1 DP

1.1 0-1 Knapsack (DP)

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

int main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(nullptr);

  int N, K;
  cin >> N >> K;

  // weight, value.
  vector<int> W(N+1);
  vector<int> V(N+1);

  for (int i = 1; i <= N; i++)
    cin >> W[i] >> V[i];

  vector<vector<int>> dp(N+1, vector<int>(K+1));

  for (int i = 1; i <= N; i++)
  {
    for (int j = 1; j <= K; j++)
    {
      // the value when the current item is ignored.
      dp[i][j] = dp[i-1][j];

      // the value when the current item is taken.
      if (W[i] <= j)
        dp[i][j] = max(dp[i][j], dp[i-1][j-W[i]] + V[i]);
    }
  }

  cout << dp[N][K] << '\n';
  return 0;
}
```

1.2 0-1 Knapsack (Recursive)

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

int knapsack(int i, int j, vector<int>& W, vector<int>& V,
vector<vector<int>>& dp)
{
  if (i == 0)
    return 0;

  if (dp[i][j] != -1)
    return dp[i][j];

  int __cache = knapsack(i-1, j, W, V, dp);

  if (W[i] <= j)
    __cache = max(__cache, knapsack(i-1, j-W[i], W, V, dp) + V[i]);

  dp[i][j] = __cache;
  return __cache;
}

int main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(nullptr);

  int N, K;
  cin >> N >> K;

  // weight, value.
  vector<int> W(N+1);
  vector<int> V(N+1);

  for (int i = 1; i <= N; i++)
    cin >> W[i] >> V[i];

  vector<vector<int>> dp(N+1, vector<int>(K+1, -1));
  cout << knapsack(N, K, W, V, dp) << '\n';
  return 0;
}
```

1.3 Berlekamp-Massey

```cpp
#include <iostream>
#include <vector>
#include <random>
#include <tuple>
using namespace std;
using ll = long long;

const ll mod = 1000000007;

ll ipow(ll x, ll p) {
  ll ret = 1, piv = x;
  while (p) {
    if (p & 1) ret = ret * piv % mod;
```

```
    piv = piv * piv % mod;
    p >>= 1;
  }
  return ret;
}
vector<ll> berlekamp_massey(vector<ll> x) {
  vector<ll> ls, cur;
  ll lf, ld;
  for (size_t i = 0; i < x.size(); i++) {
    ll t = 0;
    for (size_t j = 0; j < cur.size(); j++) {
      t = (t + 1ll * x[i - j - 1] * cur[j]) % mod;
    }
    if ((t - x[i]) % mod == 0) continue;
    if (cur.empty()) {
      cur.resize(i + 1);
      lf = i;
      ld = (t - x[i]) % mod;
      continue;
    }
    ll k = -(x[i] - t) * ipow(ld, mod - 2) % mod;
    vector<ll> c(i - lf - 1);
    c.push_back(k);

    for (auto& j : ls) c.push_back(-j * k % mod);

    if (c.size() < cur.size()) c.resize(cur.size());

    for (size_t j = 0; j < cur.size(); j++) {
      c[j] = (c[j] + cur[j]) % mod;
    }

    if (i - lf + (int)ls.size() >= (int)cur.size()) {
      tie(ls, lf, ld) = make_tuple(cur, i, (t - x[i]) % mod);
    }

    cur = c;
  }
  for (auto& i : cur) i = (i % mod + mod) % mod;
  return cur;
}
ll get_nth(vector<ll> rec, vector<ll> dp, ll n) {
  ll m = rec.size();
  vector<ll> s(m), t(m);
  s[0] = 1;
  if (m != 1) t[1] = 1;
  else t[0] = rec[0];
  auto mul = [&rec](vector<ll> v, vector<ll> w) {
    int m = v.size();
    vector<ll> t(2 * m);
    for (ll j = 0; j < m; j++) {
      for (ll k = 0; k < m; k++) {
        t[j + k] += 1ll * v[j] * w[k] % mod;
        if (t[j + k] >= mod) t[j + k] -= mod;
      }
    }
    for (ll j = 2 * m - 1; j >= m; j--) {
      for (ll k = 1; k <= m; k++) {
        t[j - k] += 1ll * t[j] * rec[k - 1] % mod;
        if (t[j - k] >= mod) t[j - k] -= mod;
      }
    }
    t.resize(m);
    return t;
  };
  while (n) {
    if (n & 1) s = mul(s, t);
    t = mul(t, t);
    n >>= 1;
  }
  ll ret = 0;
  for (ll i = 0; i < m; i++) ret += 1ll * s[i] * dp[i] % mod;
  return ret % mod;
}

int guess_nth_term(vector<ll> x, size_t n) {
  if (n < x.size()) return x[n];
  vector<ll> v = berlekamp_massey(x);
  if (v.empty()) return 0;
  return get_nth(v, x, n);
}
```

1.4 Li-Chao Tree

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

#define X first
#define Y second
#define all(v) v.begin(),v.end()

using namespace std;

typedef long long ll;

const ll inf = 2e18;
```

```
struct Line{
  ll a, b;
  ll get(ll x){
    return a * x + b;
  }
};

struct Node{
  int l, r; //child
  ll s, e; //range
  Line line;
};

struct Li_Chao{
  vector<Node> tree;

  void init(ll s, ll e){
    tree.push_back({ -1, -1, s, e, { 0, -inf } });
  }

    void update(int node, Line v){
  ll s = tree[node].s, e = tree[node].e;
  ll m = s + e >> 1;

  Line low = tree[node].line, high = v;
  if (low.get(s) > high.get(s)) swap(low, high);

      if (low.get(e) <= high.get(e)){
        tree[node].line = high; return;
      }

      if (low.get(m) < high.get(m)){
        tree[node].line = high;
        if (tree[node].r == -1){
          tree[node].r = tree.size();
          tree.push_back({ -1, -1, m + 1, e, { 0, -inf } });
        }
        update(tree[node].r, low);
      }
      else{
        tree[node].line = low;
        if (tree[node].l == -1){
          tree[node].l = tree.size();
          tree.push_back({ -1, -1, s, m, { 0, -inf } });
        }
        update(tree[node].l, high);
      }
    }

    ll query(int node, ll x){
        if (node == -1) return -inf;
        ll s = tree[node].s, e = tree[node].e;
        ll m = s + e >> 1;
        if (x <= m) return max(tree[node].line.get(x), query(tree[node].l,
x));
        else return max(tree[node].line.get(x), query(tree[node].r, x));
    }
};

int main()
{
    ios::sync_with_stdio(false);
    cin.tie(0); cout.tie(0);

