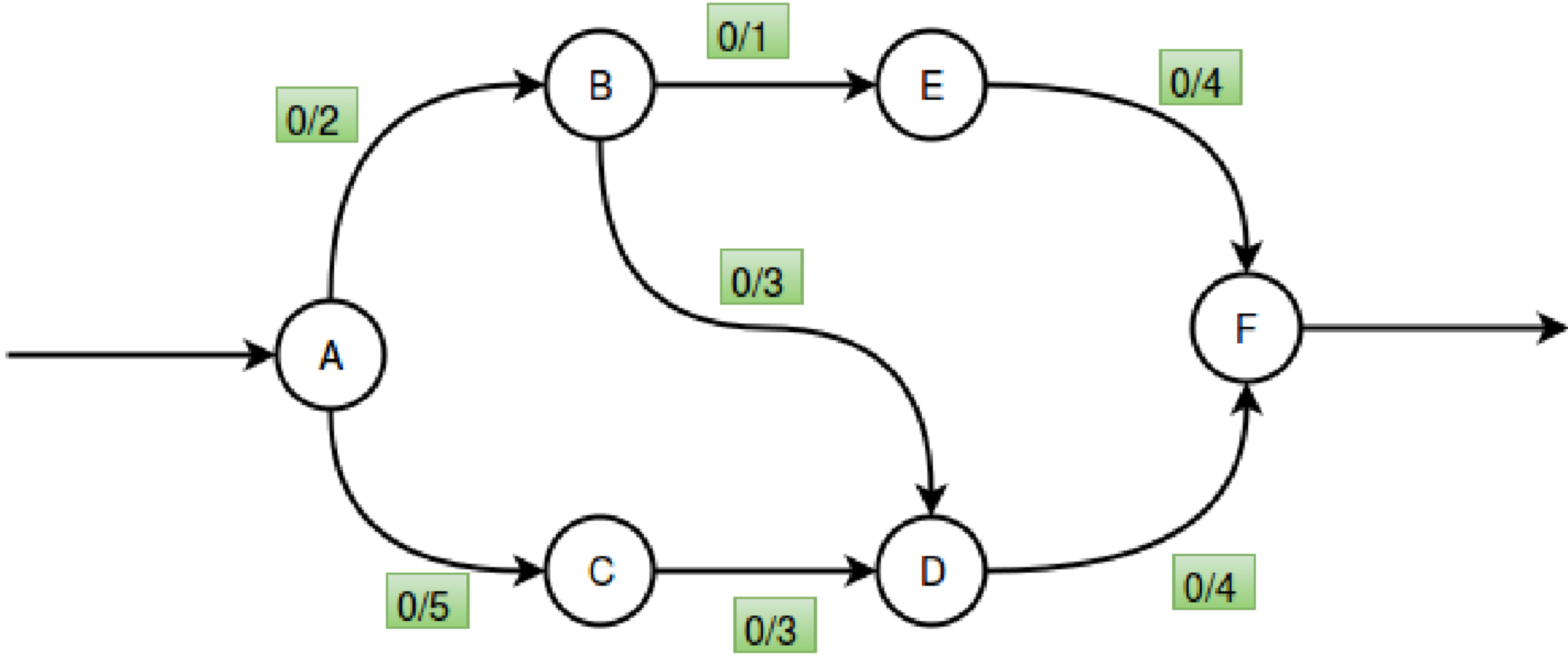
ম্যাক্সিমাম ফ্লো


A
B
C
D
E
F
0/2
0/1
0/4
0/3
0/5
0/3
0/4

Residual গ্রাফ

❝ **residual** ক্যাপাসিটি = এজ এর ক্যাপাসিটি – ব্যবহৃত ক্যাপাসিটি বা ফ্লো এর পরিমাণ ।

❝ ১. গ্রাফের প্রতিটা এজ (u,v) এর ক্যাপাসিটি হবে এজ টার **residual** ক্যাপাসিটির সমান।

২. প্রতি এজ (u,v) এর জন্য উল্টা এজ (v,u) এর **residual** ক্যাপাসিটি হবে (u,v) এজ এ ফ্লো এর সমান।

