Online Technical Interview Bootcamp at Stanford

Session 3

Yongwhan Lim

Thursday, April 27, 2023

Yongwhan Lim

Education

Stanford University

Part-time Jobs

Full-time Job

Google Research

TWO SIGMA

Workshops

Stanford ENGINEERING | Stanford Computer Forum

Carnegie Mellon University

UNIVERSITY OF CALIFORNIA

NUS National University of Singapore

Coach/Judge

icpc International Collegiate Programming Contest

icpc.foundation

advancing the art and sport of competitive programming

https://www.yongwhan.io

Yongwhan Lim

- Currently:
  - **CEO** (Co-Founder) in a Stealth Mode Startup;
  - **Co-Founder** in Christian and Grace Consulting;
  - ICPC **Internship Manager**;
  - ICPC **North America Leadership** Team;
  - Columbia ICPC **Head Coach**;
  - ICPC **Judge** for NAQ and Regionals;
  - **Lecturer** at MIT;
  - **Adjunct** (Associate in CS) at Columbia;

https://www.yongwhan.io

Session 3: Overview

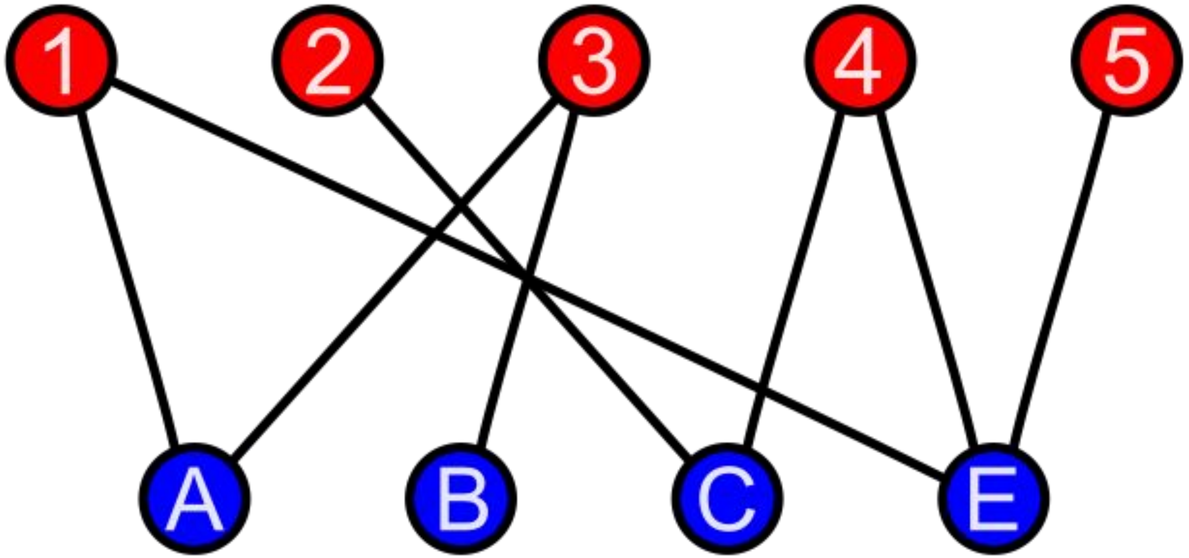
- **Part I**
  - Bipartite Graph Check
  - Bellman-Ford
  - Finding Bridges in O(N+M)
- **Part II: Problem Walkthroughs**
  - LeetCode Weekly 342
  - AtCoder Beginner Contest 299
  - Educational Codeforces Round 147 (Rated for Div. 2)
- **Important Reminders**

1. Bipartite Graph

- A **bipartite graph** is a graph whose vertices can be divided into **two** disjoint sets so that every edge connects two vertices from different sets (i.e. there are no edges which connect vertices from the same set). These sets are usually called **sides**.

- You are given an undirected graph. Check whether it is bipartite, and if it is, output its sides.