    Li_Chao seg;
    seg.init(-2e12,2e12);
    int q; cin>>q;
    while(q--)
    {
        int op; cin>>op;
        if(op==1)
        {
            int a; ll b; cin>>a>>b;
            seg.update(0,{(ll)a,b});
        }
        else
        {
            ll x; cin>>x;
            cout<<seg.query(0,x)<<'\n';
        }
    }
    return 0;
}
//세그먼트 트리+직선의 방성식
//간선 삭제가 없는 컨벡스 헐 트릭의 온라인 쿼리
//주어진 코드는 직선의 방정식이 여러개 주어질 때 x좌표 고정시 y좌표의 최댓값
```

1.5 Longeset Increasing Subsequence

```
#include <bits/stdc++.h>
#define ref(i,n) for(int i = 0; i < n; i++)
using namespace std;

int main(){
```

```
    int N;    cin >> N;
    vector<int> arr(N,0);
    vector<int> dp;
    for(int i = 0;i<N; i++) cin >> arr[i];
    vector<int> index;
    for(auto x : arr){ //dp에서 x가처음나오는위치 이분탐색
        auto it = lower_bound(dp.begin(),dp.end(),x);
        index.push_back(it - dp.begin());//x가부분순열에서
        if(it == dp.end()) {//       위치하는 인덱스(길이-1)
            dp.push_back(x);//만약에dp끝이면pushback.
        } else{            //아니면 바꾸기.
            *it = x;
        }
    }
    int ct = dp.size();//LIS길이
    cout << ct-- <<"\n";vector<int> result;
    for(int i = N-1; i >= 0; i--){
        if(ct == index[i]) {
            result.push_back(arr[i]);
            ct--;
        }
    } //인덱스 배열을 뒤에서부터 보면서 수열복원
    reverse(result.begin(),result.end());
    for(auto x : result) cout << x << " ";
    return 0;
}
```

2 Geometry

2.1 CCW Intersection Check, Convex Hull

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef pair<int, int> pi;

static inline int __reduce(ll a)
{
  if (a > 0)
    return 1;

  if (a < 0)
    return -1;

  return 0;
}

// integer square function.
static inline ll __isq(int a)
{
  return (ll) a*a;
}

// vector subtraction.
static inline pi __pi_sub(const pi& a, const pi& b)
{
  return pi(a.first-b.first, a.second-b.second);
}

// vector cross product.
static inline ll __pi_cross(const pi& a, const pi& b)
{
  return (ll) a.first*b.second - (ll) a.second*b.first;
}

// square distance between two points.
static inline ll __pi_sqdist(const pi& a, const pi& b)
{
  return __isq(a.first-b.first) + __isq(a.second-b.second);
}

// orientation of 3 points: -1 (clockwise), 0 (linear), 1 (counterclockwise)
static inline int __pi_ort(const pi& a, const pi& b, const pi& c)
{
  return __reduce(__pi_cross(__pi_sub(b, a), __pi_sub(c, b)));
}

// convex hull algorithm (graham scan method).
int cvx_hull(vector<pi>& coords, int n)
{
  pi pivot = pi(INT_MAX, INT_MAX);
  int __pivot_idx;

  for (int i = 0; i < n; i++)
  {
    if (coords[i] < pivot)
    {
      pivot = coords[i];
      __pivot_idx = i;
    }
  }

  swap(coords[0], coords[__pivot_idx]);
  sort(coords.begin()+1, coords.end(), [&pivot](const pi& a, const pi& b)
  {
    int ort = __pi_ort(pivot, a, b);
```

```
    if (ort > 0)
      return true;

    if (ort < 0)
      return false;

    return __pi_sqdist(pivot, a) < __pi_sqdist(pivot, b);
  });

  vector<pi> res;
  res.push_back(coords[0]);
  res.push_back(coords[1]);

  for (int i = 2; i < n; i++)
  {
    while (res.size() > 1 && __pi_ort(res.end()[-2], res.end()[-1],
coords[i]) <= 0)
      res.pop_back();

    res.push_back(coords[i]);
  }

  return res.size();
}

// line segment intersection check.
bool sgmt_check(ppi l1, ppi l2)
{
  pi a = l1.first;
  pi b = l1.second;
  pi c = l2.first;
  pi d = l2.second;

  int ab = __pi_ort(a, b, c) * __pi_ort(a, b, d);
  int cd = __pi_ort(c, d, a) * __pi_ort(c, d, b);

  if (ab == 0 && cd == 0)
  {
    if (a > b) swap(a, b);
    if (c > d) swap(c, d);
    return a <= d && b >= c;
  }

  return ab <= 0 && cd <= 0;
}
```

3 Graph

3.1 Articulation Point, Bridge

```
vector<int> g[s], ans;
int order[s], par[s], low[s], t;

void dfs(int v){
  order[v] = t++;
  low[v] = t;
  int sub = 0; //자식 수
  for(auto i : g[v]){
    if(i == par[v]) continue;

    if(!order[i]){
      par[i] = v;
      sub++;
      dfs(i);
      if(!par[v] && sub > 1) ans.push_back(v); //루트 노드
      else if(par[v] && low[i] >= order[v]) ans.push_back(v);

      low[v] = min(low[v], low[i]);
    }
    else low[v] = min(low[v], order[i]);
  }
}
//단절점
//ans에 있음

vector<int> g[100010];
int order[100010]; //발견 순서
int par[100010];  //부모
int low[100010]; //i를 루트로 하는 서브 트리에서 가장 위로 갈 수 있는 노드
vector<pair<int,int>> ans;
int t;

void dfs(int v){
  order[v] = t++;
  low[v] = t;

  for(auto i : g[v]){ //i : 다음에 갈 노드
    if(i == par[v]) continue; //내 부모 == 나 임면 스킵

    if(order[i] == 0){ //아직 발견이 안되었다면
      par[i] = v; //다음 노드의 붑모는 현재 노드
      dfs(i);

      if(low[i] > order[v]) ans.push_back({min(v, i), max(v, i)});
      /*서브 트리에서 위로 더 올라갈 수 없으면 단절점 확정*/
```

```
      low[v] = min(low[v], low[i]);
    }
    else low[v] = min(low[v], order[i]);
  }
}
//단절선
//ans에 있음
```

