Problem A
Maximum Subarray Sum Revisited

Time limit: 1 second

Memory limit: 2048 megabytes

Problem Description

Given an array of n integers $a_1, a_2, \ldots, a_n$, find a non-empty and contiguous subarray with the largest sum. Your program needs to calculate this largest sum.

Input Format

The first line of the input contains an integer n denoting the length of the array. The second line of the input contains n space-separated integers $a_1, a_2, \ldots, a_n$.

Output Format

Output the maximum subarray sum in one line.

Technical Specification

- $1 \leq n \leq 3 \times 10^5$
- $-10^9 \leq a_i \leq 10^9$ for $i = 1, 2, \ldots, n$

Scoring

1. (4 points) $1 \leq n \leq 1000$
2. (16 points) No additional constraints.

Sample Input 1

```
5
-5 6 -3 4 -1
```

Sample Output 1

```
7
```

Sample Input 2

```
4
1000000000 1000000000 1000000000 1000000000
```

Sample Output 2

```
4000000000
```

Sample Input 3

```
7
-3 -5 -100 -2 -23 -15 -9
```

Sample Output 3

```
-2
```

Sample Input 4

```
8
25 -32 8 13 47 -44 45 -100
```

Sample Output 4

```
69
```

Hint

```
// Finding the single source shortest paths from vertex 0 (0-indexed)
// Assume undirected graph
#include <iostream>
#include <queue>
#include <vector>
using namespace std;

int main() {
    int n, m;
    cin >> n >> m;
    vector<vector<int>> G(n);
    for (int i = 0; i < m; i++) {
    int u, v;
    cin >> u >> v;
    G[u].push_back(v);
    G[v].push_back(u);
    }
    vector<int> dist(n, -1);
    queue<int> q;
    dist[0] = 0;
    q.push(0);
    while (!q.empty()) {
    int u = q.front();
    q.pop();
    for (int v : G[u])
        if (dist[v] == -1) {
        dist[v] = dist[u] + 1;
        q.push(v);
        }
    }
    // dist stores the shortest distance from vertex 0
}
```

Listing 1: BFS algorithm

```
// Perform DFS from vertex 0
// Assume undirected graph
#include <iostream>
#include <queue>
#include <vector>
using namespace std;

void dfs(int u, vector<vector<int>> &G, vector<int> &visited) {
    visited[u] = 1;
    for (int v : G[u])
    if (!visited[v])
        dfs(v, G, visited);
}

int main() {
    int n, m;
    cin >> n >> m;
    vector<vector<int>> G(n);
    vector<int> visited(n);
    for (int i = 0; i < m; i++) {
    int u, v;
    cin >> u >> v;
    G[u].push_back(v);
    G[v].push_back(u);
    }
    dfs(0, G, visited);
}
```

Listing 2: DFS algorithm

```
#include <algorithm>
#include <queue>
#include <vector>
using namespace std;

struct Dinic {
    struct edge {
    int to, cap, flow, rev;
    };
    static constexpr int MAXN = 1000, MAXF = 1e9;
    vector<edge> v[MAXN];
    int top[MAXN], deep[MAXN], side[MAXN], s, t;
    void make_edge(int s, int t, int cap) {
    v[s].push_back({t, cap, 0, (int)v[t].size()});
    v[t].push_back({s, 0, 0, (int)v[s].size() - 1});
    }
    int dfs(int a, int flow) {
    if (a == t || !flow)
        return flow;
    for (int &i = top[a]; i < v[a].size(); i++) {
        edge &e = v[a][i];
        if (deep[a] + 1 == deep[e.to] && e.cap - e.flow) {
        int x = dfs(e.to, min(e.cap - e.flow, flow));
        if (x) {
            e.flow += x, v[e.to][e.rev].flow -= x;
            return x;
        }
        }
    }
    deep[a] = -1;
    return 0;
    }
    bool bfs() {
    queue<int> q;
    fill_n(deep, MAXN, 0);
    q.push(s), deep[s] = 1;
    int tmp;
    while (!q.empty()) {
        tmp = q.front(), q.pop();
        for (edge e : v[tmp])
        if (!deep[e.to] && e.cap != e.flow)
            deep[e.to] = deep[tmp] + 1, q.push(e.to);
    }
    return deep[t];
    }
    int max_flow(int _s, int _t) {
    s = _s, t = _t;
    int flow = 0, tflow;
    while (bfs()) {
        fill_n(top, MAXN, 0);
        while ((tflow = dfs(s, MAXF)))
        flow += tflow;
    }
    return flow;
    }
    void reset() {
    fill_n(side, MAXN, 0);
    for (auto &i : v)
        i.clear();
    }
};
```

Listing 3: Dinic algorithm