# Name : TeamGVP

# College : Gayatri vidya parishad.

**Contestants :**
PVSM Praveen Kumar
S Bhargava Ram
S Raghuveer Sharma

# Index :

**Segment Tree Lazy : (Range Max/Min)**

```
const int N=5;
#define MAX (1+(1<<6)) // Why? :D
#define inf 0x7fffffff
int arr[N+25];
int tree[MAX];
int lazy[MAX];
void build_tree(int node, int a, int b) {
    if(a > b) return;
    if(a == b) { tree[node] = arr[a]; return; }
    build_tree(node*2, a, (a+b)/2);
    build_tree(node*2+1, 1+(a+b)/2, b
    tree[node] = max(tree[node*2],
    tree[node*2+1]);
}

void update_tree(int node, int a, int b, int i, int j, int
value) {
    if(lazy[node] != 0) {
                tree[node] += lazy[node];
                if(a != b) {
            lazy[node*2] += lazy[node];
    lazy[node*2+1] += lazy[node];
        }
        lazy[node] = 0;
         }
    if(a > b || a > j || b < i)return;
    if(a >= i && b <= j) {
        tree[node] += value;
        if(a != b) {
            lazy[node*2] += value;
            lazy[node*2+1] += value;
        }
        return;
    }
    update_tree(node*2, a, (a+b)/2, i, j, value);
    update_tree(1+node*2, 1+(a+b)/2, b, i, j, value);
    tree[node] = max(tree[node*2],
    tree[node*2+1]);
}
```

```
int query_tree(int node, int a, int b, int i, int j) {
    if(a > b || a > j || b < i) return -inf;  //+inf in case
Min
    if(lazy[node] != 0) {
        tree[node] += lazy[node];
        if(a != b) {
            lazy[node*2] += lazy[node];
    lazy[node*2+1] += lazy[node];
                }
        lazy[node] = 0; }
    if(a >= i && b <= j)return tree[node];
    int q1 = query_tree(node*2, a, (a+b)/2, i, j);
    int q2 = query_tree(1+node*2, 1+(a+b)/2, b, i, j
    int res = max(q1, q2)
    return res;
}
//update_tree(1, 1, N, L, R, x);
//query_tree(1, 1, N, L, R);
```

**Note :** For sum in range query in lazy propagation.

1. Change tree[node] += lazy[node] to tree[node] += (b-a+1)*lazy[node]
2. Change max(q1,q2) to q1+q2
3. Out of range : -INF,+INF to 0.

**BIT With Binary Search :**

```
// Binary indexed tree supporting binary search.
struct BIT {
  int n;
  vector<int> bit;
// BIT can be thought of as having entries f[1], ..., f[n]
//0 Initialized.
  BIT(int n):n(n), bit(n+1) { }
   // returns f[1] + ... + f[idx-1] precondition idx <= n+1
  int read(int idx) {
    idx--;
    int res = 0;
    while (idx > 0) { res += bit[idx];
      idx -= idx & -idx;
    } return res;
  }

// returns f[idx1] + ... + f[idx2-1] precondition idx1 <= idx2 <=
n+1
  int read2(int idx1, int idx2) {
    return read(idx2) - read(idx1);
  }
```

```
// adds val to f[idx]
// precondition 1 <= idx <= n (there is no element 0!)
  void update(int idx, int val) {
    while (idx <= n) { bit[idx] += val;
      idx += idx & -idx;    }    }

  // returns smallest positive idx such that read(idx) >= target
  int lower_bound(int target) {
    if (target <= 0) return 1;
    int pwr = 1; while (2*pwr <= n) pwr*=2;
    int idx = 0; int tot = 0;
    for (; pwr; pwr >>= 1) {
      if (idx+pwr > n) continue;
      if (tot + bit[idx+pwr] < target) {
        tot += bit[idx+=pwr];    }  }
    return idx+2;
}
  // returns smallest positive idx such that read(idx) > target
  int upper_bound(int target) {
    if (target < 0) return 1;
    int pwr = 1; while (2*pwr <= n) pwr*=2;
    int idx = 0; int tot = 0;
    for (; pwr; pwr >>= 1) {
      if (idx+pwr > n) continue;
      if (tot + bit[idx+pwr] <= target) {
        tot += bit[idx+=pwr];
      }
    }
    return idx+2;
  }
};
```

**BIT with Range Updates :**

```
// BIT with range updates, inspired by Petr Mitrichev

struct BIT {
  int n;
  vector<int> slope;
  vector<int> intercept;
// BIT can be thought of as having entries f[1], ..., f[n]
// which are 0-initialized
  BIT(int n): n(n), slope(n+1), intercept(n+1) {}
// returns f[1] + ... + f[idx-1]  precondition idx <= n+1
 int query(int idx) {
      int m = 0, b = 0;
      for (int i = idx-1; i > 0; i -= i&-i) {
        m += slope[i];
        b += intercept[i];
      }
      return m*idx + b; }

// adds amt to f[i] for i in [idx1, idx2)
// precondition 1 <= idx1 <= idx2 <= n+1 (you   can't  //
update element 0)
void update(int idx1, int idx2, int amt) {
      for (int i = idx1; i <= n; i += i&-i) {
        slope[i] += amt;
        intercept[i] -= idx1*amt;
      }
      for (int i = idx2; i <= n; i += i&-i) {
        slope[i] -= amt;
        intercept[i] += idx2*amt;
      }
   }
};
```

struct **FenwickTree (Normal BIT)**

```
{
    typedef ll T;
    vector<T> v;
    FenwickTree(int n): v(n, 0) {}
    void add(int i, T x) {
        for(; i < (int)v.size(); i |= i+1) v[i] += x;
    }
    T sum(int i) {   //[0, i)
        T r = 0;
        for(-- i; i >= 0; i = (i & (i+1)) - 1) r += v[i];
        return r;
    }
    T sum(int left, int right) {    //[left, right)
        return sum(right) - sum(left);
    }
};
```

**BIT Example :**

```
int tree[(1<<LOGSZ)+1];
int N = (1<<LOGSZ);
        // add v to value at x
void set(int x, int v) {
  while(x <= N) {    tree[x] += v;    x += (x & -x);  }
}
        // get cumulative sum up to and including x
int get(int x) {
  int res = 0;
  while(x) {    res += tree[x];    x -= (x & -x);  }
  return res;
}
        // get largest value with cumulative sum
less than or equal to x;
        // for smallest, pass x-1 and add 1 to result
int getind(int x) {
  int idx = 0, mask = N;
  while(mask && idx < N) {
    int t = idx + mask;
    if(x >= tree[t]) {
      idx = t;
      x -= tree[t];
    }
    mask >>= 1;
  }
  return idx;
```

---

struct **UnionFind**

```
{
    vector<int> data;
    void init(int n) { data.assign(n+1, -1); }
    bool unionSet(int x, int y)  { x = root(x); y =
root(y);
     if(x != y) { if(data[y] < data[x]) swap(x, y);
        data[x] += data[y]; data[y] = x; }
        return x != y;
    }
    bool findSet(int x, int y)   {   return root(x) ==
root(y);  }
    int root(int x)  {     return data[x] < 0 ? x : data[x]
        = root(data[x]);  }
    int size(int x) { return -data[root(x)]; }
};
```

---

vector<int> **longestIncreaseSequence**(const vector<int>& a) *(NlogN) Binary Search + Greedy*

```
{
    const int n = a.size();
    vector<int> A(n, INF);
    vector<int> id(n);
    for(int i = 0; i < n; ++i) {
        id[i] = lower_bound(all(A), a[i]) - A.begin();
        A[id[i]] = a[i];
    }
    int m = *max_element(id.begin(), id.end());
    vector<int> b(m+1);
    for(int i = n-1; i >= 0; --i)
        if(id[i] == m) b[m--] = a[i];
    return b;
}
```