3.2 Bellman Ford

```
// A C++ program for Bellman-Ford's single source
// shortest path algorithm.
#include <bits/stdc++.h>
using namespace std;

// a structure to represent a weighted edge in graph
struct Edge {
    int src, dest, weight;
};

// a structure to represent a connected, directed and
// weighted graph
struct Graph {
    // V-> Number of vertices, E-> Number of edges
    int V, E;

    // graph is represented as an array of edges.
    struct Edge* edge;
};

// Creates a graph with V vertices and E edges
struct Graph* createGraph(int V, int E)
{
    struct Graph* graph = new Graph;
    graph->V = V;
    graph->E = E;
    graph->edge = new Edge[E];
    return graph;
}

// A utility function used to print the solution
void printArr(int dist[], int n)
{
    printf("Vertex   Distance from Source\n");
    for (int i = 0; i < n; ++i)
        printf("%d \t\t %d\n", i, dist[i]);
}

// The main function that finds shortest distances from src
// to all other vertices using Bellman-Ford algorithm.  The
// function also detects negative weight cycle
void BellmanFord(struct Graph* graph, int src)
{
    int V = graph->V;
    int E = graph->E;
    int dist[V];

    // Step 1: Initialize distances from src to all other
    // vertices as INFINITE
    for (int i = 0; i < V; i++)
        dist[i] = INT_MAX;
    dist[src] = 0;

    // Step 2: Relax all edges |V| - 1 times. A simple
    // shortest path from src to any other vertex can have
    // at-most |V| - 1 edges
    for (int i = 1; i <= V - 1; i++) {
        for (int j = 0; j < E; j++) {
            int u = graph->edge[j].src;
            int v = graph->edge[j].dest;
            int weight = graph->edge[j].weight;
            if (dist[u] != INT_MAX
                && dist[u] + weight < dist[v])
                dist[v] = dist[u] + weight;
        }
    }

    // Step 3: check for negative-weight cycles.  The above
    // step guarantees shortest distances if graph doesn't
    // contain negative weight cycle.  If we get a shorter
    // path, then there is a cycle.
    for (int i = 0; i < E; i++) {
        int u = graph->edge[i].src;
        int v = graph->edge[i].dest;
        int weight = graph->edge[i].weight;
        if (dist[u] != INT_MAX
            && dist[u] + weight < dist[v]) {
            printf("Graph contains negative weight cycle");
            return; // If negative cycle is detected, simply
                    // return
        }
    }

    printArr(dist, V);
```

```
    return;
}

// Driver's code
int main()
{
    /* Let us create the graph given in above example */
    int V = 5; // Number of vertices in graph
    int E = 8; // Number of edges in graph
    struct Graph* graph = createGraph(V, E);

    // add edge 0-1 (or A-B in above figure)
    graph->edge[0].src = 0;
    graph->edge[0].dest = 1;
    graph->edge[0].weight = -1;

    // add edge 0-2 (or A-C in above figure)
    graph->edge[1].src = 0;
    graph->edge[1].dest = 2;
    graph->edge[1].weight = 4;

    // add edge 1-2 (or B-C in above figure)
    graph->edge[2].src = 1;
    graph->edge[2].dest = 2;
    graph->edge[2].weight = 3;

    // add edge 1-3 (or B-D in above figure)
    graph->edge[3].src = 1;
    graph->edge[3].dest = 3;
    graph->edge[3].weight = 2;

    // add edge 1-4 (or B-E in above figure)
    graph->edge[4].src = 1;
    graph->edge[4].dest = 4;
    graph->edge[4].weight = 2;

    // add edge 3-2 (or D-C in above figure)
    graph->edge[5].src = 3;
    graph->edge[5].dest = 2;
    graph->edge[5].weight = 5;

    // add edge 3-1 (or D-B in above figure)
    graph->edge[6].src = 3;
    graph->edge[6].dest = 1;
    graph->edge[6].weight = 1;

    // add edge 4-3 (or E-D in above figure)
    graph->edge[7].src = 4;
    graph->edge[7].dest = 3;
    graph->edge[7].weight = -3;

      // Function call
    BellmanFord(graph, 0);

    return 0;
}
```

3.3 Dijkstra

```
vector<pair<int,int>> adj[20005];
int d[20005];
const int INF = INT_MAX;
int main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0);

    int v,e,st; cin >> v >> e >> st;
    fill(d,d+v+1,INF);
    while(e--) {
        int u , v , w;
        cin >> u >> v >> w;
        adj[u].push_back({w,v});
    }

priority_queue<pair<int,int>,vector<pair<int,int>>,greater<pair<int,int>>>
pq;
    d[st] = 0;
    pq.push({d[st],st});
    while(!pq.empty()){
        auto cur = pq.top(); pq.pop();
        if(d[cur.second]!= cur.first) continue; //가중치가 같지 않음?
        for(auto nxt : adj[cur.second]) {
            if(d[nxt.second] <= d[cur.second] + nxt.first) continue;
            d[nxt.second] = d[cur.second] + nxt.first;
            pq.push({d[nxt.second],nxt.second});
        }
    }
    for(int i = 1; i <= v; i++) {
        if(d[i] == INF) cout << "INF";
        else cout << d[i];
        cout << '\n';
    }
}
```

3.4 Lowest Common Ancestor

```
const int siz = 1e5+10;
vector<int> ed[siz];
int depth[siz];
int dp[siz][30];
int vis[siz];
int n;

void dfs(int v, int d){
  vis[v] = 1;
  depth[v] = d;
  for(auto i : ed[v]){
    if(!vis[i]){
      dp[i][0] = v;
      dfs(i, d+1);
    }
  }
}

void make_table(){
  for(int j=1; j<30; j++){
    for(int i=1; i<=n; i++){
      dp[i][j] = dp[ dp[i][j-1] ][j-1];
    }
  }
}

int lca(int u, int v){
  if(depth[u] < depth[v]) swap(u, v);
  int diff = depth[u] - depth[v];
  for(int i=0; diff; i++){
    if(diff & 1) u = dp[u][i];
    diff >>= 1;
  }
  if(u == v) return u;

  for(int i=29; i>=0; i--){
    if(dp[u][i] != dp[v][i]) u = dp[u][i], v = dp[v][i];
  }
  return dp[u][0];
}

//간선 입력받은 후, dfs(root,0); make_table()로 LCA구축하기
```

