ครอบคลุมทั่วถิ่น (Cover)

Subtask 1

ใน subtask นี้ เราจะนิยาม $dist[0][i]$ คือระยะทางที่สั้นที่สุดจาก A ไปเมืองที่ i และ $dist[1][i]$ คือระยะทางที่สั้นที่สุดจาก B ไปเมืองที่ i ซึ่งเราสามารถทำได้ด้วย **breadth first search (BFS)** หรือ **depth first search (DFS)** เพื่อหาค่าอาเรย์ $dist$ ส่วนในขั้นตอนการตอบคำถาม เราจะหาค่ามากที่สุดของ $dist[0][t]$ เมื่อ $dist[1][t] \leq X_i$ โดยเราจะไล่ลูปค่า t ตั้งแต่ 1 ถึง N หากไม่มีเมืองใดที่สอดคล้องเงื่อนไข เราจะตอบ 0

Time Complexity: $\mathcal{O}(NP)$

Subtask 2

ใน subtask นี้ เราจะสังเกตเพิ่มเติมต่อจาก subtask 1 ได้ว่า หาก $X_i \geq max_{1 \leq t \leq N} dist[0][t]$ จะตอบ 0 ในทุกกรณี นอกจากนั้นจะตอบ $max_{1 \leq t \leq N} dist[0][t]$

Time Complexity: $\mathcal{O}(N + P)$

Subtask 3

ใน subtask นี้จะได้ว่ากราฟที่ได้รับมาจะเป็นกราฟเส้นตรง ซึ่งเราจะมีตัวชี้ทางฝั่งซ้าย (pl) และตัวชี้ทางฝั่งขวา (pr) เพื่อใช้ในการเก็บค่า max ของระยะทางมากที่สุด แล้วเราจะค่อยๆ BFS จากจุด B โดยค่อยๆ ปรับระยะทางขึ้นไป 1

ในขั้นตอนการตอบนั้น เราจะนิยาม $ans[t]$ แทนคำตอบเมื่อ $X_i = t$ ซึ่งเราจะขยับ pl ไปทางขวาจนกว่า $dist[1][pl] < t$ และเราจะขยับ pr ไปทางซ้ายจนกว่า $dist[1][pr] < t$ ซึ่ง $ans[t]$ จะมีค่าเป็น $max(dist[0][pl], dist[0][pr])$ หาก pl อยู่ทางขวาของ pr เราจะเก็บค่าเป็น 0

Time Complexity: $\mathcal{O}(N + P)$

Subtask 4

ใน subtask นี้จะได้ว่ากราฟที่ได้รับมาจะเป็นต้นไม้ที่มี leaf ไม่เกิน 1 000 ซึ่งเราสามารถทำวิธีในลักษณะเดียวกับ subtask 3 ได้ โดยเราจะนิยาม $leaf[t]$ แทนตัวชี้ที่ leaf ที่ t และจะนิยาม $par[t]$ คือเมืองที่เป็นพ่อของเมือง t เมื่อ fix root ไว้ที่เมือง A

ในขั้นตอนการตอบนั้นจะนิยาม $ans[c]$ แทนคำตอบเมื่อ $X_i = c$ เราจะดูทุก $leaf$ และปรับ $leaf[t] := par[leaf[t]]$ เมื่อ $dist[1][leaf[t]] \leq c$ ทำให้ $ans[c] = max_{1 \leq t \leq T} dist[0][leaf[t]]$ ซึ่งตัวชี้ทุกตัวจะมี

การปรับค่าไม่เกิน $N-1$ ครั้ง จึงสามารถทำได้ใน Time Complexity $\mathcal{O}(NT)$ เมื่อ T คือจำนวน leaf

Time Complexity: $\mathcal{O}(NT+P)$ เมื่อ T คือจำนวน leaf

Subtask 5

ใน subtask นี้จะได้ว่ากราฟที่ได้รับมาจะเป็นกราฟวงกลม(cycle graph) ซึ่งเราสามารถดัดแปลงต่อจาก subtask 3 ได้ เราจะนิยาม $ans[t]$ แทนคำตอบเมื่อ $X_i = t$ ซึ่งเราจะมี pl และ pr ชี้ที่เมือง B เราจะขยับ pl ไปทางขวาจนกว่า $dist[1][pl] < t$ และเราจะขยับ pr ไปทางซ้ายจนกว่า $dist[1][pr] < t$ ซึ่ง $ans[t]$ จะมีค่าเป็น $max(dist[0][pl], dist[0][pr])$ และเราจะหยุดขยับ pl หรือ pr เมื่อ $pl = A$ หรือ $pr = A$

