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

struct SegmentTreeNode {
    SegmentTreeNode *left, *right;
    int L, R;
    int value;

    SegmentTreeNode() {
        left = right = NULL;
        L = R = -1;
        value = 0;
    }
};

struct SegmentTree {
    vector<int> array;
    SegmentTreeNode* root;

    SegmentTree() {
        root = new SegmentTreeNode();
    }

    void destroy() {
        _destroy(root);
    }

    void _destroy(SegmentTreeNode* node) {
        if (!node)
            return;
        _destroy(node->left);
        _destroy(node->right);
        delete node;
    }

    void build() {
        _build(0, array.size(), root);
    }

    int query(int x, int y) {
        return _query(x, y, root);
    }

    void _build(int L, int R, SegmentTreeNode* node) {
        node->L = L;
        node->R = R;

        if (L+1 == R) {
            node->value = array[L];
            return;
        }

        int M = (L + R) / 2;

        if (!node->left) node->left = new SegmentTreeNode();
        if (!node->right) node->right = new SegmentTreeNode();

        _build(L, M, node->left);
        _build(M, R, node->right);

        node->value = node->left->value + node->right->value;
    }

    int _query(int x, int y, SegmentTreeNode* node) {
        if (!node)
            return 0;
        if (y <= node->L || node->R <= x)
            return 0;
        if (x <= node->L && node->R <= y)
            return node->value;

        int a = _query(x, y, node->left);
        int b = _query(x, y, node->right);

        return a + b;
    }
};
```

```
#include <iostream>
#include <algorithm>
#include <vector>
#define N 10000000
using namespace std;

vector<long long> array;
long long dp[N], len[N], pos[N];
const long long INF = 0x3fffffffffffffff;

vector<long long> findLIS()
{
    int n = array.size();

    for (int i = 0; i <= n; i++)
        len[i] = INF;
    for (int i = 0; i < n; i++) {
        int k = lower_bound(len+1, len+n+1, array[i]) - len;
        dp[i] = k;
        len[k] = array[i];
    }
    long long L = 0;
    for (int i = 0; i < n; i++)
        L = max(L, dp[i]);

    vector<long long> lis = vector<long long>(L, 0);
    int tmp = L;
    for (int i = n-1; i >= 0; i--) {
        if (dp[i] == tmp) {
            lis[tmp-1] = array[i];
            tmp--;
        }
    }
    return lis;
}
```

```
int LCIS(vector<int>& A, vector<int>& B)
{
    vector<int> C(B.size(), 0);

    for (int i = 0; i < A.size(); i++) {
        int cur = 0;
        for (int j = 0; j < B.size(); j++) {
            if (A[i] == B[j] && cur+1 > C[j]) {
                C[j] = cur + 1;
            } else if (A[i] > B[j] && cur < C[j]) {
                cur = C[j];
            }
        }
    }
    int ret = 0;
    for (int i = 0; i < C.size(); i++) {
        ret = max(ret, C[i]);
    }
    return ret;
}
```

```cpp
#include <iostream>
#include <vector>
#include <cstdio>
#include <cstring>
#define N 15
using namespace std;

const int INF = 1000;
int graph[N][N];
int dp[1<<N][N];

int _TSP(int mask, int x, int n)
{
    if (!(mask & (mask-1)))
        return 0;
    if (dp[mask][x] != -1)
        return dp[mask][x];
    int res = INF;
    for (int i = 0; i < n; i++) {
        if (i == x)
            continue;
        if (mask & (1<<i)) {
            int tmp = _TSP(mask^(1<<x), i, n) + graph[x][i];
            res = min(res, tmp);
        }
    }
    return dp[mask][x] = res;
}

int TSP(int n)
{
    int res = INF;
    int e = (1<<n) - 1;
    for (int i = 0; i < n; i++) {
        memset(dp, -1, sizeof(dp));
        int tmp = _TSP(e, i, n);
        res = min(res, tmp);
    }
    return res;
}
```

```cpp
#include <vector>
#include <cstring>
#define N 1000
using namespace std;

vector<int> graph[N];
int match[N];
bool vst[N];

bool _bipartite(int x)
{
    for (int i = 0; i < graph[x].size(); i++) {
        int y = graph[x][i];
        if (!vst[y]) {
            vst[y] = true;
            if (match[y] == -1 || _bipartite(match[y])) {
                match[y] = x;
                return true;
            }
        }
    }
    return false;
}

void bipartite(int n)
{
    memset(match, -1, sizeof(match));
    for (int i = 0; i < n; i++) {
        memset(vst, false, sizeof(vst));
        _bipartite(i);
    }
}
```

