BigInt

```cpp
#include <bits/stdc++.h>

using namespace std;

typedef long long ll;
typedef long double ld;
const ll M = 1e9 + 7;

__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;
}
void print(__int128 x) {
    if (x < 0) {
        putchar('-');
        x = -x;
    }
    if (x > 9) print(x / 10);
    putchar(x % 10 + '0');
}
bool cmp(__int128 x, __int128 y) { return x > y; }

using lll = __int128;

int main() {
    lll a, b;
    a = read(), b = read();
    print(a + b);
}
```

Fast Combination

```cpp
#include <bits/stdc++.h>

using namespace std;

typedef long long ll;
typedef long double ld;
const ll M = 1e9 + 7;

ll binomialCoeff(ll n, ll r) {
    if (r > n)
        return 0;
    ll m = 1000000007;
    ll inv[r + 1] = { 0 };
    inv[0] = 1;
    if(r+1>=2)
    inv[1] = 1;
```

```
    // Getting the modular inversion
    // for all the numbers
    // from 2 to r with respect to m
    // here m = 1000000007
    for (ll i = 2; i <= r; i++) {
        inv[i] = m - (m / i) * inv[m % i] % m;
    }

    ll ans = 1;

    for (ll i = 2; i <= r; i++) {
        ans = ((ans % m) * (inv[i] % m)) % m;
    }

    // for (n)*(n-1)*(n-2)*...*(n-r+1) part
    for (ll i = n; i >= (n - r + 1); i--) {
        ans = ((ans % m) * (i % m)) % m;
    }
    return ans;
}
```

Inverse pow

```
ll power(ll x, unsigned int y, ll p) {
    ll res = 1;
    x = x % p;
    if (x == 0) return 0;
    while (y > 0) {
        if (y & 1) res = (res*x) % p;
        y = y>>1;
        x = (x*x) % p;
    }
    return res;
}

ll inverse(ll a, ll p) {
    return power(a, p-2, p);
}

ll factorial(ll n, ll p) {
    ll result = 1;
    for (ll i = 1; i <= n; i++)
        result = (result * i) % p;

    return result;
}
```

Combination from n by k

```
vector<int> ans;

void gen(int n, int k, int idx, bool rev) {
    if (k > n || k < 0)
        return;

    if (!n) {
        for (int i = 0; i < idx; ++i) {
            if (ans[i])
                cout << i + 1;
```

```
        }
        cout << "\n";
        return;
    }

    ans[idx] = rev;
    gen(n - 1, k - rev, idx + 1, false);
    ans[idx] = !rev;
    gen(n - 1, k - !rev, idx + 1, true);
}

void all_combinations(int n, int k) {
    ans.resize(n);
    gen(n, k, 0, false);
}

int main() {
    all_combinations(6, 2);
}
```

Minimum spanning tree

```
vector<ll> parent, ranked;

void make_set(ll v) {
    parent[v] = v;
    ranked[v] = 0;
}

ll find_set(ll v) {
    if (v == parent[v])
        return v;
    return parent[v] = find_set(parent[v]);
}

void union_sets(ll a, ll b) {
    a = find_set(a);
    b = find_set(b);
    if (a != b) {
        if (ranked[a] < ranked[b])
            swap(a, b);
        parent[b] = a;
        if (ranked[a] == ranked[b])
            ranked[a]++;
    }
}

struct Edge {
    ll u, v, weight;
    bool operator<(Edge const& other) {
        return weight < other.weight;
    }
};

int main() {
    ll n, v; cin >> n >> v;
    vector<Edge> edges(n+1);

    ll cost = 0;
    vector<Edge> result(v);
    parent.resize(v);
```

```
    ranked.resize(v);
    for (ll i = 0; i < v; i++)
        make_set(i);

    for (ll i = 0; i < v; i++) {
        cin >> edges[i].u >> edges[i].v >> edges[i].weight;
    }

    sort(edges.begin(), edges.end());

    for (Edge e : edges) {
        if (find_set(e.u) != find_set(e.v)) {
            cost += e.weight;
            result.push_back(e);
            union_sets(e.u, e.v);
        }
    }
    cout << cost << endl;
}
```

