ShanghaiTech University

CS101 Algorithms and Data Structures
Fall 2024

Homework 7

Due date: November 20, 2024, at 23:59

1. Please write your solutions in English.
2. Submit your solutions to Gradescope.
3. Set your FULL name to your Chinese name and your STUDENT ID correctly in Gradescope account settings.
4. If you want to submit a handwritten version, scan it clearly. CamScanner is recommended.
5. When submitting, match your solutions to the problems correctly.
6. No late submission will be accepted.
7. Violations to any of the above may result in zero points.

1. (12 points) Multiple Choices

Each question has **one or more** correct answer(s). Select all the correct answer(s). For each question, you will get 0 points if you select one or more wrong answers, but you will get half points if you select a non-empty subset of the correct answers.

Write your answers in the following table.

(a)	(b)	(c)	(d)	(e)	(f)

(a) (2') A union tree, used in a disjoint set with only union-by-rank optimization, has height 7. The number of nodes contained in that tree **can not** be:

A. 127

B. 128

C. 129

D. 2024

(b) (2') Which of the following statements are true for disjoint set?

A. The two main operations of disjoint set are Find and Union.

B. Union-by-rank optimization alone is sufficient to ensure that the height of the union trees remains $O(\alpha(n))$, where n is the number of nodes and α is the inverse Ackermann function.

C. After applying the Find operation with path compression in a disjoint set, the height of the union tree will always decrease.

D. The combination of path compression and union by rank ensures that both the time complexity of the Find and Union operations is nearly constant in practice.

(c) (2') Undirected Graph $G = (V, E)$ is stored in an adjacency matrix A. We let $A_{i,j} = 1$ if and only if there is an edge between V_i and V_j in G, otherwise $A_{i,j} = 0$. We want to know whether there is a path **with length** m between V_i and V_j by visiting $P_{i,j}$; if $P_{i,j}$=0, then there is no such path. What should P be?

A. A

B. mA

C. A^{m-1}

D. A^m

(d) (2') Which of the following statements are true for graph properties?

A. An undirected simple graph with n vertices can have at most $n(n-1)$ edges.

B. An undirected graph is acyclic if and only if it is a tree.

C. The minimum number of edges required to connect n vertices in a graph is $n-1$.

D. In a connected undirected graph, the removal of any edge will always result in a graph that is still connected.

(e) (2') Which of the following statements are true for graph properties?

A. Any tree is a bipartite graph.

B. A graph with an odd number of vertices can not be a bipartite graph.

C. A DAG(Directed Acyclic Graphs) must have at least one vertex with no incoming edges (source).

D. The sum of the degrees of all vertices in a graph is always equal to the number of edges in the graph.

(f) (2') Which of the following statements are true for graph traversal?

A. Given two vertices s and t in a graph G, we can use both BFS and DFS to determine whether there exists a path from s to t.

B. Both the time complexity of DFS and that of BFS are $\Theta(|V| + |E|)$.

C. In DFS, all vertices are visited in the order of their distance from the start vertex.

D. In BFS, let $d(v)$ be the minimum number of edges between a vertex v and the start vertex. For any two vertices u and v in the queue, $|d(u) - d(v)|$ is always less than 2.

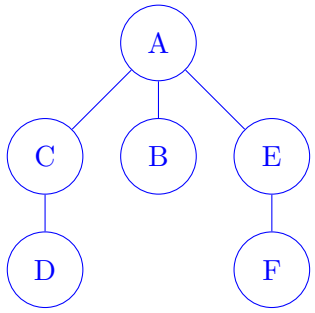
2. (10 points) Disjoint Set Practice

Given the following set of operations on a disjoint set(initially there are six disjoint elements), show the final disjoint set tree for each of the following optimization strategies:

- $set\_union(A, B)$
- $set\_union(C, D)$
- $set\_union(B, C)$
- $find(D)$
- $set\_union(E, F)$
- $set\_union(E, A)$

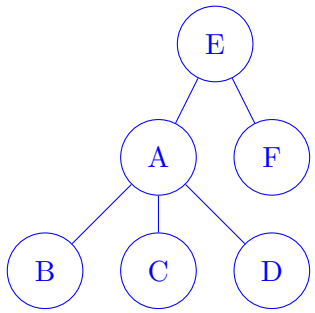
(a) (5') Only with union-by-rank optimization. (When two trees have the same height, the set specified first in the union will be the root of the merged set.)

Solution:



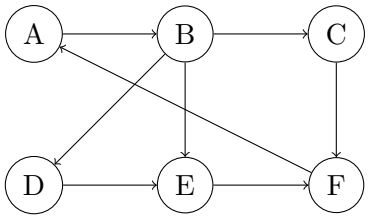
(b) (5') Only with path compression. (The set specified first in the union will always be the root of the merged set.)

