TEMPLATE

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

#define optimize ios::sync_with_stdio(false); cin.tie(NULL); cout.tie(NULL)
#define all(x) x.begin(), x.end()
#define endl '\n'
#define yes() cout << "YES\n"
#define no() cout << "NO\n"
typedef long long ll;

template<typename T> ostream& operator<<(ostream &os, const vector<T> &v) { os << "{"; for (typename vector<T>::const_iterator vi = v.begin(); vi != v.end(); ++vi) { if (vi != v.begin()) os << ", "; os << *vi; } os << "}"; return os; }
template<typename A, typename B> ostream& operator<<(ostream &os, const pair<A, B> &p) { os << '(' << p.first << ", " << p.second << ')'; return os; }

#define INF 0x3f3f3f3f
#define INFLL 0x3f3f3f3f3f3f3f3f
#define PI 3.14159265358979323846264338327950288419716939937510582097494459230781640628
#define MOD 1000000007

//------------------solution starts below------------------

void solve(ll tt)
{

}

int main()
{
   optimize;
   ll tt=1;
   //cin >> tt;
   while(tt--) solve(tt);
   //while(cin >> tt && tt) solve(tt);

   return 0;
}
```

BRUTE FORCE

Generate all Subsets

```
const int MAXN = 12; // Maximum solutuion size
vector<int> s; // Set of all elements
vector<bool> p; // Stores partial solutions

void print(int n)
{
   for(int i = 0 ; i < n ; i++)
       if(p[i])
           cout << s[i] << ' ';
   cout << endl;
}

void generate(int pos, int n) // pos is the current position in s
{
   if(pos == n) // Stops in the last position
   {
       print(n);
       return;
   }

   p[pos] = true; // subset that contains element in s[pos]
   generate(pos+1, n);
   p[pos] = false; // subset that does not contain element in s[pos]
   generate(pos+1, n);
}

int main()
{
   p.resize(MAXN, 1);
   s = {1, 2, 3};

   generate(0, s.size());

   return 0;
}
```

DATA STRUCTURES

Segtree

```
/*
   Segment Tree for range minimum query
   tl and tr: boundaries of the current segment
   l and r: query boundaries
   NULLVALUE: For minimum, a very high value works
*/

const int NULLVALUE = 0x3f3f3f3f;
const int MAXN = 1e5;

struct Seg
{
   vector<ll> input;
   ll seg[4*MAXN];
   ll n;

   ll build(int node, int tl, int tr)
   {
       if(tl == tr)
           return seg[node] = input[tl];

       int tm = (tl+tr)/2;
       return seg[node] = min(build(2*node, tl, tm), build(2*node+1, tm+1, tr));
   }

   void build(vector<ll>& v)
   {
       n = v.size();
       input = v;
       build(1, 0, n-1);
   }

   ll query(int node, int tl, int tr, int l, int r)
   {
       if(tr < l or tl > r)
           return NULLVALUE;
       if(tr <= r and tl >= l)
           return seg[node];
       int tm = tl+(tr-tl)/2;
       return min(query(2*node, tl, tm, l, r), query(2*node+1, tm+1, tr, l, r));
   }

   ll query(int l, int r)
   {
       return query(1, 0, n-1, l, r);
   }

   ll update(int node, int tl, int tr, int i, int k)
   {
       if(i < tl or i > tr)
```

```
        return seg[node];
    if(tl == tr)
        return seg[node] = k;
    int tm = tl+(tr-tl)/2;
    return seg[node] = min(update(2*node, tl, tm, i, k), update(2*node+1, tm+1,
tr, i, k));
  }

  void update(int i, ll k)
  {
    update(1, 0, n-1, i, k);
  }
};
```

Segtree (count values)

