COP 3503: Backtracking Worksheet

1) Consider printing out all strings of x A's and y B's, where x ≥ y-1 such that no two consecutive letters are Bs, in alphabetical order. For example, if x = 5 and y = 3, one of the valid strings printed would be AABABABA. One way to solve this problem would be to use backtracking, where a string is built up, letter by letter (first trying A, then trying B in the current slot). **Complete the code below to implement this backtracking solution idea.** The correct condition for when you can place As is already in the code. (Hint: You can only place Bs if there are Bs left to place. If there are Bs left, then you must ensure that if there is a previous letter, it is not a B.)

```
import java.util.*;

public class AB {

    // Prints all strings with exactly x A's and y B's in alphabetical
    // order.
    public static void printWrapper(int x, int y) {
        printAll(new char[x+y], 0, x, y);
    }

    public static void printAll(char[] buffer, int k, int x, int y) {
        // FILL IN CODE HERE
    }

    public static void main(String[] args) {
        printWrapper(5, 3);
    }
}
```

2) A "unique" positive integer of n digits is such that no two adjacent digits differ by less than 2. Specifically, given an n digit number, $d_0d_1…d_{n-1}$, where d_0 is the most significant digit, (and thus, this one digit can't be 0), $|d_i – d_{i+1}| \geq 2$ for all i ($0 \leq i \leq n-2$). Consider the problem of printing out all "unique" positive integers of n digits via backtracking, in numerical order. Fill in the code below to complete the task. (To run the code, one would have to call printWrapper with their desired parameter.)

```
import java.util.*;

public class Separated {

    public static void printWrapper(int n) {
        printAll(new int[n], 0);
    }

    public static void printRec(int[] number, int k) {
        // FILL IN CODE HERE
    }

    public static void main(String[] args) {
        printWrapper(5);
    }
}
```

3) Consider an arbitrary permutation of the integers 0, 1, 2, …, *n*-1. We define the "jumps" in a permutation array *perm* to be the set of values of the form *perm[i]* - *perm[i-1]*, with 1 ≤ *i* ≤ *n-1*. For this problem you will write a backtracking solution count the number of permutations that can be created given a limited set of jumps. The function will take in arrays *perm*, representing the current permutation array, *used*, storing which items were used in the current permutation, *k*, the number of fixed items in the current permutation, *jumps*, an array storing the valid jumps allowed, and *len*, representing the length of the *jumps* array. The length of the *perm* and *used* arrays will be the constant N. Note that the jumps array contains both positive and negative values. For example, the permutation 3, 0, 2, 1 has the following jumps: -3, 2 and -1. **Complete the framework that has been given below to solve the problem. (Write your own main to test.)**

```
import java.util.*;

public class JumpPerm {

    public static int N = 10;

    public static int numperms(int[] perm, boolean[] used, int k,
                               int[] jumps) {

        int i, j, res = 0;

        if (k == N) return ___;
        for (i=0; i<N; i++) {

            if (used[i]) _____________;

            boolean flag = false;
            if (k == 0)
                flag = ___;
            else {
                for (j=0; j < ____; j++)

                    if ( ____________ == jumps[j])

                        flag = ___;
            }

            if (flag) {
                used[i] = ___;
                perm[k] = ___;
                res += numperms(perm, used, _____, jumps);
                used[i] = ___;
            }
        }

        return res;
    }
}
```

4) Consider the problem of placing 8 kings on an 8 x 8 chessboard, so that no two of the kings can attack each other ***AND no two kings are on the same row or column***. (Recall that a King can move one space in each of the eight possible directions of movement: up, down, left, right or any of the four diagonals.) Write your own code from scratch to print all of the solutions to the 8 Kings problem. Output your solutions nicely, like this:

```
_ _ _ _ _ _ _ K
_ _ _ K _ _ _ _
K _ _ _ _ _ _ _
_ _ _ _ K _ _ _
_ K _ _ _ _ _ _
_ _ _ _ _ K _ _
_ _ K _ _ _ _ _
_ _ _ _ _ _ K _
```