অগমেন্টেড পাথ

residual গ্রাফে আমরা যখন একটা পথ বের করি , সেটা হলো অগমেন্টেড পাথ |

Ford-Fulkerson

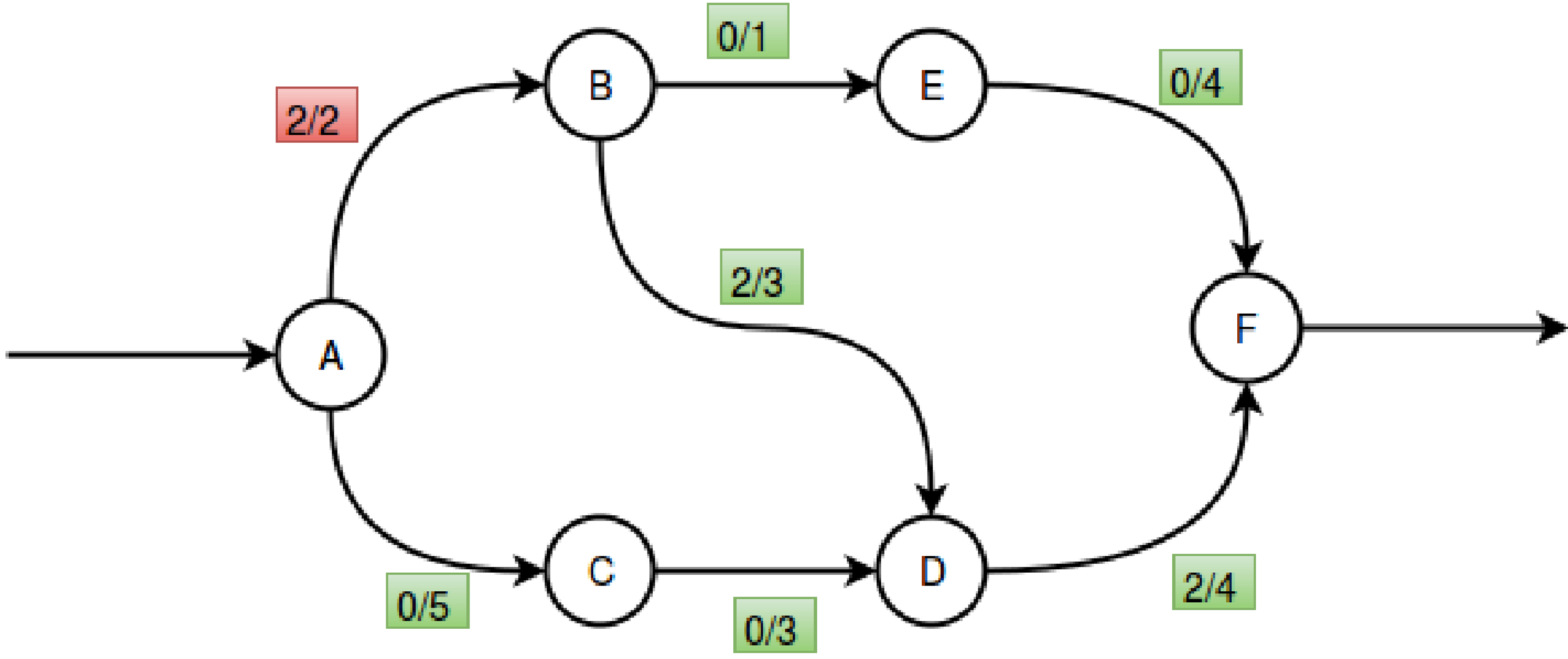
যতক্ষণ সম্ভব আমরা residual graph এ একটা অগমেন্টেড পাথ খুজে বের করবো এবং সেই পথে ফ্লো পাঠিয়ে দিবো!

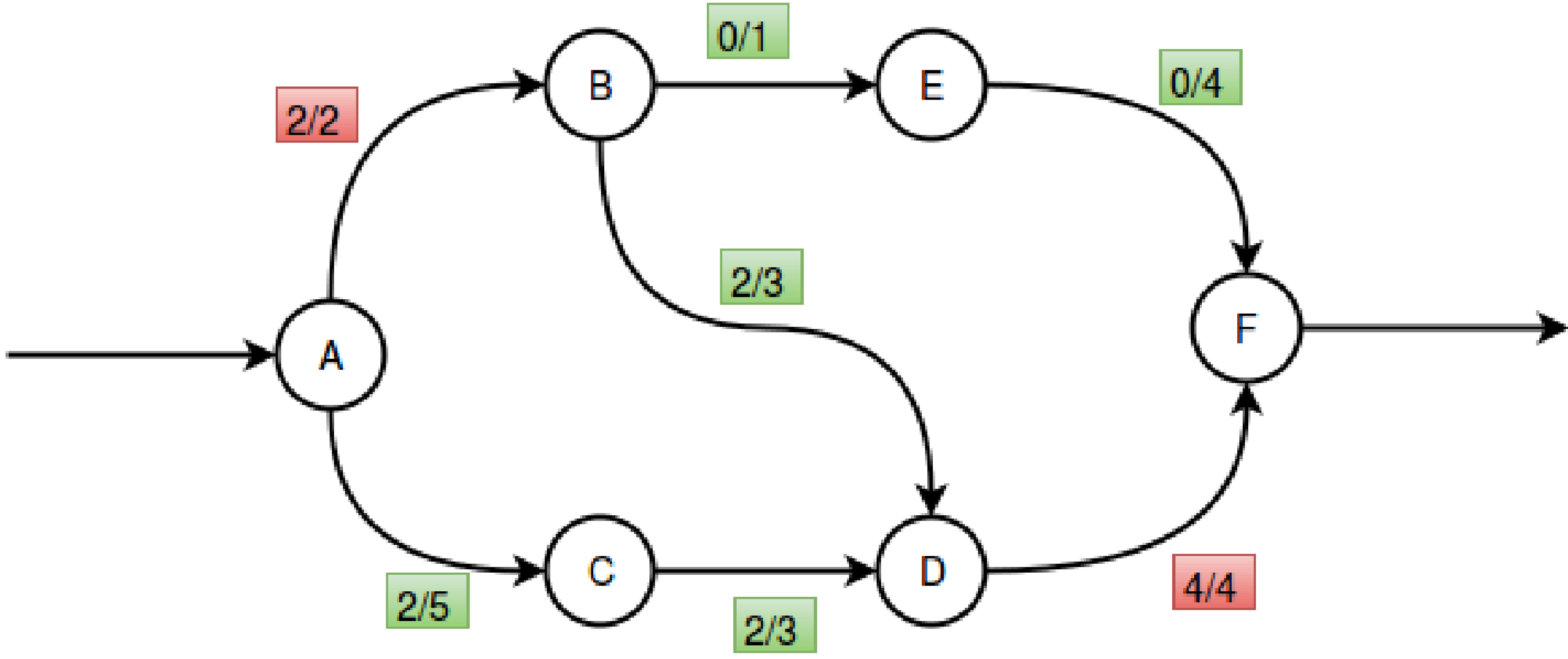
while (যতক্ষন Aug Path পাবো)