---

struct SegTree  (Without Lazy.)

```
{
    vector<int> Tree;
        void init(int Arr[],int N) {
        Tree.assign(4*N,0); Build(Arr,1,N,1); }
    void Build(int Arr[],int L,int R,int POS)
    {
      if(L==R)  {   Tree[POS]=Arr[L]; return; }
        int Mid=(L+R)/2;
                Build(Arr,L,Mid,2*POS);
                Build(Arr,Mid+1,R,2*POS+1);
      Tree[POS]=max(Tree[POS*2],Tree[POS*2+1]);
    }

    int Query(int L,int R,int QL,int QR,int POS)
    {
      if(QL<=L && QR>=R) return Tree[POS];
      if (QL>R || QR<L)  return -1;
      int Mid=(L+R)/2;   return max(
                Query(  L,  Mid  , QL,QR ,  2*POS),
                Query(Mid+1 ,R   , QL,QR ,
                2*POS+1) );
    }
```

```
void Update(int L,int R,int QL,int QR,int POS,int
VALUE)
    {
            if(L>R || QL>R || QR<L) { return; }
            if(L==R)   { if(QL==L && QR==R)
            Tree[POS]=VALUE; return;  }
            int Mid=(L+R)/2;
            Update(L,Mid,QL,QR,2*POS,VALUE);
            Update(Mid+1,R,QL,QR,2*POS+1,VALUE);
        Tree[POS]=max(Tree[POS*2],Tree[POS*2+1]);
    }
};
        Seg.Update(1,N,L,R,1,VALUE));
        Seg.Query(1,N,L,R,1);
```

**Matrix Exponentation :**

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N=4;
ll MOD=1e9+7;
void Copy(ll A[N][N],ll B[N][N]) {
  for(int i=0;i<N;i++)
     for(int j=0;j<N;j++)
        A[i][j]=B[i][j];
}
void Mul(ll A[N][N],ll B[N][N]) {
     ll C[N][N];
          for(int i=0;i<N;i++){
              for(int j=0;j<N;j++){
                 C[i][j]=0;
                 for(int k=0;k<N;k++)  {
C[i][j]=(C[i][j]+(ll)A[i][k]*(ll)B[k][j])%MOD;
                 }  }   }
        Copy(A,C);
}
void Power(ll A[N][N],int Exp) {
 ll Ans[4][4]={{1,0,0,0},{0,1,0,0},{0,0,1,0},{0,0,0,1}};
    while(Exp){
            if(Exp&1)  Mul(Ans,A);
            Mul(A,A);
            Exp=Exp>>1;
        }
        Copy(A,Ans);
}
ll Solve(int K)
{
   if(K==0)      return 0;
   if(K==1)      return 1;
   if(K==2)      return 2;
    ll A[4][4]={{1,1,1,0},{1,0,0,0},{0,0,1,1},{0,0,0,1}};
   Power(A,K-1);
  return
((A[0][0]*1+A[0][1]*0+A[0][2]*1+A[0][3]*1)
%MOD);
}
```

The above code is for **f(n) = f(n-1)+f(n-2) + n-1**

$$\begin{vmatrix} F(n) \\ F(n-1) \\ N \\ 1 \end{vmatrix} = \begin{vmatrix} 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{vmatrix} \begin{vmatrix} F(N-1) \\ F(N-2) \\ N-1 \\ 1 \end{vmatrix}$$

---

**Square Root Decomposition :**

```
#include <bits/stdc++.h>
using namespace std;
   // Variable to represent block size. This is made
global
   // so compare() of sort can use it.
int block;
   // Structure to represent a query range
struct Query{
    int L, R;
 };
   // Function used to sort all queries so that all queries
   // of same block are arranged together and within a
block,
   // queries are sorted in increasing order of R values.
bool compare(Query x, Query y){
    // Different blocks, sort by block.
  if (x.L/block != y.L/block)
     return x.L/block < y.L/block;
   // Same block, sort by R value
  return x.R < y.R;
}
// Prints sum of all query ranges. m is number of queries
// n is size of array a[].
void queryResults(int a[], int n, Query q[], int m){
  // Find block size
  block = (int)sqrt(n);
  // Sort all queries so that queries of same blocks
```

```
  // are arranged together.
  sort(q, q + m, compare);
  // Initialize current L, current R and current sum
  int currL = 0, currR = 0;
  int currSum = 0;
  // Traverse through all queries
 for (int i=0; i<m; i++){
    // L and R values of current range
    int L = q[i].L, R = q[i].R;
    // Remove extra elements of previous range. For
    // example if previous range is [0, 3] and current
    // range is [2, 5], then a[0] and a[1] are subtracted
    while (currL < L){
       currSum -= a[currL];
       currL++;
    }
    // Add Elements of current Range
    while (currL > L){
       currSum += a[currL-1];
       currL--;
    }
    while (currR <= R){
       currSum += a[currR];
       currR++;
    }
    // Remove elements of previous range.  For example
    // when previous range is [0, 10] and current range
    // is [3, 8], then a[9] and a[10] are subtracted
    while (currR > R+1){
       currSum -= a[currR-1];
       currR--;
    }
 // Print sum of current range
 cout << "Sum of [" << L << ", " << R << "] is "  << currSum
<< endl;
} //end for loop
}//End queryResults

// Driver program
int main()
{
  int a[] = {1, 1, 2, 1, 3, 4, 5, 2, 8};
  int n = sizeof(a)/sizeof(a[0]);
  Query q[] = {{0, 4}, {1, 3}, {2, 4}};
  int m = sizeof(q)/sizeof(q[0]);
  queryResults(a, n, q, m);
  return 0;
}
```

## **Graph Algorithms :**

### **Dijkstra :**

```
typedef double T;
typedef vector<T> VT;
typedef vector<VT> VVT;
typedef vector<int> VI;
typedef vector<VI> VVI;

// This function runs Dijkstra's algorithm for single source
// shortest paths.  No negative cycles allowed!
//
// Running time: O(|V|^2)
//
//   INPUT:   start, w[i][j] = cost of edge from i to j
//   OUTPUT:  dist[i] = min weight path from start to i
//            prev[i] = previous node on the best path from the
//                      start node

void Dijkstra (const VVT &w, VT &dist, VI &prev, int
start){
  int n = w.size();
  VI found (n);
  prev = VI(n, -1);
  dist = VT(n, 1000000000);
  dist[start] = 0;



while (start != -1){
   found[start] = true;
   int best = -1;
   for (int k = 0; k < n; k++) if (!found[k]){
    if (dist[k] > dist[start] + w[start][k]){
      dist[k] = dist[start] + w[start][k];
      prev[k] = start;
    }
    if (best == -1 || dist[k] < dist[best]) best = k;
   }
   start = best;
  }
}
```

**FloydWarshall :**

```
// This function runs the Floyd-Warshall algorithm for all-
pairs
// shortest paths.  Also handles negative edge weights.
Returns true
// if a negative weight cycle is found.
// Running time: O(|V|^3)
//   INPUT:  w[i][j] = weight of edge from i to j
//   OUTPUT: w[i][j] = shortest path from i to j
//   prev[i][j] = node before j on the best path starting at i

bool FloydWarshall (VVT &w, VVI &prev){
  int n = w.size();  prev = VVI (n, VI(n, -1));

  for (int k = 0; k < n; k++){    for (int i = 0; i < n;
i++){
     for (int j = 0; j < n; j++){  if (w[i][j] > w[i][k] +
w[k][j]){
          w[i][j] = w[i][k] + w[k][j];
          prev[i][j] = k;  }     }     }   }

  // check for negative weight cycles
  for(int i=0;i<n;i++)
   if (w[i][i] < 0) return false;
  return true;
}
```

**Euclidean Path :**

```
struct Edge;
typedef list<Edge>::iterator iter;

struct Edge{
 int next_vertex;
 iter reverse_edge;
Edge(int next_vertex)  :next_vertex(next_vertex)  { }
};
const int max_vertices = ??;
int num_vertices;
list<Edge> adj[max_vertices];   // adjacency list
vector<int> path;
void find_path(int v){
            while(adj[v].size() > 0)  {
            int vn = adj[v].front().next_vertex;
            adj[vn].erase(adj[v].front().reverse_edge);
            adj[v].pop_front();
            find_path(vn);
            }
    path.push_back(v);
}
void add_edge(int a, int b){
  adj[a].push_front(Edge(b));
  iter ita = adj[a].begin();
  adj[b].push_front(Edge(a));
  iter itb = adj[b].begin();
  ita->reverse_edge = itb;
  itb->reverse_edge = ita;
}
```

**Single Source Shortest Path :**

