ShanghaiTech University

CS101 Algorithms and Data Structures

Fall 2019

Homework 6

Due date: 23:59, November 3, 2019

1. Please write your solutions in English.
2. Submit your solutions to gradescope.com.
3. Set your FULL Name to your Chinese name and your STUDENT ID correctly in 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 according problem numbers correctly.
6. No late submission will be accepted.
7. Violations to any of above may result in zero score.

1 (6') Graph and Disjoint Set

Each question has one or more correct answer(s). Select all the correct answer(s). For each question, you get 0 point if you select one or more wrong answers, but you get 0.5 point if you select a non-empty subset of the correct answers.
Note that you should write you answers of section 1 in the table below.

Question 1	Question 2	Question 3	Question 4	Question 5	Question 6
D	BC	B	CD	B	ABCDE

Question 1. *Undirected Graph G = (V, E) is stored in an adjacency matrix A using 0 and 1. If we want to know whether there is a path with length m between V_i and V_j, then we only need to check whether or not M[i][j] is zero where M is ? (if M[i][j] is 0, then this means there doesn't exist such a path)*

(A) A

(B) mA

(C) A^{m-1}

(D) A^m

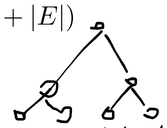
Question 2. *Which of following statements is true?*

(A) In a directed graph, the maximal number of edges is $|V|(|V|-1)$ $\times$

(B) In an undirected graph, a simple path has no repetitions of $\cancel{\textit{edges}}$.

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

(D) If Graph G = (V, E) is also a tree, then $|V| = |E| + 1$ ✓



Question 3. *Undirected Graph G = (V, E) is stored in an adjacency matrix A. What is the degree of V_i?*

(A) $\sum_{i=1}^{|V|} A[i][j]$

(B) $\sum_{j=1}^{|V|} A[i][j]$

(C) $\sum_{i=1}^{|V|} A[j][i]$

(D) $\sum_{i=1}^{|V|} A[i][j] + \sum_{j=1}^{|V|} A[j][i]$

Question 4. *At the beginning of fall semester, Keyi, an experienced TA of CS101, was assigned a task to divide students into different discussion classes. He was a master of Data Structure and thus he decided to use disjoint sets based on votes for discussion sessions. Each class corresponds to a union tree after implementation of disjoint sets with union-by-size optimization. We know that Class1 has a union tree with height 5, thus the possible number of students in Class1 is ?*

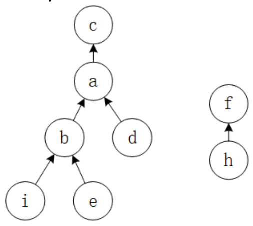
(A) 25

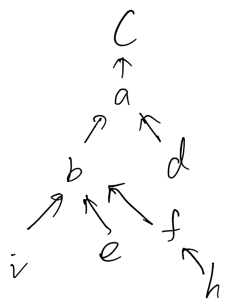
(B) 30

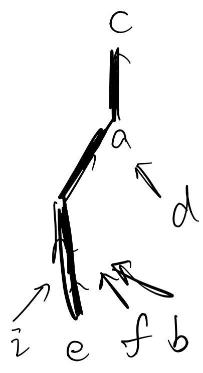
(C) 35

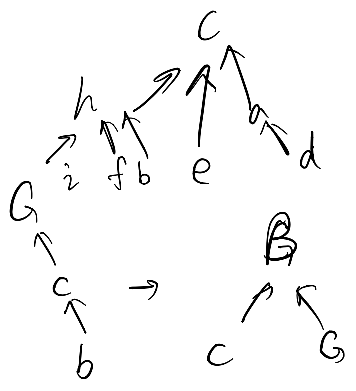
(D) 40

Question 5. *Suppose we have disjoint sets shown as the following figure. After we do set_union(e, h) and find(e) (consider all optimization), what's the depth of node i?*
A. 1
B. 2
C. 3
D. 4









Question 6. *Which of the following is TRUE?* ACD

(A) We should use an adjacency list instead of adjacency matrix to permanently store the graph in which there are 100 vertices and 200 edges ✓

(B) Adjacency list is better than adjacency matrix when inserting a new vertex ✗

(C) We can use DFS to determine whether a graph is bipartite ✓

(D) In a connected undirected graph, we can use BFS to find the shortest path between any vertices. ✓

(E) A directed graph with n vertices has at least n edges to ensure the whole graph is strongly connected.

2 (9') Weighted Graph

A *weighted graph* is a graph such that each edge has a weight (a number). The weights can be represented with an adjacency matrix, whose elements contain the weights (instead of boolean values). Note that this representation does not distinguish the case that there is an edge with weight 0 from the case that there is no edge. For this reason, we interpret a 0 in the matrix to mean that there is no edge.
Let A be the adjacency matrix of a weighted graph, G, with the columns and rows labelled, in order, by the vertices of the set $V = v_0, v_1, v_2, v_3, v_4, v_5$ That is, the first row of the matrix represents the adjacencies of vertex v_0, the second row those of vertex v_1, etc. Note that there are some non-zero diagonals which means that there are some edges of the form (v, v).

$$A = \begin{bmatrix} 0 & 1 & 3 & 0 & 0 & 2 \\ 1 & 1 & 0 & 4 & 5 & 0 \\ 3 & 0 & 0 & 5 & 4 & 0 \\ 0 & 4 & 5 & 2 & 0 & 6 \\ 0 & 5 & 4 & 0 & 0 & 2 \\ 2 & 0 & 0 & 6 & 2 & 3 \end{bmatrix}$$

(1)(3pts) Write down the adjacency list for the graph, such that the edges in the list are ordered *by increasing weight*. (If two edges have the same weight, then order them by their index order.)