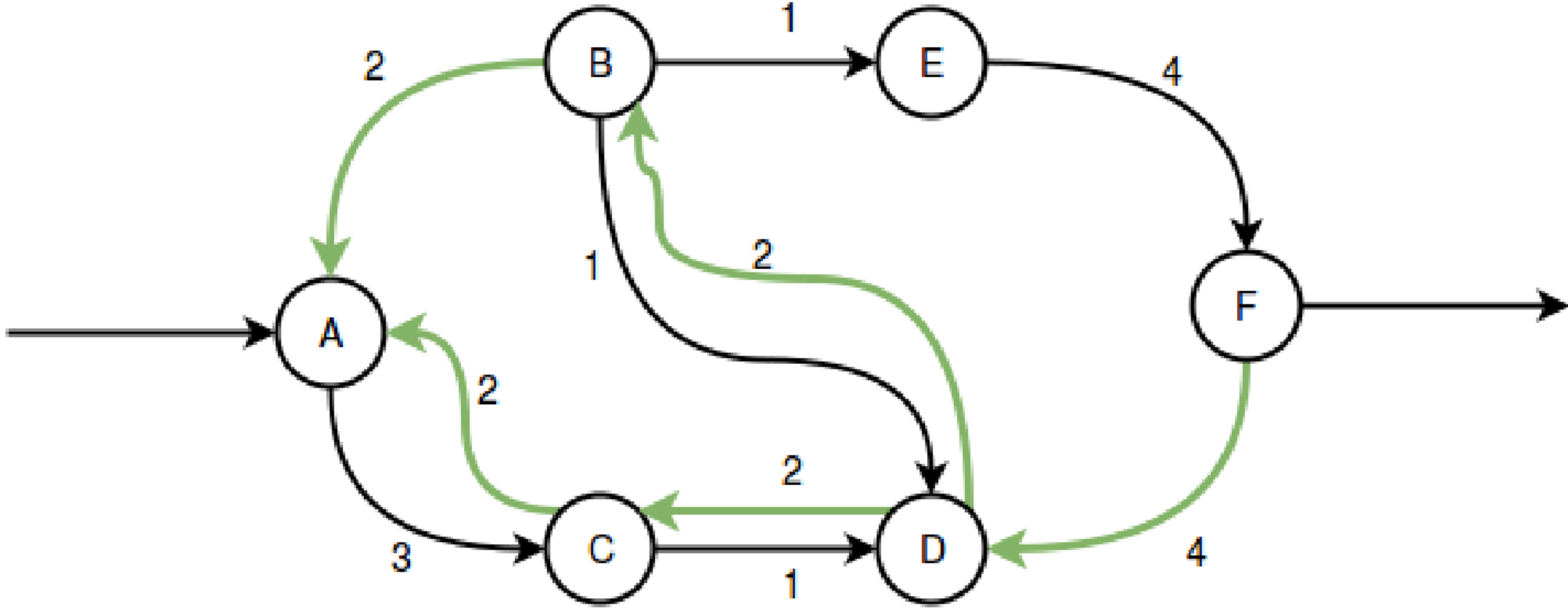
{

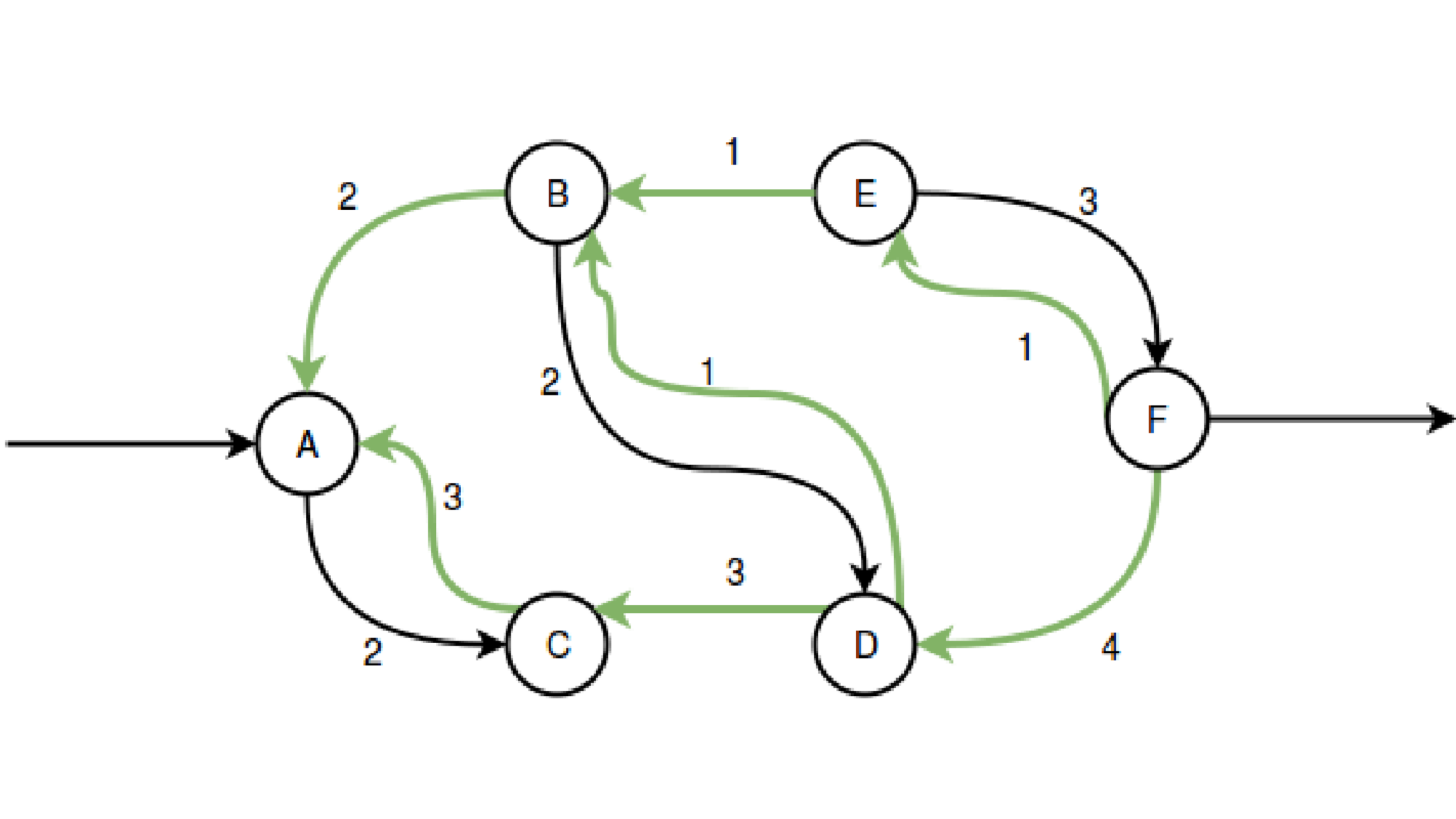
maxflow += Avg path এর flow

}


A
B
C
D
E
F
2/2
0/1
0/4
2/3
0/5
0/3
2/4


A
B
C
D
E
F
2/2
0/1
0/4
2/3
2/5
2/3
4/4


A
B
C
D
E
F
1
2
3
4
1
2
2
2
1
4


A
B
C
D
E
F
1
2
3
2
