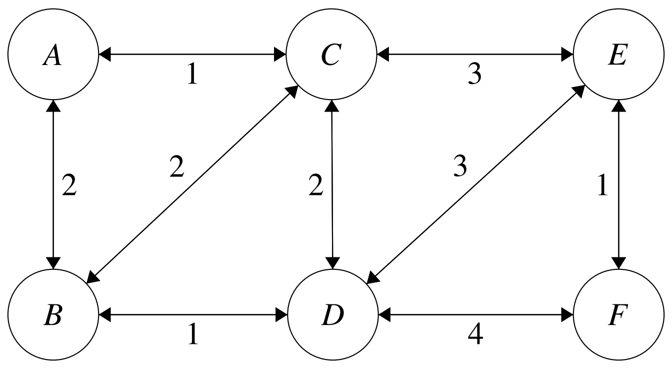
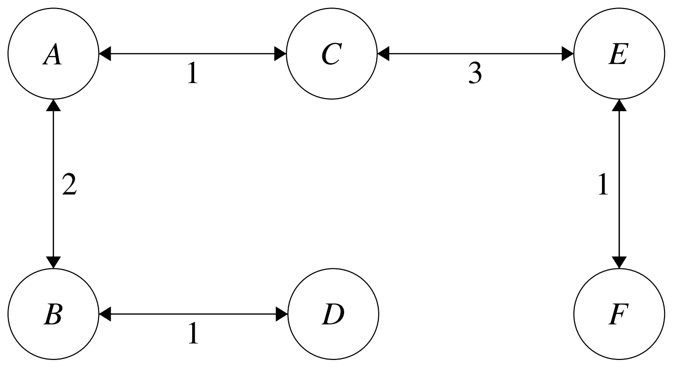
CS 61B Fall 2014 MSTs and Dynamic Programming

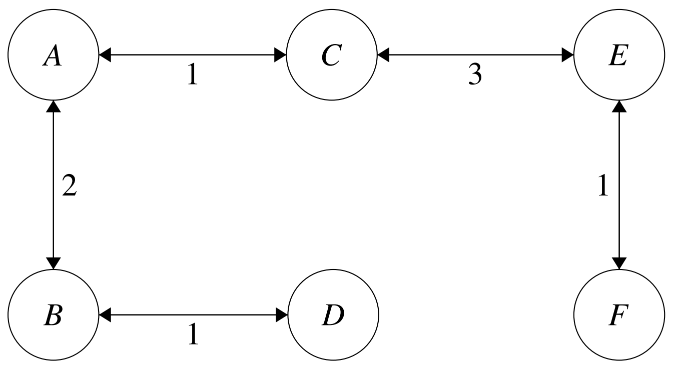
1 Minimum Spanning Trees



a) Perform Prim's algorithm to find the minimum spanning tree of the following graph. Pick A as the initial node. Whenever there are more than one node with the same cost, process them in alphabetical order.



b) Use Kruskal's algorithm to find a minimum spanning tree.



c) Bonus! There are quite a few MSTs here. How many can you find?

2 Dynamic Programming: Fibonacci

a) Write a recursive memoized version of the Fibonacci function. As a reminder, fib(n) = fib(n-1) + fib(n-2). fib(0) = 0 and fib(1) = 1. Hint: You may want to define a helper function

```
public static int fib(int n) {
        return fib(n, new HashMap<Integer, Integer>());
}

public static int fib(int n, HashMap<Integer, Integer> memo) {
        if (n == 0 || n == 1) {
                return n;
        }
        if (memo.containsKey(n)) {
                return memo.get(n);
        } else {
                memo.put(n, fib(n-1, memo) + fib(n-2, memo));
                return memo.get(n);
        }
}
```

b) What is the running time of your method?
$\Theta(n)$

3 Dynamic Programming: Maximum Subarray

You are given an array of integers, A. Find the subarray with the maximum sum. Let's suppose we were given an array containing the elements $\{-2,1,-3,4,-1,2,1,-5,4\}$. The maximum subarray is $\{4,-1,2,1\}$ with a sum of 6. Note that the empty subarray is valid, with a sum of 0. For example, given $\{-1,-2,-3\}$, you would return 0 for the subarray $\{\}$

a) Sometimes, we can define a problem in terms of subproblems. What might be an appropriate subproblem for this problem? Hint: If we know the the maximum sum of the array ending at index $i-1$, what do we know about the maximum sum of the array ending at index i?

Comments:
Let $S(i)$ be the maximum sum of the subarray ending at i.

$$S(i) = max(S(i-1)+A[i], 0)$$

If you have the maximum sum of the prefix of the array A ending at $i-1$, then to calculate the maximum sum of the prefix of the array ending at i we consider two possibilities. If the sum of the subarray ending at index $i-1$ plus $A[i]$ is positive, then that must be the maximum subarray sum ending at i including $A[i]$. In other words, given the maximum subarray ending at $i-1$, we check to see if tacking on item $A[i]$ gives us a positive sum. Otherwise, if the maximum sum is negative when we include $A[i]$, then we're better off just choosing the empty subarray.

b) Write an iterative method to solve the problem.

```
public static int maxSubarraySum(int[] A) {
        int maxSoFar = 0;
        int[] memo = new int[A.length];
        memo[0] = Math.max(A[0], 0);
        for (int i = 1; i < A.length; i++) {
                memo[i] = Math.max(0, A[i] + memo[i-1]);
                maxSoFar = Math.max(memo[i], maxSoFar);
        }
        return maxSoFar;
}
```

Comments:
You can actually optimize this solution further. As an exercise, try solving this problem without creating an array.