1. Bipartite Graph



1. Bipartite Graph

```
int n;
vector<vector<int>> adj;
vector<int> side(n, -1);
bool is_bipartite = true;
queue<int> q;
```

1. Bipartite Graph

```
for (int st = 0; st < n; ++st) {
    if (side[st] == -1) {
        q.push(st), side[st] = 0;
        while (!q.empty()) {
            int v = q.front(); q.pop();
            for (int u : adj[v])
                if (side[u] == -1)
                    side[u] = side[v]^1, q.push(u);
                else is_bipartite &= side[u] != side[v];
        }
    }
}
cout << (is_bipartite ? "YES" : "NO") << endl;
```

1. Bipartite Graph: Problem

- https://codeforces.com/contest/1144/problem/F

1. Bipartite Graph: Solution

- https://codeforces.com/blog/entry/66307 (1144F)

1. Bipartite Graph: Model Code

- https://codeforces.com/contest/1144/submission/77873752

1. Bipartite Graph: Reference

- https://cp-algorithms.com/graph/bipartite-check.html

2. Bellman-Ford

- Single source shortest path with **negative** weight edges.

- Quick Reminder
  - Unweighted: **BFS** or **DFS**
  - Weighted, non-negative weights: **Dijkstra**
  - All pair shortest paths: **Floyd-Warshall**

2. Bellman-Ford

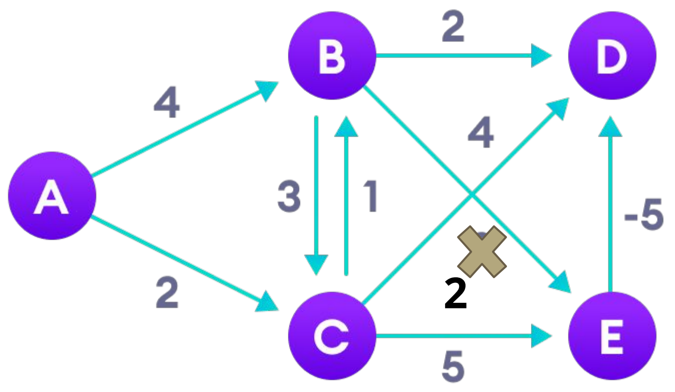
- We will create an array of distances d , which after execution of the algorithm will contain the answer to the problem.
- Initialization: d[v] = 0 and all other elements d[] equal to infinity.
- In each phase, for each edge (a,b) with weight c, relax

d[b] := min(d[b], d[a] + c)

- **n-1** phases of the algorithm are sufficient to correctly calculate the lengths of all shortest paths in the graph. So, if we run **n** phases and see a weight change, we know there is a cycle with negative total weight.
- For unreachable vertices the distance d[] will remain equal to infinity.

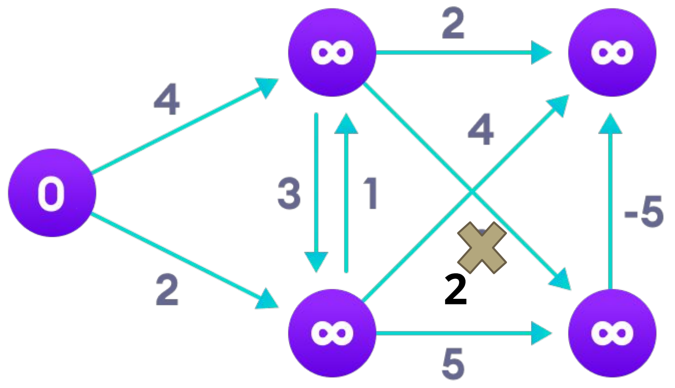
2. Bellman-Ford

Step 1: Start with the weighted graph



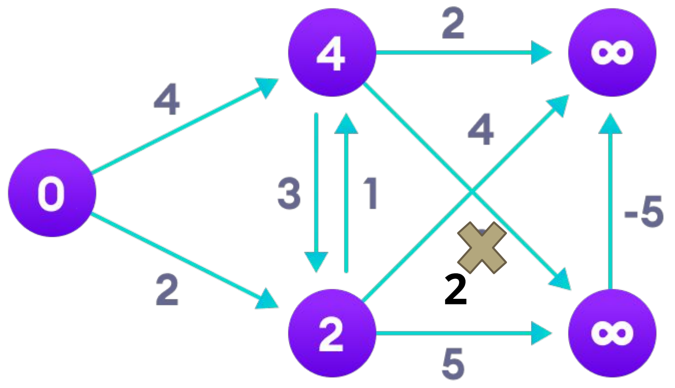
2. Bellman-Ford

Step 2: Choose a starting vertex and assign infinity path values to all other vertices



