % n = ((a % n) x (b % n)) % n
```

Modular Exponentiation:

```
#define mod_pow(base, exp, mod)
power((base), (exp)) % (mod)
//Function modInv() the modular inverse of
i mod 10^9+7
/// Here is the single line compressed
function code
long long modInv(long long i) { return (i
<= 1) ? i : MOD - (MOD / i) * modInv(MOD %
i) % MOD; }
```

Modular Exponentiation

```
const int MOD = 1e9 + 7;
int modularFastExpo(int x, int n)
{
    int ans = 1;
    x = x % MOD;
    if (x == 0)
        return 0;

    while (n > 0)
    {
        if (n & 1)
        {
            ans = (ans % MOD * x % MOD) %
MOD;
            --n;
        }
        x = (x * x) % MOD;
        // n /= 2;
        n >>= 1;
    }
    return ans;
}
```

Tips

```
  #define ull unsigned long long
  ull uint_pow(ull base, ull exp)
  {
      ull result = 1;
      while (exp)
      {
          if (exp % 2)
             result *= base;
          exp /= 2;
          base *= base;
      }

      return result;
  }
```

- To accurately calculate the logarithm value for an integer number use the following code.

```
int n = 32;
int power = __lg(n);
```

To accurately calculate the square root value for an integer number use the following code.

```
void safer_SQRT_int_2()
{
    long double res = exp(log(n) / 2);
    int floorRes = floor(res);

    if (res * res == n)
        cout << res << endl;
    else
        cout << floorRes << endl;
}
```

- **Avoid using float for fractional numbers.**

Comparing double values

Filling or initialization with data values

Filling 1D arrays and containers

memset

```
char chArr[n];
memset(chArr, 'a', sizeof(chArr)); //
fills with character 'a'

bool boolArr[n];
memset(boolArr, true, sizeof(boolArr)); //
fills with 1 or true
```

It is much useful to use with 2D arrays.

```
char arr[n][n];
memset(arr, 'R', sizeof(arr));

bool bool2D[n][n];
memset(bool2D, false, sizeof(bool2D)); //
fills with false or 0
```

⚠️ memset doesn't work fine with other than character and boolean arrays. So, one should not use memset() for integer or float arrays. For more information, you can still use memset fill to integer array but this is limited to values 0 and -1 only.

fill

Unlike memset(), the fill() function can be used with any type of array.

```
int arr[n];
fill(arr, arr + n, 11); // fills with
value 11

vector<int> arr2(n);
fill(arr2.begin(), arr2.begin() + n, 12);
// fills with value 12
```

iota

To store or initialize values in increasing order.

```
int arr[5];
iota(arr, arr + 5, 10);
vector<int> arr2(6);
iota(arr2.begin(), arr2.begin() + 6, 33);
```

Even and odd count within a given range

```
#define EVEN_ODD_COUNT(n, evenCnt, oddCnt)
\
    evenCnt = (n / 2),
\
    oddCnt = (n & 1) ? (n / 2) + 1 : (n /
2)
```

Multiset

Set with multiple same values.
By default stores ascending value:

```
multiset<int> mset;
```

Descending order storing:

```
multiset<int,
greater<int>> mset;
```

Insertion:

```
mset.insert(10);
```

Delete value:

```
mset.erase(10);
```

Iteration

1.
```
for(auto it = mset.begin(); it !=
   mset.end(); cout << *it << ' ',
   ++it);
```
2.
```
for(auto i : mset) cout << i << " ";
```

Find value/check exitance of value

1.
```
cout << (mset.find(element) !=
   mset.end() ? "Exist" : "Not Exist");
```
2.
```
cout << (mset.count(element) ?
   "Exist" : "Not Exist");
```

Empty whole multiset

```
mset.clear();
cout << (mset.empty() ? "Multiset is
Empty" : "Elements are available");
```

Copy array elements into multiset

```
multiset<int, greater<int>>
mset2(arr.begin(), arr.end());
```

Deque (Double-ended queue)

```
deque<int> dq{5, 10, 15};
dq.push_front(3);
dq.push_back(5);
cout << dq.front() << endl;
cout << dq.back() << endl;
dq.pop_front();
dq.pop_back();
for (auto i : dq) cout << i << " ";
```

String matching - Rabin Karp