1
1
3
3
2
4

এডমন্ড কার্প

Augment Path
Find = BFS

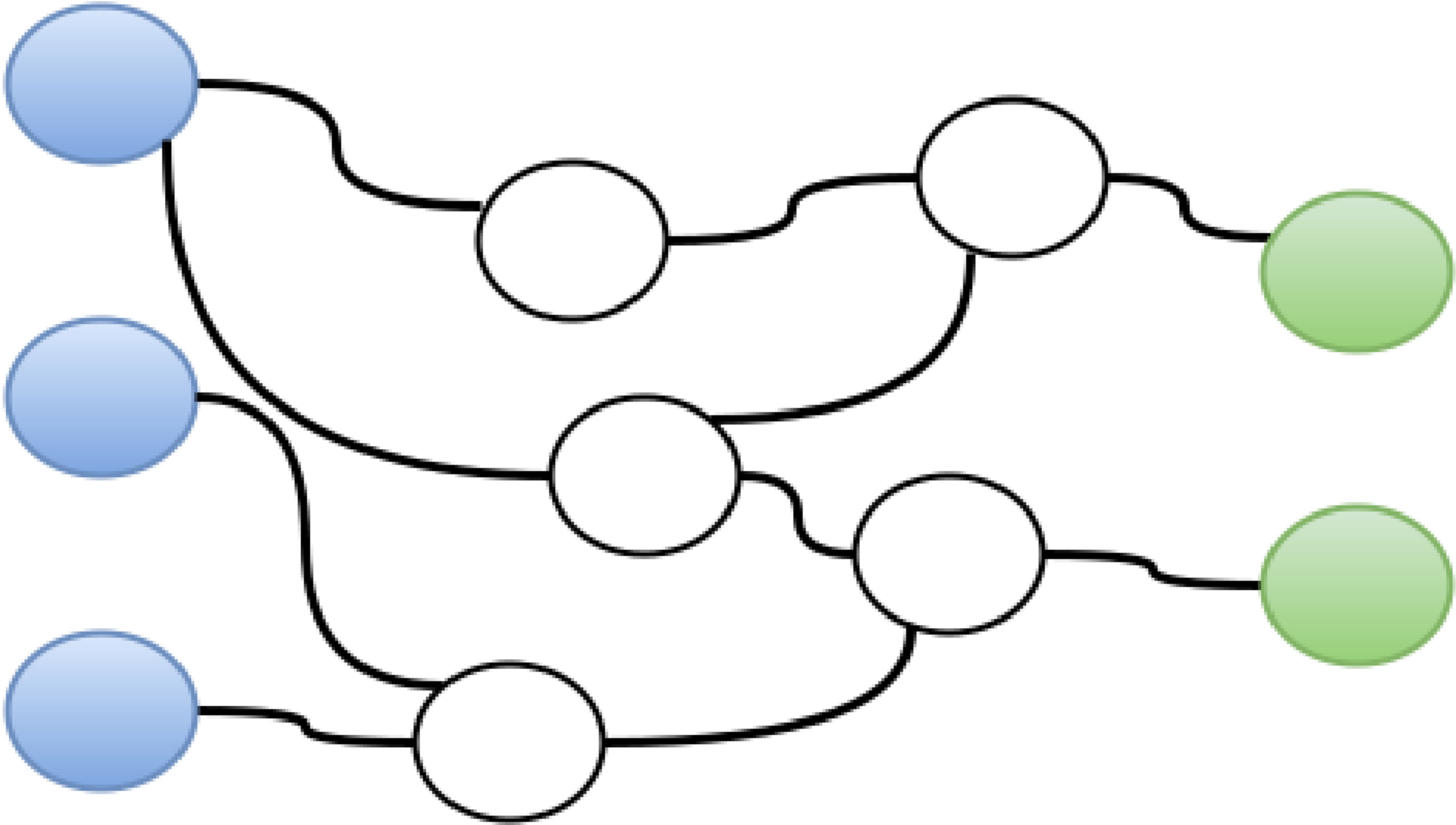
ফোর্ড-ফুলকার্সন অ্যালগোরিদমে শুধু পাথ খুজে বের করার কথা বলে হয়েছে, সেটা যেকোনোভাবে বের করা যেতে পারে, আর এডমন্ড-কার্প বিএফএস ব্যবহার করে কাজটা করতে বলা হয়েছে |

