# Midterm 1 Sample Solution

**NOTE:** Throughout the exam a *simple* graph is an undirected, unweighted graph with no multiple edges (i.e., no exact repeats of the same edge) and no self-loops (i.e., no edges from a vertex to itself). Graphs are simple unless stated otherwise, and even where we explicitly contradict one of these, the rest remain true. So, for example, a "directed graph" with no other information specified would be unweighted with no multiple edges and no self-loops.

## 1 Asymptotically yours

Here are two statements about non-negative functions $f(n)$ and $g(n)$ over the positive integers. For each, indicate whether it is

- **Always** true for every possible pair of non-negative functions $f(n)$ and $g(n)$.
- **Sometimes** true for pairs of non-negative functions $f(n)$ and $g(n)$ (and false for other pairs $f(n)$, $g(n)$).
- **Never** true for any possible pair of non-negative functions $f(n)$ and $g(n)$.

Here are the statements:

1. If $f(n) = cg(n) + o(g(n))$ for some constant $c$ then $f(n) = O(g(n))$.

   ● Always ○ Sometimes ○ Never

   **Additional explanations:** $o(g(n))$ is a lower order term, and can be ignored.

2. If $\log f(n) = O(\log g(n))$ then $f(n) = O(g(n))$.

   ○ Always ● Sometimes ○ Never

   **Additional explanations:** Note that $\log n^2 = 2 \log n$. So the statement is true for $f(n) = n$ and $g(n) = n^2$, but false for $f(n) = n^2$ and $g(n) = n$.

Now, give examples of nonnegative functions $f(n)$ and $g(n)$ satisfying the following relationships (you should use a different example for each relationship):

3. $f(n) \neq g(n)$ but $f(n) = g(n) + h(n)$ where $h(n) = o(g(n))$.

$$f(n) = \boxed{n+1} \qquad g(n) = \boxed{n}$$

4. $f(n) = O(g(n))$ but $f(n) \neq \Theta(g(n))$.

$$f(n) = \boxed{n} \qquad g(n) = \boxed{n^2}$$

5. $f(n) = O(g(n))$ but $2^{f(n)} \neq O(2^{g(n)})$.

$f(n) = \boxed{2n}$ $\qquad g(n) = \boxed{n}$

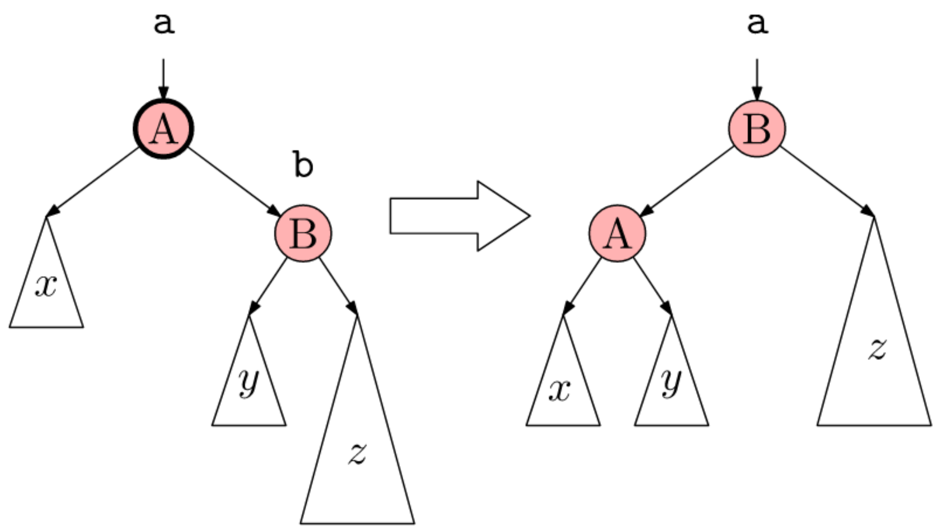
## 2 Knowing your trees

1. Let $T$ be a binary search tree with $n$ keys. In Assignment #1, we considered storing in each node $N$ of the tree the size of the subtree rooted at $N$, and you described an algorithm that uses this information to count in $O(\log n)$ time the number of keys $x$ of $T$ such that $k_1 < x < k_2$ for some given parameters $k_1$ and $k_2$.