```
struct Node{
  int node, cost;
  // i.e Node o is higher up in priority in the queue
  bool operator<(Node const &o) const {
    return cost > o.cost;  }
};

vector<Node> AdjList[MAXN];
int dist[MAXN];

void ShortestPath(int src){
    Node start,curr,next;
    start.node = src;
    start.cost = 0;
    memset(dist, -1, sizeof(dist));
            // Initialize distance
    priority_queue<Node> q;
            // Initialize queue
    q.push(start);
    while(!q.empty()){
        curr = q.top();
        q.pop();
        if(dist[curr.node] == -1){
            dist[curr.node] = curr.cost;
            for(int i = 0; i < AdjList[curr.node].size();
i++){
                if(dist[AdjList[curr.node][i].node] != -
1) continue;
            next.node = AdjList[curr.node][i].node;
            next.cost = curr.cost +
AdjList[curr.node][i].cost;
            q.push(next);
```

```
        }
    }   } }

scanf("%d %d %d",&x,&y,&c);
AdjList[x].push_back({y,c});
AdjList[y].push_back({x,c});
ShortestPath(src);
```

## Fast Dijkstra :

```
typedef pair<int, int> PII;
vector< vector<PII> > edges(N);
edges[i].push_back(make_pair(dist, vertex));
            // note order of arguments here
   // use priority queue in which top element has the
"smallest" priority
priority_queue<PII, vector<PII>, greater<PII> > Q;
vector<int> dist(N, INF), dad(N, -1);
Q.push(make_pair(0, s));
 dist[s] = 0;

  while (!Q.empty()) {
          PII p = Q.top();
          Q.pop();
          int here = p.second;
          if (dist[here] != p.first) continue;
    //vector<PII>::iterator it
    for (auto it=edges[here].begin();
                    it!=edges[here].end(); it++) {
      if (dist[here] + it->first < dist[it->second]) {
            dist[it->second] = dist[here] + it->first;
            dad[it->second] = here;
           Q.push(make_pair(dist[it->ss], it->ss));
     }
    }       //ss = second
}
```

## BFS :

```
vector<int> Graph[V];
int Color[V]; // {0,-1,1} = {White,Grey,Black}  =
{Unvisited,Discovered,Explored}
int Dist[V],Pre[V];
//Set Dist[],Pre[] = INF , Color[],Dtime[],Ftime[] = 0
```

```
void BFS(int s){
   queue <int> q;    q.push(s);
   Color[s] = -1;   Dist[s]=Pre[s]=0;
   while(!q.empty()){  int u = q.front(); q.pop();
      for(int i=0;i<Graph[u].size();i++){
        int v = Graph[u][i];  if(Color[v]==0){
            Color[v] = -1; Dist[v] = Dist[u]+1;
            Pre[v] = u; q.push(v);  }
      }  Color[u] = 1;
  }
return;
}
```

## DFS :

```
void DFS(); void DFS_VISIT(int u);
list<int> TopoSort;

void DFS(int N){
     for(int u=1;u<=N;u++){
      if(Color[u]==0){
         DFS_VISIT(u);
         }
} }
void DFS_VISIT(int u){
   Time++;
   DTime[u] = Time;    Color[u] = -1;
   for(int i=0;i<Graph[u].size();i++){
      int v = Graph[u][i]; if(Color[v]==0){
           Pre[v]=u; DFS_VISIT(v); }
     }
   Color[u]=1; Time++;
   FTime[u]=Time;  TopoSort.push_front(u);
}
```

## Biparatite Graph

```
vector<int> Graph[VMax];
int Color[VMax];
int Visited[VMax];
```

```
bool isBipartite(int s)
{
    Color[s] = 1;
     queue <int> q;
    q.push(s); Visited[s]=1;
    while (!q.empty()){
            int u = q.front();
            Visited[u]=1;
            q.pop();
            for (int i=0; i<Graph[u].size();i++) {
                int v = Graph[u][i];
                    if (Color[v] == -1){
                    Color[v] = 1 - Color[u];
                    q.push(v);
                    Visited[v]=1;
                }
                    else if (Color[v] == Color[u])
                        return false;
            }
    }
    return true;
}
```

**Kruskal :**

```
//Kruskal
// inside int main()
vector< pair<int, ii> > EdgeList; // (weight, two vertices) of the .
edge
  for (int i = 0; i < E; i++) {
   scanf("%d %d %d", &u, &v, &w);    // read the triple: (u, v, w)
   EdgeList.push_back(make_pair(w, ii(u, v))); }
// (w, u, v)
   sort(EdgeList.begin(), EdgeList.end()); //  sort by edge weight                    .
O(ElogE)
   // note: pair object has built-in comparison function
int mst_cost = 0;
UnionFind UF(V);       // all V are disjoint sets initially
  for (int i = 0; i < E; i++) { // for each edge, O(E)
    pair<int, ii> front = EdgeList[i];
    if (!UF.isSameSet(front.second.first, front.second.second)){
       mst_cost += front.first;          // add the weight of e to MST
     UF.unionSet(front.second.first, front.second.second);
    }
}

// note: the runtime cost of UFDS is very light
// note: the number of disjoint sets must eventually be 1 for a valid MST

printf("MST cost = %d (Kruskal’s)\n", mst_cost);
```

**PRIMS(ADDING VERTICES-SPANNING TREE)--(O(V+E)logV):**

```
#include <iostream>
#include <vector>
#include <queue>
#include <functional>
#include <utility>

using namespace std;
const int MAX = 1e4 + 5;
typedef pair<long long, int> PII;
bool marked[MAX];
vector <PII> adj[MAX];

long long prim(int x){
priority_queue<PII, vector<PII>, greater<PII> > Q;
   int y;
   long long minimumCost = 0;
   PII p;
   Q.push(make_pair(0, x));
   while(!Q.empty())    {
      // Select the edge with minimum weight
      p = Q.top();
      Q.pop();
      x = p.second;
      // Checking for cycle
      if(marked[x] == true)
         continue;
      minimumCost += p.first;
      marked[x] = true;
      for(int i = 0;i < adj[x].size();++i)
      {
         y = adj[x][i].second;
         if(marked[y] == false)
            Q.push(adj[x][i]);
```

```
        }
    }
    return minimumCost;
}

int main(){
    int nodes, edges, x, y;
    long long weight, minimumCost;
    cin >> nodes >> edges;
    for(int i = 0;i < edges;++i){
        cin >> x >> y >> weight;
        adj[x].push_back(make_pair(weight, y));
        adj[y].push_back(make_pair(weight, x));
    }
    // Selecting 1 as the starting node
    minimumCost = prim(1);
    cout << minimumCost << endl;
    return 0;
}
```

**Flood fill :**

```
int dr[] = {1,1,0,-1,-1,-1, 0, 1};
// trick to explore an implicit 2D grid
int dc[] = {0,1,1, 1, 0,-1,-1,-1};
 // S,SE,E,NE,N,NW,W,SW neighbors
int floodfill(int r, int c, char c1, char c2) {
// returns the size of CC
    if (r < 0 || r >= R || c < 0 || c >= C) return 0;
// outside grid
    if (grid[r][c] != c1) return 0;
// does not have color c1
    int ans = 1;
 // adds 1 to ans because vertex (r, c) has c1 as its color
    grid[r][c] = c2;
 // now recolors vertex (r, c) to c2 to avoid .cycling!
    for (int d = 0; d < 8; d++)
      ans += floodfill(r + dr[d], c + dc[d], c1, c2);
    return ans;    // the code is neat due to dr[] and dc[]
}
```

**Connected components**

```
numCC = 0;
dfs_num.assign(V, UNVISITED);
// sets all vertices' state to UNVISITED
for (int i = 0; i < V; i++)
// for each vertex i in [0..V-1]
  if (dfs_num[i] == UNVISITED)
// if vertex i is not visited yet
    printf("CC %d:", ++numCC), dfs(i), printf("\n");
```

**LCA (Least Common Ancestor)**

```
const int max_nodes, log_max_nodes;
int num_nodes, log_num_nodes, root;

vector<int> children[max_nodes];
// children[i] contains the children of node i

// A[i][j] is the 2^j-th ancestor of node i, or -1 if that
ancest or does not exist
int A[max_nodes][log_max_nodes+1];