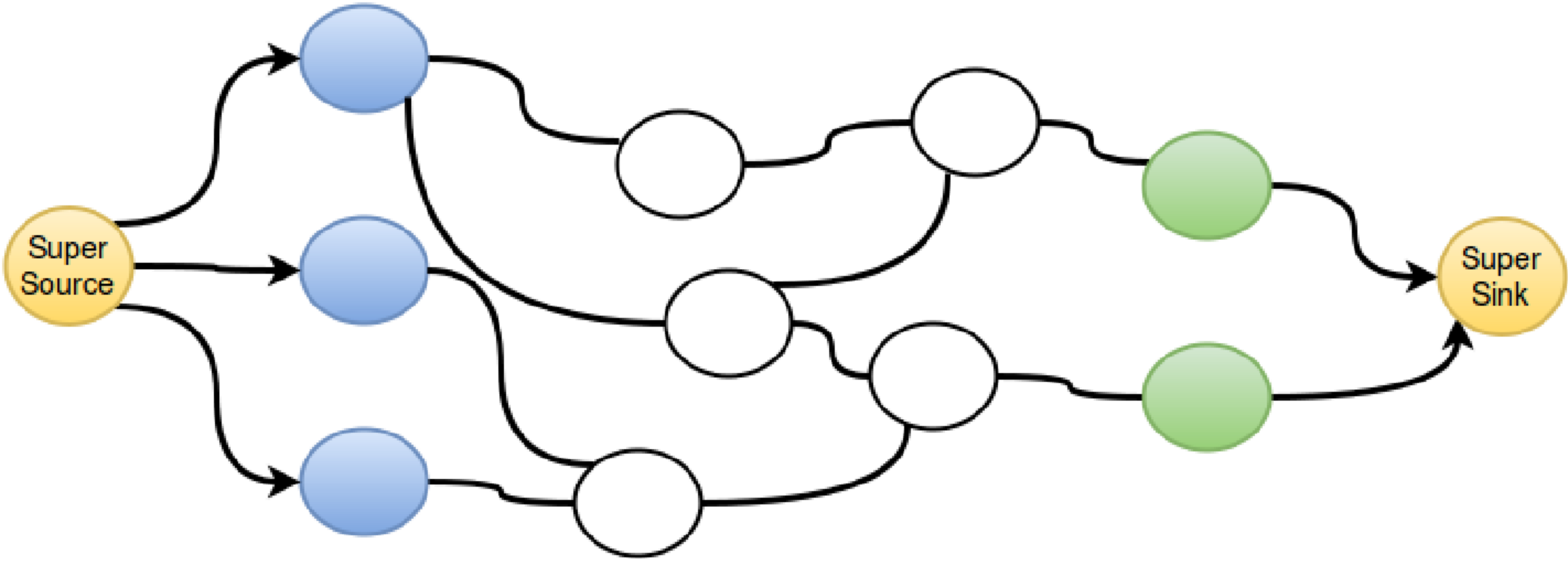
কমপ্লেক্সিটি O(VE^2)