```
void rabinKarpSearch(string S, string P)
{
    int Ns = S.length();
    int Np = P.length();

    int prime = 31;
    int mod = 1e9 + 9;

    vector<long long> p_pow(Ns);
    p_pow[0] = 1;
    for (int i = 1; i < Ns; i++)
        p_pow[i] = (p_pow[i - 1] * prime)
% mod;

    vector<long long> h(Ns + 1, 0);
    for (int i = 0; i < Ns; i++)
        h[i + 1] = (h[i] + (S[i] - 'a' +
1) * p_pow[i]) % mod;

    long long hash_P = 0;
    for (int i = 0; i < Np; i++)
    {
        hash_P = (hash_P + (P[i] - 'a' +
1) * p_pow[i]) % mod;
    }
    vector<int> occurredPos;
    for (int i = 0; i + Np - 1 < Ns; i++)
    {
        long long curr_hash = (h[i + Np] +
mod - h[i]) % mod;
        if (curr_hash == hash_P * p_pow[i]
% mod) occurredPos.push_back(i);
    }
    for(auto i : occurredPos) cout << i <<
' ';
}
```

Prefix, Suffix and Range Sum

```
void prefix_sum()
{
    int prefix_arr[n + 1]{0};
    prefix_arr[0] = 0;
    for (int i = 0; i < n; ++i)
    prefix_arr[i + 1] = prefix_arr[i] +
arr[i];
}

void suffix_sum()
{
    int suffix_arr[n + 1]{0};
    suffix_arr[0] = 0;
    for (int i = n - 1, j = 1; i >= 0;
--i, ++j)
        suffix_arr[j] = suffix_arr[j - 1]
+ arr[i];
}
/// pre-compute prefix sum
int rangeSum(int i, int j, int pre[])
{
    if (i == 0)
        return pre[j];

    return pre[j] - pre[i - 1];
}
```

Prefix sum library

```
vector<int> a(n, 0);
for (auto &i : a) cin >> i;
vector<int> prefix1(n + 1, 0), prefix2(n +
1, 0);
partial_sum(a.begin(), a.end(),
prefix1.begin() + 1);
```

Suffix sum library

```
vector<int> nums = {1, 2, 3, 4, 5};
vector<int> suffixSum(nums.size());
partial_sum(nums.rbegin(), nums.rend(),
suffixSum.rbegin());
```

String subsequence check

```
bool is_subsequence(string s, string p)
{
    int n = s.length(), m = p.length(), i
= 0, j = 0;
    for (; i < n and j < m; ++i) if (s[i]
== p[j]) ++j;
    return j >= m;
}
```

Graph Template

```
/// Global Variables for graph
const int N = 1e5 + 5;
const long long int INF = 1e18;
/// Adjacency list representation -
unweighted
vector<int> adjList[N];
/// Adjacency list representation -
weighted
vector<pair<int, int>> adjListWeighted[N];
/// Adjacency matrix representation
vector<vector<int>> adjMat(n,
vector<int>(n, 0));
bool visited[N];
int level[N];
long long int d[N];
vector<pair<int, int>> dijkstraAdjList[N];
/// for dijkstra
int n, m;

int main()
{
    cin >> n >> m;
    bool undirected = false;
    /// Adjacency matrix
    /// ------------------------------
    /// with weight and direction
    while (m--)
    {
        int u, v, wt;
        cin >> u >> v >> wt;
        adjMat[u - 1][v - 1] = wt;
        if (undirected)
            adjMat[v - 1][u - 1] = wt;
    }

    /// unweighted
    while (m--)
    {
        int a, b;
        cin >> a >> b;
        adj[a][b] = 1;
        if (undirected) adj[b][a] = 1;
    }

    /// display adjacency matrix
    for (int i = 0; i < n; ++i)
    {
        for (int j = 0; j < n; ++j)
            cout << adjMat[i][j] << ' ';
        cout << endl;
    }

    /// Adjacency list
    /// ------------------------------
    /// with weight and direction
    while (m--)
    {
        int u, v, wt;
        cin >> u >> v >> wt;
        adjListWeighted[u -
1].push_back({v, wt});
        if (undirected) adjListWeighted[v
- 1].push_back({u, wt});
    }
    /// unweighted
    while (m--)
    {
        int a, b;
        cin >> a >> b;
        adjList[a].push_back(b);
        if(undirected)
adjList[b].push_back(a);
    }
    /// display adjacency list graph
    for (int i = 0; i < n; ++i)
    {
```