// L[i] is the distance between node i and the root
// floor of the binary logarithm of n
int L[max_nodes];

int lb(unsigned int n){
    if(n==0)
        return -1;
    int p = 0;
    if (n >= 1<<16) { n >>= 16; p += 16; }
    if (n >= 1<< 8) { n >>= 8; p += 8; }
    if (n >= 1<< 4) { n >>= 4; p += 4; }
    if (n >= 1<< 2) { n >>= 2; p += 2; }
    if (n >= 1<< 1) { p += 1; }
    return p;
}
void DFS(int i, int l){
    L[i] = l;
    for(int j = 0; j < children[i].size(); j++)
        DFS(children[i][j], l+1);
}

int LCA(int p, int q){
    // ensure node p is at least as deep as node q
    if(L[p] < L[q])
        swap(p, q);
    // "binary search" for the ancestor of node p
situated on the same level as q
    for(int i = log_num_nodes; i >= 0; i--)
        if(L[p] - (1<<i) >= L[q])
```

```
        p = A[p][i];
    if(p == q)
        return p;
    // "binary search" for the LCA
    for(int i = log_num_nodes; i >= 0; i--)
        if(A[p][i] != -1 && A[p][i] != A[q][i])  {
            p = A[p][i];
            q = A[q][i];
        }
    return A[p][0];
}

int main(int argc,char* argv[])
{
// read num_nodes, the total number of nodes
log_num_nodes=lb(num_nodes);
for(int i = 0; i < num_nodes; i++)
{
    int p;
    // read p, the parent of node i or -1 if node i is
the root
    A[i][0] = p;
    if(p != -1)

        children[p].push_back(i);
    else
        root = i;
}
// precompute A using dynamic programming
for(int j = 1; j <= log_num_nodes; j++)
    for(int i = 0; i < num_nodes; i++)
        if(A[i][j-1] != -1)
            A[i][j] = A[A[i][j-1]][j-1];
        else
            A[i][j] = -1;
    // precompute L
    DFS(root, 0);
    return 0;
}
```

## LCA using RMQ

```
//LCA(U,V) = MIN(E[H[U]....H[V])

int L[2*MAX_N], E[2*MAX_N], H[MAX_N], idx;
void dfs(int cur, int depth) {
    H[cur] = idx;
    E[idx] = cur;
    L[idx++] = depth;
    for (int i = 0; i < children[cur].size(); i++) {
        dfs(children[cur][i], depth+1);
        E[idx]= cur; // backtrack to current node
        L[idx++]= depth;
    }
}
void buildRMQ(){
    idx= 0;
    memset(H,-1, sizeof H);
    dfs(0, 0);
    // we assume that the root is at index 0
}
```

## Arithematic :

### Modular Multiplication :

```
ull MulMod(ull A,ull B,ull Mod)  (user if doubt.)
{
  ull x=0,y=A%Mod;
   while(B>0)
   {
     if(B&1)
       x=(x+y)%Mod;
     y=(y*2)%Mod;
     B=B/2;
   }
   return x%Mod;
}
```

### Modular Exponent :

```
ll PowerMod(ll Base,ll Exp,ll Mod)
{
   ll Ans=1;
   Base = Base%Mod;
   while(Exp>0)
   {
     if(Exp&1) Ans=MulMod(Ans,Base,Mod);
     Base=MulMod(Base,Base,Mod);
     Exp = Exp>>1LL;
   }
   return Ans;
}
```

**Fermet Prime Test :**

```
bool FermatPrimeTest(ll N,ll Iterations)
{
   if(N==1)
      return 0;
   srand(time(NULL));

for(ll i=0;i<Iterations;i++)
   {
      ll a=rand()%(N-1)+1;
      if(PowerMod(a,N-1,N)!=1)
          return 0;
   }
   return 1;
}
```

---

**Extended Euclid :**

```
int d, x, y;
void extendedEuclid(int A, int B) {
if(B == 0) {
    d = A;
   x = 1;   y = 0;
 }
 else {
        extendedEuclid(B, A%B);
        int temp = x;
        x = y;
        y = temp - (A/B)*y;
 }
}
extendedEuclid(16, 10);
cout << ”The GCD of 16 and 10 is ” << d << endl;
cout << ”Coefficients x and y are ”<< x <<  “and “;
cout << y << endl;
```

---

**Mod Multiplicative Inverse** : (IF M is composite)

```
int d,x,y;
int modInverse(int A, int M) {
        extendedEuclid(A,M);
        return (x%M+M)%M; //x may be negative
}
```

**Miller Rabbin Test :**

```
bool MillerRabinTest(ll N,ll Iterations)
{
  if(N<2) return 0;  if(N==2) return 1;
  if(N%2==0) return 0;
  ll M = N-1;
   while(M%2==0)  M=M/2;
    for(int i=0;i<Iterations;i++){
     ll a=rand()%(N-1)+1 , K=M;
     ll Mod = ModExp(a,K,N);
     while(K!=N-1 && Mod!=1 && Mod!=N-1){
        Mod = MulMod(Mod,Mod,N);
        K=K*2;
     }
     if(Mod!=N-1 && K%2==0)  return 0;
  }
  return 1;
}
```

---

**Pollard Rho Factorization** :

```
   ll pollardRho(ll n) {
        if(n<=1) return -1;
        if(n%2==0)
          return 2 ;
        srand (time(NULL)) ;
        ll x, y , g=1 , a;
        x = rand() % n + 1 ;
        y = x ;
        a = rand() % n + 1 ;
        while(g==1) {
          x = ((x*x) + a)%n ;
          y = ((y*y) + a)%n ;
          y = ((y*y) + a)%n ;
          g = gcd(abs(x - y), n) ;
        }
    return g ; }
```

---

**Highest Prime Factor:**

```
bool Prime[MAX+25];
int HPF[MAX+25]; //Highest Prime Factor
void sieve(){
   int i,j;
   memset(Prime,1,sizeof(Prime));
   Prime[0]=Prime[1]=0;
```

```
  for(i=2;i<=sqrt(MAX);i+=2) Prime[i]=0;
   HPF[i] =2;
  for(i=3;i<=sqrt(MAX);i+=2) {
    if(Prime[i]){
      HPF[i]=i;
        K=i;
      for(j=i*i;j<=MAX;j+=i){
        Prime[j]=0;
      }
    }
  }
}
```

---

**Prime Factorization :**

```
//returns 60 = (2,2)(3,1)(5,1)
auto ExpCal(ll N)
{
   vector< pair<int,int> > Fac;
   ll NN = N,Pow,prime;
   while(NN>1){
      Pow=0;
      prime=HPF[NN];
      while(NN%prime==0){
         NN=NN/prime;
         Pow++;
      }
      Fac.push_back(make_pair(prime,Pow));
   }
   return Fac;
}
```

---

**Small Number Prime Tests :**

```
 bool PrimeI(ll x){
        if(x==2 || x==3) return 1;
        if(x<=1 || x%2==0 || x%3==0) return 0;
        if(x%6!=1 && x%6!=5) return 0;
        ll i=5;
        while(i*i<=x){
          if(x%i==0 || x%(i+2)==0) return 0;
          i+=6;
         }
        return 1;
 }
```

```
bool PrimeII(ll x){
  if(x<=1)return 0;
  if(x<=3)return 1;
  if(x%6==1||x%6==5){
    ll y=sqrt(x);
    for(ll i=2;i<=y;i++)
      if(x%i==0)
      return 0;
    return 1;
  }
  return 0;
}
```

---

```
#define MAX 1000000005
#define setval(a,val) memset(a,val,sizeof(a))
bitset<MAX+25>Prime;

//bool Prime[MAX] gave RunTime error in
//HackerRank but accepted in Codechef and Spoj.
//Have to Check this in Practice Contest.
```

**Sieve of Erathothesis :**

```
void Sieve(){
  int i,j,POS; Prime[0]=Prime[1]=1;
  for(i=2;i<=180;i++)
    if(!Prime[i])
      for(j=2;(POS=j*i)<=32000;j++)
        Prime[POS]=1;
}
```

---

**Segmented Sieve :**

```
void SegmentSieve(int L,int R){
   int lim = sqrt(R);
   int i, j;
   for (i = 2; i <= lim; i++)   {
      if (!Prime[i])      {
         for (j = L - L%i; j <= R; j += i)
            if (j>=L && !Prime[j] && j!=i)
               Prime[j] = 1;
      }
   }
}
If(R>200)
   SegmentSieve(L,R);
```

**Modular Inverse DP:**

