Notebook

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

#define fi first
#define se second
#define _all(v) v.begin(), v.end()
#define __(v) memset(v,0,sizeof(v))
#define arrayin(a,n) for(int i=0;i<n;i++) cin>>a[i];
#define FOR(i,a,b) for (int i=(a),_b_=(b);i<_b_;i++)
#define FOD(i,a,b) for (int i=(a),_b_=(b);i>_b_;i--)
#define arrayout(a,n) for(int i=0;i<n;i++) cout<<a[i]<<" ";cout<<"\n";
#define _it(i,v) for (typeof((v).begin()) i = (v).begin(); i != (v).end(); ++i)


using namespace std;
typedef long long LL;
typedef unsigned long long ULL;
typedef pair<int, int> PII;
template<typename T> vector<T> &operator += (vector<T> &v, T x) {v.push_back(x);return v;}


void solve() {

}

int main(){
    ios_base::sync_with_stdio(0);
//    cin.tie(0);
    #ifdef HIEUNV
    freopen("input.txt","r",stdin);
//  freopen("output.txt","w",stdout);
    #endif
    solve();
}
```

Segment tree (Interval tree)

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

const int N = 1e5 + 10;

int node[4*N];

void modify(int seg, int l, int r, int p, int val){
    if(l == r){
        node[seg] += val;
        return;
    }
    int mid = (l + r)/2;
    if(p <= mid){
        modify(2*seg + 1, l, mid, p, val);
    }else{
        modify(2*seg + 2, mid + 1, r, p, val);
    }
    node[seg] = node[2*seg + 1] + node[2*seg + 2];
}

int sum(int seg, int l, int r, int a, int b){
    if(l > b || r < a) return 0;
    if(l >= a && r <= b) return node[seg];
    int mid = (l + r)/2;
    return sum(2*seg + 1, l, mid, a, b) + sum(2*seg + 2, mid + 1, r, a, b);
}
```

```
int main(){
    int n, m;
    scanf("%d %d", &n, &m);
    for(int i = 1; i <= m; i++){
        char t;
        scanf(" %c", &t);
        if(t == 'A'){
            int p, x;
            scanf("%d %d", &p, &x);
            modify(0, 1, n, p, x);
        }else{
            int a, b;
            scanf("%d %d", &a, &b);
            printf("%d\n", sum(0, 1, n, a, b));
        }
    }
    return 0;
}
```