```
        // cout << "List " << i + 1 << ":
";
        for (auto j : adjList[i])
        {
            /// if weighted
            // cout << j.first << " " <<
j.second << endl;
            /// if unweighted
            cout << j << " ";
        }
        cout << endl;
    }

    return 0;
}
```

Graph with directions

```
const int N = 1e3 + 9;
bool vis[N][N];
char s[N][N];

// direction array for go all the way to 8
points
int di[] = {0, -1, 0, +1, -1, -1, +1, +1};
int dj[] = {+1, 0, -1, 0, +1, -1, -1, +1};

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

void dfs(int i, int j)
{
    vis[i][j] = true;
    if (s[i][j] == '@')
    {
        for (int k = 0; k < 8; ++k)
        {
            int nxt_i = i + di[k];
            int nxt_j = j + dj[k];
            if (!vis[nxt_i][nxt_j] and
is_valid(nxt_i, nxt_j))
                dfs(nxt_i, nxt_j);
        }
    }
}
```

DFS

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

const int N = 1e5 + 9;
vector<int> g[N];
bool vis[N];

void dfs(int u) {
  vis[u] = true;
  for (auto v: g[u]) {
    if (!vis[v]) {
      dfs(v);
    }
  }
}
int32_t main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);
  int n, m; cin >> n >> m;
  while (m--) {
    int u, v; cin >> u >> v;
    g[u].push_back(v);
    g[v].push_back(u);
  }
  // dfs(1);
  // for (int i = 1; i <= n; i++) {
  //   if (!vis[i]) {
  //     cout << "Disconnected Graph\n";
  //     return 0;
  //   }
  // }
  // cout << "Connected Graph\n";
  int ans = 0;
  for (int i = 1; i <= n; i++) {
    if (!vis[i]) {
      dfs(i);
      ++ans;
```

```
    }
  }
  cout << "Connected Components = " << ans
<< '\n';
  return 0;
}
```

BFS

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

const int N = 1e5 + 9;
vector<int> g[N];
bool vis[N]; int dis[N], par[N];

int32_t main() {
 ios_base::sync_with_stdio(0);
 cin.tie(0);
 int n, m; cin >> n >> m;
 while (m--) {
  int u, v; cin >> u >> v;
  g[u].push_back(v);
  g[v].push_back(u);
 }
 queue<int> q;
 q.push(1); vis[1] = true; dis[1] = 0;
 while (!q.empty()) {
  int u = q.front();
  q.pop();
  for (auto v: g[u]) {
   if (!vis[v]) {
    q.push(v);
    par[v] = u;
    dis[v] = dis[u] + 1;
    vis[v] = true;
   }
  }
 }
 for (int i = 1; i <= n; i++) {
  cout << dis[i] << ' ';
 }
 cout << '\n';
 int v = 4;
 while (v != 1) {
  cout << v << ' ';
  v = par[v];
 }
 cout << 1 << '\n';
}
```

Cycle Finding

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

const int N = 1e5 + 9;
vector<int> g[N];
int col[N], par[N];
bool cycle;
void dfs(int u) {
  col[u] = 1;
  for (auto v: g[u]) {
    if (col[v] == 0) {
      par[v] = u;
      dfs(v);
    }
    else if (col[v] == 1) {
      cycle = true;
      // you can track the cycle using par
array
    }
  }
  col[u] = 2;
}
int32_t main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);
  int n, m; cin >> n >> m;
  for (int i = 1; i <= m; i++) {
    int u, v; cin >> u >> v;
    g[u].push_back(v);
  }
  cycle = false;
  for (int i = 1; i <= n; i++) {
    if (col[i] == 0) dfs(i);
  }
  cout << (cycle ? "YES\n" : "NO\n") <<
'\n';
  return 0;
}
```

Topological Sort Using DFS O(V+E)

