Junhao Zhang “Freddie”

USC ID:

CSCI 570 – HW1

07/06/2021

*jzhang49@usc.edu*

**1. Graded Problems**

* **[Question 1]**

True. Because Gale-Shapely algorithm guarantees to find at least one solution. If the instance has exactly one solution, then doesn’t matter what versions are, they all find the one same solution.

* **[Question 2]**

False. For the graph below, its BFS tree and DFS tree are identical.

A picture containing person

Description automatically generated

* **[Question 3]**

True. Because based on the definition of the DAG, *a directed graph is a DAG if and only if it can be topologically ordered*[[1]](#footnote-1).

* **[Question 4]**

No, doesn’t exist such stable matching. One of the examples is:

|  |  |
| --- | --- |
| Student | The student’s preference list (from high to low) |
| a | |  |  |  | | --- | --- | --- | | b | c | d | |
| b | |  |  |  | | --- | --- | --- | | c | a | d | |
| c | |  |  |  | | --- | --- | --- | | a | b | d | |
| d | |  |  |  | | --- | --- | --- | | b | c | a | |

Whoever is paired with *d*, the student and a student from the other pair will prefer each other to their current roommates.

* **[Question 5]**

|  |  |
| --- | --- |
| order | function |
| 1 |  |
| 2 |  |
| 3 |  |
| 4 |  |
| 5 |  |
| 6 |  |
| 7 |  |

* **[Question 6]**

Let’s denote *G* to be the given graph, and *T* to be the BFS tree of *G*. Since a tree doesn’t have cycle, thus if all edges of *G* exist in *T*, that means *G* doesn’t have cycle. Otherwise, *G* has at least one cycle, then we output one. The time complexity of BFS is *O* (*m + n*), thus the time complexity of this algorithm is also *O* (*m + n*). The next thing is outputting a cycle if there is one. If *G* has a cycle, then there has to be an edge *e* such that *e* exists in *G* but not in *T*. Let *x* and *y* be the two nodes that are connected by *e*. In the BFS tree *T*, *x* and *y* must exist in *T* and they must have a least common ancestor, and they must both have a unique path to that least common ancestor. The path from *x* to the LSA, the path from *y* to the LSA, and the edge *e* they together construct a cycle, and we can output this cycle.

**2. Practice Problems**

* **[Question 1]**

Our algorithm is this: for each student that hasn’t been assigned to a hospital, the student applies to the most preferred hospital that hasn’t been applied yet on the list, and if the hospital still has positions, then the student is assigned to that hospital, and the hospital keeps the hired students in the order of preference. Otherwise, if the hospital doesn’t have available positions, then compares the student with the least preferred hired student, if hospital prefers the new student to the least preferred one, then removes (“fires”) the least preferred one and hire the new student and keeps the new list of hired students in the order of preference. But if the hospital still prefers the least preferred one to the new student, then rejects the new student and the student applies to the next hospital in the preference list that hasn’t applied yet. Move to the next student if either this student is assigned to a hospital, or all hospitals rejected the student. Keep doing until no student can be assigned to a hospital.

*Proof*

We are going to prove the correctness of this algorithm by contradiction, for doing that we first assume that there are either first type or second type of instabilities after we ran the algorithm, and then we will show the contradiction case by case.

If have first type instability, then we have a contradiction: since student *s’* is unassigned, based on our algorithm *s’* must had been rejected by all hospitals, so there cannot be a hospital *h* that still prefers *s’* to any of its existing hired students *s*.

If have second type instability, then we have two cases.

Case 1: *s’* applied *h* before. But we would have a contradiction because based on our algorithm *h* must rejected *s’* due to still prefers the least preferred hired student to *s’*. Thus, *h* cannot still prefer *s’* to any of its existing hired students *s*.

Case 2: *s’* didn’t apply *h*. Then we have contradiction as well, because based on our algorithm, *s’* must prefers *h’* to *h*. Thus, *s’* cannot prefer *h* to *h’*.

* **[Question 2]**

|  |  |
| --- | --- |
| order | function |
| 1 |  |
| 2 |  |
| 3 |  |
| 4 |  |
| 5 |  |
| 6 |  |

* **[Question 3]**

1. No. One of the counterexample is:

*f* (*n*) = 2 \* (1 + ), *g* (*n*) = 1 +

As we can see, *f* (*n*) = *O* (*g* (*n*)). However, , since:

,

, and the greater *n* is, the closer is to 0.

Thus, we can observe that when *n* becomes infinitely large, would be equal to , which is equal to *1*. And = *0*. But *1*.

1. No. One of the counterexample is:

*f* (*n*) = 2*n*, *g* (*n*) = *n*

As we can see, *f* (*n*) = *O* (*g* (*n*)). However, , since , and .

1. Yes. Based on the definition, there exists a positive real number *c* such that:

*0* ≤ *f* (*n*)≤ *cg* (*n*)

And we can observe that:

*0* ≤≤

Thus,  *= O.*

* **[Question 4]**

We will prove it by contradiction. First, we assume there is an edge *e* such that *e* exists in *G* but doesn’t exist in *T*. Since *T* has all nodes that *G* has, it must also have the two nodes that *e* connects, and because *T* is a DFS tree, thus one of the two nodes must be the ancestor of another. Let’s denote *x* be the ancestor, and *y* be the descendent. Moreover, since *e* connects *x* and *y*, thus the distance between these must be 1. However, there is a contradiction: since *x* is the ancestor of *y* and their distance is 1, thus when running BFS from *x*, *y* must be added to *T* as the direct child of *x*, which means *e* exists in T, but we already assumed *e* doesn’t exist in T. Contradiction! Therefore, we proved *G* cannot contain any edges that do not belong to *T*.

1. # *Directed acyclic graph*, wikipedia.org, https://en.m.wikipedia.org/wiki/Directed\_acyclic\_graph

   [↑](#footnote-ref-1)