Binary Indexed tree (Fenwick Tree)

```
#include <iostream>
using namespace std;

int n, m, k;
long long arr[1000005];
long long tree[1000005];

void update(int idx, int val) {
  while (idx <= n) {
    tree[idx] += val;
    idx += (idx & -idx); // 1인 비트 중 가장 최하단 비트 찾음
  }
}

long long read(int idx) {
  long long ret = 0;

  while (idx > 0) {
    ret += tree[idx];
    idx -= (idx & -idx); // 1인 비트 중 가장 최하단 비트 찾음
  }

  return ret;
}

int main() {
  scanf("%d %d %d", &n, &m, &k);
  for (int i = 1; i <= n; i++) {
    scanf("%lld", &arr[i]);
    update(i, arr[i]);
  }
  int a, b, c;
  for (int i = 0; i < m + k; i++) {
    scanf("%d %d %d", &a, &b, &c);
    if (a == 1) {
      update(b, c - arr[b]);
      arr[b] = c;
    }
    else {
      printf("%lld\n", read(c) - read(b - 1));
    }
  }

  return 0;
}
```

```
LL gcd(LL a, LL b) {
  return b?gcd(b,a%b):a;
}
__gcd(a,b); // Builtin
```

```
//Tính a*b%m
LL mulmod(LL a, LL b, LL m) {
  LL res = 0;
  if (!a || !b) {
    return 0;
  }
  if (a<b) swap(a,b);

  while (b>1) {
    if (b&1) res = (res + a)%m;
    a = (a+a)%m;
    b >>= 1;
  }
  return (res+a)%m;
}
```

```
LL xGCD(LL a, LL b, LL &x, LL &y) {
  if (!b) {
    x = 1; y = 0;
    return a;
  }
  LL xx,yy,t = xGCD(b,a%b,xx,yy);
  x = yy;
  y = xx - a/b*yy;;
  return t;
}
```

```
// Tính a^e%m
LL pm(LL a, LL e, LL m) {
  if (m==1) return 0;
  if (!e) return 1;
  LL t = 1;
  while (e > 1) {
    if (e&1) t = t*a%m;
    a = a*a%m;
    e >>= 1;
  }
  return t*a%m;
}
```

```
//Tính n^(-1)%m
LL moduloInverse(LL a, LL m){
  LL q,r,y0=0,y1=1,y,m0=m;
  while (a>0) {
    q = m/a;
    r = m%a;
    if (!r) return (y%m0 + m0) % m0;
    y = y0-y1*q;
    y0=y1;
    y1=y;
    m=a;
    a=r;
  }
}
```

```
Sàng số nguyên tố sử dụng Sieve of Eratosthenes
const int N = 1e7+7;
bool m[N];

void sieveEra() {
  memset(m, 0, sizeof(m));
  int i, lim = sqrt(N);
  for (i=2; i <= lim; i++) {
    if (!m[i]) {
      for (LL j=i*i; j < N; j+=i)
        if (!m[j]) m[j] = true;
    }
  }
}
```

```
Kiểm tra xác suất 1 số là số nguyên tố (Rabin - Miller)
const int RAB[] = {3,5,7,11,13,17}, R = sizeof(RAB)/sizeof(RAB[0]);
LL pm(LL a, LL e, LL m) {
  if (m==1) return 0;
  if (!e) return 1;
  LL t = 1;
  while (e > 1) {
    if (e&1) t = t*a%m;
    a = a*a%m;
    e >>= 1;
  }
  return t*a%m;
}
bool primeTest(LL n) {
  if (n==2) return true;
  if (n<2 || (n&1)==0) return false;
```

```
  LL m = n-1, s = 0;
  while ((m&1)==0) {
    m >>= 1;
    s++;
  }

  _for(i,0,R) {
    LL k = RAB[i], b = pm(k,m,n);
    if (n == k) return true;
    if (n%k == 0) return false;

    if (b == 1) continue;
    bool pass = false;
    _for (j,0,s) {
      if (b == n-1) {
        pass = true;
        break;
      }
      b = b*b%n;
    }
    if (!pass) return false;
  }
  return true;
}
```

```
Prim algorithm (cây khung nhỏ nhất)
#include<bits/stdc++.h>
#define pb push_back
#define mp make_pair
using namespace std;
const int MAX = 5005;
typedef pair<int, int> PII;
vector<PII> adj[MAX];
int visited[MAX];
long long prim(int s){
    long long minimumCost = 0;
    priority_queue<PII, vector<PII>, greater<PII> > q;
    q.push(mp(0, s)); //insert (0, s) to retrieve first node as the starting one
    while(!q.empty()){
        PII p = q.top();
        q.pop();
        int length = p.first;
        s = p.second;
        if(visited[s])  continue; //if the current node is visited earlier check the next value
        visited[s] = 1; //mark the node as visited
        minimumCost+=length;
        for(int i=0;i<adj[s].size();i++){
            if(!visited[adj[s][i].second]) q.push((adj[s][i])); //push all the neighbours of current node in the priority queue
        }
    }
    return minimumCost;
}
int main(){
    int nodes, edges, x, y, weight, s;
    cin >> nodes >> edges;  //Number of Nodes and Edges
    memset(visited, 0, sizeof(visited));  //Initialise values to 0 as none of the nodes are visited
    for(int i = 0;i<edges;i++){
        cin>> x >> y >> weight;
        adj[x].pb(mp(weight, y));
        adj[y].pb(mp(weight, x));
    }
    cin >> s;
    long long minimumCost = prim(s);
    cout<<minimumCost;
    return 0;
}
```

Dijkstra (Đường đi ngắn nhất từ 1 đỉnh đến các đỉnh còn lại):