Sieves of eratosthenes

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

typedef long long ll;
typedef long double ld;
const ll M = 1e9 + 7;
#define MAXN 100001

ll spf[MAXN];

void sieve() {
    spf[1] = 1;
    for (ll i=2; i<MAXN; i++)
        spf[i] = i;

    for (ll i=4; i<MAXN; i+=2)
        spf[i] = 2;

    for (ll i=3; i*i<MAXN; i++) {
        if (spf[i] == i) {
            for (ll j=i*i; j<MAXN; j+=i)
                if (spf[j]==j)
                    spf[j] = i;
        }
    }
}

vector<ll> getFactorization(ll x) {
    vector<ll> ret;
    while (x != 1) {
        ret.push_back(spf[x]);
        x = x / spf[x];
    }
```

```
    return ret;
}

int main() {
    sieve();
    ll x = 128;
    cout << "prime factorization for " << x << " : ";

    vector <ll> p = getFactorization(x);

    for (ll i=0; i<p.size(); i++)
        cout << p[i] << " ";
    cout << endl;
    return 0;
}
```

Matrix expo

```cpp
ll add(ll a, ll b) { return (a + b) % MOD; }
ll mul(ll a, ll b) { return (a * b) % MOD; }

struct Matrix {
    ll mat[3][3];

    Matrix operator * (Matrix b) {
        Matrix res;
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                res.mat[i][j] = 0;
                for (int k = 0; k < 3; k++)
                    res.mat[i][j] = add(res.mat[i][j], mul(mat[i][k], b.mat[k][j]));
            }
        }
        return res;
    }
};

Matrix powmod(Matrix a, ll b, ll p = MOD){
    Matrix res;
    for (int i = 0; i < 3; i++)
        for (int j = 0; j < 3; j++)
            res.mat[i][j] = (i == j);

    while (b > 0){
        if (b & 1) res = res * a;
        a = a * a;
        b >>= 1;
    }
    return res;
}

void solve() {
    ll k;
    cin >> k;
    Matrix a;
    for (int i = 0; i < 3; i++)
        for (int j = 0; j < 3; j++)
            cin >> a.mat[i][j];

    a = powmod(a, k);
    for (int i = 0; i < 3; i++)
        for (int j = 0; j < 3; j++)
            cout << a.mat[i][j] << " \n"[j == 2];
}
```

BFS

```
class Graph
{
    int V;    // No. of vertices

    // Pointer to an array containing adjacency
    // lists
    list<int> *adj;
public:
    Graph(int V);  // Constructor

    // function to add an edge to graph
    void addEdge(int v, int w);

    // prints BFS traversal from a given source s
    void BFS(int s);
};

Graph::Graph(int V)
{
    this->V = V;
    adj = new list<int>[V];
}

void Graph::addEdge(int v, int w)
{
    adj[v].push_back(w); // Add w to v's list.
}

void Graph::BFS(int s)
{
    // Mark all the vertices as not visited
    bool *visited = new bool[V];
    for(int i = 0; i < V; i++)
        visited[i] = false;

    // Create a queue for BFS
    list<int> queue;

    // Mark the current node as visited and enqueue it
    visited[s] = true;
    queue.push_back(s);

    // 'i' will be used to get all adjacent
    // vertices of a vertex
    list<int>::iterator i;

    while(!queue.empty())
    {
        // Dequeue a vertex from queue and print it
```

```
        s = queue.front();
        cout << s << " ";
        queue.pop_front();

        // Get all adjacent vertices of the dequeued
        // vertex s. If a adjacent has not been visited,
        // then mark it visited and enqueue it
        for (i = adj[s].begin(); i != adj[s].end(); ++i)
        {
            if (!visited[*i])
            {
                visited[*i] = true;
                queue.push_back(*i);
            }
        }
    }
}

// Driver program to test methods of graph class
int main()
{
    // Create a graph given in the above diagram
    Graph g(4);
    g.addEdge(0, 1);
    g.addEdge(0, 2);
    g.addEdge(1, 2);
    g.addEdge(2, 0);
    g.addEdge(2, 3);
    g.addEdge(3, 3);

    cout << "Following is Breadth First Traversal "
         << "(starting from vertex 2) \n";
    g.BFS(2);

    return 0;
}
```

DFS

```
class Graph
{
public:
    map<int, bool> visited;
    map<int, list<int>> adj;

    // function to add an edge to graph
    void addEdge(int v, int w);

    // DFS traversal of the vertices
    // reachable from v
    void DFS(int v);
};
```

```
void Graph::addEdge(int v, int w)
{
    adj[v].push_back(w); // Add w to v's list.
}

void Graph::DFS(int v)
{
    // Mark the current node as visited and
    // print it
    visited[v] = true;
    cout << v << " ";