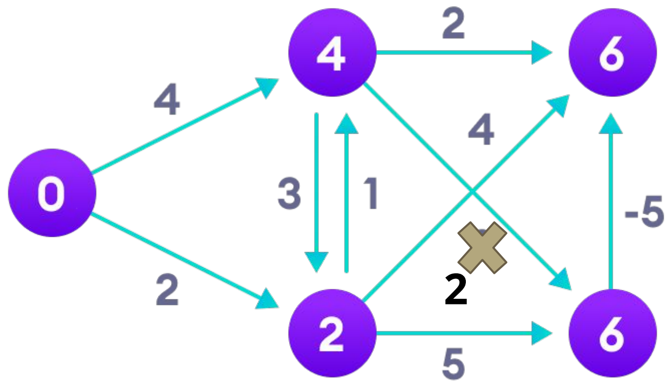
2. Bellman-Ford

Step 3: Visit each edge and relax the path distances if they are inaccurate



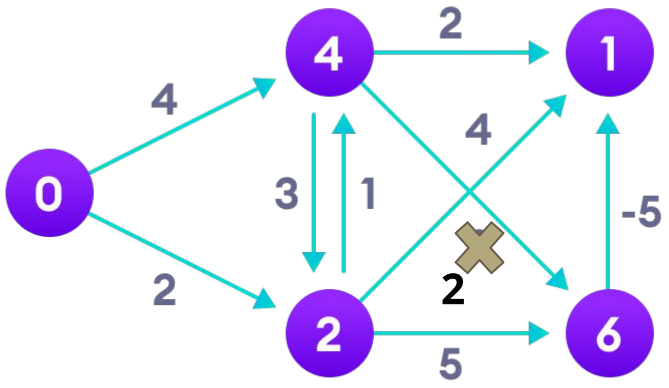
2. Bellman-Ford

Step 4: We need to do this V times because in the worst case, a vertex's path length might need to be readjusted V times



2. Bellman-Ford

Step 5: Notice how the vertex at the top right corner had its path length adjusted



2. Bellman-Ford: Implementation

```
struct Edge {
    int a, b, cost;
};

int n, m, v;
vector<Edge> edges;
const int INF = 1000000000;
```

2. Bellman-Ford: Implementation

```cpp
void bellman_ford() {
    vector<int> d(n, INF);
    d[v] = 0;
    for (int i = 0; i < n - 1; ++i)
        for (Edge e : edges)
            if (d[e.a] < INF)
                d[e.b] = min(d[e.b], d[e.a] + e.cost);
    // display d, for example, on the screen
}
```

2. Bellman-Ford: Problem

- https://onlinejudge.org/external/5/558.pdf

2. Bellman-Ford: Solution

- (Direct) application of Bellman-Ford!