```
typedef pair<ll, ll> pll;
/*
   Segment Tree for range minimum query just like the one from segBasics.cpp, but
finds minimum and counts it appears
*/

const int NULLVALUE = 0x3f3f3f3f;
const int MAXN = 1e5;

pll compare(pll a, pll b)
{
   if(a.first == b.first)
       return {a.first, a.second+b.second};
   return min(a, b);
}

struct Seg
{
   vector<ll> input;
   pll seg[4*MAXN];
   ll n;

   pll build(int node, int tl, int tr)
   {
       if(tl == tr)
           return seg[node] = {input[tl], 1};

       int tm = (tl+tr)/2;
       return seg[node] = compare(build(2*node, tl, tm), build(2*node+1, tm+1, tr));
   }

   void build(vector<ll>& v)
   {
       n = v.size();
```

```
        input = v;
        build(1, 0, n-1);
    }

    pll query(int node, int tl, int tr, int l, int r)
    {
        if(l > r)
            return {NULLVALUE, 0};
        if(l == tl and r == tr)
            return seg[node];
        int tm = tl+(tr-tl)/2;
        return compare(query(2*node, tl, tm, l, min(r, tm)), query(2*node+1, tm+1,
tr, max(l, tm+1), r));

    }

    pll query(int l, int r)
    {
        return query(1, 0, n-1, l, r);
    }

    pll update(int node, int tl, int tr, int i, ll k)
    {
        if(tl > i or tr < i)
            return {NULLVALUE, 0};
        if(tl == tr)
            return seg[node] = {k, 1};
        int tm = tl+(tr-tl)/2;
        return seg[node] = compare(update(2*node, tl, tm, i, k), update(2*node+1,
tm+1, tr, i, k));
    }

    void update(int i, ll k)
    {
        update(1, 0, n-1, i, k);
    }
};
```

Fenwick Tree

```
/*
    Binary Indexed Tree for range sum query (PURQ)
*/

#define lsOne(n) (n&-n)
struct Bit
{
    int n;
    vector<ll> bit;
```

```
    Bit(int _n=0)
    {
        n = _n;
        bit.assign(n+1, 0);
    }

    Bit(vector<ll>& v)
    {
        n = v.size();
        bit.assign(n+1, 0);

        for (int i = 1 ; i <= n ; i++)
        {
            bit[i] += v[i - 1];
            int j = i + lsOne(i);
            if (j <= n)
                bit[j] += bit[i];
        }
    }

    void update(int i, ll k)
    {
        for (i++ ; i <= n ; i += lsOne(i))
            bit[i] += k;
    }

    ll query(int r)
    {
        ll sum = 0;
        for (r++ ; r ; r -= lsOne(r))
            sum += bit[r];
        return sum;
    }

    ll query(int l, int r)
    {
        return query(r) - query(l - 1);
    }
};
```

DIVIDE AND CONQUER

Binary Search

```
bool ok(int x){return x>=30;}
bool okc(double x){return x >= 30.51122;}

int firstTrue(int l, int r)
```

```
{
   int mid;
   while (l<r)
   {
       mid = l+(r-l)/2;

       if(ok(mid))
           r = mid;
       else
           l = mid+1;
   }

   if(!ok(l)) // [l, r] is all False
       return -1;
   else
       return l;
}

// Using this function but with f2(x){return !f(x)} finds the last true
int lastFalse(int l, int r) // Finds last False in a function (FFFFFTTT)
{
   int mid;
   while (l<r)
   {
       mid = l+(r-l+1)/2; // +1 so it rounds up

       if(ok(mid))
           r = mid-1;
       else
           l = mid;
   }

   if(ok(l)) // [l, r] is all True
       return -1;
   else
       return l;
}

double firstTrue(double l, double r) // Finds first true in a continuous interval
(FFFFTTT)
{
   double mid;

   int p = 100; // more iterations = more precision
   for(int i = 0 ; i < p ; i++)
   {
       mid = l+(r-l)/2;
       if(okc(mid))
           r = mid;
       else
           l = mid;
   }
```

```
    if(!okc(l)) // [l, r] is all False
        return -1;
    else
        return l;
}
```

Ternary Search

```
double f(double x)
{return abs(500.0-x); // f -> \/}

double tsearch(double l, double r) // Finds minimum in continuous interval
{
    double eps = 1e-9;
    double m1, m2;

    while (fabs(r-l) > eps)
    {
        m1 = l+(r-l)/3;
        m2 = r-(r-l)/3;

        if(f(m1) > f(m2))
            l = m1;
        else
            r = m2;
    }

    return l;
}

int main()
{
    cout << tsearch(0, 1000) << endl;
    return 0;
}
```

DYNAMIC PROGRAMMING

Change

