COP 3503 Exam #1 Review Solutions

Java API

1) The country of PaperTrailLandia runs its national presidential election via a very simple paper voting system. Each citizen submits a single piece of paper with the name of a single person, for whom they wish to vote. For the purposes of this problem, we assume that each name is a string of uppercase letters and that each distinct person has a distinct name. Complete the method below which takes all pieces of paper as a String array, and returns a HashMap<String,Integer> which maps each person receiving a vote to the number of votes they received.

```
public static HashMap<String,Integer> getMap(String[] names) {

    HashMap<String,Integer> map = new HashMap<String,Integer>();

    for (int i=0; i< names.length; i++) {

        if ( map.containsKey(names[i]) )

            map.put(names[i], map.get(names[i]) + 1);

        else

            map.put(names[i], 1);

    }

    return map;
}
```

Backtracking

2) Consider arranging 16 values into a 4 x 4 square such that no two adjacent values (directly up, down, left or right) differ by more than the absolute value of a given maximum. If the input values are spaced out enough and the maximum is small enough, we can utilize backtracking to speed up a solution to count the number of different ways to arrange the values in the square. (In particular, we skip trying a value in a square if its difference with the neighbor above or to the left the square is greater than the given maximum.) Below is an incomplete implementation of 2 methods of a program that solves this problem. Complete the 2 methods.

```
final public static int N = 4;
final public static int[] DX = {-1,0};
final public static int[] DY = {0,-1};
public static int go(int[][] grid, boolean[] used, int[] list, int k, int
max) {

      if (k == N*N) return 1;

      int res = 0;
      for (int i=0; i<N*N; i++) {

            if (used[i]) continue;

            if ( conflict(grid, k, list[i], max ) continue;

            grid[ k/N ][ k%N ] = list[i];

            used[i] = true;

            res += go(grid, used, list, k+1, max) ;

            used[i] = false;
      }
      return res;
}

public static boolean conflict(int[][] grid, int pos, int value, int max) {

      int cX = pos/N;
      int cY = pos%N;

      for (int i=0; i<DX.length; i++) {

            if (!inbounds( cX + DX[i], cY + DY[i] )) continue;

            if (Math.abs( grid[cX + DX[i]][ cY + DY[i]] - value )> max)
                  return true;
      }
      return false;
}

public static boolean inbounds(int x, int y) {
      return x >= 0 && x < N && y >= 0 && y < N;
}
```

3) Consider the following program description. Complete the code on the following page so it solves the problem.

A k-gap sequence of integers $a_1, a_2, \ldots, a_n$ is such that for every i, $1 \leq i \leq n-1$, $|a_i - a_{i+1}| \geq k$. For example, the sequence 2, 8, 3, 6, 1, 7 is a 3-gap sequence but not a 4-gap sequence, since the difference the consecutive terms, 3 and 6 is 3.

Write a program that reads in a sequence of integers and an integer k and outputs the permutation of the input sequence that is first lexicographically that is a valid k-gap sequence.

Input

The first line of the input file will contain two positive integers, *n (1 ≤ n ≤ 12)*, and *k (1 ≤ k ≤ 10^6)*, where n represents the length of the input sequence for which we are looking for the ordering that is a valid k-gap sequence that is first, lexicographically. The next line will contain the *n* integers.

Output

Output the sequence specified by the problem description with each integer separated by a comma and a space.