```
void DpModInv(int N){
   int i;
   ModInverse[1]=1;
   for(i=2;i<N;i++){
         ModInverse[i]=( -ll)(MOD/i)*
         (ll)ModInverse[MOD%i]) % MOD + MOD;
  }
}
```

---

**Mobius Function Precompute :**

```
vector<int> mobiusMu;
void calcMobiusMu(int n) {
     mobiusMu.assign(n+1, 1);
     for(int i = 2; i <= n; i ++)
      if(isprime[i]){
         if((ll)i * i <= n) {
              for(int j = i * i; j <= n; j += i * i)
                   mobiusMu[j] = 0;
         }
         for(int j = i; j <= n; j += i)
              mobiusMu[j] *= -1;
     }
}
```

---

**Totient Function :**

```
#include <stdlib.h>
// This code took less than 0.5s to calculate with
MAX = 10^7
#define MAX 10000000
int phi[MAX];
bool pr[MAX];
void totient(){
  for(int i = 0; i < MAX; i++){
    phi[i] = i;
    pr[i] = true;
  }

  for(int i = 2; i < MAX; i++)
    if(pr[i]){
      for(int j = i; j < MAX; j+=i){
        pr[j] = false;
        phi[j] = phi[j] - (phi[j] / i);
      }
      pr[i] = true;
    }
}
```

The sum of all values of Totient Function of all divisors of N
is equal to N.

$\varphi(p^k) = p^k - p^{k-1} = p^{k-1}(p - 1)$ // p is prime.

$$\begin{aligned}\varphi(n) &= \varphi(p_1^{k_1})\varphi(p_2^{k_2})\cdots\varphi(p_r^{k_r}) \\ &= p_1^{k_1}\left(1-\frac{1}{p_1}\right)p_2^{k_2}\left(1-\frac{1}{p_2}\right)\cdots p_r^{k_r}\left(1-\frac{1}{p_r}\right) \\ &= p_1^{k_1}p_2^{k_2}\cdots p_r^{k_r}\left(1-\frac{1}{p_1}\right)\left(1-\frac{1}{p_2}\right)\cdots\left(1-\frac{1}{p_r}\right) \\ &= n\left(1-\frac{1}{p_1}\right)\left(1-\frac{1}{p_2}\right)\cdots\left(1-\frac{1}{p_r}\right).\end{aligned}$$

$GCD(A,B) == 1$ then $Totient(A) \times Totient(B) = Totient(A \cdot B)$.

---

**Linear Equation Solver :**

```
// finds all solutions to ax = b (mod n)
VI mod_linear_equation_solver(int a, int b, int n) {
      int x, y;
      VI solutions;
      int d = extended_euclid(a, n, x, y);
      if (!(b%d)) {
      x = mod (x*(b/d), n);
      for (int i = 0; i < d; i++)
          solutions.push_back(mod(x + i*(n/d), n));
  }
  return solutions;
}
```

**Chinese Remainder Theorem 1:**

```
// Chinese remainder theorem (special case): find z such that
// z % x = a, z % y = b.  Here, z is unique modulo M = lcm(x,y).
// Return (z,M).  On failure, M = -1.
PII chinese_remainder_theorem(int x, int a, int y, int b) {
     int s, t;
     int d = extended_euclid(x, y, s, t);
     if (a%d != b%d) return make_pair(0, -1);
  return make_pair(mod(s*b*x+t*a*y,x*y)/d, x*y/d);
}
```

**Chinese Remaider Theorem 2 :**

```
// Chinese remainder theorem: find z such that
// z % x[i] = a[i] for all i.  Note that the solution is
// unique modulo M = lcm_i (x[i]).  Return (z,M).  On
// failure, M = -1.  Note that we do not require the a[i]'s
// to be relatively prime.
PII chinese_remainder_theorem(const VI &x, const VI &a) {
     PII ret = make_pair(a[0], x[0]);
      for (int i = 1; i < x.size(); i++) {
           ret = chinese_remainder_theorem(ret.first,
          ret.second, x[i],   a[i]);
           if (ret.second == -1) break;
     }
  return ret;
}
```

**Longest Common Substring :**

```
int LCSubStr(char *X, char *Y, int m, int n){
   // Create a table to store lengths of longest common suffixes of
   // substrings.   Notethat LCSuff[i][j] contains length of longest
   // common suffix of X[0..i-1] and Y[0..j-1]. The first row and
   // first column entries have no logical meaning, they are used only
   // for simplicity of program
   int LCSuff[m+1][n+1];
   int result = 0;  // To store length of the longest common substring
   /* Following steps build LCSuff[m+1][n+1] in bottom up fashion. */
   for (int i=0; i<=m; i++) {
      for (int j=0; j<=n; j++){
         if (i == 0 || j == 0)
            LCSuff[i][j] = 0;
         else if (X[i-1] == Y[j-1]){
            LCSuff[i][j] = LCSuff[i-1][j-1] + 1;
            result = max(result, LCSuff[i][j]);
         }
         else LCSuff[i][j] = 0;
      }
   }
   return result;
}
```

```
int binomialCoeff(int n, int k)
{
   int C[n+1][k+1];    int i, j;
    for (i = 0; i <= n; i++){
     for (j = 0; j <= min(i, k); j++){
         if (j == 0 || j == i)
            C[i][j] = 1;
         else
            C[i][j] = C[i-1][j-1] + C[i-1][j];
      }
   }
   return C[n][k];
}
```

**Geometry :**

```
double INF = 1e100;
double EPS = 1e-12;

struct PT {
 double x, y;
 PT() {}
 PT(double x, double y) : x(x), y(y) {}
 PT(const PT &p) : x(p.x), y(p.y)    {}
 PT operator + (const PT &p)  const { return PT(x+p.x, y+p.y); }
```

```
 PT operator - (const PT &p)  const { return PT(x-
p.x, y-p.y); }
 PT operator * (double c)     const { return PT(x*c,
y*c  ); }
 PT operator / (double c)     const { return PT(x/c,
y/c  ); }
};

double dot(PT p, PT q)     { return p.x*q.x+p.y*q.y; }
double dist2(PT p, PT q)   { return dot(p-q,p-q); }
double cross(PT p, PT q)   { return p.x*q.y-p.y*q.x; }
ostream &operator<<(ostream &os, const PT &p) {
  os << "(" << p.x << "," << p.y << ")";
}


// rotate a point CCW or CW around the origin
PT RotateCCW90(PT p)   { return PT(-p.y,p.x); }
PT RotateCW90(PT p)    { return PT(p.y,-p.x); }
PT RotateCCW(PT p, double t) {
  return PT(p.x*cos(t)-p.y*sin(t),
p.x*sin(t)+p.y*cos(t));
}

// project point c onto line through a and b
// assuming a != b
PT ProjectPointLine(PT a, PT b, PT c) {
  return a + (b-a)*dot(c-a, b-a)/dot(b-a, b-a);
}

// project point c onto line segment through a and
b
// if the projection doesn't lie on the segment,
returns closest vertex
PT ProjectPointSegment(PT a, PT b, PT c) {
        double r = dot(b-a,b-a);
         if (fabs(r) < EPS) return a;
         r = dot(c-a, b-a)/r;
         if (r < 0) return a;
         if (r > 1) return b;
         return a + (b-a)*r;
}

// compute distance from c to segment between a
and b
double DistancePointSegment(PT a, PT b, PT c) {
  return sqrt(dist2(c, ProjectPointSegment(a, b, c)));
}

// determine if lines from a to b and c to d are
parallel or collinear
bool LinesParallel(PT a, PT b, PT c, PT d) {
  return fabs(cross(b-a, c-d)) < EPS;
}

bool LinesCollinear(PT a, PT b, PT c, PT d) {
  return LinesParallel(a, b, c, d)
      && fabs(cross(a-b, a-c)) < EPS
      && fabs(cross(c-d, c-a)) < EPS;
}