3.5 Union Find, Minimal Spanning Tree

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

struct itriple
{
  int x, y, z;

  itriple() {}
  itriple(int x, int y, int z): x(x), y(y), z(z) {}
  bool operator<(const itriple& a) const
  {
    if (x < a.x)
      return true;
    else if (x > a.x)
      return false;

    if (y < a.y)
      return true;
    else if (y > a.y)
      return false;

    if (z < a.z)
      return true;

    return false;
  }
};

// find the root of an element in a forest; the paths are recursively
compressed to improve performance of future calls.
int disjoint_root(vector<int>& forest, int a)
{
  if (forest[a] == a)
    return a;

  forest[a] = disjoint_root(forest, forest[a]);
  return forest[a];
}

// merge the two elements in a forest (size based).
// returns zero if merging is unnecessary (already in the same group);
otherwise, the elements are merged and 1 is returned.
int disjoint_merge_size(vector<int>& forest, vector<int>& __size, int a, int
b)
{
  int root_a = disjoint_root(forest, a);
  int root_b = disjoint_root(forest, b);

  if (root_a == root_b)
```

```
    return 0;

  if (__size[root_a] < __size[root_b])
  {
    forest[root_a] = root_b;
    __size[root_b] += __size[root_a];
  }
  else
  {
    forest[root_b] = root_a;
    __size[root_a] += __size[root_b];
  }

  return 1;
}

// merge the two elements in a forest (rank based).
// returns zero if merging is unnecessary (already in the same group);
otherwise, the elements are merged and 1 is returned.
int disjoint_merge_rank(vector<int> &forest, vector<int>& __rank, int a, int
b)
{
  int root_a = disjoint_root(forest, a);
  int root_b = disjoint_root(forest, b);

  if (root_a == root_b)
    return 0;

  if (__rank[root_a] < __rank[root_b])
    forest[root_b] = root_a;
  else if (__rank[root_a] > __rank[root_b])
    forest[root_a] = root_b;
  else
  {
    forest[root_b] = root_a;
    __rank[root_a]++;
  }

  return 1;
}

int main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(nullptr);

  int v, e;
  cin >> v >> e;

  vector<itriple> edges(e);
  for (int i = 0; i < e; i++)
  {
    int a, b, c;
    cin >> a >> b >> c;
    edges[i] = itriple(c, a, b);
  }
  sort(edges.begin(), edges.end());

  vector<int> forest(v+1);
  vector<int> __size(v+1);
  vector<int> __rank(v+1);
  for (int i = 1; i <= v; i++)
  {
    forest[i] = i;
    __size[i] = 1;
    __rank[i] = 1;
  }

  int ans = 0;
  for (const itriple& edge : edges)
    ans += disjoint_merge_rank(forest, __rank, edge.y, edge.z) * edge.x;
    // ans += disjoint_merge_size(forest, __size, edge.y, edge.z) * edge.x;

  cout << ans << '\n';
  return 0;
}
```

3.6 Strongly Connected Components

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
/*
scc 는 사이클의 더 확장된 개념이라고 볼 수 있다. 유향 그래프에서, 모든 정점에
대해서 서로도달가능한
최대부분 그래프를 의미한다. 방향 비순환 그래프로 그래프를 압축할 수 있음.*/

//정점x부터 dfs하면서 scc 찾기.
int findscc(int x, int &id,int (& P)[], bool (& finished)[],
stack<int> &S, vector<vector<int>> &adj,vector<vector<int>> &scc,
vector<int> &sccID, int &sccCnt) {
    P[x] = ++id;  S.push(x);  int parent = P[x];
    //정점x와 인접한 정점들의 부모노드를결정. 사이클이 돌면 호출멈추기.
    for(auto y : adj[x]) {
        if(P[y]==0) parent =
min(parent,findscc(y,id,P,finished,S,adj,scc,sccID,sccCnt));
```

```
        else if(!finished[y]) parent = min(parent,P[y]);
    } //같은 scc들을 가장작은 정점의 값으로 같은 소속임을 나타냄.
    if(parent == P[x]) {
        vector<int> SCC;
        while(true) {
            int y = S.top();
            S.pop();
            SCC.push_back(y);
            finished[y] = true;
            sccID[y] = sccCnt;
            if(y==x) break;
        }
        sccCnt++;
        scc.push_back(SCC);
    }
    return parent; //P[x]에 부모 넣어서 scc 까지 확인하기 <- 반례가 나오긴
했는데 약한 데이터에서는 괜찮을수도
} //P[x] == 0인 정점에 대해서만 탐색을 실시

int main(void)
{
    int n,m;
    cin >> n >> m;
    int id = 0, sccCnt = 0; //sccID 로 그 노드들이 같은 scc인지 확인
    int P[2*n+1] = {0};//id:정점번호,부모체크
    vector<int> sccID(2*n+1);
    bool finished[2*n+1] = {0};//scc에 속한 정점체크
    vector<vector<int>> adj(2*n+1); //정점x 인접그래프
    stack<int> S; //dfs에 사용하는 스택
    vector<vector<int>> scc; //scc 저장.

    for(int i = 0; i < m; i++) {
        int u,v; cin >> u >> v;
        if(u<0) u = n-u;
        if(v<0) v = n-v;
        if(u<=n) u += n; else u -= n;
        adj[u].push_back(v);
        if(u<=n) u += n; else u -= n;
        if(v<=n) v += n; else v -= n;
        adj[v].push_back(u);
    }
    for(int i = 1; i <=2*n; i++) {
        if(P[i]==0) findscc(i,id,P,finished,S,adj,scc,sccID,sccCnt);
    }
    for(int i = 1; i <= n; i++) {
        if(sccID[i] == sccID[n+i]) {
            cout << 0;
            return 0;
        }
    }
    cout << 1;

}
```

3.7 Sparse Table

```
const int siz = 100005;
int n;
vector<int> ed[siz];
int sp[20][100005];
int vis[siz];
void dfs(int v){
  vis[v] = 1;
  for(auto i : ed[v]){
    if(!vis[i]){
      sp[0][i]=v;
      dfs(i.X);
    }
  }
}
void make_sparse()
{
  for(int p=1; p<20; p++)
  {
    for(int i=1; i<=n; i++)
    {
      sp[p][i]=sp[p-1][sp[p-1][i]];
    }
  }
}
int f(int v, int k)
{
  for(int i=19; i>=0; i--)
  {
    if(k&(1<<i))
    {
      v=sp[i][v];
    }
  }
  return v;
}
```

