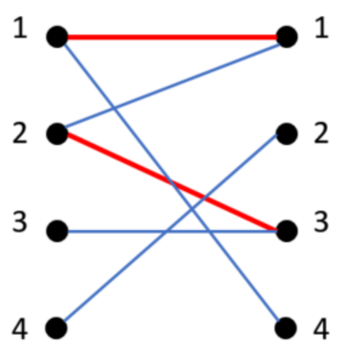
COMPSCI330 Design and Analysis of Algorithms
Assignment 8

Due Date: Wednesday, November 15, 2017

Guidelines

- **Describing Algorithms** If you are asked to provide an algorithm, you should clearly define each step of the procedure, and then analyze its overall running time. There is no need to write pseudo-code; an unambiguous description of your algorithm in plain text will suffice. If the running time of your algorithm is worse than the suggested running time, you might receive partial credits.
- **Typesetting and Submission** Please submit each problem as an *individual* pdf file for the correct problem on Sakai. LaTeX is preferred, but answers typed with other software and converted to pdf is also accepted. Please make sure you submit to the correct problem, and your file can be opened by standard pdf reader. **Handwritten answers or pdf files that cannot be opened will not be graded.**
- **Timing** Please start early. The problems are difficult and they can take hours to solve. The time you spend on finding the proof can be much longer than the time to write. If you submit within one week of the deadline you will get half credit. Any submission after that will not receive any credit.
- **Collaboration Policy** Please check this page for the collaboration policy. You are **not** allowed to discuss homework problems in groups of more than 3 students. **Failure to adhere to these guidelines will be promptly reported to the relevant authority without exception.**

Problem 1 (Maximum Matching)**.** (15 points)
We will run the perfect matching algorithm on the following graph:



Here the red edges are edges in the current matching. The blue edges are edges in the graph, but are not in the current matching.
(a) (10 points) Find two augmenting paths on the graph with respect to the red matching.
(b) (5 points) Find a maximum matching of this graph (list all the edges in the matching).

Problem 2 (Expensive Counters)**.** (20 points) In order to represent large integers, we can store them in a binary array. As an example, the number 11 can be represented by an array $A[] = [1, 1, 0, 1]$. The first element in the array represents the least significant bit, and in general the k-th element in the array represents 2^{k-1}. The array $A[]$ is a representation for 11 because $11 = 1+2+8$.

You are implementing a binary counter on a strange hardware. Your counter consists of n binary bits. The counter starts at 0, and has a function called "increase". If the counter is representing number x now, after calling "increase" it should represent the number $x + 1$ (counter resets to 0 if x becomes 2^n).

The hardware is very strange, so flipping the k-th bit in the counter actually takes 2^k time. Show that the amortized cost for the increase operation is $O(n)$.

(Hint: You can use any method that we talked about in class, but aggregate method and charging method are easier for this problem.)

(Hint: Your need to first figure out in the first n operations, how many times does the k-th bit get flipped.)

Problem 3 (Waiting in lines)**.** (20 points) Suppose there are n people that are trying to form lines to enter a basketball game. Initially, each person is in his/her own line. After that, two kinds of operations might happen: 1. Two lines might merge into one line. 2. A person enters the stadium, and is therefore not in any of the lines (and he/she will not come back and join any other lines).

You are now asked to design a data structure that can keep track of how many people are in each line. Your data structure should support the following operations:

- Merge(A, B). Merge the two lines where A and B are in. (A and B are not already in stadium, and they are guaranteed to be in different lines.)
- Enter(A). Person A enters the stadium.

- Count(A). Output the number of people in the same line with A. Person A is not already in the stadium.

As an example, after Merge(A, B), Merge(B, C), Enter(B), A and C will be in the same line. In this case Count(A) should return 2. Count(D) should return 1 because D has not joined the line with anyone else.

The amortized cost for all the operations should be $O(\alpha(n))$ where α is the inverse Ackermann function.

You need to analyze the running time of the algorithm (if you are using disjoint set with union-by-rank and path compression, you don't need to analyze the running time of that algorithm, but you should specify how you modified the algorithm and how much additional cost your algorithm incurs). You don't need to prove the correctness of your algorithm.