// determine if line segment from a to b intersects
with
// line segment from c to d
bool SegmentsIntersect(PT a, PT b, PT c, PT d) {
  if (LinesCollinear(a, b, c, d)) {
    if (dist2(a, c) < EPS || dist2(a, d) < EPS ||
      dist2(b, c) < EPS || dist2(b, d) < EPS) return true;
    if (dot(c-a, c-b) > 0 && dot(d-a, d-b) > 0 && dot(c-
b, d-b) > 0)
      return false;
    return true;
  }
  if (cross(d-a, b-a) * cross(c-a, b-a) > 0) return
false;
  if (cross(a-c, d-c) * cross(b-c, d-c) > 0) return false;
  return true;
}

// determine if c and d are on same side of line
passing through a and b
bool OnSameSide(PT a, PT b, PT c, PT d) {
  return cross(c-a, c-b) * cross(d-a, d-b) > 0;
}

// compute center of circle given three points
PT ComputeCircleCenter(PT a, PT b, PT c) {
  b=(a+b)/2;
  c=(a+c)/2;
  return ComputeLineIntersection(b,
b+RotateCW90(a-b), c, c+RotateCW90(a-c));
}
```

```
bool PointInPolygon(const vector<PT> &p, PT q) {
  bool c = 0;
  for (int i = 0; i < p.size(); i++){
    int j = (i+1)%p.size();
    if ((p[i].y <= q.y && q.y < p[j].y ||
      p[j].y <= q.y && q.y < p[i].y) &&
      q.x < p[i].x + (p[j].x - p[i].x) * (q.y - p[i].y) /
(p[j].y - p[i].y))
      c = !c;
  }
  return c;
}

// determine if point is on the boundary of a
polygon
bool PointOnPolygon(const vector<PT> &p, PT q)
{
  for (int i = 0; i < p.size(); i++)
    if (dist2(ProjectPointSegment(p[i],
p[(i+1)%p.size()], q), q) < EPS)
      return true;
    return false;
}


// compute intersection of circle centered at a with
radius r
// with circle centered at b with radius R
vector<PT> CircleCircleIntersection(PT a, PT b,
double r, double R) {
  vector<PT> ret;
  double d = sqrt(dist2(a, b));
  if (d > r+R || d+min(r, R) < max(r, R)) return ret;
  double x = (d*d-R*R+r*r)/(2*d);
  double y = sqrt(r*r-x*x);
  PT v = (b-a)/d;
  ret.push_back(a+v*x + RotateCCW90(v)*y);
  if (y > 0)
    ret.push_back(a+v*x - RotateCCW90(v)*y);
  return ret;
}
```

**Convex Hull :**

```
// A C++ program to find convex hull of a set of points.
Refer
// http://www.geeksforgeeks.org/orientation-3-
ordered-points/
// for explanation of orientation()
#include <iostream>
#include <stack>
#include <stdlib.h>
using namespace std;

struct Point{  int x, y; };
 // A globle point needed for  sorting points with
reference
// to  the first point Used in compare function of
qsort()
Point p0;
// A utility function to find next to top in a stack
Point nextToTop(stack<Point> &S){
   Point p = S.top();     S.pop();
   Point res = S.top();   S.push(p);
   return res;
}

// A utility function to swap two points
int swap(Point &p1, Point &p2){
   Point temp = p1;  p1 = p2;
   p2 = temp;
}
// A utility function to return square of distance
// between p1 and p2
int distSq(Point p1, Point p2){
   return (p1.x - p2.x)*(p1.x - p2.x) +
        (p1.y - p2.y)*(p1.y - p2.y);
}
// To find orientation of ordered triplet (p, q, r).
// The function returns following values
// 0 --> p, q and r are colinear
// 1 --> Clockwise
// 2 --> Counterclockwise
int orientation(Point p, Point q, Point r){
   int val = (q.y - p.y) * (r.x - q.x) -
          (q.x - p.x) * (r.y - q.y);
```

```
  if (val == 0) return 0;  // colinear
  return (val > 0)? 1: 2; // clock or counterclock wise
}
 // A function used by library function qsort() to sort
an array of
// points with respect to the first point
int compare(const void *vp1, const void *vp2){
  Point *p1 = (Point *)vp1;
  Point *p2 = (Point *)vp2;
  // Find orientation
  int o = orientation(p0, *p1, *p2);
  if (o == 0)
   return (distSq(p0, *p2) >= distSq(p0, *p1))? -1 : 1;
  return (o == 2)? -1: 1;
}
// Prints convex hull of a set of n points.
void convexHull(Point points[], int n){
  // Find the bottommost point
  int ymin = points[0].y, min = 0;
  for (int i = 1; i < n; i++){
   int y = points[i].y;
   // Pick the bottom-most or chose the left
   // most point in case of tie
   if ((y < ymin) || (ymin == y &&
       points[i].x < points[min].x))
      ymin = points[i].y, min = i;
  }
  // Place the bottom-most point at first position
  swap(points[0], points[min]);
  // Sort n-1 points with respect to the first point.
  // A point p1 comes before p2 in sorted ouput if p2
  // has larger polar angle (in counterclockwise
  // direction) than p1
  p0 = points[0];
  qsort(&points[1], n-1, sizeof(Point), compare);
  // If two or more points make same angle with p0,
  // Remove all but the one that is farthest from p0
  // Remember that, in above sorting, our criteria was
  // to keep the farthest point at the end when more
than
  // one points have same angle.
  int m = 1; // Initialize size of modified array
  for (int i=1; i<n; i++){
     // Keep removing i while angle of i and i+1 is same
     // with respect to p0
     while (i < n-1 && orientation(p0, points[i],
                          points[i+1]) == 0)
       i++;
     points[m] = points[i];
     m++;  // Update size of modified array
  }
  // If modified array of points has less than 3 points,
  // convex hull is not possible
  if (m < 3) return;
  // Create an empty stack and push first three points
  // to it.
  stack<Point> S;
  S.push(points[0]);
  S.push(points[1]);
  S.push(points[2]);
  // Process remaining n-3 points
  for (int i = 3; i < m; i++){
    // Keep removing top while the angle formed by
    // points next-to-top, top, and points[i] makes
    // a non-left turn
    while (orientation(nextToTop(S), S.top(), points[i])
!= 2)
       S.pop();
    S.push(points[i]);
  }
  // Now stack has the output points, print contents
of stack
  while (!S.empty()){
     Point p = S.top();
     cout << "(" << p.x << ", " << p.y <<")" << endl;
     S.pop();
  }
}
// Driver program to test above functions
int main(){
   Point points[] = {{0, 3}, {1, 1}, {2, 2}, {4, 4},
                {0, 0}, {1, 2}, {3, 1}, {3, 3}};
   int n = sizeof(points)/sizeof(points[0]);
   convexHull(points, n);
   return 0;}
```

**Utilities :**

queue < type > q;
1. q.push(element)  2. q.pop()
3. q.front()  4. q.back()
5. q.empty()(!)  6. q.size()
7. q.swap(q2)

stack <int> sk;
1. sk.empty() (!)  2. sk.pop()
3. sk.push(element)  4. sk.top()
5. sk.size()  6. sk.swap(sk2)

priority_queue < type , vector < type > , cmp > pq;

```
// cmp :     //opposite
struct cmp {
    bool operator()(const type &aa,const type &bb){
            return aa.ff < b.ff;
        }
};
```

1. pq.push(element)  2. pq.pop()
3. pq.top()  4. pq.empty() (!)
5. pq.size()  6. pq.swap(pq2)

1. stoi(string)  //string to integer
2. to_string(integer)  // integer to string
3. // normal :
   fill(a,0); fill(a,-1); fill(a,0x3f); //INF
4. //bool :
   fill(a,0); fill(a,1);
5. set_intersection(first.begin(),first.end(),second.begin(),s econd.end(),inserter(ans,ans.begin()));
6. // erase from back
minheap.erase(--minheap.end()); // c++ 4.3.2
minheap.erase(std::prev(minheap.end())); // c++11
7. setbase , setfill , setw , setprecision

8.numeric limits:
   numeric_limits<int>::max()
   numeric_limits<int>::min() ... etc..

string s;
1. operator " + "  2. operator " = "
3. operator " [] "  4. s.size()
5. s.substr(pos)  s.substr(pos,length)
6. s.swap(s2)  7. s.find(string) s.find(string,pos)

8. s.replace(pos,len,string)  9. we can sort a string
char array to string --> string str(ch_arr);
string to char array --> str.c_str()

vector < type > vi;
1. vi.assign(val,no.of)  vi.assign(it first,it last)
2. vi.clear()  vi.assign(arr , arr+N)
3. vi.empty() (!)  4. vi.push_back()
5. vi.pop_back()
6. vi.erase(it)  vi.erase(it first,it last)
7. vi.front()  8. vi.back()
9. vi.insert(it pos,val,no.of)  vi.insert(it pos,it first,itlast)
   vi.insert(it pos,arr, arr+N)
10. operator " = " ( vector1 = vector2 )
11. operator " [] " (vi[i]) 12. vi.begin()
13. vi.end() (!)  vector<typ>::iterator it
14. vi.rbegin()  vector<typ>::reverse_iterator it
15. vi.rend()  16. vi.size()
17. vi.swap(vii)  18. lower_bound(all(vi),element)
19. upper_bound(all(vi),element) 20. vi.find()
21. set<int> uniq(vi.begin(),vi.end()) // uniq will contain unique elements

