CS 310 'Algorithms' **Assignment-3**
Due date: Monday, November 19, 2018 at 11:55 PM. [Total Marks = 65]

---

**IMPORTANT:** Read the instructions at the end of this document and follow the naming conventions.
Marks for writing well-structured and efficient code. **[4 marks]**

---

## Stable Matching

---

**Q1.** **[10 marks]**
We have a set $A=\{a_1, a_2, a_3, \ldots a_n\}$ of $n$ airports and a set $P=\{p_1, p_2, p_3, p_n\}$ of $n$ airplanes. There is hangar space available at each of the airports for a **single** plane. Each of the airports have a preference list where they rank all the $n$ airplanes and each of the airplanes have a preference list where they rank all of the $n$ airports.

Some of the airplanes, however, are big in size and can only be accommodated in hangars large enough to hold them. There will, thus, be some airport-airplane pairs which are invalid because of the mismatch in their sizes. We define a set $X$ which contains all such invalid pairs.

Design an algorithm that takes as input the preference lists of each of the airports and airplanes and the set $X$ and produces a stable matching. **Note that it is possible that the matching produced by your algorithm is not perfect.** This stable matching should not have any instability and should not contain any pairs from set $X$.

For this question, you have to do the following:

1. Code an **efficient** C/C++ algorithm that takes as input the preference lists of airports and airplanes and the set X and produces a stable matching as described above. [7]
2. In the comments at the beginning of your code, describe how your algorithm works and terminates. **What is the running time of your algorithm and how much space does it use?** Your should include clear comments that explain your algorithm's time and space complexity. [1]
3. You should create at least two different input files to test your algorithm for different scenarios (see format below). [2]

Your code will read input from a text file. The format of the file is as follows. The first line contains the value of 'n', followed by the preference lists of 'n' airplanes (represented by P) and preference lists of 'n' airports (represented by A), respectively. The last line contains the set X. The excluded pairs in X are separated by commas as shown below. E.g., Pair P2 A2 means that the plane P2 cannot be housed in airport A2.

**Input :**
```
n 4
P1: A1, A3, A2, A4
P2: A3, A1, A2, A4
P3: A4, A3, A2, A1
P4: A2, A1, A4, A3
```

A1: P1, P2, P3, P4
A2: P3, P2, P1, P4
A3: P4, P3, P2, P1
A4: P2, P1, P4, P3
X: P2 A2, P1 A4

**Output:**
Stable Matching: A1 P1, A2 P3, A3 P4, A4 P2

**Q2.** **[10 marks]**

The archeological department is organizing an excavation and there are $n$ teams participating in it. This excavation is spread over $n$ different locations and the teams move from one location to another after finishing work at the previous location. The organizers want that no two teams should visit the same location at the same time. They have made different visiting schedules for these teams. Due to lack of funds, they want to stop the movement of the teams and assign each of them to one location for a year. Remember that the teams will be moving according to their schedules and we want to find where each of the teams should finally stop, while fulfilling the following conditions:

1. No two teams can be at the same location at the same time. For example, if a team $T_1$ has chosen location $L_3$ as its final stopping destination then no other team can visit $L_3$ after that.
2. Each team should be assigned a different final destination.

You have to do the following:

1. Design an **efficient** C/C++ algorithm that takes as input the location schedules for each of the teams and outputs a final destination for each of the teams, while fulfilling the conditions mentioned above. [8]
2. In the comments at the beginning of your code, describe how your algorithm works and terminates. **What is the running time of your algorithm and how much space does it use?** Your should include clear comments that explain your algorithm's time and space complexity. ***Hint:*** *Do not reinvent the wheel! Use your knowledge of the stable matching algorithm to solve this problem. You can think of these schedules as location preference lists for the teams.* [2]

**EXAMPLE:** Suppose following is the time schedule for the teams.

| | Day 1 | Day 2 | Day 3 | Day 4 | Day 5 | Day 6 |
|---|---|---|---|---|---|---|
| Team 1 ($T_1$) | $L_1$ | — | $L_2$ | — | $L_3$ | — |
| Team 2 ($T_2$) | — | $L_1$ | — | $L_2$ | — | $L_3$ |
| Team 3 ($T_3$) | $L_2$ | — | $L_3$ | — | $L_1$ | — |

The correct final destinations of each of the teams is ($L_1$,$T_3$), ($L_2$,$T_2$), ($L_3$,$T_1$).

Your code will read input from a text file. The format of the file is as follows. The first line contains the value of 'n', followed by the schedule of each of the teams. The input of above example is shown below.