$v_0 - v_1, v_5, v_2$
$v_1 - v_0 v_1 v_3 v_4$
$v_2 - v_0 v_4 v_3$
$v_3 - v_3 v_1 v_2 v_5$
$v_4 - v_5 v_2 v_1$
$v_5 - v_4 v_5 v_3$

(2)(3pts) Perform a (preorder) depth first search on G, beginning at vertex v_0. Use the ordering in the adjacency list to determine the ordering of vertices visited.

recursion : $v_0 v_1 v_3 v_2 v_4 v_5$
stack: $v_0 v_1 v_5 v_2 v_4 v_3$

(3)(3pts) Perform a breadth first search of the graph, G, again beginning at v_0 and using the ordering in the adjacency list.

$v_0 v_1 v_5 v_2 v_3 v_4$

3 (9') Disjoint Set

Part(A): Consider a list of cities $c_1, c_2, \ldots, c_n$. Assume we have a relation R such that, for any i, j, $R(c_i, c_j)$ is 1 if cities c_i and c_j are in the same province, and 0 otherwise.

(1)(1pts) If R is stored as a table, how much space does it require? Because R must have an entry for every pair of cities. and there are $\Theta(n^2)$ of these

(2)(2pts) Using a Disjoint Sets ADT, how do we construct it from R? Briefly explain it and analyze the time complexity. (Optimization should be implemented)

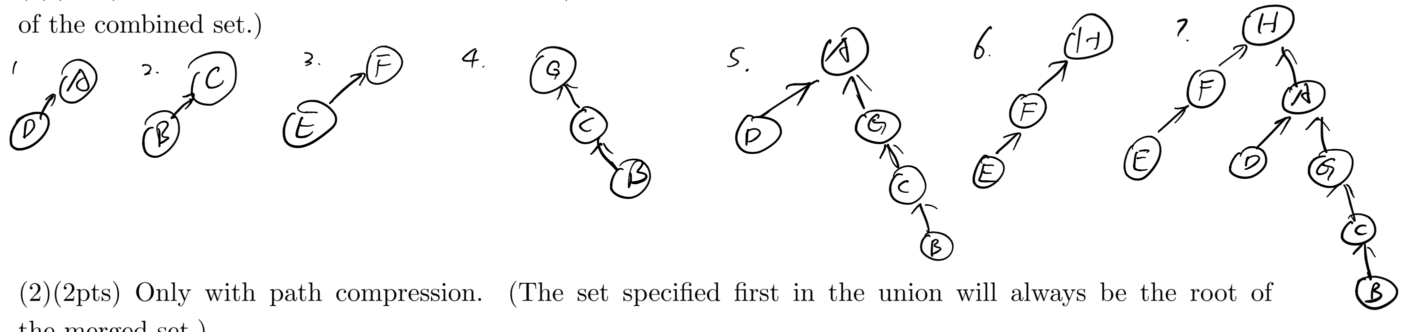
```
for i = 1 to n do
  make-set(c_i)
  for j = 1 to n do
    if R(c_j, c_i) then
      union(c_j c_i)
      break
```

(3)(2pts) When the cities are stored in the Disjoint Sets ADT, if you are given two cities c_i and c_j, how do you check if they are in the same province? (specific function mentioned in courseware should be used here)

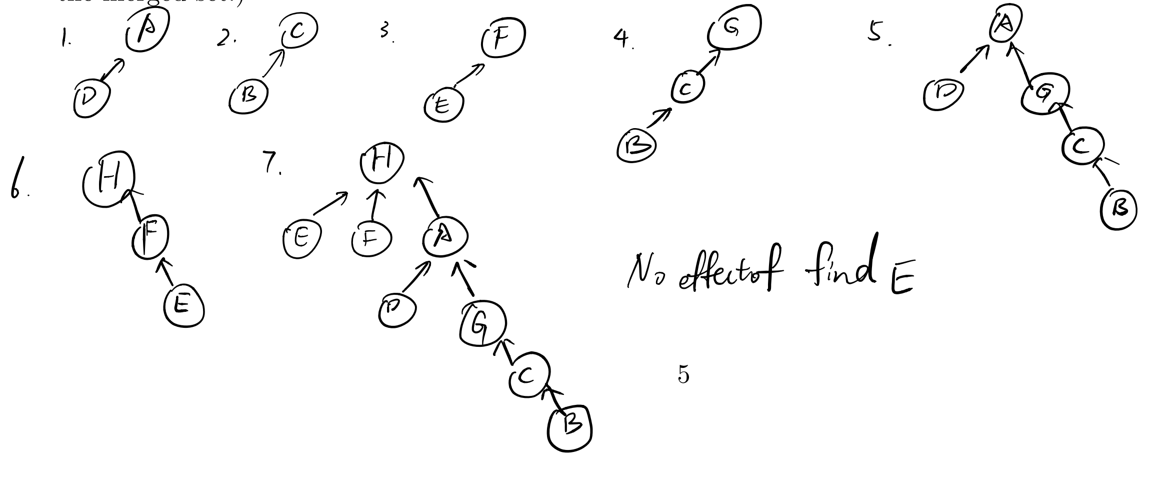
c_i and c_j are in the same province iff find-set(c_i)=find-set c_j

Part(B): Given the following set of operations, show the final disjoint set tree for each of the following union strategies.
set_union(A, D), set_union(C, B), set_union(F, E), set_union(G, C),
set_union(D, G), find(A), set_union(H, E), set_union(E, G), find(E)

(1)(2pts) Without considering optimization. (The set specified first in the union will always be the root of the combined set.)



(2)(2pts) Only with path compression. (The set specified first in the union will always be the root of the merged set.)



No effect of find A

No effect of find E