```
// comparision :
bool cmp(type aa,type bb){
    return aa.ff < bb.ff;      //direct
}
```

map < key , value > mp;  //different elements
multimap < key , value > mmp;  //same elements
unordered_map < key , value > u_mp;
unordered_multimap < key , value > um_mp;

1. operator " = "  2. operator " [] "
3. mp.begin()  4. mp.end()  // same to rbegin() and rend()
5. mp.insert(element)  6. mp.empty() (!)
7. mp.size()  8. mp.erase(it)
   mp.erase(it first,it last)
9. mp.clear()  10. mp.count(element)
11. mp.find(element)  // != mp.end()

12. mp.lower_bound(element) // returns iterators
13. mp.upper_bound(element) 14. mp.swap(mp2)

---

```
set < type , cmp > st;
multiset < type , cmp > mst;
unordered_set < type , cmp > u_st;
unordered_multiset < type , cmp > um_st;
```

1. st.begin() 2. st.end() (!)
3. st.size() 4. st.empty()
5. st.find() // != st.end() 6. st.insert(type)
7. st.erase(it) st.erase(it first,it last)
8. st.lower_bound(element) O(logN)
9. st.upper_bound(element) O(logN)
10 st.swap(st2) 11. st.clear()
12. st.count(element)

```
// cmp :
    struct cmp {
        bool operator()(const int &aa, const int &bb)
{
            return aa < bb;
        }
    };
```

//inbuit cmp functions.
greater<int>()
ex: sort (arr, arr+5, std::greater<int>())
less<int>()

deque < type > dq;
1. operator " = "
2. operator " [] "
3. dq.clear()
4. dq.size()
5. dq.begin()
6. dq.end()
7. dq.erase(it) dq.erase(it first,it last)
8. dq.push_back(element)
9. dq.pop_back()
10. dq.push_front(element)
11. dq.pop_front()
12. dq.empty()

**Set a bit**

Use the bitwise OR (|) operator.
number |= 1 << x; // Sets bit x

**Clear a bit**

Use the bitwise AND (&) and NOT (~) operators.
number &= ~(1 << x); // Clears bit x
number &= !(1 << x); // Also works

**Toggle a bit**

Use the bitwise XOR (^) operator.
number ^= 1 << x; // Toggles bit x

**Check a bit**

Use the bitwise AND (&) operator.
bit = number & (1 << x); // Store value of bit x

```
#define isOn(S, j)   (S & (1 << j))
#define setBit(S, j)      (S |= (1 << j))
#define clearBit(S, j)  (S &= ~(1 << j))
#define toggleBit(S, j)      (S ^= (1 << j))
#define lowBit(S)  (S & (-S))
#define setAll(S, n)     (S = (1 << n) - 1)
#define modulo(x, N) ((x) & (N - 1))
// returns x % N, where N is a power of 2

#define isPowerOfTwo(x)    ((x & (x - 1)) == 0)
#define nearestPowerOfTwo(x)
 ((int)pow(2.0, (int)((log((double)x) / log(2.0)) +
0.5)))
```

**String Tokenizer :**

```
vector<string> makeStringTokens (string s, string
div) {
  vector<string> items;
 //vector<int> items; for int ex : "1,2,3,4"
  int i=0, j;
  //items.push_back(atoi(s.substr(i, j-i).c_str())); for
integers.
for (i=s.find_first_not_of(div, i); i!=-1;
             i=s.find_first_not_of(div, i=j)) {
     j = s.find_first_of(div, i);
     items.push_back(s.substr(i, j-i));
  }
  return items;
}
```

setbase - cout << setbase (16); cout << 100 << endl;
---- Prints 64

setfill - cout << setfill ('x') << setw (5); cout << 77 << endl;
----- Prints xxx77
setprecision - cout << setprecision (4) << f << endl;
-----Prints x.xxxx

- **Java :** **Big Integer**

```
import java.math.BigInteger
BigInteger sum = BigInteger.ZERO;   //zero Constant
BigInteger two = new BigInteger("2");
sum = sum.add(V);                 //addition
BigInteger x = BigInteger(sc.next(), b);
// the second parameter is base.

BigInteger gcd_pq = p.gcd(q);  //p and q are also
                               //BigIntegers
```

**Methods :**

| | | |
|---|---|---|
| abs(x) | clearBit(p) | modInverse(m) |
| add(x) | compareTo(x) | modPower(e,m) |
| and(x) | divide(x) | nextProbablePrime(x,Random) |
| bitCount(x) | gcd(x) | probablePrime(x,Random r) |
| bitLength(x) | mod(m) | |

**Python :**

**Fast IO :**

```
from sys import stdin,stdout

T = int(stdin.readline())
n,q = map(int,stdin.readline().split())
stdout.write(str(ans)+" ")
stdout.write("\n")

Int('string',base)    oct(number)    hex(number)
    bin(number) isinstance(data/variable,datatype)
```

Catalan numbers :
Cat(m) = ((2m×(2m−1))/( (m+1)×m))× Cat(m − 1).

```
Import  itertools
print (list((itertools.product("abc",repeat=3))))
('a', 'a', 'a'), ('a', 'a', 'b'), ('a', 'a', 'c'), ('a', 'b', 'a')……..]

print (list((itertools.permutations("abc",r=2))))
[('a', 'b'), ('a', 'c'), ('b', 'a'), ('b', 'c'), ('c', 'a'), ('c', 'b')]
```

Similarily,

```
combinations('ABCD', 2)
combinations_with_replacement('ABCD', 2)
```

**Dynamic Programming :**

**Knuth-Morris-Prath algorithm**

```
#define MAX_N 100010
char T[MAX_N], P[MAX_N];
// T = text, P = pattern
int b[MAX_N], n, m;
// b = back table, n = length of T, m = length of P
// call this before calling kmpSearch()
void kmpPreprocess() {
int i = 0, j = -1; b[0] = -1; // starting values
  while (i < m) { // pre-process the pattern string P
  while (j >= 0 && P[i] != P[j]) j = b[j];
            // different, reset j using b
   i++; j++; // if same, advance both pointers
  b[i] = j; // observe i=8, 9, 10,11,12,13 with j = 0, 1, 2, 3, 4,
  5
  }
} // in the example of P = "SEVENTY SEVEN" above


void kmpSearch() {
      // this is similar as kmpPreprocess(), but on string T
      int i = 0, j = 0; // starting values
      while (i < n) { // search through string T
      while (j >= 0 && T[i] != P[j]) j = b[j];
      // different, reset j using b
      i++; j++; // if same, advance both pointers
       if (j == m) {
      // a match found when j == m
      printf("P is found at index %d in T\n", i - j);
          j = b[j]; // prepare j for the next possible match
   }
  }
}
```

**Longest Common Prefix**

```
void computeLCP() {
  int i, L;
  Phi[SA[0]] = -1; // default value
  for (i = 1; i < n; i++) // compute Phi in O(n)
     Phi[SA[i]] = SA[i-1];
```

```
// remember which suffix is behind this suffix

for (i = L = 0; i < n; i++) { //Compute LCP in O(n)
        if (Phi[i] == -1) { PLCP[i] = 0; continue; }
                              // special case
        while (T[i + L] == T[Phi[i] + L]) L++;
                              // L increased max n times
        PLCP[i] = L;
        L = max(L-1, 0); // L decreased max n times
   }
   for (i = 0; i < n; i++) // compute LCP in O(n)
        LCP[i] = PLCP[SA[i]];
        // put the permuted LCP to the correct position
}
```

**Rectangle Sum DP**