Time Complexity: $\mathcal{O}(N+P)$

Subtask 6

ใน subtask นี้จะเห็นว่ากราฟสามารถเป็รูปแบบอย่างไรก็ได้ ดังนั้นเราจะดัดแปลงต่อจาก subtask 1 โดยที่จะเพิ่ม $da[t]$ แทนจำนวนเมืองที่มีระยะทางสั้นที่สุดที่ห่างจาก A คือ t พอดี และยังไม่มีสัญญาณจากเมือง B ส่งมาถึง ซึ่งเราจะลูป d ตั้งแต่ 0 ถึง $N-1$ เพื่อค่อยๆ ลด $da[dist[0][T]]$ ตามลำดับเมื่อ T คือเมืองทุกเมืองที่ $dist[1][T] = d$ ส่วนค่า $ans[d]$ จะหาได้จากการลูปบนเพื่อดูว่าอาเรย์ da ช่องที่มากที่สุดที่ยังมีค่ามากกว่า 0 คือช่องใด ซึ่งทำให้ Time Complexity โดยรวมคือ $\mathcal{O}(1000N+P)$

Time Complexity: $\mathcal{O}(1000N+P)$

Subtask 7

ใน subtask นี้จะเห็นได้ว่าหากเราใช้วิธีเดียวกับ subtask 6 จะทำไม่ทันเวลาเนื่องจากคำตอบอยู่ในช่วง $[0, N-1]$ เราจึงจำเป็นต้องหา data structure ที่สามารถหาค่า max ได้เร็วๆ ซึ่งสามารถทำได้หลายวิธี(set, multiset, segment tree, fenwick tree) ซึ่งเราจะใช้ multiset ในการเฉลยครั้งนี้ เราจะนิยาม max_disc คือ multiset ที่เก็บค่า $dist[0][t]$ เมื่อ t คือทุกเมืองที่สัญญาณ B ยังมาไม่ถึง โดยเราจะทำในลักษณะเดียวกับ subtask 6 ส่วนค่าอาเรย์ ans จะสามารถหาได้จากการเรียก $*max\_disc.rbegin()$ ซึ่งสามารถทำได้ภายใน $\mathcal{O}(1)$ ซึ่งทำให้ Time Complexity โดยรวมจากการใช้ multiset คือ $\mathcal{O}(NlogN)$

Model solution

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

const int MAX_N = 1e5 + 5;

vector <int> graph[MAX_N];
int dist[2][MAX_N];
vector <int> nodes[MAX_N];
int ans[MAX_N];
```

```
void bfs(int t, int s) {
    queue <int> q;
    q.emplace(s);
    dist[t][s] = 0;
    while(!q.empty()) {
        int u = q.front();
        q.pop();

        for(auto v : graph[u]) {
            if(dist[t][v] == -1) {
                dist[t][v] = dist[t][u] + 1;
                q.push(v);
            }
        }
    }
}

int main() {
    cin.tie(nullptr)->sync_with_stdio(false);

    int N, M, P, A, B;
    cin >> N >> M >> P >> A >> B;

    while(M--) {
        int u, v;
        cin >> u >> v;

        graph[u].push_back(v);
        graph[v].push_back(u);
    }

    memset(dist, -1, sizeof(dist));

    bfs(0, A);
    bfs(1, B);

    multiset <int> max_dist;
    max_dist.insert(0);
    for(int i = 1; i <= N; i++) {
        nodes[dist[1][i]].push_back(i);
        max_dist.emplace(dist[0][i]);
    }
    for(int i = 0; i <= N; i++) {
        for(auto v : nodes[i]) {
```

```
            max_dist.erase(max_dist.lower_bound(dist[0][v]));
        }
        if(!max_dist.empty()) {
            ans[i] = *max_dist.rbegin();
        }
    }

    while(P--) {
        int X;
        cin >> X;

        cout << ans[X] << '\n';
    }
    return 0;
}
```

Time Complexity: $\mathcal{O}(NlogN + P)$