**Input :**

```
n 3
T1: L1,-, L2,-, L3,-
T2: -,L1,-, L2,-, L3
T3: L2,-, L3,- L1,-
```

**Output:**

```
Final Destinations: L1 T3, L2 T2, L3 T1
```

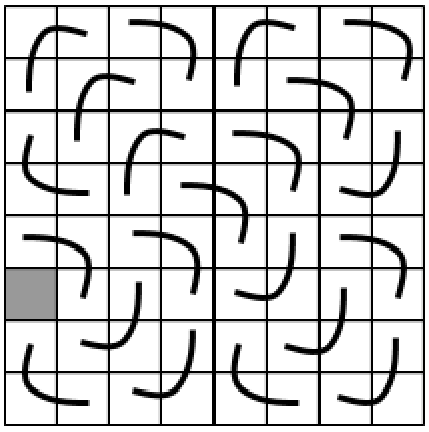
---

## Divide and Conquer

---

**Q3.** **[11 marks]**

In this question you will implement a board game using the Divide and Conquer approach. You are given an **nxn** square board, where n is a power of 2. **One** of the squares on the board is blank and you have to fill the remaining squares with boomerangs. Each boomerang occupies **three** squares that form a **right angle**. See the illustration below. The greyed square is blank and all the remaining squares should be completely filled with boomerangs. No boomerangs will share any squares.



1. Code an **efficient** algorithm in C/C++ that asks the user for the value of 'n' and the row and column indices of the blank square. It will then fill all the remaining squares with boomerangs. In order to print boomerangs on your screen, you will use different alphabet letters as shown in the illustration below. You program should output an nxn matrix with alphabet letters on the screen. [8]

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| A | A | B | B | D | D | E | E |
| A | C | C | B | D | F | F | E |
| K | C | J | J | H | H | F | G |
| K | K | J | I | I | H | G | G |
| L | L | M | M | I | N | O | O |
| ▨ | L | T | M | N | N | R | O |
| U | T | T | S | Q | R | R | P |
| U | U | S | S | Q | Q | P | P |

2. Give a clear description of your algorithm and the data structures used to implement it in the comments at the beginning of your code. **What is the recurrence relation of your algorithm? What is the running time of your algorithm and how much space does it use?** Your should include clear comments that explain your algorithm's time and space complexity. [2+1]

**Q4.** **[15 marks]**

You are given a **complete binary tree** of height h and $n=2^h$ leaves. Each vertex in the tree has an integer value associated with it. We have numbered the leaves from $x_1$ to $x_n$ starting from left to right. An ancestor of a vertex is its parent, grandparent etc., up to the root of the tree. We define the following:

Ancestry($x_i$) of a leaf nodes as the **set of vertices** including $x_i$ and all its ancestors.

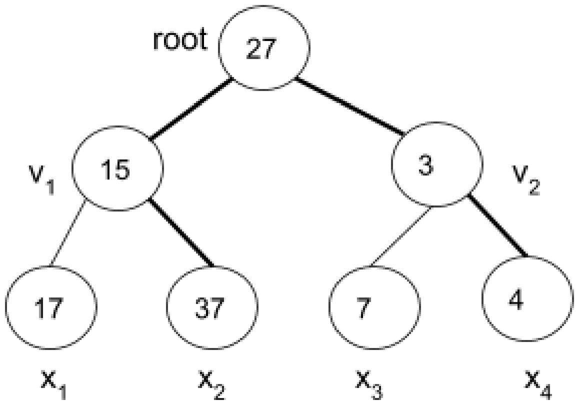
Ancestry($x_{i,}$ $x_j$) of a pair of leaves is defined as **union** of their respective ancestries, i.e., Ancestry($x_i$) U Ancestry($x_j$)

*Example:* Consider the following binary tree.

Ancestry($x_2$) = {$x_2$, $v_1$, root}

Ancestry($x_4$) = {$x_4$, $v_2$, root}

Ancestry($x_2$,$x_4$) = Ancestry($x_2$) U Ancestry($x_4$) = {$x_2$, $v_1$, root, $x_4$, $v_2$}



Value of Ancestry($x_2$) = 37+15+27

Value of Ancestry($x_4$) = 4+3+27

Value of Ancestry($x_2, x_4$) = 37+15+27+3+4 = 86

You have to describe a **Divide and Conquer (D&C)** algorithm that finds **two** leaves $x_i$ and $x_j$ such that **Value** of Ancestry ($x_i$,$x_j$) is **maximum**.