Sample Input

```
5 10
13 19 3 2 17
```

Sample Output

```
13, 3, 19, 2, 17
```

```
import java.util.*;

public class gap {

    public static int diff;
    public static int[] values;

    public static void main(String[] args) {
        Scanner stdin = new Scanner(System.in);
        int n = stdin.nextInt();
        diff = stdin.nextInt();
        values = new int[n];
        for (int i=0; i<n; i++)
            values[i] = stdin.nextInt();
        Arrays.sort(values);
        printSeq(new int[n], 0, new boolean[n]);
    }

    public static void print(int[] arr) {
        for (int i=0; i<arr.length-1; i++)
            System.out.print(arr[i]+", ");
        System.out.println(arr[arr.length-1]);
    }
```

```
    public static boolean printSeq(int[] cur, int k, boolean[] used) {

        if (k == cur.length) {
            print(cur);
            return true;
        }

        boolean ans = false;
        for (int i=0; i<used.length; i++) {

          if (k == 0 || (!used[i] &&
              Math.abs(cur[k-1]-values[i]) >= diff)) {

              used[i] = true;
              cur[k] = values[i];
              ans = printSeq(cur, k+1, used);
              if (ans) break;
              used[i] = false;
           }

        }

        return ans;
    }
}
```

4) On the Sudoku assignment, the following test case took too much time on a majority of the submissions:

```
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 9 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 9
```

Why would this case in particular take a long time to run, using a standard backtracking solution? What is a relatively simple "fix" to the code that would allow for this test case to be processed quickly?

In a standard backtracking solution, we would fill in a value in row 1 col 1, then row 1 col 2, etc. until there were no possible values to fill in. The problem here is that no values are given in the first 60 squares that the backtracking algorithm searches for. Thus, many possible values satisfy the constraints of these squares before the algorithm ever notices that the two 9s in the bottom right corner are prohibiting any solution.

The quick fix is simply to check to see if the input board is consistent before starting the backtracking. If it isn't, then you can answer no solutions with confidence.

Disjoint Sets

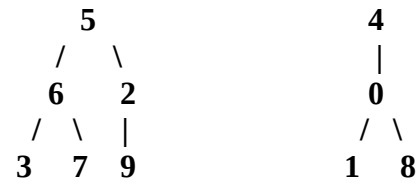
5) Show the state of the array storing a disjoint set AFTER it had undergone the following operations.(Note: Assume that the items in the disjoint set of size n are 0 through n-1.) Assume that the shorter tree is always attached to the longer tree and that if two trees of equal height are put together that the tree with the higher root value is attached to the tree with the lower root value.

```
DisjointSet dj = new DisjointSet(10);
dj.union(3, 8);
dj.union(2, 4);
dj.union(8, 7);
dj.union(4, 7);
dj.union(9, 1);
dj.union(9, 6);
dj.union(3, 0);
dj.union(5, 9);
```

Index	0	1	2	3	4	5	6	7	8	9
Value	**2**	**1**	**2**	**2**	**2**	**1**	**1**	**3**	**3**	**1**

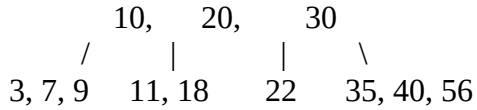
6) The array below stores a disjoint set. Draw the corresponding tree representation of that disjoint set.

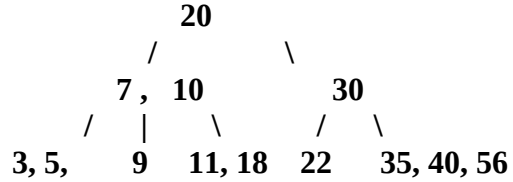
index	0	1	2	3	4	5	6	7	8	9
value	4	0	5	6	4	5	5	6	0	2



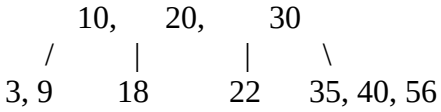
2-4 Trees

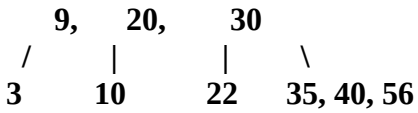
7) Show the result of inserting the value 5 into the 2-4 tree shown below:



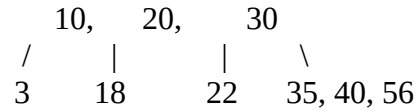


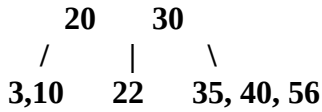
8) Show the result of deleting the value 18 from the 2-4 tree shown below:





