Tech Interview Prep
Lecture 19

Christian Yongwhan Lim

5:40pm ET, Thursday, November 9, 2023

Overview

- Roll Call
- 1:1 Meeting Request
- Regular Office Hours Reinstated!
- ICPC Practice Contest
- Presenter Feedback Form
- Presenters: Wo (wl2834); Ken (km3635); Tsai-Chen (th2990);
- Algorithms: Topological Sorting; Strongly Connected Component (Kosaraju); 2-SAT;

Roll Call

- **Please be on-time!**

- Your attendance will count only if you come to lecture prior to your roll call time.

1:1 Meeting Request

- Happy to meet with you to discuss anything you'd like.

- Please use https://calendly.com/yongwhan/quick-chat-blitz to sign up!

Regular Office Hours Reinstated!

- I will hold a **regular office hour** from **3pm ET** to **4pm ET** in **7th floor CEPSR** on Mondays.
- This will start **Monday, November 13**!
- If anything, I will discuss LeetCode problems more carefully.

ICPC Practice Contest

- We will have a practice contest on **Saturday, November 11, 2023**!
- **from 11am ET to 4pm ET**
- **Computer Cluster** @ Mudd (2nd floor)

Presenter Feedback Form

- Please use **https://bit.ly/techprep-feedback** to **provide feedback** for presenters today!

Presenters

- Wo (wl2834)
  - 1291. Sequential Digits

Presenters

- Ken (km3635)
    - 287. Find the Duplicate Number

Presenters

- Tsai-Chen (th2990)
  - [1362. Closest Divisors]

Topological Sorting

- In a directed acyclic graph (DAG),

Topological Sorting

- In a directed acyclic graph (DAG), put into queue all nodes with indegree zero.

Topological Sorting

- In a directed acyclic graph (DAG), put into queue all nodes with indegree zero.

- Each time a node is dequeued, decrement their children's indegree by 1 and anytime it hits 0, put in that node into queue.

Topological Sorting

- In a directed acyclic graph (DAG), put into queue all nodes with indegree zero.

- Each time a node is dequeued, decrement their children's indegree by 1 and anytime it hits 0, put in that node into queue.

- Rinse and repeat!

Topological Sorting

- Of course, you can do it using DFS too!

Topological Sorting: Implementation

```
void dfs(int v) {
  visited[v] = true;
  for (int u : adj[v]) {
    if (!visited[u])
      dfs(u);
  }
  ans.push_back(v);
}
```

Topological Sorting: Implementation

```
void topological_sort() {
  visited.assign(n, false);
  ans.clear();
  for (int i = 0; i < n; ++i)
    if (!visited[i])
      dfs(i);
  reverse(ans.begin(), ans.end());
}
```

Strongly Connected Component (Kosaraju) & 2-SAT

- Kosaraju for SCC!
  - Do topological sorting (DFS);
  - Reverse the order;
  - Do DFS again!

Kosaraju: Implementation

```
vector<vector<int>> adj, adj_rev;
vector<bool> used;
vector<int> order, component;
```

Kosaraju: Implementation

```
void dfs1(int v) {
  used[v] = true;

  for (auto u : adj[v])
    if (!used[u])
      dfs1(u);

  order.push_back(v);
}
```

Kosaraju: Implementation

```
void dfs2(int v) {
  used[v] = true;
  component.push_back(v);

  for (auto u : adj_rev[v])
    if (!used[u])
      dfs2(u);
}
```

Kosaraju: Implementation

```
int main() {
  int n;
  for (;;) {
    int a, b;
    // ... read next directed edge (a,b) ...
    adj[a].push_back(b);
    adj_rev[b].push_back(a);
  }
  used.assign(n, false);
```

Kosaraju: Implementation

```
for (int i = 0; i < n; i++)
  if (!used[i])
    dfs1(i);
used.assign(n, false);
reverse(order.begin(), order.end());
```

Kosaraju: Implementation

```
for (int i = 0; i < n; i++)
  if (!used[i])
    dfs1(i);
used.assign(n, false);
reverse(order.begin(), order.end());
for (auto v : order)
  if (!used[v]) {
    dfs2 (v);
    // ... processing next component ...
    component.clear();
  }
}
```

2-SAT

- **CNF** (conjunctive normal form):

$$(a \vee \neg b) \wedge (\neg a \vee b) \wedge (\neg a \vee \neg b) \wedge (a \vee \neg c)$$

2-SAT

- **CNF** (conjunctive normal form):

$$(a \vee \neg b) \wedge (\neg a \vee b) \wedge (\neg a \vee \neg b) \wedge (a \vee \neg c)$$

- Find an assignment of a, b, c such that the above formula is true!

2-SAT

- **CNF** (conjunctive normal form):

$$(a \vee \neg b) \wedge (\neg a \vee b) \wedge (\neg a \vee \neg b) \wedge (a \vee \neg c)$$

- Find an assignment of a, b, c such that the above formula is true!

- In 2-SAT, every clause has exactly two literals as above!

2-SAT

- **CNF** (conjunctive normal form):

$$(a \vee \neg b) \wedge (\neg a \vee b) \wedge (\neg a \vee \neg b) \wedge (a \vee \neg c)$$

- Find an assignment of a, b, c such that the above formula is true!

- In 2-SAT, every clause has exactly two literals as above!