   Using the size information stored in each node is only half the job, however. Write appropriate code in each of the blanks in the following implementation of `insert_key` to keep the size information accurate when nodes are inserted in $T$. Assume that the size field of a node `node` is stored in `node.size`. Note that some of the blanks can (and should) be left blank, and the code does **not** have to maintain balance in the tree.

```
define insert_key(start_node, new_key)

    [                    ]

    if start_node == null:

        [                    ]

        newnode = new node(new_key)

        [ newnode.size = 1 ]

        return newnode

    if new_key < start_node.key():

        [ start_node.size++ ]

        start_node.left_child = insert_key(start_node.left_child, new_key)

    else if new_key > start_node.key():

        [ start_node.size++ ]

        start_node.right_child = insert_key(start_node.right_child, new_key)

    return start_node
```

2. Now consider a self-balancing AVL tree. Describe how to update the size fields of nodes $A$ and $B$ after a single rotation as illustrated in the figure is performed (we will spare you the other types of rotations). Note that $a$ refers to the parent of the node containing key $A$, and that $x$, $y$ and $z$ represent subtrees. Ignore the label $b$ that appears in the picture; it plays no role in this question.




A.size = $\boxed{\text{x.size + y.size + 1}}$ or $\boxed{\text{A.left\_child.size + A.right\_child.size + 1}}$

B.size = $\boxed{\text{A.size + z.size + 1}}$ or $\boxed{\text{A.size + B.right\_child.size + 1}}$

## 3 More exchange of kidneys

Recall that an instance of the Kidney Exchange Problem (KEP) is ($n$,M,R), where $n$ is the number of patients (and also the number of donors), M is an initial perfect matching of patients and donors, and R is a collection of complete rankings R[ $p$ ] for each of the $n$ patients $p$. The donor $d$ initially matched with $p$ may not be at the top of $p$'s ranking.

For an instance $I$ of KEP, the **best-preference graph** $G_I$ has one node per pair $(d, p)$ in the initial matching M. There is a directed edge from $(d, p)$ to $(d', p')$ if $d'$ is the most highly ranked donor on $p$'s preference list.

The *Exchange* algorithm of Assignment 2 for matching patients and donors ignores the reality that for large $n$, it is impractical to get $n$ pairs of patients and donors in the same hospital and do all the exchanges (surgeries) at the same time. In practice, exchanges typically involve just two pairs.

Recall that for a given instance $I$ of KEP, a valid solution $S$ is a perfect matching between patients and donors. $S$ is a *2-exchange* if the following holds for all pairs of patients $p$ and $p'$: If $(p, d)$ and $(p', d')$ are in the initial matching $M$ and $(p, d')$ is in $S$, then $(p', d)$ is also in $S$.

We say that 2-exchange $S$ is *stable* if there do not exist distinct pairs $(d, p)$ and $(d', p')$ such that patient $p$ prefers $d'$ to $d$ **and also** patient $p'$ prefers $d$ to $d'$.

1. Show an instance of KEP with $n = 3$ for which there is no stable 2-exchange.

**Solution:** The preference lists are as follows, and the initial matching is $p_1 : d_1$, $p_2 : d_2$, and $p_3 : d_3$.

$$\begin{aligned} p_1 &: \quad d_2, d_3, d_1 \\ p_2 &: \quad d_3, d_1, d_2 \\ p_3 &: \quad d_1, d_2, d_3 \end{aligned}$$

2. Is the number of 2-exchanges upper bounded by a polynomial in $n$, for all instances $I$ with $n$ patients?

   ○ Yes   ● No

   **Additional explanations:** The number of 2-exchanges is equal to the number of ways we can divide the $n$ patients into pairs. This is $(n-1)\cdot(n-3)\cdots 3\cdot 1$ which is approximately equal to $\sqrt{n!}$.