3.8 Topology Sort

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

```
void solve(int n, const vector<int>& time, const vector<vector<int>>& adj,
vector<int>& res)
{
  vector<int> start(n+1);
  vector<int> indeg(n+1);
  queue<int> tsort_queue;

  // pre-compute indegree.
  for (int i = 1; i <= n; i++)
  {
    for (int j : adj[i])
      indeg[j]++;
  }

  // search starting positions;
  for (int i = 1; i <= n; i++)
  {
    if (!indeg[i])
      tsort_queue.push(i);
  }

  while (!tsort_queue.empty())
  {
    int cur = tsort_queue.front(); tsort_queue.pop();
    res[cur] = start[cur] + time[cur];

    for (int i : adj[cur])
    {
      indeg[i]--;
      start[i] = max(start[i], res[cur]);

      if (!indeg[i])
        tsort_queue.push(i);
    }
  }
}

int main()
{
  std::ios_base::sync_with_stdio(false);
  cin.tie(nullptr);

  int n;
  cin >> n;

  vector<int> time(n+1);
  vector<vector<int>> adj(n+1);

  for (int i = 1; i <= n; i++)
  {
    int pre;
    cin >> time[i] >> pre;

    while (pre != -1)
    {
      adj[pre].push_back(i);
      cin >> pre;
    }
  }

  vector<int> res(n+1);
  solve(n, time, adj, res);

  for (int i = 1; i <= n; i++)
    cout << res[i] << '\n';

  return 0;
}
```

4 IMOS

4.1 IMOS (1-Dimensional), Value Compression

```
#include <bits/stdc++.h>
using namespace std;
typedef pair<int, int> pi;

int main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(nullptr);

  int N;
  cin >> N;

  // value collection.
  vector<pi> T;
  for (int i = 0; i < N; i++)
  {
    int e, x;
    cin >> e >> x;
    T.push_back(pi(e, 1)); // 1 marks an entry.
    T.push_back(pi(x, 0)); // 0 marks an exit.
  }
  sort(T.begin(), T.end());
```

```cpp
// value compression.
int prev_val;
vector<int> orig(T.size());
orig[0] = T[0].first;
prev_val = T[0].first;
T[0].first = 0;

int comp = 0; // compressed value.
for (int i = 1; i < T.size(); i++)
{
  if (T[i].first != prev_val)
  {
    orig[++comp] = T[i].first;
    prev_val = T[i].first;
  }

  T[i].first = comp;
}

// difference array construction.
vector<int> diff(comp+1);
for (pi& p : T)
{
  if (p.second)
    diff[p.first]++;
  else
    diff[p.first]--;
}

for (int i = 1; i <= comp; i++)
  diff[i] += diff[i-1];

// finally, we find the segment with the largest diff-value.
int ans = diff[0];
int seg_begin = 0;
int seg_end = 1;

for (int i = 1; i <= comp; i++)
{
  if (diff[i] > ans)
  {
    ans = diff[i];
    seg_begin = i;
    seg_end = i+1;
  }
  else if (diff[i] == ans && i == seg_end)
    seg_end++;
  }

  cout << ans << '\n';
  cout << orig[seg_begin] << ' ' << orig[seg_end] << '\n';
  return 0;
}
```

4.2 IMOS (2-Dimensional), Rectangle, Diamond

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

int rect_img[2502][2502];
int diag_img[2502][2502];
int W, H;

// mark start/terminal points on a rectangular basis.
void mark_rect(int px, int py, int qx, int qy)
{
  rect_img[py][px]++;
  rect_img[py][qx+1]--;
  rect_img[qy+1][px]--;
  rect_img[qy+1][qx+1]++;
}

// mark start/terminal points on a diagonal basis.
void mark_diag(int px, int py, int r)
{
  diag_img[py][px-r]++;

  if (py+r+1 < H)
    diag_img[py+r+1][px+1]--;
  else
    diag_img[py+r][px+2]--;

  if (py-r-1 >= 0)
    diag_img[py-r-1][px+1]--;

  diag_img[py][px+r+2]++;
}

// rectangular imos sweeping.
void sweep_rect()
{
  // sweep from left to right.
  for (int i = 0; i < H; i++)
  {
```

```
    for (int j = 1; j < W; j++)
      rect_img[i][j] += rect_img[i][j-1];
  }

  // sweep from top to bottom.
  for (int i = 1; i < H; i++)
  {
    for (int j = 0; j < W; j++)
      rect_img[i][j] += rect_img[i-1][j];
  }
}

// diagonal imos sweeping.
void sweep_diag()
{
  // sweep diagonally from bottom-left to top-right.
  for (int i = 1; i < H; i++)
  {
    for (int j = 1; i-j >= 0 && j < W; j++)
      diag_img[i-j][j] += diag_img[i-j+1][j-1];
  }

  for (int i = 1; i < W-1; i++)
  {
    for (int j = 1; j < H && i+j < W; j++)
      diag_img[H-1-j][i+j] += diag_img[H-j][i+j-1];
  }

  // sweep diagonally from top-left to bottom-right.
  for (int i = 0; i < H-1; i++)
  {
    for (int j = 1; i+j < H && j < W; j++)
      diag_img[i+j][j] += diag_img[i+j-1][j-1];
  }

  for (int i = 1; i < W-1; i++)
  {
    for (int j = 1; j < H && i+j < W; j++)
      diag_img[j][i+j] += diag_img[j-1][i+j-1];
  }
}

int main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(nullptr);

  int K;
  cin >> W >> H >> K;

  while (K--)
  {
    // type variable.
    int t;
    cin >> t;

    if (t == 1)
    {
      int px, py, qx, qy;
      cin >> px >> py >> qx >> qy;
      mark_rect(px, py, qx, qy);
    }
    else
    {
      int px, py, r;
      cin >> px >> py >> r;
      mark_diag(px, py, r);
      mark_diag(px, py, r-1);
    }
  }

  sweep_rect();
  sweep_diag();

  // final output.
  for (int i = 0; i < H; i++)
  {
    for (int j = 0; j < W; j++)
    {
      int __value = rect_img[i][j] + diag_img[i][j];
      cout << (__value % 2 ? '#' : '.');
    }

    cout << '\n';
  }

  return 0;
}
```

5 Math

5.1 Extended GCD

```
ll POW(ll a, ll b, ll MMM) {
    ll ret = 1; for (; b; b >>= 1, a = (a * a) % MMM)
    if (b & 1)ret = (ret * a) % MMM; return ret; }
```