```
/*
   dp aproach to get the number of ways we can get to value n with a set of coins
(O(n*m) where n is the value and m the size of the coin set)
*/

vector<int> c; // All coin values
vector<int> ways; // Ways to get to value i

void fill(int n) // Fill the interval [0, n] with the results
```

```
{
   ways.assign(n+1, 0);
   ways[0] = 1;
   for(auto coin: c)
       for(int i = 0 ; i+coin <= n ; i++)
           if(ways[i])
               ways[i+coin] += ways[i];
}

int main()
{
   int n, value; cin >> n >> value;
   c.resize(n);
   for(int i = 0 ; i < n ; i++)
       cin >> c[i];
   fill(value);
   cout << ways[value] << endl;
}
```

Knapsack

```
/*
   Famous knapsack problem, this solution uses dp to find the best subset of items
possible in O(n*sum)
*/

#define value first
#define weight second
vector<pair<int, int>> v; // knapsack
vector<vector<int>> dp; // dp table (has to be initialized)

int maximize(int i, int j) // Greatest value using items in [0, i) using at max j
capacity
{
   if(!i or !j) return 0;
   if(dp[i][j] != -1) return dp[i][j];

   if(j-v[i-1].weight >= 0)
       return dp[i][j] = max(maximize(i-1, j), maximize(i-1,
j-v[i-1].weight)+v[i-1].value);
   else
       return maximize(i-1, j);
}

int main()
{
   int n, maxCapacity, sum = 0;
   cin >> n >> maxCapacity;
   v.resize(n);
   for(int i = 0 ; i < n ; i++)
```

```
    {
        cin >> v[i].value >> v[i].weight;
        sum+=v[i].value;
    }
    dp.resize(n+1, vector<int>(sum+1, -1));
    cout << maximize(n, maxCapacity) << endl;
}
```

Longest Common Subsequence

```
/*
    Longest common subsequence in 2 strings
    ex: bride bridge
        11111 111101 -> 5
*/

string a, b;
int n, m;

vector<vector<int>> dp;

int lcs(int i, int j) // longest common subsequence (lcs) ate in [0, i) and [0, j)
{
    if(!i or !j) return 0;
    if(dp[i][j] != -1) return dp[i][j];

    if(a[i-1] == b[j-1])
        return dp[i][j] = 1 + lcs(i-1, j-1);
    else
        return dp[i][j] = max(lcs(i, j-1), lcs(i-1, j));
}

int main()
{
    getline(cin, a);
    getline(cin, b);
    n = a.size();
    m = b.size();

    dp.resize(n+1, vector<int>(m+1, -1));
    cout << lcs(n, m) << endl;
}
```

Longest Increasing Subsequence

```
/*
    Larger increasing subsequence
*/

vector<int> v;
```

```
vector<int> dp;

int lis(int i) // lis in v[0, i]
{
   if(i < 0) return 0;
   if(dp[i] != -1) return dp[i];

   int res = 1;
   for(int j = 0 ; j < i ; j++)
   {
       if(v[j] < v[i])
           res = max(res, lis(j)+1);
       else
           res = max(res, lis(j));
   }
   return dp[i] = res;
}

int main()
{
   int n; cin >> n;
   v.resize(n);
   dp.resize(n, -1);
   for(int i= 0 ; i < n ; i++)
       cin >> v[i];
   cout << lis(n-1) << endl;
}
```

Range Sum (Kadane)

```
/*
   Kadane's algorithm to find greatest sum in a subarray of v
*/

int kadane(vector<int>& v)
{
   int curr = 0; // Current sum
   int best = 0; // Best sum
   int i = 0, j = 0; // Best range
   int last = 0; // Last reset
   for(int idx = 0 ; idx < v.size() ; idx++)
   {
       curr+=v[idx];

       if(curr > best)
       {
           best = curr;
           i = last;
           j = idx;
       }
       if(curr < 0)
```

```
        {
            last = idx+1;
            curr = 0;
        }
    }
    return best;
}

int main()
{
    vector<int> v(5);
    for(int i = 0 ; i < 5 ; i++)
        cin >> v[i];
    cout << kadane(v) << endl;
}
```

GEOMETRY