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

#define _for(i,a,b) for (int i=(a),_b_=(b);i<_b_;i++)
#define _fod(i,a,b) for (int i=(a),_b_=(b);i>_b_;i--)
#define _it(i,v) for (typeof((v).begin()) i = (v).begin(); i != (v).end(); ++i)
#define _all(v) v.begin(), v.end()
#define __(v) memset(v,0,sizeof(v))
#define fi first
#define se second
#define arrayin(a,n) for(int i=0;i<n;i++) cin>>a[i];
#define arrayout(a,n) for(int i=0;i<n;i++) cout<<a[i]<<" ";cout<<"\n";

using namespace std;
typedef long long LL;
typedef unsigned long long ULL;
template<typename T> vector<T> &operator += (vector<T> &v, T x) {v.push_back(x);return v;}
template <class T> string toStr(const T &x){ stringstream s; s << x; return s.str(); }
template <class T> int toInt(const T &x){ stringstream s; s << x; int r; s >> r; return r; }

struct Node{
  int edge;
  int weight;
};

bool operator < (Node a, Node b){
  return a.weight < b.weight;
}

#define INF INT_MAX
const int MAX = 1e5 + 1;
int dist[MAX];
vector<Node> G[MAX];
int n,m;
int first, last;

void dijkstra(int n, int first, int last){
  _for(i,0,n) dist[i] = INF;
  priority_queue<Node> q;
  q.push((Node){first, 0});
  dist[first] = 0;

  while(!q.empty()){
    Node p = q.top();
    q.pop();

    _for(i,0,G[p.edge].size()){
      Node k = G[p.edge][i];
      if(dist[p.edge] + k.weight < dist[k.edge]){
        dist[k.edge] = dist[p.edge] + k.weight;
        q.push(k);
      }
    }
  }

  if(dist[last] != INF){
    cout<<dist[last]<<"\n";
  }else cout<<"NONE\n";
}

void solve() {
  cin>>n>>m>>first>>last;
  first--;
  last--;
  __(G);
  int u,v,w;
  _for(i,0,m){
    cin>>u>>v>>w;
    G[u-1] += ((Node) {v-1,w});
    G[v-1] += ((Node) {u-1,w});
  }
  dijkstra(n,first,last);
```

```
}

int main(){
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    #ifdef HIEUNV
    freopen("input.txt","r",stdin);
//  freopen("output.txt","w",stdout);
    #endif
    int T;
    cin>>T;
    while(T--)
      solve();
}
```

```
Floyd-Warshall
Đường đi ngắn nhất từ 1 đỉnh đến 1 đỉnh khác
#include <bits/stdc++.h>
using namespace std;

const int N = 110;

int dist[N][N];

void floyd(int n){
    for(int k = 1; k <= n; k++){
        for(int i = 1; i <= n; i++){
            for(int j = 1; j <= n; j++){
                dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]);
            }
        }
    }
}

int main(){
    int n, m;
    scanf("%d %d", &n, &m);
    for(int i = 1; i <= n; i++){
        for(int j = 1; j <= n; j++){
            dist[i][j] = 1e9;
        }
        dist[i][i] = 0;
    }
    for(int i = 0; i < m; i++){
        int a, b, c;
        scanf("%d %d %d", &a, &b, &c);
        dist[a][b] = min(dist[a][b], c);
    }
    floyd(n);
    for(int i = 1; i <= n; i++){
        for(int j = 1; j <= n; j++){
            printf("dist[%d][%d] = ", i, j);
            if(dist[i][j] == 1e9){
                printf("infinito\n");
            }else{
                printf("%d\n", dist[i][j]);
            }
        }
        printf("\n");
    }
    return 0;
}
```

```
Tarjan (tìm Strong Connected Components)
/* Complexity: O(E + V)
 Tarjan's algorithm for finding strongly connected
components.
 *d[i] = Discovery time of node i. (Initialize to -1)
 *low[i] = Lowest discovery time reachable from node
```

```
 i. (Doesn't need to be initialized)
 *scc[i] = Strongly connected component of node i. (Doesn't
 need to be initialized)
 *s = Stack used by the algorithm (Initialize to an empty
 stack)
 *stacked[i] = True if i was pushed into s. (Initialize to
 false)
 *ticks = Clock used for discovery times (Initialize to 0)
 *current_scc = ID of the current_scc being discovered
 (Initialize to 0)
*/
vector<int> g[MAXN];
int d[MAXN], low[MAXN], scc[MAXN];
bool stacked[MAXN];
stack<int> s;
int ticks, current_scc;
void tarjan(int u){
  d[u] = low[u] = ticks++;
  s.push(u);
  stacked[u] = true;
  const vector<int> &out = g[u];
  for (int k=0, m=out.size(); k<m; ++k){
    const int &v = out[k];
    if (d[v] == -1){
      tarjan(v);
      low[u] = min(low[u], low[v]);
    }else if (stacked[v]){
      low[u] = min(low[u], low[v]);
    }
  }
  if (d[u] == low[u]){
    int v;
    do{
      v = s.top();
      s.pop();
      stacked[v] = false;
      scc[v] = current_scc;
    }while (u != v);
    current_scc++;
  }
}
```