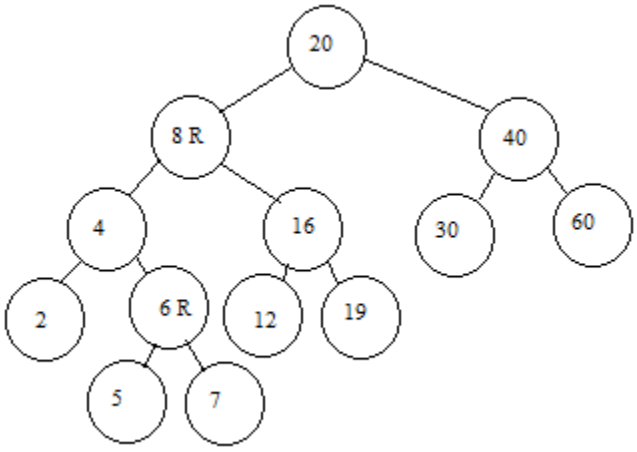
9) Show the result of deleting the value 18 from the 2-4 tree shown below:





Red-Black Trees

10) Draw the result of deleting the value 40 from the red-black tree shown below. In the drawing below, red nodes are indicated with a letter 'R' next to the number stored in the node. In your solution, please put an 'R' in each node that is red. (Note: use the regular binary tree rules for deleting a value which is stored in a node with 2 children.) If you explain your thinking, you may get partial credit for incorrect solutions.

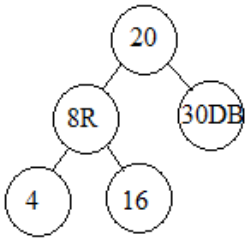


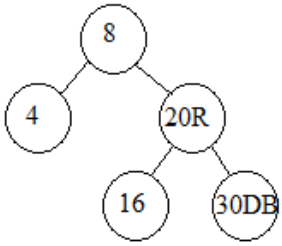
Note: When deleting 40, we can either delete the physical location of node 30 or 60. Due to the length of the solution, this solution only shows the result of deleting the physical location of the 30. A second correct answer exists (and was properly graded for) where the physical node storing 60 is deleted.

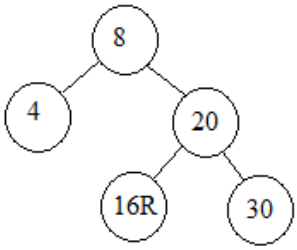
After we denote the physical node storing 30 to be deleted, we copy 30 into the node that originally stored 40. Here is the state of affairs from that point, just for the subtree rooted at 30:

Initially, the null node is a double black. **We push the double black up one level.**

Then, we deal with a double black that has a red sibling:

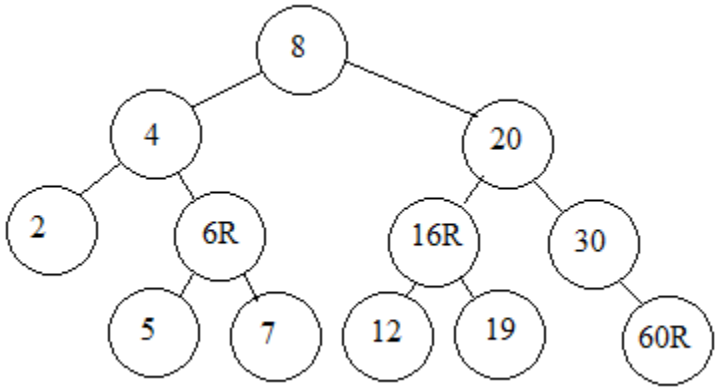






This is the original picture. **Here is when we right rotate making 8 the root and 20 red.** **Push the double black up one level, fixing the issue.**

Here is the final tree after those changes are made:



Note: if we had deleted the physical node storing 60 instead of the one storing 30, the only difference in the final answer is that 60 would be a black child of 20 and 30 would be a red left child of 60.