```
/*
   This is a struct representing Points and Vectors in 2d
*/
struct Point2d
{
   double x, y;

   Point2d(double x=0, double y=0): x(x), y(y) {} // Constructor

   // Standard operations
   Point2d operator+(Point2d other) {return Point2d(x+other.x, y+other.y);}
   Point2d operator-(Point2d other) {return Point2d(x-other.x, y-other.y);}
   Point2d operator*(double s) {return Point2d(s*x, s*y);}
   Point2d operator/(double s) {return Point2d(x/s, y/s);}

   double operator*(Point2d other) {return x*other.x + y*other.y;} // Dot product
   double operator^(Point2d other) {return x*other.x - y*other.y;} // Cross Product

   // Comparison
   bool operator==(Point2d other) {return x==other.x and y==other.y;}
   bool operator!=(Point2d other) {return !(*this==other);}

   // cout for debug
   friend ostream &operator<<(ostream &os, Point2d const &p) {return os << "(" <<
p.x << ", " << p.y << ")";}
};

// returns vector from a to b
Point2d toVector(Point2d a, Point2d b){return b-a;}

// If cross product equals 0, a, b and c are collinear
bool collinear(Point2d a, Point2d b, Point2d c){return toVector(a, b) ^ toVector(a,
c) == 0;}
```

```
// Triangle Area
double area(Point2d a, Point2d b, Point2d c){return 0.5 * (toVector(a, b) ^ toVector(a, c));}

/*
The cross product can be used to check if a vector is closer to another rotating clockwise or counter-clockwise.
Take 2 vectors v and u.
v^u == 0 -> the vectors are in the same line;
v^u > 0 -> v rotates counter-clockwise to meet u;
v^u < 0 -> v rotates clockwise to meet u.
*/

// Returns vec's norm^2
double norm2(Point2d vec){return vec.x*vec.x + vec.y*vec.y;}

// Returns if p is between p1 and p2 but NOT equal to p1 or p2
bool between(Point2d p1, Point2d p2, Point2d p)
{
   return collinear(p1, p2, p) and toVector(p, p1)*toVector(p, p2) < 0;
}
```

GRAPHS

Max Flow / Min cut

```
#define INF 0x3f3f3f3f
// Max flow using Edmonds-Karp algorithm

int n;
vector<vector<int>> capacity;
vector<vector<int>> adj;

int bfs(int s, int t, vector<int>& parent)
{
   parent.assign(parent.size(), -1);
   parent[s] = -2;
   queue<pair<int, int>> q;
   q.push({s, INF});

   while (!q.empty())
   {
       int cur = q.front().first;
       int flow = q.front().second;
       q.pop();

       for (int next : adj[cur])
       {
           if (parent[next] == -1 && capacity[cur][next])
```

```
            {
                parent[next] = cur;
                int new_flow = min(flow, capacity[cur][next]);
                if (next == t)
                    return new_flow;
                q.push({next, new_flow});
            }
        }
    }

    return 0;
}

int maxflow(int s, int t)
{
    int flow = 0;
    vector<int> parent(n);
    int new_flow;

    while (new_flow = bfs(s, t, parent))
    {
        flow += new_flow;
        int cur = t;
        while (cur != s)
        {
            int prev = parent[cur];
            capacity[prev][cur] -= new_flow;
            capacity[cur][prev] += new_flow;
            cur = prev;
        }
    }

    return flow;
}
```

Breadth First Search (BFS)

```
/*
   Using bfs to find shortest path in a graph
*/

#define INF 0x3f3f3f3f
vector<int> d, p;
vector<vector<int>> adj;

void bfs(int start=0, int end=-1)
{
    int n = adj.size();
    d.assign(n, INF);
    p.assign(n, -1);
```

```
    d[start] = 0;

    queue<int> q;
    q.push(start);

    while (!q.empty())
    {
        int curr = q.front(); q.pop();
        if(curr == end) return;

        for(auto nei: adj[curr]) if(d[nei] == INF)
        {
            q.push(nei);
            d[nei] = d[curr]+1;
            p[nei] = curr;
        }
    }
}

void restore_path(int start, int end, vector<int>& res)
{
    res.clear();
    for(int u = end ; u != start ; u = p[u])
        res.push_back(u);

    res.push_back(start);
    reverse(res.begin(), res.end());
}
```

Bipartite Graph