1. Code an **efficient** algorithm in C/C++ that solves the above problem. Your code will first generate a complete binary tree and assign random integer values to all the nodes in the tree. Your code will then run the divide and conquer algorithm on the tree to find $x_i$ and $x_j$ that maximize the value of Ancestry. Your algorithm should list the value of nodes in Ancestry ($x_i$), value of nodes in Ancestry ($x_j$) and Value of Max Ancestry ($x_i$, $x_j$). See output format below. [12]
2. Give a clear description of your algorithm and the data structures used to implement it in the comments at the beginning of your code. **What is the recurrence relation of your algorithm? What is the running time of your algorithm and how much space does it use?** Your should include clear comments that explain your algorithm's time and space complexity. [2+1]
3. You algorithm should run for large values of 'n'. The above example is only for illustration purposes. For testing, remember to change the seed of your random number generator in different executions of the algorithm.
4. You will get partial credit if you design a $O(n\log_2 n)$ D&C algorithm. It is possible to design a D&C algorithm for this problem that runs in $O(n)$ time.

**Output format:**
xi = x1
xj = x2
Value of nodes in Ancestry (x1) = {17, 15, 27}
Value of nodes in Ancestry (x2) = {37, 15, 27}
Value of Max Ancestry (x1, x2) = 96 /* **NOTE:** 17+37+15+27 = 96. Common ancestors $v_1$ and root are only counted ***once*** since it is a ***union*** of sets. */

**Q5. [15 marks]**

You are given 'n' encrypted photos. The images in the photos belong to different biological species. You have to determine if there are more than $\frac{n}{2}$ photos that belong to the same species or not. Since the photos are encrypted, you can't examine them directly and have to **call a function** that takes as input **two** photos and tells you if they belong to the same species or not. Assume there can be at most 'm' different species, where $m<n$.

1. Code an **efficient** Divide & Conquer (D&C) algorithm in C/C++ that solves the above problem. Your code will ask the user for positive integer values of 'n' and 'm'. Your code will generate 'n' photos and randomly assign species to the photos. You will also write a helper function **decode(),** that takes two photos as parameters and returns "Y" if they belong to the same species, otherwise returns "N" (decode() is only allowed to compare two photos at a time and there is no way to inspect or compare the photos except through this function). If more than $\frac{n}{2}$ photos belong to

the same species, then your D&C algorithm will report “success” and list the indices of those photos, otherwise report “failure”. [12]

2. Give a clear description of your algorithm and the data structures used to implement it in the comments at the beginning of your code. **What is the recurrence relation of your algorithm? What is the running time of your algorithm and how much space does it use?** Your should include clear comments that explain your algorithm’s time and space complexity. [2+1]
3. The time complexity of your algorithm should not be more than $O(n\log_2 n)$.

## Instructions and policies

1. You must submit your ***own*** work. You may discuss the problems with other classmates but must not reveal the solution to others or copy someone’s work. Remember to acknowledge other classmates if discussions with them has helped you.
2. You should name your code files using the following convention: Qx-rollno.c
3. Type your answer to the theoretical questions and submit a separate pdf file for each using the naming convention above.
4. Upload all your files in Assignment-3 folder on LMS. **There will be a 20% deduction for assignments submitted up to one day late. Assignments submitted 24 hours after the deadline will not be marked.**
5. There will be vivas during grading of the assignment. The TAs will announce a schedule and ask you to sign up for viva time slots. Failure to show up for vivas will result in a **70% marks reduction** in the assignment.
6. In the questions where you are asked to create test cases. Think carefully about good test cases that check different conditions and corner cases. The examples given in the assignment are for clarity and illustration purposes. You should not assume that those are the only test cases your code should work for. Your code should be able to scale up to larger input sizes and more complex scenarios.
7. Do not make arbitrary assumptions about the input or the structure of the problem without clarifying them first with the Instructor or the TAs.
8. Make sure that your code compiles and runs on Ubuntu. You may choose to develop your code in your favourite OS environment, however, the code must be properly tested on Linux before submission. During vivas, your code should not have any compatibility issues. It's a good idea to use gcc -Wall option flag to generate the compiler's warnings. Pay attention to those warnings as fixing them will lead to a much better and robust code.
9. For full credit, comment your code clearly and state any assumptions that you have made. As mentioned in the beginning of the assignment, there are marks for writing well-structured and efficient code.
10. Familiarize yourself with LUMS policy on plagiarism and the penalties associated with it. We will use a tool to check for plagiarism in the submissions.