- SAT is NP-complete, but 2-SAT can be solved in linear time!

Key Insight

- We can see that:

$$a \lor b$$

is equivalent to:

$$\neg a \Rightarrow b \land \neg b \Rightarrow a$$

Key Insight

- We now construct a directed graph of these implications: for each variable x there will be two vertices v_x and $v_{\neg x}$.

- The edges will correspond to the implications.

Key Insight

- So, for:

$$(a \vee \neg b) \wedge (\neg a \vee b) \wedge (\neg a \vee \neg b) \wedge (a \vee \neg c)$$

Example (con't)

- So, for:

$$(a \vee \neg b) \wedge (\neg a \vee b) \wedge (\neg a \vee \neg b) \wedge (a \vee \neg c)$$

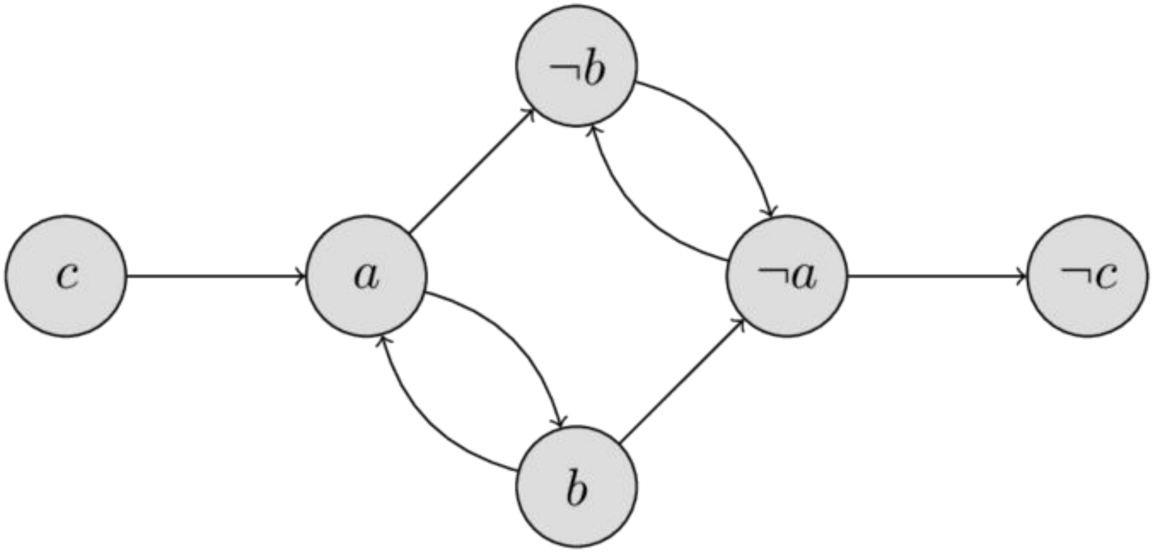
- We will have the following vertices and edges:

$$\neg a \Rightarrow \neg b \qquad a \Rightarrow b \qquad a \Rightarrow \neg b \qquad \neg a \Rightarrow \neg c$$

$$b \Rightarrow a \qquad \neg b \Rightarrow \neg a \qquad b \Rightarrow \neg a \qquad c \Rightarrow a$$

Example (con't)

- Which will result in the following (implication) graph:



Key Insight

- In order for this 2-SAT problem to have a solution, *it is necessary and sufficient* that for any variable x the vertices **x** and **¬x** are in **<u>different</u>** strongly connected components of the implication graph.

- So, 2-SAT can be solved using SCC, or Kosaraju!

2-SAT: Implementation

```
int n;
vector<vector<int>> adj, adj_t;
vector<bool> used;
vector<int> order, comp;
vector<bool> assignment;
```

2-SAT: Implementation

```
void dfs1(int v) {
  used[v] = true;
  for (int u : adj[v]) {
    if (!used[u])
      dfs1(u);
  }
  order.push_back(v);
}
```

```
void dfs2(int v, int cl) {
  comp[v] = cl;
  for (int u : adj_t[v]) {
    if (comp[u] == -1)
      dfs2(u, cl);
  }
}
```

2-SAT: Implementation

```
bool solve_2SAT() {
  order.clear();
  used.assign(n, false);
  for (int i = 0; i < n; ++i) {
    if (!used[i])
      dfs1(i);
  }
```

2-SAT: Implementation

```
comp.assign(n, -1);
for (int i = 0, j = 0; i < n; ++i) {
  int v = order[n - i - 1];
  if (comp[v] == -1)
    dfs2(v, j++);
}
```

2-SAT: Implementation

```
    assignment.assign(n / 2, false);
    for (int i = 0; i < n; i += 2) {
      if (comp[i] == comp[i + 1])
        return false;
      assignment[i / 2] = comp[i] > comp[i + 1];
    }
    return true;
  }
```

2-SAT: Implementation

```
void add_disjunction(int a, bool na, int b, bool nb) {
  a = 2*a ^ na;
  b = 2*b ^ nb;
  int neg_a = a ^ 1;
  int neg_b = b ^ 1;
  adj[neg_a].push_back(b);
  adj[neg_b].push_back(a);
  adj_t[b].push_back(neg_a);
  adj_t[a].push_back(neg_b);
}
```


THANK YOU