```
ll mod(ll a,ll m) {return (a % m + m) % m;}
ll binary_gcd(ll a,ll b) { //큰 수에 대해서 사용.
    if(a==0||b==0) return a+b;
    int shift = __builtin_ctzll(a|b);
    a >>= __builtin_ctzll(a);
    while(b != 0) {
        b >>= __builtin_ctzll(b);
        if(a>b) swap(a,b); b-=a;
    }
    return a << shift;
}
struct egcdResult {
    ll gcd,x,y;
};
egcdResult egcd(ll a,ll b) {
    if(b==0) {
        return {a,1,0};
    }
    egcdResult res = egcd(b,a%b);
    ll x = res.y;
    ll y = res.x - (a/b) * res.y;
    return {res.gcd,x,y};
}   // x인 해와 그 mod
pair<ll,ll> crt(ll a1, ll m1,ll a2, ll m2) {
    ll g = gcd(m1,m2) , m = m1 / g * m2;
    if((a2 - a1) % g) return {-1,-1};//crt의 해가 존재X
    ll md = m2/g, s = mod((a2-a1)/g,m2/g);
    ll t = mod(egcd(m1/g%md,m2/g).x ,md);
    return {a1 + s * t % md * m1,m};
}

pair<ll,ll> crtmany(const vector<ll> &a,const vector<ll> &m) {
    ll ra = a[0], rm = m[0];
    for(int i = 1; i < m.size(); i++) {
        auto [aa,mm] = crt(ra,rm,a[i],m[i]);
        if(mm == -1) return {-1,-1}; else tie(ra,rm) = tie(aa,mm);
    }return {ra,rm};
}
```

5.2 Floor Sum

```
ll floorsum(ll a, ll b,ll c,ll n){ //[(ax+b)/c] x=0~n sum.
  if(!a) return (b / c) * (n + 1);
  if(a >= c or b >= c) return ( ( n * (n + 1) ) / 2) * (a / c) + (n + 1) *
(b / c) + floorsum(a % c, b % c, c, n);
  long long m = (a * n + b) / c;
  return m * n - floorsum(c, c - b - 1, a, m - 1);
}
```

5.3 Integer Division Algorithm

```
// computes a/b and prints out p digits after the decimal point (rounded).
void printd(int a, int b, int p)
{
  // int sign = a*b < 0;
  a = abs(a);
  b = abs(b);
  vector<int> digits;

  digits.push_back(a/b);
  a %= b;

  for (int i = 0; i <= p; i++)
  {
    a *= 10;
    digits.push_back(a/b);
    a %= b;
  }

  if (digits[p+1] >= 5)
    digits[p]++;

  for (int i = p; i >= 1; i--)
  {
    if (digits[i] == 10)
    {
      digits[i] = 0;
      digits[i-1]++;
    }
    else
      break;
  }

  // if (sign)
  //  cout << '-';

  cout << digits[0] << '.';
  for (int i = 1; i <= p; i++)
    cout << digits[i];
  cout << '\n';
}
```

5.4 Linear Sieve

```
// computes a/b and prints out p digits after the decimal point (rounded).
void printd(int a, int b, int p)
```

```
{
  // int sign = a*b < 0;
  a = abs(a);
  b = abs(b);
  vector<int> digits;

  digits.push_back(a/b);
  a %= b;

  for (int i = 0; i <= p; i++)
  {
    a *= 10;
    digits.push_back(a/b);
    a %= b;
  }

  if (digits[p+1] >= 5)
    digits[p]++;

  for (int i = p; i >= 1; i--)
  {
    if (digits[i] == 10)
    {
      digits[i] = 0;
      digits[i-1]++;
    }
    else
      break;
  }

  // if (sign)
  //  cout << '-';

  cout << digits[0] << '.';
  for (int i = 1; i <= p; i++)
    cout << digits[i];
  cout << '\n';
}
```

5.5 Modular Integer Power (Nonnegative)

```
#include <bits/stdc++.h>
typedef long long ll;
//
ll POW(ll a, ll b, ll MMM) { ll ret = 1; for (; b; b >>= 1, a = (a * a) %
MMM)if (b & 1)ret = (ret * a) % MMM; return ret; }
```

6 Network Flow

6.1 Bimatch

```
#define MXN 1001
int A[MXN], B[MXN];
vector<int> adj[MXN]; bool visited[MXN];
bool dfs(int a) {
    visited[a]=true;
    for(int b : adj[a]){
        if(B[b]==-1||!visited[B[b]]&&dfs(B[b])) {
            A[a]=b;B[b]=a;return true;
        }}
    return false;}
int main() {
    int n,m; cin >> n >> m;
    for(int i=0; i<n;i++){
        int s; cin >> s;
        for(int j=0; j<s;j++){
            int z; cin >> z;
            adj[i].push_back(z-1);
        }
    }//A는 왼쪽,B는오른쪽
    fill(A,A+n,-1); fill(B,B+m,-1);
    int match = 0;
    while(true) {
        bool flow = false;
        fill(visited,visited+n,false);
        for(int i=0;i < n;i++){
            if(A[i]==-1){
                if(dfs(i)){
                    flow=true; match++;
                }
            }
        }
        if(!flow) break;
    }