    // Recur for all the vertices adjacent
    // to this vertex
    list<int>::iterator i;
    for (i = adj[v].begin(); i != adj[v].end(); ++i)
        if (!visited[*i])
            DFS(*i);
}

// Driver code
int main()
{
    // Create a graph given in the above diagram
    Graph g;
    g.addEdge(0, 1);
    g.addEdge(0, 2);
    g.addEdge(1, 2);
    g.addEdge(2, 0);
    g.addEdge(2, 3);
    g.addEdge(3, 3);

    cout << "Following is Depth First Traversal"
            " (starting from vertex 2) \n";
    g.DFS(2);

    return 0;
}
```

Gcd lcm

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

long long lcm (long long a, long long b) {
    return a / gcd(a, b) * b;
```

```
}
```

Determinant matrix

```
const double EPS = 1E-9;
int n;
vector < vector<double> > a (n, vector<double> (n));

double det = 1;
for (int i=0; i<n; ++i) {
    int k = i;
    for (int j=i+1; j<n; ++j)
        if (abs (a[j][i]) > abs (a[k][i]))
            k = j;
    if (abs (a[k][i]) < EPS) {
        det = 0;
        break;
    }
    swap (a[i], a[k]);
    if (i != k)
        det = -det;
    det *= a[i][i];
    for (int j=i+1; j<n; ++j)
        a[i][j] /= a[i][i];
    for (int j=0; j<n; ++j)
        if (j != i && abs (a[j][i]) > EPS)
            for (int k=i+1; k<n; ++k)
                a[j][k] -= a[i][k] * a[j][i];
}

cout << det;
```

Extended gcd

```
pair <LL,LL> extgcd(LL a, LL b){
bool swapped = false;
if (a < b){
swapped = true; swap(a,b);
}
LL x, y;
if (b == 0){
x = 1; y = 0; // gcd(a,0) = a*1 + 0*0
}
else{
LL xp, yp;
tie(xp,yp) = extgcd(b,a%b);
x = yp; y = xp - (a/b)*yp;
}
if (swapped)
swap(x,y);
return {x,y};
}
pair<LL,LL> ee2 = extgcd(a,b);
x = ee2.first, y = ee2.second;
```

D = ax + by

```
tuple<LL,LL,LL> xgcd(LL a, LL b){
// returns a triple (d,x,y) s.t. d = gcd(a,b) = a*x + b*y
if (b == 0){
return make_tuple(a,1,0);
// gcd(a,0)= a = 1*a + 0*0
}
else{
LL dp,xp,yp;
tie(dp,xp,yp) = xgcd(b,a%b);
LL d = dp, x = yp, y = xp - (a/b)*yp;
return make_tuple(d,x,y);
}
}

tuple<LL,LL,LL> ee1 = xgcd(a,b);
LL d = get<0>(ee1), x = get<1>(ee1), y = get<2>(ee1);
```

Sum of subset is in array

```
bool isSubsetSum(int set[], int n, int sum)
{
    // The value of subset[i][j] will be true if
    // there is a subset of set[0..j-1] with sum
    // equal to i
    bool subset[n + 1][sum + 1];

    // If sum is 0, then answer is true
    for (int i = 0; i <= n; i++)
        subset[i][0] = true;

    // If sum is not 0 and set is empty,
    // then answer is false
    for (int i = 1; i <= sum; i++)
        subset[0][i] = false;

    // Fill the subset table in bottom up manner
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= sum; j++) {
            if (j < set[i - 1])
                subset[i][j] = subset[i - 1][j];
            if (j >= set[i - 1])
                subset[i][j] = subset[i - 1][j]
                               || subset[i - 1][j - set[i - 1]];
        }
    }

    /*   // uncomment this code to print table
     for (int i = 0; i <= n; i++)
     {
       for (int j = 0; j <= sum; j++)
```

```
            printf ("%4d", subset[i][j]);
        cout <<"\n";
      }*/

    return subset[n][sum];
}
```

Sort by alpha

```
bool comp(st a, st b) {
    if (accumulate(a.sc.begin(), a.sc.end(), 0) > accumulate(b.sc.begin(),
b.sc.end(), 0)) return 1;
    else if (accumulate(a.sc.begin(), a.sc.end(), 0) < accumulate(b.sc.begin(),
b.sc.end(), 0)) return 0;
        else {
        if (a.sc > b.sc) return 1;
        else if (a.sc < b.sc) return 0;
        else {
            if (a.id < b.id) return 1;
            else return 0;
        }
    }
}
```