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

const int N = 1e5 + 9;
```

```
int indeg[N];
vector<int> g[N];
bool vis[N];
int32_t main() {
 ios_base::sync_with_stdio(0);
 cin.tie(0);
 int n, m; cin >> n >> m;
 while (m--) {
  int u, v; cin >> u >> v;
  indeg[v]++;
  g[u].push_back(v);
 }
 vector<int> z;
 for (int i = 1; i <= n; i++) {
  if (indeg[i] == 0) {
   z.push_back(i);
   vis[i] = true;
  }
 }
 vector<int> ans;
 while (ans.size() < n) {
  if (z.empty()) {
   cout << "IMPOSSIBLE\n";
   return 0;
  }
  int cur = z.back();
  z.pop_back();
  ans.push_back(cur);
  for (auto v: g[cur]) {
   indeg[v]--;
   if (!vis[v] and indeg[v] == 0) {
    z.push_back(v);
    vis[v] = true;
   }
  }
 }
 for (auto x: ans) cout << x << ' ';
 return 0;
}
```

Dijkstra

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

#define INF 0x3f3f3f3f

typedef pair<int, int> iPair;

class Graph {
    int V;
    list<iPair> *adj;

public:
    Graph(int V){
      this->V = V;
      adj = new list<iPair>[V];
      }
    void addEdge(int u, int v, int w) {
      adj[u].push_back(make_pair(v, w));
      adj[v].push_back(make_pair(u, w));
      }
    void shortestPath(int src) {
      priority_queue<iPair, vector<iPair>,
greater<iPair>> pq;

      vector<int> dist(V, INF);

      pq.push(make_pair(0, src));
      dist[src] = 0;

      while (!pq.empty()) {
            int u = pq.top().second;
            pq.pop();
            for (auto &neighbor : adj[u]) {
                  int v = neighbor.first;
                  int weight =
neighbor.second;

                  if (dist[v] > dist[u] +
weight) {
                  dist[v] = dist[u] +
weight;

pq.push(make_pair(dist[v], v));
                  }
            }
      }
      cout << "Vertex Distance from Source"
<< endl;
      for (int i = 0; i < V; ++i)
            cout << i << "   " << dist[i] <<
endl;
      }
```

```
};

void Graph::
int main() {
    int V = 9;
    Graph g(V);

    g.addEdge(0, 1, 4);
    g.addEdge(0, 7, 8);
    g.addEdge(1, 2, 8);
    g.addEdge(1, 7, 11);

    g.shortestPath(0);

    return 0;
}
```

Floyd warshall

```
const int INF = 1e7;
int main()
{
    int n, e;
    cin >> n >> e;
    int dis[n + 1][n + 1];
    for (int i = 1; i <= n; i++)
    {
        for (int j = 1; j <= n; j++)
        {
            dis[i][j] = INF;
            if (i == j)
                dis[i][j] = 0;
        }
    }
    while (e--)
    {
        int a, b, w;
        cin >> a >> b >> w;
        dis[a][b] = w;
    }
    for (int i = 1; i <= n; i++)
    {
        for (int j = 1; j <= n; j++)
        {
            if (dis[i][j] == INF)
                cout << "INF"
                     << " ";
            else
                cout << dis[i][j] << " ";
        }
        cout << endl;
    }
    for (int k = 1; k <= n; k++)
    {
        for (int i = 1; i <= n; i++)
        {
            for (int j = 1; j <= n; j++)
            {
                if (dis[i][k] + dis[k][j]
< dis[i][j])
                {
                    dis[i][j] = dis[i][k]
+ dis[k][j];
                }
            }
        }
    }
    cout << "Updated" << endl;
    for (int i = 1; i <= n; i++)
    {
        for (int j = 1; j <= n; j++)
        {
            if (dis[i][j] == INF)
                cout << "INF"
                     << " ";
            else
                cout << dis[i][j] << " ";
        }
        cout << endl;
    }
    return 0;
}
```