```
void BuildSum(){
     for(int i = 1; i <= NAX; i++)
       for(int j = 1; j <= NAX; j++)
            Sum[i][j]+=Sum[i][j-1]+Arr[i][j];
     for(int j=1;j<=NAX;j++)
       for(int i=1;i<=NAX;i++)
          Sum[i][j]=Sum[i-1][j]+Sum[i][j];
    return;
}
```

**Sum of Rectangle (x1,y1) and (x2,y2)** = Sum[x2][y2] - Sum[x1-1][y2] - Sum[x2][y1-1] + Sum[x1-1][y1-1].

**Stable Marriage Problem:**

```
// Gale-Shapley algorithm for the stable marriage problem.
// madj[i][j] is the jth highest ranked woman for man i.
// fpref[i][j] is the rank woman i assigns to man j.
// Returns a pair of vectors (mpart, fpart), where mpart[i]
gives the partner of man i, and fpart is analogous

pair<vector<int>, vector<int> > stable_marriage
(vector<vector<int> >& madj, vector<vector<int> >&
fpref) {
     int n = madj.size();
     vector<int> mpart(n, -1), fpart(n, -1);
     vector<int> midx(n);
     queue<int> mfree;
     for (int i = 0; i < n; i++) {
          mfree.push(i);
     }
     while (!mfree.empty()) {
          int m = mfree.front(); mfree.pop();
          int f = madj[m][midx[m]++];
          if (fpart[f] == -1) {
               mpart[m] = f; fpart[f] = m;
          } else if (fpref[f][m] < fpref[f][fpart[f]]) {
               mpart[fpart[f]] = -1; mfree.push(fpart[f]);
               mpart[m] = f; fpart[f] = m;
          } else {
               mfree.push(m);
          }
     }
     return make_pair(mpart, fpart);
}
```

**Longest Alternating Sequence :**

```
#include<bits/stdc++.h>
using namespace::std;
typedef long long ll;
ll N;
ll Arr[6000];
ll lis[6000];
ll ALTSEQ(ll Arr[], ll N){
   ll i, j, Max = 0;
   for (i = 0; i < N; i++ )
      lis[i] = 1;
   for (i = 1; i < N; i++ )
      for (j = 0; j < i; j++ )
         if ( abs(Arr[i]) > abs(Arr[j]) && lis[i] < lis[j] + 1
                   && 1LL*Arr[i]*Arr[j] < 0 )
            lis[i] = lis[j] + 1;
   for (i = 0; i < N; i++ )
      if (Max < lis[i])
         Max = lis[i];
   return Max;
}
```

**Input :** 1 2 -2 -3 5 -7 -8 10

**Output :** 5  (1 -2 5 -8 10)

**Kadane Algorithm :**

```
#include<iostream>
using namespace std;
int maxSubArraySum(int a[], int size){
    int max_so_far = a[0];
    int curr_max = a[0];
    int curStartIndex = 0;
    int maxStartIndex;
    int maxEndIndex;
    for (int i = 1; i < size; i++){
        curr_max = max(a[i], curr_max+a[i]);
        if(curr_max > max_so_far){
            max_so_far = curr_max;
            maxStartIndex = curStartIndex;
            maxEndIndex = i;
        }
    if(curr_max < 0){
        curStartIndex = i+1;
    }
}
    cout << "Max Subarray starts at : " <<
    maxStartIndex << " and ends at :" << maxEndIndex
    << " and sum is : " << max_so_far << endl;
    return max_so_far;
}
```

**Longest Increasing SubSequence (N^2) DP:**

```
int increasingSubsequece(int n) {
  int maxLength = 1, bestEnd = 0;
  dp[0] = 1;
  prev[0] = -1;
  for (int i=1; i<n; ++i) {
    dp[i] = 1;
    prev[i] = -1;
    for (int j=i-1; j>=0; --j) {
      if (dp[j] + 1 > dp[i] && a[j] < a[i]) {
        dp[i] = dp[j] + 1;
        prev[i] = j;
      }
    }
    if (dp[i] > maxLength) {
      bestEnd = i;
      maxLength = dp[i];
    }
  }
  // constructing the longest increasing subsequence
from prev[] array
  /*
    vector <int> lis;
    for (int j=0; j<maxLength; ++j)
    {
      lis.push_back(a[bestEnd]);
      bestEnd = prev[bestEnd];
    }
    reverse(lis.begin(), lis.end());
    for (int i=0; i<lis.size(); ++i)
      printf("%d ", lis[i]);
    printf("\n");
  */
  return maxLength;
}
```

**Z- Algorithm :** O(N)

Pattern : “abc”

Text : “xabcabzabc”

```
vector<int> z_function(string s, int n) {
   vector<int> z(n);
   for (int i=1, l=0, r=0; i<n; ++i) {
      if (i <= r) {
         z[i] = min(r-i+1, z[i-l]);
      }
      while (i+z[i]<n && s[z[i]] == s[i+z[i]]) {
         ++z[i];
      }
      if (i+z[i]-1 > r) {
         l = i, r = i+z[i]-1;
      }
   }
   return z;
}
z = z_function("abc$xabcabzabc",14);
z = [0 0 0 (0) 0 3 0 0 2 0 0 3 0 0]
```

where len(pat) = z[i] , it matches from that position.

**Manacher’s Algorithm [Longest Palindrome Substr]**

```
#include <stdio.h>
#include <string.h>
char text[100];
```

```
void findLongestPalindromicString(){
        int N = strlen(text);
        if(N == 0) return;
        N = 2*N + 1; //Position count
        int L[N]; //LPS Length Array
        L[0] = 0; L[1] = 1;
        int C = 1; //centerPosition
        Int R = 2; //centerRightPosition
        int i = 0; //currentRightPosition
        int iMirror; //currentLeftPosition
        int expand = -1,diff = -1;
        int maxLPSLength = 0;
        int MaxLPSCenterPosition = 0;
        int start = -1,end = -1;
   //Uncomment it to print LPS Length array
   //printf("%d %d ", L[0], L[1]);
            for (i = 2; i < N; i++) {
      //get currentLeftPosition iMirror for
currentRightPosition i
      iMirror  = 2*C-i;
//Reset expand - means no expansion required
      expand = 0; diff = R - i;
//If currentRightPosition i is within
//centerRightPosition R
      if(diff > 0) {
        if(L[iMirror] < diff) // Case 1
          L[i] = L[iMirror];
        else if(L[iMirror] == diff && i == N-1)
          L[i] = L[iMirror];
        else if(L[iMirror] == diff && i < N-1)
        {
            L[i] = L[iMirror];
            expand = 1;  // expansion required
        }
        else if(L[iMirror] > diff)
         {
          L[i] = diff;
          expand = 1;  // expansion required
        }
      }
      else{
        L[i] = 0;
        expand = 1;  // expansion required
      }
//Attempt to expand palindrome centered at
currentRightPosition i
//Here for odd positions, we compare characters and
//if match then increment LPS Length by ONE
//If even position, we just increment LPS by ONE
without
//any character comparison
        if (expand == 1){
          while (((i + L[i]) < N && (i - L[i]) > 0) &&
            ( ((i + L[i] + 1) % 2 == 0) ||
            (text[(i + L[i] + 1)/2] == text[(i-L[i]-1)/2] )))
          {
            L[i]++;
          }
        }

        if(L[i] > maxLPSLength){// Track maxLPSLength
          maxLPSLength = L[i];
          maxLPSCenterPosition = i;
        }
// If palindrome centered at currentRightPosition i
// expand beyond centerRightPosition R,adjust
// centerPosition C based on expanded palindrome.
        if (i + L[i] > R){
          C = i;
          R = i + L[i];
        }
        //Uncomment it to print LPS Length array
        //printf("%d ", L[i]);
      }
      //printf("\n");
      start = (maxLPSCenterPosition - maxLPSLength)/2;
      end = start + maxLPSLength - 1;
      //printf("start: %d end: %d\n", start, end);
      printf("LPS of string is %s : ", text);
      for(i=start; i<=end; i++)
        printf("%c", text[i]);
      printf("\n");
}
```