```
Articulation Points (or Cut Vertices) in a Graph
// A C++ program to find articulation points in an undirected graph
#include<iostream>
#include <list>
#define NIL -1
using namespace std;

// A class that represents an undirected graph
class Graph
{
    int V;    // No. of vertices
    list<int> *adj;    // A dynamic array of adjacency lists
    void APUtil(int v, bool visited[], int disc[], int low[],
                int parent[], bool ap[]);
public:
    Graph(int V);   // Constructor
    void addEdge(int v, int w);   // function to add an edge to graph
    void AP();    // prints articulation points
};

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

void Graph::addEdge(int v, int w)
{
    adj[v].push_back(w);
```

```
    adj[w].push_back(v);  // Note: the graph is undirected
}

// A recursive function that find articulation points using DFS traversal
// u --> The vertex to be visited next
// visited[] --> keeps tract of visited vertices
// disc[] --> Stores discovery times of visited vertices
// parent[] --> Stores parent vertices in DFS tree
// ap[] --> Store articulation points
void Graph::APUtil(int u, bool visited[], int disc[],
                                      int low[], int parent[], bool ap[])
{
    // A static variable is used for simplicity, we can avoid use of static
    // variable by passing a pointer.
    static int time = 0;

    // Count of children in DFS Tree
    int children = 0;

    // Mark the current node as visited
    visited[u] = true;

    // Initialize discovery time and low value
    disc[u] = low[u] = ++time;

    // Go through all vertices aadjacent to this
    list<int>::iterator i;
    for (i = adj[u].begin(); i != adj[u].end(); ++i)
    {
        int v = *i;  // v is current adjacent of u

        // If v is not visited yet, then make it a child of u
        // in DFS tree and recur for it
        if (!visited[v])
        {
            children++;
            parent[v] = u;
            APUtil(v, visited, disc, low, parent, ap);

            // Check if the subtree rooted with v has a connection to
            // one of the ancestors of u
            low[u]  = min(low[u], low[v]);

            // u is an articulation point in following cases

            // (1) u is root of DFS tree and has two or more chilren.
            if (parent[u] == NIL && children > 1)
               ap[u] = true;

            // (2) If u is not root and low value of one of its child is more
            // than discovery value of u.
            if (parent[u] != NIL && low[v] >= disc[u])
               ap[u] = true;
        }

        // Update low value of u for parent function calls.
        else if (v != parent[u])
            low[u]  = min(low[u], disc[v]);
    }
}

// The function to do DFS traversal. It uses recursive function APUtil()
void Graph::AP()
{
    // Mark all the vertices as not visited
    bool *visited = new bool[V];
    int *disc = new int[V];
    int *low = new int[V];
    int *parent = new int[V];
    bool *ap = new bool[V]; // To store articulation points

    // Initialize parent and visited, and ap(articulation point) arrays
    for (int i = 0; i < V; i++)
    {
```

```
        parent[i] = NIL;
        visited[i] = false;
        ap[i] = false;
    }

    // Call the recursive helper function to find articulation points
    // in DFS tree rooted with vertex 'i'
    for (int i = 0; i < V; i++)
        if (visited[i] == false)
            APUtil(i, visited, disc, low, parent, ap);

    // Now ap[] contains articulation points, print them
    for (int i = 0; i < V; i++)
        if (ap[i] == true)
            cout << i << " ";
}