    cout<<match;
}
```

6.2 Dinic

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

struct Edge { // u -> v
    int v,cap,ref; //ref 는 역간선 ?
    Edge(int v,int cap,int ref) :v(v),cap(cap),ref(ref) {}
```

```
};

class Dinic {
    int S,T;
    vector<vector<Edge>> edges; //그래프
    vector<int> level, next_v; //레벨그래프, flow계산시 역추적 사용.
public:
    Dinic(int MAX_V, int S,int T) : S(S),T(T) {
        edges.resize(MAX_V);
        level.resize(MAX_V);
        next_v.resize(MAX_V);
    } //역방향 간선 있으면 추가.
    void addEdge(int u,int v,int cap,bool inv) {
        edges[u].emplace_back(v,cap,(int)edges[v].size());
        edges[v].emplace_back(u,inv?cap : 0, (int) edges[u].size()-1);
    } //  이 간선의 역방향 간선이 v의 인접 리스트에서 어떤 위치에 있는지 저장.
    void reset_next_v() {
        fill(next_v.begin(),next_v.end(),0);
    }
    bool bfs() {
        fill(level.begin(),level.end(),-1);
        queue<int> Q;
        level[S] = 0;
        Q.push(S);
        while(!Q.empty()) {
            int u = Q.front(); Q.pop();
            for(auto edge: edges[u]) {
                int v = edge.v, cap = edge.cap;
                if(level[v]==-1 && cap > 0) {
                    level[v] = level[u] + 1;
                    Q.push(v);
                }
            }
        }
        return level[T] != -1;//sink T 에 도달가능여부.
    }
    int dfs(int u,int max_flow) {
        if(u==T) return max_flow; //싱크에 도달
        for(int &i = next_v[u]; i<edges[u].size(); i++) {
            int v = edges[u][i].v, cap = edges[u][i].cap;
            if(level[u]+1 == level[v] && cap > 0) { //level satisfy
                int flow = dfs(v,min(max_flow,cap));
                if(flow>0) {
                    edges[u][i].cap -= flow;//유량갱신
                    edges[v][edges[u][i].ref].cap += flow;//역간선용량증가
                    return flow;
                }
            }
        }
        return 0;
    }
};
int main(void)
{
    ios_base::sync_with_stdio(0);
    cin.tie(0);

    Dinic netflow(58,0,25);
    int tt; cin >> tt;
    while(tt--) {
        int cap;
        char a,b; cin >> a >> b >> cap;
        a -= 'A';
        b -= 'A';
        netflow.addEdge(a,b,cap,true);
    }
    int result = 0;
    while(netflow.bfs()) {
        netflow.reset_next_v();
        while(true) {
            int flow = netflow.dfs(0,INT_MAX);
            if(!flow) break;
            result += flow;
        }
    }
    cout << result;
}
```

6.3 Network Flow

```
int c[60][60], f[60][60];
int MaxFlow(int source,int sink) {
    int totalFlow = 0;
    while(true) {
        vector<int> parent(60,-1);
        queue<int> Q; parent[source] = source; Q.push(source);
        while(!Q.empty() && parent[sink] == -1) {
            int here = Q.front(); Q.pop();
            for(int t_here = 0; t_here < 60; t_here++) {
                if(c[here][t_here] - f[here][t_here] > 0 && parent[t_here]
== -1) {
                    Q.push(t_here); parent[t_here] = here; } } }
        if(parent[sink] == -1) break;
        int amount = INT_MAX;
        for(int p = sink; p != source; p = parent[p])
```

```
            amount = min(c[parent[p]][p]-f[parent[p]][p],amount);
        for(int p = sink; p != source; p = parent[p]) {
            f[parent[p]][p] += amount; f[p][parent[p]] -= amount;
        }
        totalFlow += amount; }
    return totalFlow; }
int main() {
    int n; cin >> n;
    for(int i = 0; i < n; i++) {
        int cap,fr,to; cin >> fr >> to >> cap;
        c[fr][to] += cap; //간선마다용량추가
        } //시작과끝노드지정
    cout << MaxFlow(0,25);
    return 0;
}
```

7 Segment Tree

7.1 Persistent Segment Tree

```
int c[60][60], f[60][60];
int MaxFlow(int source,int sink) {
    int totalFlow = 0;
    while(true) {
        vector<int> parent(60,-1);
        queue<int> Q; parent[source] = source; Q.push(source);
        while(!Q.empty() && parent[sink] == -1) {
            int here = Q.front(); Q.pop();
            for(int t_here = 0; t_here < 60; t_here++) {
                if(c[here][t_here] - f[here][t_here] > 0 && parent[t_here]
== -1) {
                    Q.push(t_here); parent[t_here] = here; } } }
        if(parent[sink] == -1) break;
        int amount = INT_MAX;
        for(int p = sink; p != source; p = parent[p])
            amount = min(c[parent[p]][p]-f[parent[p]][p],amount);
        for(int p = sink; p != source; p = parent[p]) {
            f[parent[p]][p] += amount; f[p][parent[p]] -= amount;
        }
        totalFlow += amount; }
    return totalFlow; }
int main() {
    int n; cin >> n;
    for(int i = 0; i < n; i++) {
        int cap,fr,to; cin >> fr >> to >> cap;
        c[fr][to] += cap; //간선마다용량추가
        } //시작과끝노드지정
    cout << MaxFlow(0,25);
    return 0;
}
```

7.2 Lazy Segment Tree

```
#include <iostream>
#include <cmath>
#include <vector>
using namespace std;
typedef long long ll;

void init(vector<ll> &a, vector<ll> &tree, int node, int start, int end) {
    if (start == end) {
        tree[node] = a[start];
    } else {
        init(a, tree, node*2, start, (start+end)/2);
        init(a, tree, node*2+1, (start+end)/2+1, end);
        tree[node] = tree[node*2] + tree[node*2+1];
    }
}
void update_lazy(vector<ll> &tree, vector<ll> &lazy, int node, int start,
int end) {
    if (lazy[node] != 0) {
        tree[node] += (end-start+1)*lazy[node];
        if (start != end) {
            lazy[node*2] += lazy[node];
            lazy[node*2+1] += lazy[node];
        }
        lazy[node] = 0;
    }
}
void update_range(vector<ll> &tree, vector<ll> &lazy, int node, int start,
int end, int left, int right, ll diff) {
    update_lazy(tree, lazy, node, start, end);
    if (left > end || right < start) {
        return;
    }
    if (left <= start && end <= right) {
        tree[node] += (end-start+1)*diff;
        if (start != end) {
            lazy[node*2] += diff;
            lazy[node*2+1] += diff;
        }
        return;
    }
    update_range(tree, lazy, node*2, start, (start+end)/2, left, right,
diff);
    update_range(tree, lazy, node*2+1, (start+end)/2+1, end, left, right,
diff);
```

```
    tree[node] = tree[node*2] + tree[node*2+1];
}
ll query(vector<ll> &tree, vector<ll> &lazy, int node, int start, int end,
int left, int right) {
    update_lazy(tree, lazy, node, start, end);
    if (left > end || right < start) {
        return 0;
    }
    if (left <= start && end <= right) {
        return tree[node];
    }
    ll lsum = query(tree, lazy, node*2, start, (start+end)/2, left, right);
    ll rsum = query(tree, lazy, node*2+1, (start+end)/2+1, end, left,
right);
    return lsum + rsum;
}
int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(nullptr);
    int n, m, k;
    cin >> n >> m >> k;
    vector<ll> a(n);
    int h = (int)ceil(log2(n));
    int tree_size = (1 << (h+1));
    vector<ll> tree(tree_size);
    vector<ll> lazy(tree_size);
    m += k;
    for (int i=0; i<n; i++) {
        cin >> a[i];
    }
    init(a, tree, 1, 0, n-1);
    while (m--) {
        int what;
        cin >> what;
        if (what == 1) {
            int left, right;
            ll diff;
            cin >> left >> right >> diff;
            update_range(tree, lazy, 1, 0, n-1, left-1, right-1, diff);
        } else if (what == 2) {
            int left, right;
            cin >> left >> right;
            cout << query(tree, lazy, 1, 0, n-1, left-1, right-1) << '\n';
        }
    }
    return 0;
}
```