DSU

```
int parent[1000];
int parentSize[1000];
int parentLevel[1000];

void dsu_set(int n)
{
    for (int i = 1; i <= n; i++)
    {
        parent[i] = -1;
        parentSize[i] = 1;
        parentLevel[i] = 0;
    }
}

/// Time Complexity: O(n)
int dsu_find(int node)
{
    while (parent[node] != -1)
    {
        node = parent[node];
    }

    return node;
}
void dsu_union_normal(int u, int v)
{
    int leaderA = dsu_find(u);
    int leaderB = dsu_find(v);

    if (leaderA != leaderB)
    {
        // parent[leaderB] = leaderA;
        /// or vice versa
        parent[leaderA] = leaderB;
    }
}

/*
 Union by size and union by rank or level
 Time complexity: O(α(n))
*/
void dsu_union_by_size(int u, int v)
{
    int leaderA = dsu_find(u);
    int leaderB = dsu_find(v);

    if (leaderA != leaderB)
    {
        if (parentSize[leaderA] >
parentSize[leaderB])
        {
            parent[leaderB] = leaderA;
            parentSize[leaderA] +=
parentSize[leaderB];
        }
        else
        {
            parent[leaderA] = leaderB;
            parentSize[leaderB] +=
parentSize[leaderA];
        }
    }
}

void dsu_union_by_rank(int u, int v)
{
    int leaderA = dsu_find(u);
    int leaderB = dsu_find(v);

    if (leaderA != leaderB)
    {
        if (parentLevel[leaderA] >
parentLevel[leaderB])
        {
            parent[leaderB] = leaderA;
        }
        else
        {
            parent[leaderA] = leaderB;
            if (parentLevel[leaderA] ==
parentLevel[leaderB])
                ++parentLevel[leaderB];
        }
    }
}
```

```
int main()
{
    int n, e;
    cin >> n >> e;
    dsu_set(n);

    while (e--)
    {
        int u, v;
        cin >> u >> v;
        dsu_union_normal(u, v);
        // dsu_union_by_size(u, v);
        // dsu_union_by_rank(u, v);
    }

    cout << dsu_find(1);

    return 0;
}
```

0-1 knapsack

```
int dp[1005][1005];
int knapsack(int n, int s, int v[], int
w[])
{
    // base case
    if (n == 0 || s == 0)
        return 0;
    if (dp[n][s] != -1)
    {
        return dp[n][s];
    }
    if (w[n - 1] <= s)
    {
        int op1 = knapsack(n - 1, s - w[n
- 1], v, w) + v[n - 1];
        int op2 = knapsack(n - 1, s, v,
w);
        return dp[n][s] = max(op1, op2);
    }
    else
    {
        return dp[n][s] = knapsack(n - 1,
s, v, w);
    }
}
int main()
{
    int n;
    cin >> n;
    int v[n], w[n];
    for (int i = 0; i < n; i++)
    {
        cin >> v[i];
    }
    for (int i = 0; i < n; i++)
    {
        cin >> w[i];
    }
    int s;
    cin >> s;
    for (int i = 0; i <= n; i++)
    {
        for (int j = 0; j <= s; j++)
        {
            dp[i][j] = -1;
        }
    }
    cout << knapsack(n, s, v, w);
    return 0;
}
```

LCS

```
int dp[1005][1005];
int lcs(string a, int n, string b, int m)
{
    if (n == 0 || m == 0)
        return 0;
    if (dp[n][m] != -1)
        return dp[n][m];
    if (a[n - 1] == b[m - 1])
    {
        int ans = lcs(a, n - 1, b, m - 1);
        return dp[n][m] = ans + 1;
    }
    else
```

```
    {
        int ans1 = lcs(a, n - 1, b, m);
        int ans2 = lcs(a, n, b, m - 1);
        return dp[n][m] = max(ans1, ans2);
    }
}
int main()
{
    string a, b;
    cin >> a >> b;
    // memset(dp, -1, sizeof(dp));
    for (int i = 0; i <= a.size(); i++)
    {
        for (int j = 0; j <= b.size();
j++)
        {
            dp[i][j] = -1;
        }
    }
    cout << lcs(a, a.size(), b, b.size());
```

C++ Factorial Class

```
#define factorial(n) ({long long result =
1; for (int i = 2; i <= (n); ++i) result
*= i; result; })
```

Python Factorial Class

```
class Factorial:
    def calculate(self, n):
        if n == 0 or n == 1:
            return 1
        else:
            return n * self.calculate(n-1)
```

Floor, ceil, round

- Floor of two integer number is automatically determined by division of two integer numbers.

```
int a = 14, b = 5;
cout << (a / b) <<endl; // 2
```

- Ceil (safer) of two integer numbers can be evaluated by following trick:

```
cout << (a + b - 1) / b << endl; // 3
```

- **Sum of first N odd positive integers**

```
int n;
cin >> n;
long long oddSum = n * 1LL * n;
cout << oddSum << endl;
```

- **Sum of first N even positive integers**

```
long long evenSum = n * 1LL * (n + 1);
cout << evenSum << endl;
```

- **Sum of first N squares**

```
long long squareSum = n / 6 * (n + 1) * (2
* n + 1);
cout << squareSum << endl;
```