// Driver program to test above function
int main()
{
    // Create graphs given in above diagrams
    cout << "\nArticulation points in first graph \n";
    Graph g1(5);
    g1.addEdge(1, 0);
    g1.addEdge(0, 2);
    g1.addEdge(2, 1);
    g1.addEdge(0, 3);
    g1.addEdge(3, 4);
    g1.AP();

    cout << "\nArticulation points in second graph \n";
    Graph g2(4);
    g2.addEdge(0, 1);
    g2.addEdge(1, 2);
    g2.addEdge(2, 3);
    g2.AP();

    cout << "\nArticulation points in third graph \n";
    Graph g3(7);
    g3.addEdge(0, 1);
    g3.addEdge(1, 2);
    g3.addEdge(2, 0);
    g3.addEdge(1, 3);
    g3.addEdge(1, 4);
    g3.addEdge(1, 6);
    g3.addEdge(3, 5);
    g3.addEdge(4, 5);
    g3.AP();

    return 0;
}
```

```
Disjoint Set
#include <iostream>

using namespace std;

const int N = 5000 + 1;

int n, m, q, pre[N], rank[N] = { 0 };

int get_father(int x) {
  if (pre[x] == x)
    return x;
  return pre[x] = get_father(pre[x]);
}

void merge(int x, int y) {
  x = get_father(x);
  y = get_father(y);
  if (rank[x] < rank[y])
```

```
    swap(x, y);
  if (rank[x] == rank[y])
    ++ rank[x];
  pre[y] = x;
}

inline bool is_relative(int x, int y) {
  return get_father(x) == get_father(y);
}

void init_disjoint_set() {
  for (int i = 1; i <= n; ++i)
    pre[i] = i;
}

void print_pre() {
  for (int i = 1; i <= n; ++i)
    cout << "pre[" << i << "] = " << pre[i] << endl;
}

int main() {
  cin >> n >> m >> q;
  init_disjoint_set();
  int x, y;
  for (int i = 0; i < m; ++i) {
    cin >> x >> y;
    merge(x, y);
    print_pre();
  }
  for (int i = 0; i < q; ++i) {
    cin >> x >> y;
    cout << (is_relative(x, y) ? "Yes" : "No") << endl;
  }
  return 0;
}
```

```
Cặp ghép cực đại
#include <stdio.h>
#include <vector>
#include <iostream>
#include <algorithm>
using namespace std;

const int N = 102;
int n, m, Assigned[N];
int Visited[N], t=0;
vector<int> a[N];

bool visit(int u){
  if (Visited[u]==t) return false;
  Visited[u] = t;
  for (int i=0;int v=a[u][i];i++) {
    if (!Assigned[v] || visit(Assigned[v])) {
      Assigned[v] = u;
      return true;
    }
  }
  return false;
}

main() {
  freopen("input.txt", "r", stdin);
    scanf("%d%d", &m, &n);
    int x, y;
    while (scanf("%d%d", &x, &y) > 0)
        a[x].push_back(y);
    for (int i=1; i<=m; i++)
        a[i].push_back(0);

    int Count = 0;
    for (int i=1; i<=m; i++) {
```

```
        t++;
        Count += visit(i);
    }
    printf("%d\n", Count);
    for (int i=1; i<=n; i++)
    if (int j=Assigned[i])
    printf("%d %d\n", j, i);
}
```

```
KMP String search
#include <bits/stdc++.h>
#define BachNX
#define _for(i,a,b) for (int i=(a),_b_=(b),_d_=(a<b?1:-1);i!=_b_;i+=_d_)

using namespace std;
typedef long long LL;
typedef unsigned long long ULL;

void prepair(char *pat, int *lps) {
  int len = 0, i = 1, N = strlen(pat);
  lps[0] = 0;
  while (i < N) {
    if (pat[i] == pat[len])
      lps[i++] = ++len;
    else {
      // Not matching, fallback
      if (len) {
        len = lps[len-1];
      } else {
        lps[i++] = 0;
      }
    }
  }
  cout << "lps:" << endl;
  _for(i,0,N) cout << lps[i] << ' ';
  cout << endl;
}

int search(char s[], char pat[]) {
  int lps[1000];
  prepair(pat, lps);
  int i = 0, j = 0, N = strlen(s), M = strlen(pat);
  while (i < N) {
    if (s[i] == pat[j]) {
      i++;
      j++;
      if (j == M) return i-j;
    } else {
      if (j) {
        j = lps[j-1];
      } else {
        i++;
      }
    }
  }
  return -1;
}

int main(){
//  freopen("input.txt","r",stdin);
//  freopen("output.txt","w",stdout);
  ios::sync_with_stdio(false);
  cin.tie(NULL);

  char s[] = "AABAACAADAABAB324CABABABABABCABABCADEFAAABABCDEABABCD";
  char pat[] = "ABCDEABABCD";
  cout << "Search result: " << search(s, pat);
  return 0;
}
```