2. Bellman-Ford: Model Code

- https://github.com/Diusrex/UVA-Solutions/blob/master/558%20Wormholes.cpp

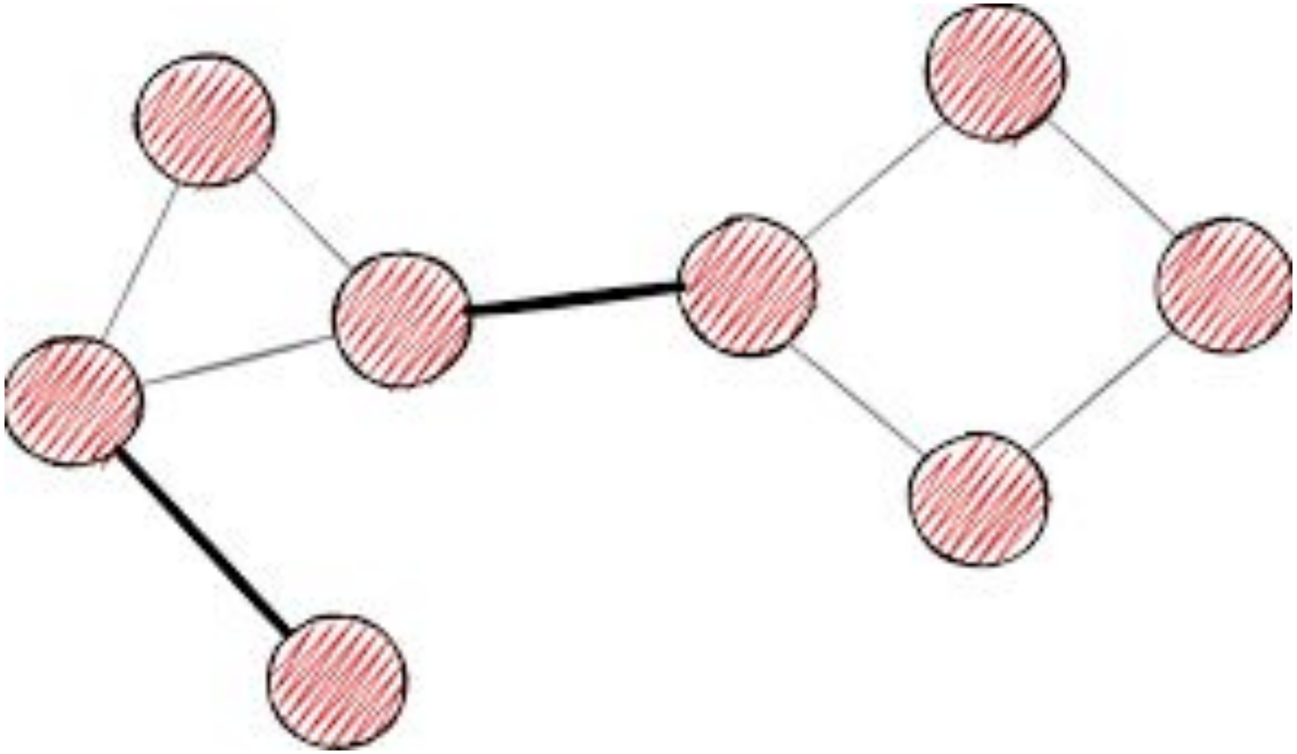
2. Bellman-Ford: Reference

- https://cp-algorithms.com/graph/bellman_ford.html
- https://www.programiz.com/dsa/bellman-ford-algorithm

3. Bridges: Definition

- An edge which, **when removed**, makes the graph **disconnected** (or more precisely, increases the number of connected components in the graph)

3. Bridges: Definition



3. Bridges: DFS Algorithm

- The algorithm described here is based on depth first search and has O(n+m) complexity where n is the number of vertices and m is the number of edges in the graph.

3. Bridges: Implementation Details

- **Fact**: Let's say we are in the DFS, looking through the edges starting from vertex **v**. The current edge (**v**, **to**) is a bridge if and only if none of the vertices to and its descendants in the DFS traversal tree has a back-edge to vertex **v** or any of its ancestors. Indeed, this condition means that there is no other way from **v** to to except for edge (**v**, **to**).
- We can do check this fact for each vertex efficiently using "time of entry into node"; let's denote tin[v] to mean the entry time for node v.

3. Bridges: Implementation Details

- Let's define low for each vertex v to be:

$$low[v] = \min \begin{cases} tin[v] & \\ tin[p] & \text{for all } p \text{ for which } (v, p) \text{ is a back edge} \\ low[to] & \text{for all } to \text{ for which } (v, to) \text{ is a tree edge} \end{cases}$$

- Then, current edge (**v**, **to**) in the DFS tree is a **bridge** if and only if **low[to] > tin[v]**.

3. Bridges: Implementation Details

- The implementation needs to distinguish three cases: when we go down the edge in DFS tree, when we find a back edge to an ancestor of the vertex, and when we return to a parent of the vertex.

- These are the cases:
    - **visited[to] = false**: the edge is part of DFS tree;
    - **visited[to] = true && to ≠ parent**: the edge is back edge to one of the ancestors;
    - **to = parent**: the edge leads back to parent in DFS tree.