```
/*
    DFS used to check if a graph is bipartite
*/

vector<int> color; // colors are 0 or 1 and -1 means it is not colored yet
vector<vector<int>> adj;

bool colorize(int curr, int value) // Tries to bipart the component, returns if it
could do it
{
    color[curr] = value;
    for(auto e: adj[curr])
    {
        if(color[e] == color[curr]) // Neighboor with the same color -> not bipartite
            return false;
        if(color[e] == -1)
        {
            color[e] = value == 0 ? 1 : 0;
            if(!colorize(e, !value))
                return false;
```

```
        }
    }
    return true;
}

bool isBipartite() // Runs a dfs starting in every node (every component needs to be
searched)
{
    for(int i = 0 ; i < adj.size() ; i++)
        if(color[i] != -1 and !colorize(i, 0))
            return false;
    return true;
}
```

Dijkstra

```
#define INFLL 0x3f3f3f3f3f3f3f3f

vector<ll> d;
vector<int> p;
vector<vector<pair<int, ll>>> adj;

void dijkstra(int start=0, int end=-1)
{
    int n = adj.size();

    d.assign(n, INFLL);
    p.assign(n, -1);
    priority_queue<pair<ll, int>, vector<pair<ll, int>>, greater<pair<ll, int>>> pq;
    d[start] = 0;
    pq.push({0, start});
    while (!pq.empty())
    {
        auto [w1, u] = pq.top(); pq.pop();

        if(w1 > d[u]) continue;
        if(u == end) return;
        for(auto [v, w2]: adj[u])
        {

            if(d[u]+w2 < d[v])
            {
                p[v] = u;
                d[v] = d[u] + w2;
                pq.push({d[v], v});
            }
        }
    }
}

void restore_path(int start, int end, vector<int>& res)
```

```
{
   res.clear();
   for(int u = end ; u != start ; u = p[u])
       res.push_back(u);

   res.push_back(start);
   reverse(res.begin(), res.end());
}
```

Disjoint Set Union

```
struct dsu
{
   vector<int> parent; // parent of a node (not always the root of the tree)
   vector<int> r; // rank
   int n; // number of nodes

   dsu(int n)
   {
       parent.resize(n);
       for(int i = 0 ; i < n ; i++)
           parent[i] = i;
       r.resize(n);
       for(int i = 0 ; i < n ; i++)
           r[i] = 0;
   }

   void make_set(int v)
   {
       parent[v] = v;
       r[v] = 0;
   }

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

   void union_sets(int a, int b)
   {
       a = find_set(a);
       b = find_set(b);
       if (a != b)
       {
           if(r[a] < r[b])
               swap(a, b);
           parent[b] = a;
           if(r[a] == r[b])
```

```
                r[a]++;
        }
    }
};
```

Floyd-Warshall

```
#define INFLL 0x3f3f3f3f3f3f3f3f
/*
   Floy-Warshall algorithm to find minimum distance in a graph with or without
negative edges
*/

vector<vector<ll>> d; // Distance from i to j
void floydWarshall()
{
   int n = d.size();
   for (int k = 0; k < n; ++k)
       for (int i = 0; i < n; ++i)
           for (int j = 0; j < n; ++j) if (d[i][k] < INFLL and d[k][j] < INFLL)
               d[i][j] = min(d[i][j], d[i][k] + d[k][j]);

}

int main()
{
   int n, m; cin >> n >> m;
   d.resize(n, vector<ll>(n, INFLL));

   int u, v;
   ll w;
   for(int i = 0 ; i < m ; i++)
   {
       cin >> u >> v >> w;
       d[u][v] = w;
   }

   return 0;
}
```

Minimum Spanning Tree (Prim)