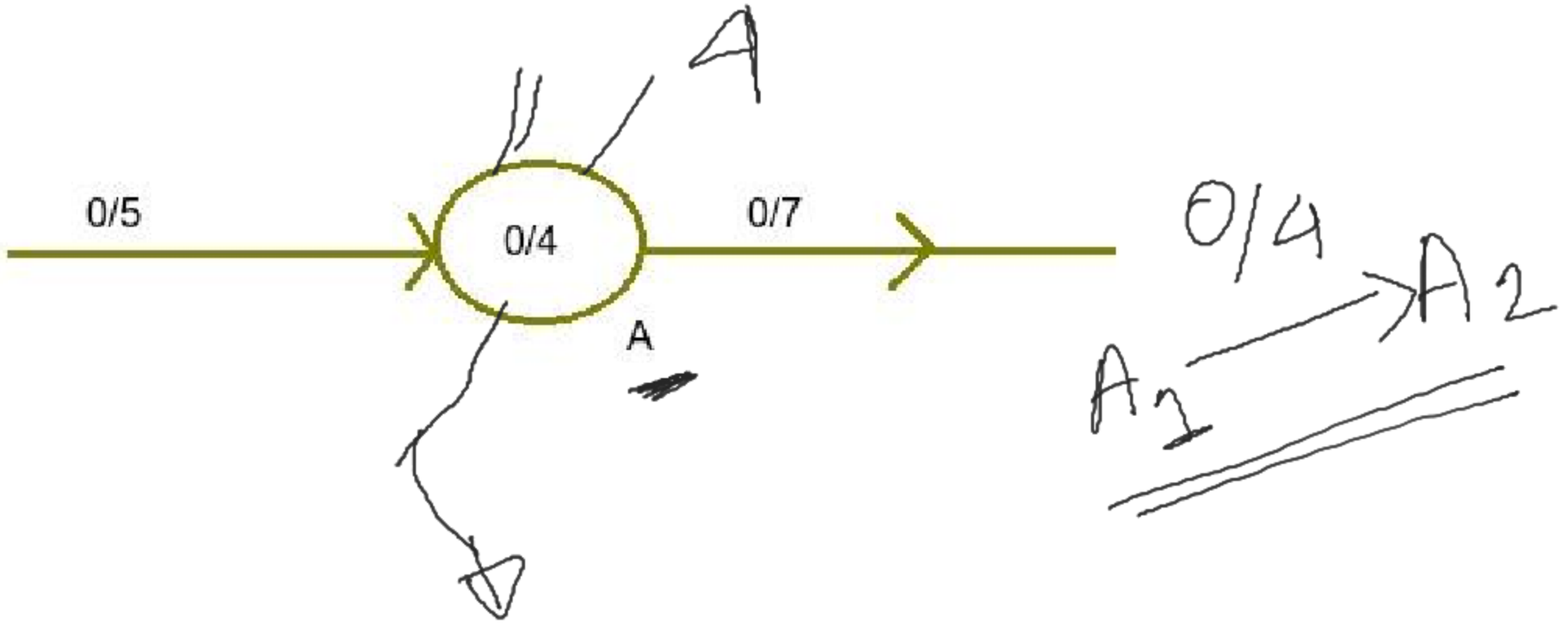
VE^2

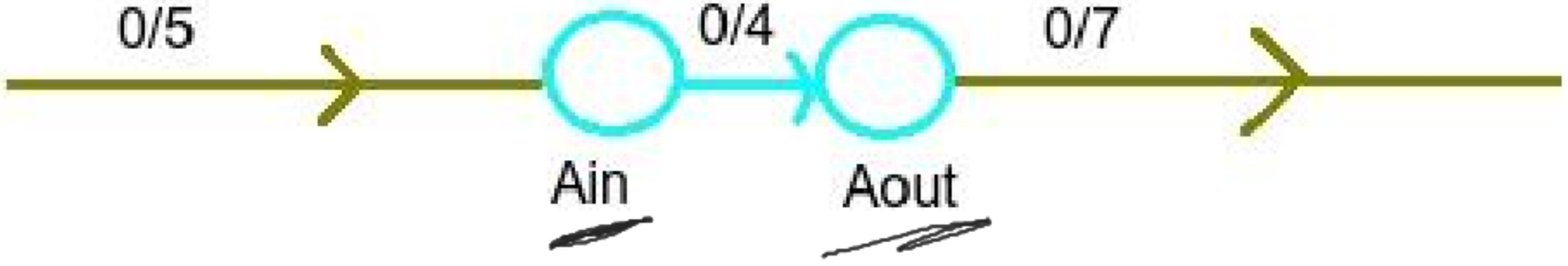
```
int maxflow(int s, int t) {
    int flow = 0;
    vector<int> parent(n);
    int new_flow;

    while (new_flow = bfs(s, t, parent)) {
        flow += new_flow;
        int cur = t;
        while (cur != s) {
            int prev = parent[cur];
            capacity[prev][cur] -= new_flow;
            capacity[cur][prev] += new_flow;
            cur = prev;
        }
    }

    return flow;
}
```

```
int n;
vector<vector<int>> capacity;
vector<vector<int>> adj;

int bfs(int s, int t, vector<int>& parent) {
    fill(parent.begin(), parent.end(), -1);
    parent[s] = -2;
    queue<pair<int, int>> q;
    q.push({s, INF});

    while (!q.empty()) {
        int cur = q.front().first;
        int flow = q.front().second;
        q.pop();

        for (int next : adj[cur]) {
            if (parent[next] == -1 && capacity[cur][next]) {
                parent[next] = cur;
                int new_flow = min(flow, capacity[cur][next]);
                if (next == t)
                    return new_flow;
                q.push({next, new_flow});
            }
        }
    }

    return 0;
}
```