- **Sum of first N natural numbers which are divisible by X or Y**

```
int Sx, Sy, Sxy, sum;
Sx = ((n / x)) * (2 * x + (n / x - 1) *
x) / 2;
Sy = ((n / y)) * (2 * y + (n / y - 1) *
y) / 2;
Sxy= ((n / (x * y))) * (2 * (x * y) +
(n / (x * y) - 1) * (x * y))/ 2;
sum = Sx + Sy - Sxy;
```

- **Find the formula for the sum of the positive integers from 1 to n that are not multiple of K.**

```
int sum_of_k_multiple = 1LL* (n / k) * (2
* k + (n / k - 1) * k) / 2;
int sum_till_n = 1LL * n / 2 * (n + 1);
```

```
cout << sum_till_n - sum_of_k_multiple <<
endl;
```

- **Arithmetic Progression**

```
Given terms a, n, d

int nthTerm(int a, int n, int d) {
    return a + (n - 1) * d;
}

int sumOfNTerms(int a, int n, int d) {
    return (n / 2) * (2 * a + (n - 1) *
d);
}
```

- **Geometric Progression**

```
Given terms a, r, n

int nthTerm(int a, int r, int n) {
    return a * pow(r, n - 1);
}
int sumOfTerms(int a, int r, int n) {
    return (a * (pow(r, n) - 1)) / (r -
1);
}
```

- **Logarithm**

```
int n, targetBase;
cin >> n >> targetBase;
int a = log2(n);        // base 2
int b = log10(n);       // base 10
int c = log(n); // base 'e' or natural
base
int d = log2(n) / log2(targetBase); //
base 'target'
```

- **Find number of digits using logarithm**

```
#define number_cnt(n) floor(log10(n)) + 1
```

- **Find number of digits in N! number using logarithm**

```
    int n;
    cin >> n;
    double digitCnt = 0;
    for (int i = 1; i <= n; ++i)
           digitCnt += log10(i);
    cout << digitCnt + 1 << endl;
```

num % n = [0 to n - 1]

- (a - b) % mod == 0 means value a and b reside in the same congruent series.

- **If result produces any negative value**

Suppose, 'x' is a negative number. Here is the formula that is used for handle such negative number with modular arithmetic formula

```
x % p = (x % p + p) % p;
```

Modular operation safety check: There may be some issues regarding MOD value overflow or underflow in your code. So, it's always better to safety check using the following method:

```
#define MOD 1000000007
#define isModResultSafe(res) ((res) >= 0
&& (res) <= MOD)
// #define negMod(x) ((x) < 0 ||
!isModResultSafe(x) ? ((x) + MOD) % MOD :
(x))

int negMod(int x)
{
    if (x < 0 or !isModResultSafe(x))
        x = (x + MOD) % MOD;

    return x;
}

int main()
{
```

```
    int a = 1e9, b = 1e9 + 1, m = 27;
    int x = ((a % m) - (b % m)) % m;

    cout << x << endl; /// -1

    x = negMod(x);
    cout << x << endl; /// 26

    return 0;
}
```

In short, one should apply safety masurements as well. Hereby the following code snippet can be used for both positive and negative numbers:

```
int arithmeticMOD(int x = 0, int y = 0)
{
    x = (x % MOD + MOD) % MOD;
    y = (y % MOD + MOD) % MOD;

    return (x + y) % MOD;
}
```

Some bit masking codes

```
bool isOddEven(int x) {
    return x & 1;
}
int clrBit(int x, int i) {
    return x & (~(1 << i));
}
int kth_bit(int x, int k) {
    return (x >> k) & 1;
}
int on_kth_bit(int x, int k) {
    return x | (1 << k);
}
int off_kth_bit(int x, int k) {
    return x & (~(1 << k));
}
int countSetBit(int n) {
    return __builtin_popcount(n);
}
bool is_power_of_two(int x) {
    return x && !(x & (x - 1));
}
int count_one(int n) {
    int count = 0;
    while (n) {
        n &= (n - 1);
        ++count;
    }
    return count;
}
```

Check if the ith bit is set in the binary form of the given number.

```
bool check(int N)
{
    if (N & (1 << i))
        return true;
    else
        return false;
}
```

How to generate all the possible subsets of a set?

```
for (int mask = 0; mask < (1 << n);
++mask)
{
    for (int i = 0; i < n; ++i)
    {
        if ((mask >> i) & 1)
        { // if ith bit is on in mask or
not
            cout << v[i] << ' ';
        }
    }
}
```