```
/*
   Prim's minimum spanning tree algorithm
   returns total weight
*/

vector<vector<pair<int, ll>>> adj; // <node, weight>

ll prim(int start=0)
```

```
{
   int n = adj.size();

   ll totalWeight = 0;
   int numTaken = 0; // Number of nodes taken

   vector<bool> taken(n, false); // Stores for every node, if it was taken
   priority_queue<pair<ll, int>, vector<pair<ll, int>>, greater<pair<ll, int>>> pq;
   pq.push({0, start});

   while (!pq.empty() and numTaken < n)
   {
       auto [w1, u] = pq.top(); pq.pop();

       if(taken[u]) continue;

       taken[u] = true;
       numTaken++;
       totalWeight += w1;

       for(auto [v, w2]: adj[u])
           if(!taken[v])
               pq.push({w2, v});
   }

   return totalWeight;
}
```

Topological Sort

```
#define all(x) x.begin(), x.end()

vector<vector<int>> adj;
vector<bool> visited;
vector<int> ans;

void dfs(int v)
{
   if(visited[v]) return;

   visited[v] = true;
   for(int u: adj[v])
       dfs(u);
   ans.push_back(v);
}

void topoSort(int n)
{
   visited.assign(n, false);
   ans.clear();
```

```
    for(int i = 0 ; i < n ; i++)
        dfs(i);
    reverse(all(ans));
}
```

NUMBER THEORY

Binary Pow

```
ll binpow(ll x, ll y) // x^y
{
    ll res = 1;
    while (y)
    {
        if(y&1) // If it's odd
            res*=x;
        x*=x;
        y >>= 1;
    }

    return res;
}
```

GCD and LCM

```
ll gcd(ll a, ll b) // Euclidean Algorithm
{
    if (!b)
        return a;
    return gcd(b, a%b);
}

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

Sieve of Erathostenes

```
const ll MAX = sqrt(10000010)+10; // Insert maximum input

vector<bool> prime;
vector<ll> allPrimes; // For calculating bigger primes
void sieve(int n) // Sieve of Eratosthenes
{
    prime.assign(n+10, true);
```

```cpp
    prime[0] = prime[1] = false;

    for(ll i = 2 ; i <= n  ; i++)
    {
        if(prime[i])
        {
            for(ll j = i*i ; j <= n ; j += i)
                prime[j] = false;

            allPrimes.push_back(i);
        }
    }
}

bool isPrime(ll n) // Uses previous algorithm
{
    if(n <= allPrimes.back())
        return prime[n];

    for(ll i = 0 ; i <= prime.size() ; i++)
    {
        if(n%allPrimes[i] == 0)
            return false;
        if(allPrimes[i]*allPrimes[i] > n) break;
    }
    return true;
}

void factorization(ll n, vector<int>& factors)
{
    for (ll d : allPrimes)
    {
        if (d * d > n)
            break;
        while (n % d == 0)
        {
            factors.push_back(d);
            n /= d;
        }
    }
    if (n > 1)
        factors.push_back(n);
}

int main()
{
    sieve(MAX);
    int n; cin >> n;
    for(int i = 0 ; i <= n*n ; i++)
        if(isPrime(i))
            cout << i << " is prime\n";

    return 0;
```

```
}
```

EXTRA

Bitmask

```
bool isPowerOf2(int n)
{
   return !(n&(n-1));
}

bool LSOne(int n) // Least Significant One
{
   return n&(-n);
}
```

New Float

```
// Float, but more assertive

double eps = 1e-9;

int comp(double a, double b)
{
   if(fabs(a-b) < eps) return 0;
   else if(a > b) return 1;
   else return -1;
}

struct newFloat
{
   double data;

   newFloat(double data): data(data) {}
   newFloat() {}

   template<typename T>
   bool operator==(const T& other) {return fabs(data-other) < eps;}
   template<typename T>
   bool operator!=(const T& other) {return !(*this == other);}
   template<typename T>
   bool operator>(const T& other) {return !(*this==other) and data > other;}
   template<typename T>
   bool operator>=(const T& other) {return (*this==other) or *this > other;}
   template<typename T>
   bool operator<(const T& other) {return !(*this==other) and data < other;}
   template<typename T>
   bool operator<=(const T& other) {return (*this==other) or *this < other;}
```

```
    friend ostream &operator<<(ostream &os, newFloat const &n) {return os << n.data;}
};
```