7.3 Segment Tree

```
int init(int node, int st, int end) {
    // st   : init 함수가 관심두는 arr의 시작 인덱스
    // end  : init 함수가 관심두는 arr의 끝 인덱스
    // node : segTree의 노드
    // -> node번째 노드가 st ~ end의 합을 저장한다.
    if (st == end) return segTree[node] = arr[st];
    int mid = (st + end) / 2;

    //재귀로 반씩 나눠서 초기화
    return segTree[node] = init(node * 2, st, mid) + init(node * 2 + 1, mid
+ 1, end);
}

void update(int n, int st, int end, int t, int diff) {
    // st   : 시작 인덱스
    // end  : 끝 인덱스
    // idx  : 수정할 원소의 인덱스
    // diff  : 수정할 값

    // 범위 안에 있을 경우
    if (st <= t && t <= end) segTree[n] += diff;

    // 범위 밖에 있을 경우
    else return;
    if (st == end) return;
    int mid = (st + end) / 2;
    update(n * 2, st, mid, t, diff);
    update(n * 2 + 1, mid + 1, end, t, diff);
}

//sum함수는 구간합 쿼리이다
int sum(int l, int r, int node, int st, int end) {
    // st   : 시작 인덱스
    // end  : 끝 인덱스
    // l~r  : 구하고자 하는 구간 합의 범위

    // [l, r]이 [st, end]를 완전히 포함하는 경우
    if (l <= st && end <= r) return segTree[node];

    // [l, r]와 [st, end]가 겹치지 않는 경우
    if (r < st || end < l) return 0;

    // 나머지 경우 (일부분 겹칠때)
    int m = (st + end) / 2;
    return sum(l, r, node * 2, st, m) + sum(l, r, node * 2 + 1, m + 1, end);
```

```
}
```

8 String

8.1 KMP

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

using namespace std;
typedef long long ll;
typedef pair<ll,ll> pl;

vector<int> failure(string & s) {
    vector<int> f(s.size());
    int j = 0;
    for(int i = 1; i < s.size(); i++) {
        while(j>0 && s[i] != s[j]) j = f[j-1];
        if(s[i] == s[j]) f[i] = ++j;
    } return f;
}
ll kmp(string s, string p) {
    vector<int> f = failure(p);
    ll cnt = 0;
    int j = 0;
    for(int i = 0; i < s.size(); i++) {
        while(j>0 && s[i] != p[j]) j = f[j-1];
        if(s[i] == p[j]) j++;
        if(j == p.size()) {
            cnt++;
            //w.emplace_back(i-j + 2); 몇번째인덱스인지카운트
        }
    } return cnt;
}
```

8.2 Suffix Array, LCP

```
int n, t;
string s;
int SA[MAX_N], LCP[MAX_N];
int tmp[MAX_N], poz[MAX_N];

//Algoritam koji generise sufiksni niz i niz najdužeg zajedničkog prefiksa
(LCP) nad datim stringom
//Slozenost: O(n log^2 n) za sufiksni niz, O(n) za LCP niz

inline bool suff_compare(int i, int j)
{
    if (poz[i] != poz[j]) return (poz[i] < poz[j]);
    i += t;
    j += t;
    if (i < n && j < n) return (poz[i] < poz[j]);
    else return (i > j);
}

inline void buildSA()
{
    for (int i=0;i<n;i++)
    {
        SA[i] = i;
        poz[i] = s[i];
    }
    for (t = 1 ; t < 2*n+1; t *= 2)
    {
        sort(SA, SA + n, suff_compare);
        for (int i=0;i<n-1;i++)
        {
            tmp[i+1] = tmp[i];
            if (suff_compare(SA[i],SA[i+1])) tmp[i+1]++;
        }
        for (int i=0;i<n;i++)
        {
            poz[SA[i]] = tmp[i];
        }
        if (tmp[n-1] == n-1) break;
    }
}

inline void buildLCP()
{
    int k = 0;
    for (int i=0;i<n;i++)
    {
        if (poz[i] != n-1)
        {
            int j = SA[poz[i]+1];
            while (s[i+k] == s[j+k]) k++;
            LCP[poz[i]] = k;
            if (k > 0) k--;
        }
    }
}

int main()
{
    n = 6;
    s = "banana";
```

```
    buildSA();
    buildLCP();

    for (int i=0;i<n;i++) printf("%d ",SA[i]);
    printf("\n");
    for (int i=0;i<n;i++) printf("%d ",LCP[i]);
    printf("\n");

    return 0;
}
```

9 Misc

9.1 Custom Comparator, Custom Hashing

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef unsigned long long ull;
typedef unsigned int uint;

struct pos
{
  int x, y;
  pos(): x(0), y(0) {}
  pos(int x, int y): x(x), y(y) {}

  // this comparator is analogous to std::less.
  bool operator<(const pos& another) const
  {
    if (x < another.x)
      return true;

    if (x > another.x)
      return false;

    if (y < another.y)
      return true;

    return false;
  }

  bool operator==(const pos& another) const
  {
    return x == another.x && y == another.y;
  }
};

hash<ull> ull_hash;

struct pos_hash
{
  size_t operator()(const pos& p) const
  {
    return ull_hash(((ull) p.y << 32) | (uint) p.x);
  }
};

int main()
{
  set<pos> test1;
  unordered_set<pos, pos_hash> test2;

  pos p1(10, 20);
  pos p2(30, 30);

  test1.insert(p1);
  test2.insert(p2);
}
```