3. Are there instances $I$ with $n$ patients for which the number of 2-exchanges exponential in $n$, for sufficiently large $n$?

   ● Yes   ○ No

   **Additional explanations:** Observe that $\sqrt{n!}$ grows faster than any single exponential $k^n$. So the number of 2-exchanges is (at least) exponential in $n$.

4. Let *2Exchange* be the following algorithm (a variant of the *Exchange* algorithm from Assignment 2), which arranges exchanges involving two patient-donor pairs, as well as degenerate exchanges involving just one patient-donor pair. The algorithm never arranges exchanges involving three or more pairs.

   **Algorithm** *2Exchange* ($I = (n,\text{M},\text{R})$)
   Initialize $S$ to be the empty set
   While $n > 0$
   Create the best-preference graph $G_I$ for instance $I$
   If $G_I$ has a directed cycle, say $v_1, v_2, ..., v_k = v_1$, for $k = 2$ or $k = 3$ then
   Let $v_i$ be the pair $(d_i, p_i)$, for $1 \le i < k$
   Update instance $I$ as follows:
   Remove the pairs $(d_i, p_i)$ from M, $1 \le i < k$
   Remove $d_i$ from all patient rankings in R, $1 \le i < k$
   Set $n$ to $n - k + 1$
   Add the pairs $(d_{i+1}, p_i)$ to $S$ for $1 \le i < k-1$, and also add the pair $(d_1, p_{k-1})$ to $S$
   Else
   Remove all remaining pairs from M and add them to $S$
   Set $n$ to 0
   Endwhile
   Return $S$

   Is the output of Algorithm *2Exchange* always the same on any instance, no matter which directed cycle (with $k = 2$ or $k = 3$) is chosen at each iteration of the While loop?

   ● Yes   ○ No

   **Additional explanations:** All of the cycles are disjoint, because a vertex has only one outgoing edge. So the order in which we choose them does not matter.

5. Does Algorithm *2Exchange* always produce a 2-stable solution on instances $I$ that have a 2-stable solution?

   ○ Yes   ● No

**Additional explanations:** Consider four (patient, donor) pairs with the following preferences, and the initial matching $p_1 : d_1$, $p_2 : d_2$, $p_3 : d_3$ and $p_4 : d_4$.

$$\begin{array}{ll} p_1: & d_2, d_3, d_4, d_1 \\ p_2: & d_3, d_4, d_1, d_2 \\ p_3: & d_4, d_1, d_2, d_3 \\ p_4: & d_1, d_2, d_3, d_4 \end{array}$$

The best-preference graph for this instance is a four-cycle, which means that algorithm *2Exchange* will do nothing and retain the initial matching. This initial matching is unstable, since $p_1$ likes $d_3$ better than $d_1$, and $p_3$ likes $d_1$ better than $d_3$. Thus algorithm *2Exchange* returns a solution that is not 2-stable.

The following solution is 2-stable: $p_1 : d_3$, $p_2 : d_4$, $p_3 : d_1$, $p_4 : d_2$.

## 4 More on graphs

In this problem, let $G$ be a simple, connected graph with at least two nodes. By the *depth* of a rooted tree, we mean the number of edges of the longest path from the root to a leaf.

For each of the following statements about graph $G$, indicate whether the statement is:

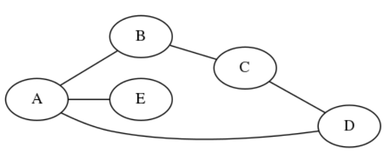
- Always true for every simple connected graph $G$ with at least two nodes
- Sometimes true for a simple connected graph $G$, but not for other simple graphs $G$ with at least two nodes
- Never true for any simple graph $G$ with at least two nodes.