PYTHON

```
def multivariable_input():
    t = int(input())
    for i in range(0, t):
        a, b, c = map(int,
input().split())
        print(f"Testcase {i + 1}: {a *
10}, {b * 10}, {c * 10}")
# multivariable_input()
""" Multiline array input"""


def array_input():
    # array size based input
    n = int(input())
    # initialize with default value
    # arr = [0 for _ in range(n)]
    arr = [0] * n


    # array input
    for i in range(n):
        arr[i] = int(input())


    for index, value in enumerate(arr):
        print(f"Index {index}: {value}")


# array_input()
""" Single array input """
def array_input2():
    arr = list(map(int, input().split()))


    for index, value in enumerate(arr):
        print(f"Index {index}: {value}")


# array_input2()

def array_of_string_input():
    arr_str = []
    n = int(input())
    for _ in range(n):
        arr_str.append(input())

    # list comprehension to print array
elements
    [print(f"Index {index}: {value}") for
index, value in enumerate(arr_str)]
# array_of_string_input()

def matrix_input():
    rows = int(input("Row: "))
    cols = int(input("Column: "))
    matrix = []
    # initialize the matrix
    # matrix = [[0 for _ in range(cols)]
for _ in range(rows)]
    """ Taking input """
    for i in range(rows):
        row = []
        for j in range(cols):
            element = int(input())
            row.append(element)
        matrix.append(row)
    """ How to print the matrix """
    for i, row in enumerate(matrix):
        for _, ele in enumerate(row):
            print(ele, end=" ")
        print()
    # [print(row) for row in matrix]
    # [[print(ele, end=" ") for i, ele in
enumerate(row)] and print() for row in
matrix]

matrix_input()

///Comparators:
Comparator for  pairs:
bool comp(pair<int,int>p1,
pair<int,int>p2){
     //descending second value;
     return p1.second > p2.second;
     //asc
ending second value:
     return p1.second < p2.second;
```

```
//for first value we can use normal sort;
}
struct Pair //use this to store pair in decreasing order;
{
   int first;
   int second;
   bool operator<(const Pair &other) const
   {
      return first > other.first;
   }
};
```

/// PYRAMID
**Polyhedron:
Surface_area = base_area + ½ (num_base_sides * slant_height * base_length)
Volume = ⅓ (base_area * height)
Slant height = (height^2+(side/2)^2)

**Square:
Base_area = a*a
Surface_area = 2 * a * s + a * a; //s = slant_height
Height = sqrt(slant_height * slant_height - (a/2) * (a /2)
Volume = ⅓ (a*a*height)

**Triangular:
Base_area = ½ (a * b)
Surface_area = ½ (a * b) + 3/2 (b * s) //s = slant_height
Volume = ⅛ (a * b * h)

**Pentagonal:
Base_area = 5/2 (a * b)
Surface_area = 5/2 (a * b) + 5/2 (b * s) //s = slant_height
Volume = ⅝ (a * b * h)

**Hexagonal:
Base_area = 3 * a * b
Surface_area = 3 * a * b + 3 * b * s
Volume = a * b * h

String Circulic Left Shift

```
string circularLeftShift(string s, int k){ int n = s.length(); k = k % n; // Avoid redundant shifts return s.substr(k) + s.substr(0, k); // Concatenate the two parts }
```

String Circulic Right Shift

```
string circularRightShift(string s, int k) { int n = s.length(); k = k % n; // Avoid redundant shifts return s.substr(n - k) + s.substr(0, n - k); // Concatenate the two parts }
```

Segment Tree Query Sum:

```
#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;
   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;

int32_t main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);

  return 0;
}
```

for Sublime Text
Build System:

```
{
    "shell_cmd": "g++ -Wl,-stack=268435456 -std=c++17
\"${file}\" -o \"${file_path}/${file_base_name}\" &&
\"${file_path}/${file_base_name}\" <
\"C:\\Users\\<UserName>\\input.txt\" >
\"C:\\Users\\<UserName>\\output.txt\"",
    "file_regex": "^(..[^:]*):([0-9]+):?([0-9]+)?:? (.*)$",
    "working_dir": "${file_path}",
    "selector":  "source.c++",
}
```

Precompile the header file "bits/stdc++.h":
1. Open (default) path
"C:\MinGW\lib\gcc\mingw32\6.3.0\include\c++\ming
w32\bits"
2. Open powerShell here.
3. Run g++ stdc++.h -std=c++17