Solution:



3. (9 points) Graph traversal

Consider the following directed graph starting with A.



(a) (3') Give the adjacency list for the graph. You should write the node in alphabetical order. (Leave it blank if the node has no neighbour)

$$adj(A) = [\_\_\_\_\_],$$
$$adj(B) = [\_\_\_\_\_],$$
$$adj(C) = [\_\_\_\_\_],$$
$$adj(D) = [\_\_\_\_\_],$$
$$adj(E) = [\_\_\_\_\_],$$
$$adj(F) = [\_\_\_\_\_],$$

Solution:

$$adj(A) = [B],$$
$$adj(B) = [CDE],$$
$$adj(C) = [F],$$
$$adj(D) = [E],$$
$$adj(E) = [F],$$
$$adj(F) = [A],$$

(b) (3') Give the visited node order using the above adjacency list for Breadth First Search.

Solution: ABCDEF

(c) (3') Give the visited node order using the above adjacency list for Depth First Search.

Solution: ABCFDE

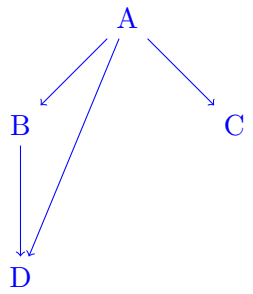
4. (6 points) DFS went wrong!

The following algorithm which runs DFS on a directed graph, but it contains a fatal error.

```
Create a stack.
Choose the initial vertex and mark it as visited.
Put the initial vertex onto the stack.
while the stack is not empty:
    Pop a vertex V from the top of the stack.
    for each neighbor of V:
        if that neighbor is not marked as visited:
            Mark that neighbor as visited.
            Push that neighbor onto the stack.
```

Please give a graph as an counterexample and briefly explain why this algorithm is wrong.
Note: In this problem, we say a node is "visited" whenever it is marked as visited. Pay special attention to how this method differs from the standard DFS in terms of the order in which nodes are marked as visited.

Solution: This is incorrect because it actually marks all neighbors of the initial node. If A follows the alphabetical order, then the result should be A, B, D, C. But in this case it will be A, B, C, D.



5. (6 points) Friend or enemy?

Consider a set of n individuals, each of whom can have two types of relationships with others: friendship and enmity. It is also possible for individuals to have no relationship at all, or even to be both friends and enemies simultaneously. The relationships satisfy the following properties:

1. If person a_i is a friend of person a_j, then person a_j is also a friend of person a_i (friendship is symmetric).
2. If person a_i is an enemy of person a_j, then person a_j is also an enemy of person a_i (enmity is symmetric).
3. Friends of friends are also considered friends. That is, if person a_i is a friend of person a_j and person a_j is a friend with person a_k, then person a_i is a friend of person a_k.
4. Enemies of enemies are considered friends. That is, if person a_i is an enemy of person a_j and person a_j is an enemy of person a_k, then person a_i is a friend of person a_k.

Given a set of relationships represented as:

- A set of friendship pairs $F \subseteq \{(a_i, a_j) \mid 1 \leq i, j \leq n, i \neq j\}$
- A set of enmity pairs $E \subseteq \{(a_i, a_j) \mid 1 \leq i, j \leq n, i \neq j\}$

Your task is to determine the number of distinct groups that can be formed, where each group consists of individuals who are all friends with each other. (When forming groups, we do not need to consider enemy relationships.)

You should ensure that your algorithm has a time complexity of $O((m+n)\alpha(n))$, where m is $|F| + |E|$ and n is the number of individuals, but no proof is required.

Hint: You may consider representing each individual as two nodes: one for their friendships and one for their enmities. Utilize the disjoint set to efficiently manage and merge these relationships.

Solution:

1. **Representation:**
   For each individual a_i, create two nodes:
   - P_i: Represents the friendship of individual a_i.
   - E_i: Represents the enmity of individual a_i.
2. **Union Operations for Friendships:**
   For each pair (a_i, a_j) in the friendship set F: Perform a union operation on their friendship nodes:

$$\text{Union}(P_i, P_j)$$

3. **Union Operations for Enmities:**
   For each pair (a_i, a_j) in the enmity set E: Perform a union operation between the friendship node of one individual and the enmity node of the other:

$$\text{Union}(P_i, E_j)$$

$$\text{Union}(P_j, E_i)$$

4. **Counting Distinct Groups:**
   After processing the pairs, count the number of distinct sets in the union-find structure(in P). Each distinct set corresponds to a group of individuals who are all friends with each other.

Complexity: The time complexity of this approach is $O(m\alpha(n) + n)$, where m is the number of relationships, n is the number of individuals.