Here are the statements:

1. Two different depth first search (dfs) trees rooted at the same node $s$ of $G$ have the same depth.

   ○ Always ● Sometimes ○ Never

   **Additional explanations:** Consider the following graph:

   A dfs that starts at B, and visits C first, will given the tree B – C – D – A – E (depth 4). If the dfs visits A first, however, then it will give a tree with depth 3.

2. Two different dfs trees rooted at two *different* nodes $s$ and $s'$ of $G$ have the same depth.

   ○ Always ● Sometimes ○ Never

   **Additional explanations:** Every dfs of the graph shown above that starts at E will give a tree with depth 4; a dfs that starts at A will only give a tree with depth 3.

3. Two different breadth first search (bfs) trees rooted at the same node $s$ of $G$ have the same depth.

   ● Always ○ Sometimes ○ Never

**Additional explanations:** The depth of a bfs starting at any node $N$ is determined by the node furthest from $N$ in the graph (in terms of the length of the shortest path from $N$ to that node). This does not depend on the order in which the neighbours of each node are visited.

4. Two different bfs trees rooted at two *different* nodes $s$ and $s'$ of $G$ have the same depth.

   ○ Always ● Sometimes ○ Never

   **Additional explanations:** Every bfs of the graph shown above that starts at E will give a tree with depth 3; a bfs that starts at A will only give a tree with depth 2.

## 5 Triathlon reductions

The *Triathlon-stable matching* problem is defined as follows. An instance involves $3n$ athletes: $n$ swimmers, $n$ runners and $n$ cyclists. Each athlete in one group has a preference list for the athletes in each of the other two groups (that is, a runner will have a preference list for the swimmers, and another separate preference list for the cyclists, and so on). You want to find $n$ triples (groups of three), with one swimmer, one runner and one cyclist per triple, to compete in a triathlon. Ideally your solution of $n$ triples should have no instabilities. The first part of this problem asks you to come up with a reasonable notion of instability.

1. Complete the following definition of instability applied to this new situation. With respect to a solution $S$, an instability consists of two triples $(s, r, c)$ and $(s', r', c')$ of $S$ such that

   **Solution**: An instability is one of the following three situations:

   - $s$ likes both $r'$, $c'$ better than $r$, $c$ respectively **and** $r'$, $c'$ both like $s$ better than $s'$.
   - $r$ likes both $s'$, $c'$ better than $s$, $c$ respectively **and** $s'$, $c'$ both like $r$ better than $r'$.
   - $c$ likes both $r'$, $s'$ better than $r$, $s$ respectively **and** $r'$, $s'$ both like $c$ better than $c'$.

   We accepted several other answers (some without any penalty, some with some penalty).

2. Let $A$ be an algorithm that, given the preference lists of all athletes, always finds solution with no instabilities, if the instance has such a solution. Complete the following reduction that uses $A$ to solve the regular stable matching problem:

   Given an instance $I$ of **Stable Matching** we transform it into an instance of **Triathlon-stable matching** as follows:

   **Solution**: Here is a reduction that seems reasonable. This was not the only reduction we gave full marks to. Interestingly, we have not yet been able to prove that **any** reduction we came up with works correctly. We'll award bonus points to anyone who manages to come up with a proof of correctness for this or any other reduction.

   We perform the following steps:

   - we add cyclists $c_1, \ldots, c_n$.
   - we rename hospital $h_i$ to swimmer $s_i$. Its preference list for the $r_i$'s (renamed from **r**esidents to **r**unners) remain the same. Its preference list for the cyclists mirrors its preference list for runners. That is, if $r_k$ occurs in position $j$ on $s_i$'s preference list, then $c_k$ occurs in position $j$ on $s_i$'s preference list
   - The preference list of $r_i$ for the $s_i$'s is the same as its preference list for the $h_i$'s in the instance of stable matching. Its preference list for the cyclists mirrors its preference list for the $s_i$'s.
   - Every cyclist $c_i$ has the same preference list for $r_1 \ldots r_n$ as $s_i$, and the same preference list for $s_1 \ldots s_n$ as $r_i$.