```
vector<int> adj[N];
int clk[N], low[N], int t;

vector<int> ap;
vector< pair<int, int> > bridge;

void dfs(int cur, int parent)
{
    int child = 0;
    bool flag = false;

    low[cur] = clk[cur] = t;
    t++;

    for (int i = 0; i < adj[cur].size(); i++) {
        int next = adj[cur][i];
        if (!clk[next]) {
            child++;
            dfs(next, cur);
            low[cur] = min(low[cur], low[next]);

            if (low[next] >= clk[cur])
                flag = true;
            if (low[next] > clk[cur])
                bridge.push_back(make_pair(cur, next));
        } else if (next != parent) {
            low[cur] = min(low[cur], clk[next]);
        }
    }
    if (parent == -1 && child >= 2)
        ap.push_back(cur);
    else if (parent != -1 && flag)
        ap.push_back(cur);
}

void tarjan(int n)
{
    t = 1;
    ap.clear();
    bridge.clear();
    memset(clk, 0, sizeof(clk));
    memset(low, 0, sizeof(low));
    dfs(0, -1);
}
```

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

vector<int> build_fail_function(string S)
{
    int len = S.length(), cur;
    vector<int> fail = vector<int>(len, 0);

    cur = fail[0] = -1;
    for (int i = 1; i < len; i++) {
        while (cur != -1 && S[cur+1] != S[i]) {
            cur = fail[cur];
        }
        if (S[cur+1] == S[i])
            cur++;
        fail[i] = cur;
    }
    return fail;
}

vector<int> match(string A, string B)
{
    int lenA = A.length(), lenB = B.length();
    int cur = -1;
    vector<int> pos;
    vector<int> fail = build_fail_function(B);
    for (int i = 0; i < lenA; i++) {
        while (cur != -1 && B[cur+1] != A[i]) {
            cur = fail[cur];
        }
        if (B[cur+1] == A[i])
            cur++;
        if (cur == lenB-1) {
            pos.push_back(i);
            cur = fail[cur];
        }
    }
    return pos;
}
```

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

struct TrieNode {
    TrieNode* to[26];
    bool word;

    TrieNode() {
        word = false;
        memset(to, 0, sizeof(to));
    }
};

struct Trie {
    TrieNode* root;

    Trie () {
        root = new TrieNode();
    }

    ~Trie() {
        freeNode(root);
    }

    void freeNode(TrieNode* ptr) {
        for (int i = 0; i < 26; i++) {
            if (ptr->to[i])
                freeNode(ptr->to[i]);
        }
        delete ptr;
    }

    void insert(string& s) {
        TrieNode* ptr = root;
        for (int i = 0; i < s.length(); i++) {
            if (!ptr->to[s[i]-'a'])
                ptr->to[s[i]-'a'] = new TrieNode();
            ptr = ptr->to[s[i]-'a'];
        }
        ptr->word = true;
    }

    bool contain(string& s) {
        TrieNode* ptr = root;
        for (int i = 0; i < s.length(); i++) {
            if (!ptr->to[s[i]-'a'])
                return false;
            ptr = ptr->to[s[i]-'a'];
        }
        return ptr->word;
    }
};
```

```
#include <iostream>
#include <cstdio>
#include <cstring>
#include <vector>
#include <queue>
#include <stack>
#define N 2000
using namespace std;

vector <int> adj[N]; //adj list
vector <int> rev[N]; //adj list reverse
vector <int> finish; //the order of leaving 1st dfs
vector <int> scc[N];
int group[N];
bool vst[N];

void dfs1(int x) //first dfs to get the order of leaving dfs
{
    vst[x] = true;
    for (int i = 0; i < adj[x].size(); i++)
        if (!vst[adj[x][i]])
            dfs1(adj[x][i]);

    finish.push_back(x);
}

void dfs2(int x, int c) //use the reverse adj list to dfs
{
    scc[c].push_back(x);
    group[x] = c;

    vst[x] = true;
    for (int i = 0; i < rev[x].size(); i++)
        if (!vst[rev[x][i]])
            dfs2(rev[x][i], c);
}
```

```
void find_scc(int n)
{
        memset(vst, false, sizeof(vst));
        for (int i = 1; i <= n; i++)// first dfs
            if (!vst[i])
                dfs1(i);

        int c = 0;
        memset(vst, false, sizeof(vst));
        for (int i = finish.size()-1; i >= 0; i--) //second dfs using the order
            if (!vst[finish[i]])
                dfs2(finish[i], c++);
}
```

```
#include <iostream>
#include <cstdio>
#include <cstring>
#include <queue>
#include <algorithm>
#define N 300
using namespace std;

long long adj[N][N];
int pre[N];
long long vst[N];
int enterNode, exitNode, edgeNum, timeStamp;

bool find_path()
{
    queue<int> q;
    q.push(enterNode);
    bool ok = false;
    while (!q.empty()) {
        int currNode = q.front(); q.pop();
        if (currNode == exitNode) {
            ok = true;
            break;
        }
        for (int i = 0; i < N; i++) {
            if (adj[currNode][i] && vst[i] < timeStamp) {
                vst[i] = timeStamp;
                q.push(i);
                pre[i] = currNode;
            }
        }
    }
    timeStamp++;
    return ok;
}

long long find_min()
{
    int currNode = pre[exitNode];
    long long ret = adj[pre[exitNode]][exitNode];
    while (currNode != enterNode) {
        ret = min(ret, adj[pre[currNode]][currNode]);
        currNode = pre[currNode];
    }
    return ret;
}

void flow(long long x)
{
    int currNode = exitNode;
    while (currNode != enterNode) {
        adj[pre[currNode]][currNode] -= x;
        adj[currNode][pre[currNode]] += x;
        currNode = pre[currNode];
    }
}

long long find_maxflow()
{
    long long ans = 0;
    timeStamp = 1;
    while (find_path()) {
        long long minEdge = find_min();
        flow(minEdge);
        ans += minEdge;
    }
    return ans;
}
```