3. Bridges: Implementation

```
int n; // number of nodes
vector<vector<int>> adj; // adjacency list of graph

vector<bool> visited;
vector<int> tin, low;
int timer;
```

3. Bridges: Implementation

```
void dfs(int v, int p = -1) {
    visited[v] = true;
    tin[v] = low[v] = timer++;
    for (int to : adj[v]) {
        if (to == p) continue;
        if (visited[to]) low[v] = min(low[v], tin[to]);
        else {
            dfs(to, v);
            low[v] = min(low[v], low[to]);
            if (low[to] > tin[v])
                IS_BRIDGE(v, to);
        }
    }
}
```

3. Bridges: Implementation

```
void find_bridges() {
    timer = 0;
    visited.assign(n, false);
    tin.assign(n, -1);
    low.assign(n, -1);
    for (int i = 0; i < n; ++i) {
        if (!visited[i])
            dfs(i);
    }
}
```

3. Bridges: Problem

- https://codeforces.com/contest/732/problem/F

3. Bridges: Solution

- https://codeforces.com/blog/entry/47890 (732F)

3. Bridges: Model Code

- https://codeforces.com/contest/732/submission/173829186

3. Bridges: Reference

- https://cp-algorithms.com/graph/bridge-searching.html
- Closely related notion is **articulation point**.

BREAK

Problem Walkthroughs

- LeetCode Weekly 342
- AtCoder Beginner Contest 299
- Educational Codeforces Round 147 (Rated for Div. 2)

Request 1:1 Meeting, through Calendly

- Use calendly.com/yongwhan/one-on-one to request 1:1 meeting:
  - Mock Interview
  - Career Planning
  - Resume Critique
  - Practice Strategy
  - Volunteering Opportunity
  - ...
- I am always **inspired** by driven students like yourself!
- Since I'd feel honored/thrilled to talk to you, do not feel shy to sign up!!!

Terse Guide Google Drive

- Browse through Terse Guides, which include:
    - Behavioral interview preparation
    - System design interview preparation
    - ICPC preparation
    - Live contests
    - Useful resources

Discord Server Invitations

- Some discord server invitations:
    - **[Online Technical Interview Bootcamp at Stanford]** https://discord.gg/aJwHBccg3n
    - **[ICPC CodeForces Zealots]** https://discord.gg/QC9ss6WJPy

Where to go from here? (for training)

- Train, train, train, BUT only go so much to **NOT** burnout. **IT IS REAL!**
- Each and every one of you can do it, from what I observed last few days!!
- Register for **Universal Cup**: ask, if interested!
- **CSES**: https://cses.fi/problemset/
- **Kattis**: https://open.kattis.com/ with its companion: https://cpbook.net/methodstosolve?oj=kattis&topic=all&quality=all
- **USACO Guide:** https://usaco.guide/ (especially Platinum and Advanced)
- **CP Algorithm:** https://cp-algorithms.com/
  - String Processing; Graphs; Linear Algebra; Data Structures; ...

Success Pathways

- Those who are just starting should focus on the **first half** of problems in Zealot Problem Set. Your main focus should be gaining some experiences with an explicit goal to enjoy the process of solving new problems and potentially making it to the ICPC North America Championship (NAC)!

- Those who are more serious should focus on the **second half** of problems in Zealot Problem Set. Your goal should be making into the World Finals and potentially winning a medal!

Practice Strategy

- If your goal is to get to a rating of **X**, you should practice on problems that are **X + 300** typically, with a spread of 100. So, picking problems within the range of:

  {X + 200, X + 300, X + 400}

  would be sensible!

- So, if you want to target becoming a **red**, which has a lower-bound of 2400, you should aim to solving {2600, 2700, 2800}.
- **(Eventual) Target**: You should focus on solving it for 30 minutes or less!

Practice Strategy

- You should focus on solving each problem for **30 minutes or less**; if you cannot solve any problem with this range, you should consider solving a problem with a lower rating.
- You should aim to solve **10 ~ 15 problems** each day within this range to expect a rank up within a quarter (3 months).
- If you cannot solve a problem, here is a sample recipe you can follow:
    - Look at editorial for hints, and try to solve the problem.
    - Look at editorial for full solutions, and try to solve the problem.
    - Look at accepted solutions, and try to solve the problem.
    - Make sure you look back after two weeks and see if you can solve it.

Contact Information

- **Email**: yongwhan@yongwhan.io
- **Personal Website**: https://www.yongwhan.io/
- **LinkedIn Profile**: https://www.linkedin.com/in/yongwhan/
    - Feel free to send me a connection request!
    - Always happy to make connections with promising students!

Q&A’s