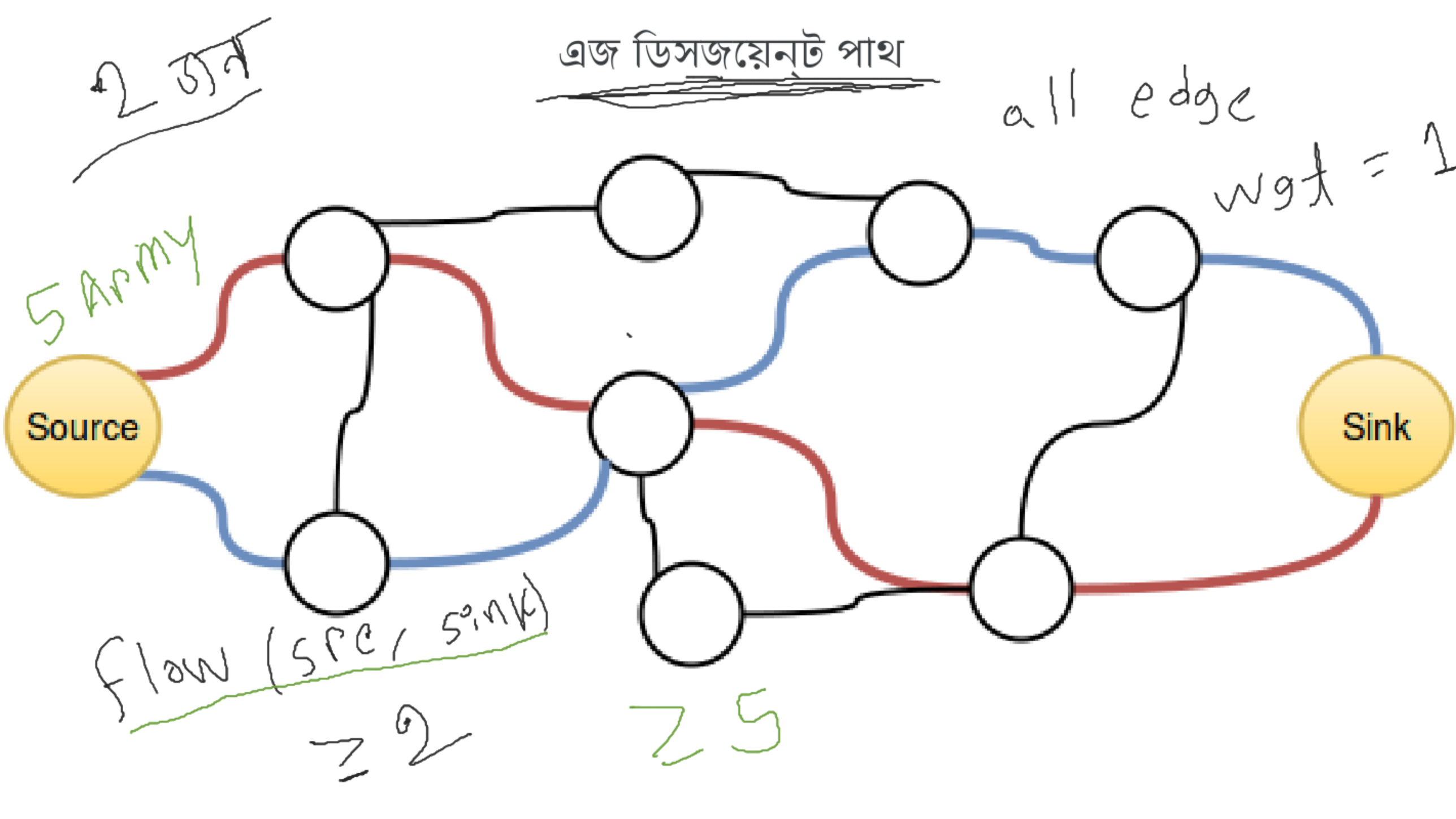



Super
Source
Super
Sink

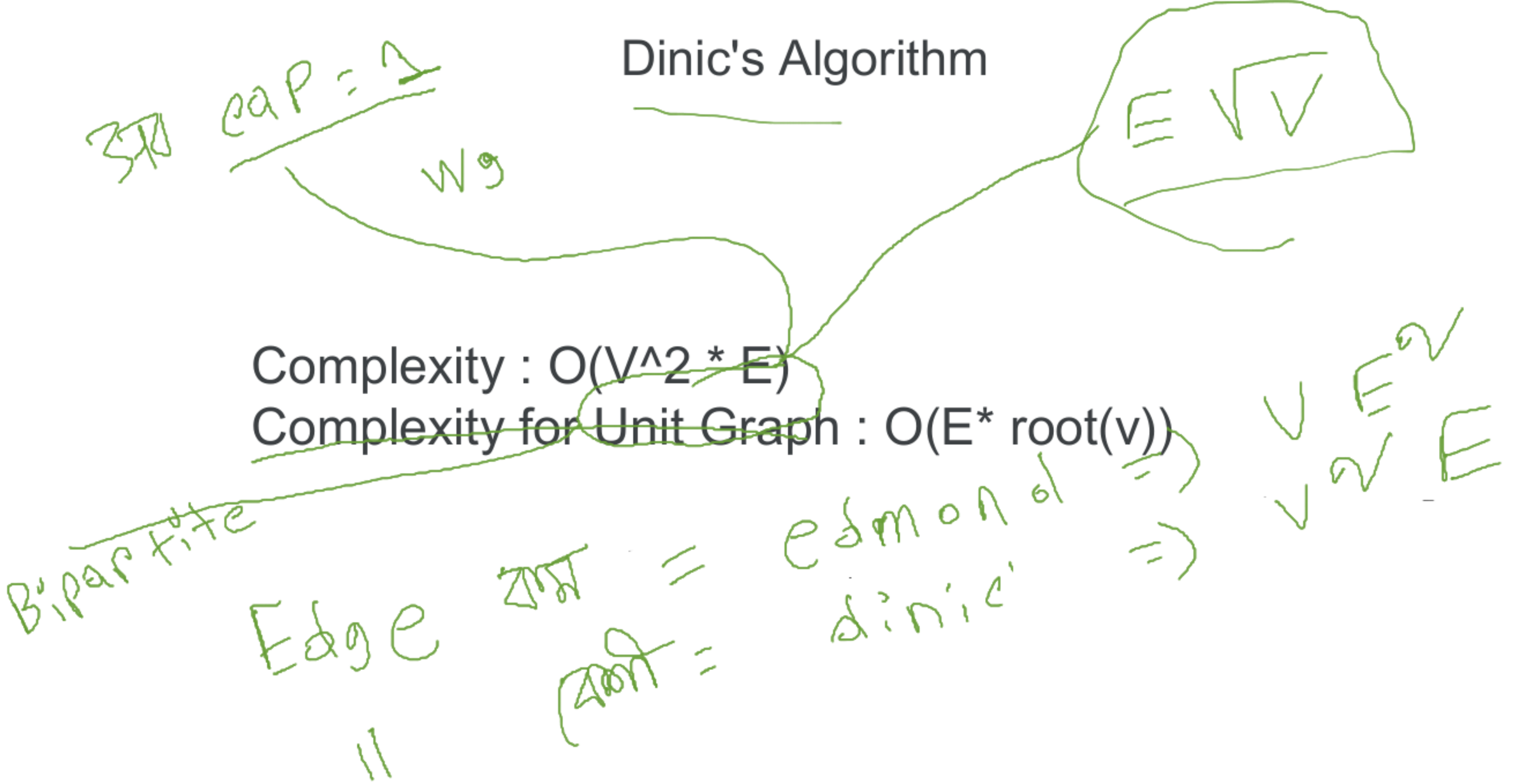

0/5
0/4
0/7
A
0/4
A1
A2


0/5
0/4
0/7
Ain
Aout

এজ ডিসজয়েনট পাথ

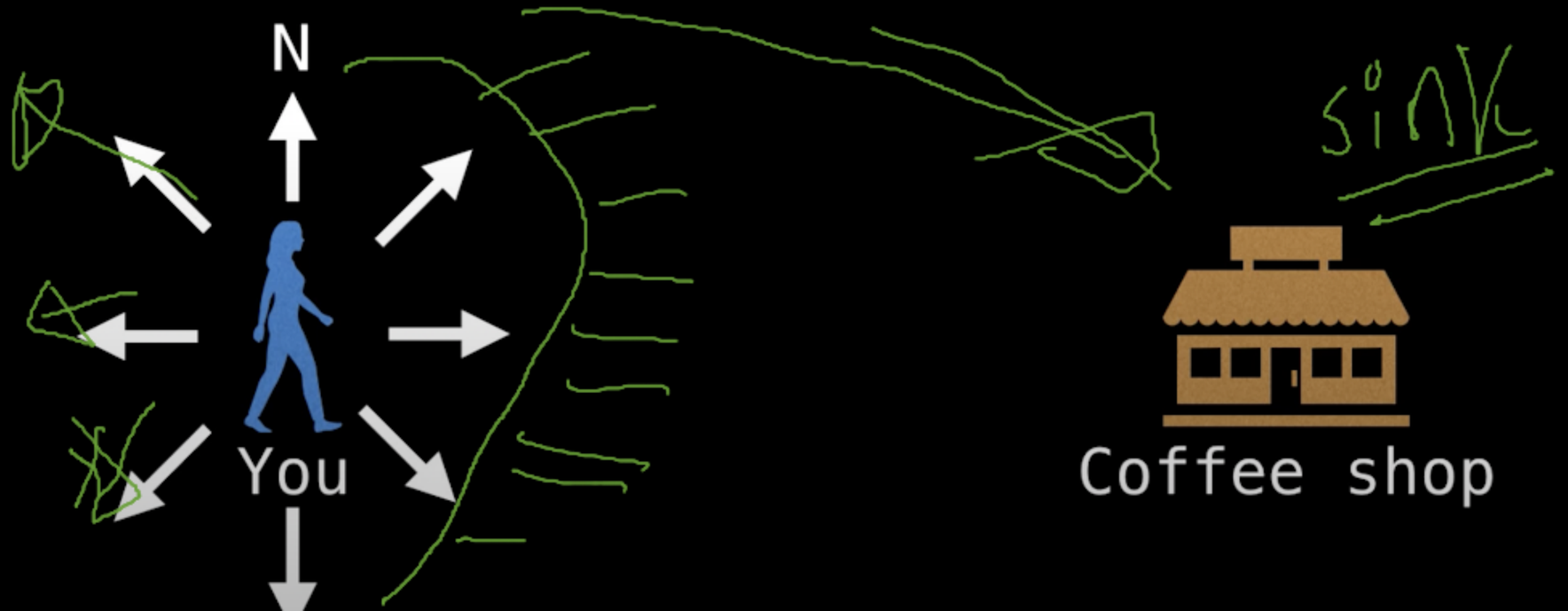


Dinic's Algorithm

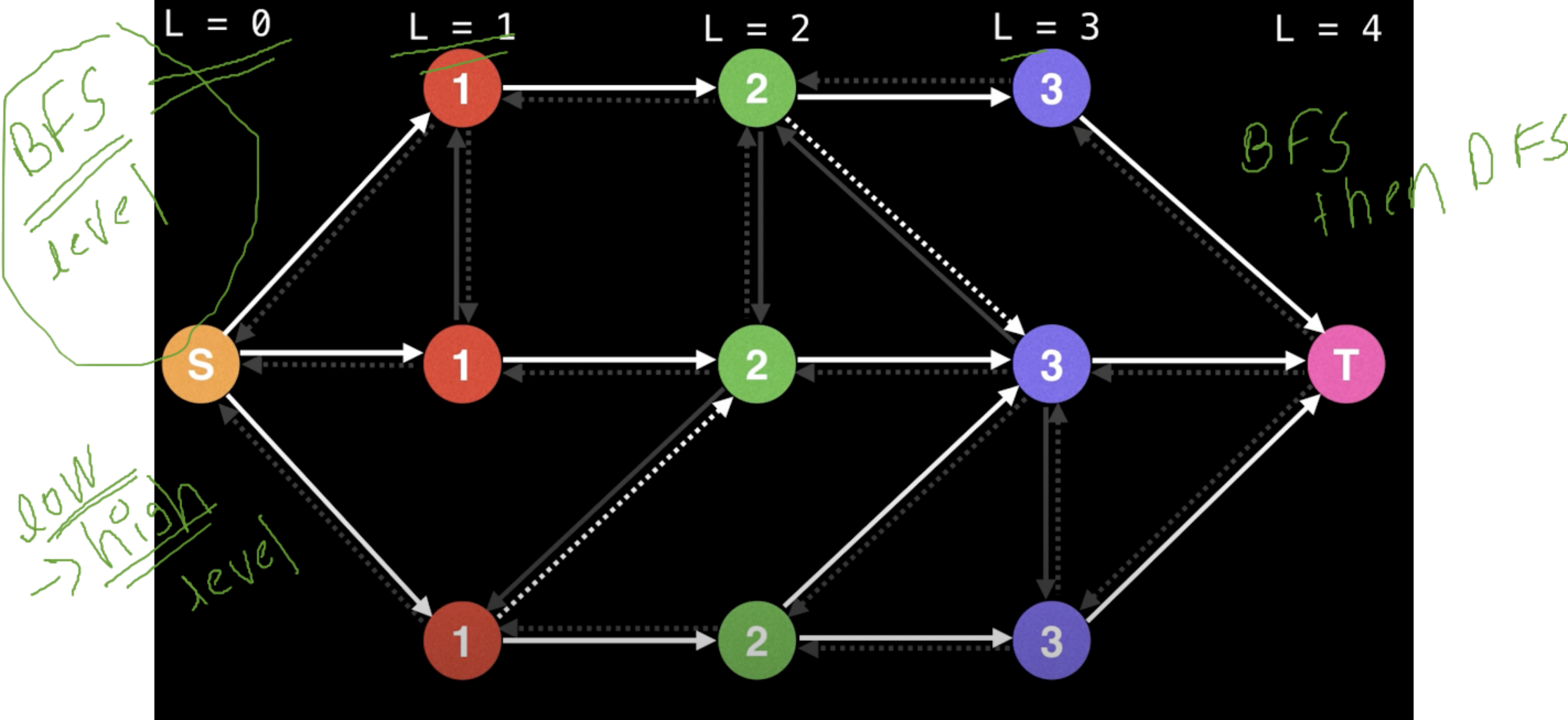


Dinic's Intuition Analogy

The only sensible directions are: east, north east and south east, this is because you know that those directions guarantee **positive progress towards** the coffee shop.



Furthermore, an edge is only part of the level graph if it makes progress towards the sink. That is, the edge must go from a node at level L to another at level L+1.



Dinic's Algorithm Steps

Step 1: Construct a level graph by doing a BFS from the source to label all the levels of the current flow graph.

Step 2: If the sink was never reached while building the level graph, then stop and return the max flow.

Step 3: Using only valid edges in the level graph, do multiple DFSs from s -> t until a **blocking flow** is reached, and sum over the bottleneck values of all the augmenting paths found to calculate the max flow.

Repeat Steps 1 to 3