Universitetet i Oslo
Institutt for Informatikk

I. Yu, D. Karabeg

INF2220: algorithms and data structures
Series 8

Topic Permutations and Pruning

Issued: 12. 10. 2016

Classroom

Exercise 1 A *derangement* is a permutation p of {`1,...,n`} such that no item is in its proper position, i.e. $\texttt{p}_i \neq \texttt{i}$ for all $\texttt{1} \leq \texttt{i} \leq \texttt{n}$. Write an efficient backtracking program with pruning that constructs all the derangements of n items.

Exercise 2 Use a random number generator that generates numbers from {`0,1,2,3,4`} with equal probability to write a random number generator that generates numbers from {`0,1,2,3,4,5,6,7`} with equal probability.

Exercise 3 There are four people (A, B, C, D) have to cross a bridge. However, the bridge is fragile and can hold at most two of them at the same time. Moreover, to cross the bridge a torch is needed to avoid traps and broken parts. The problem is that these people have only one torch that lasts for only 60 minutes. Each one of them needs a different time to cross the bridge (in either direction):

A	5 minutes
B	10 minutes
C	20 mintues
D	25 minutes

The problem now is: In which order can the four people cross the bridge in time (that is, in 60 minutes)?

Exercise 4 Solve the first exercise of the handout of Krogdahl & Maus [**?**] (linked from the course website) (the exercise in section 16: placement of 9 numbers):[1]

Find permutations $x_1, x_2, \ldots, x_9$ of the nine decimal numerals $1, 2, \ldots, 9$ which satisfies the following condition: the decimal number x_1x_2 is divisible by 2, the number $x_1x_2x_3$ is divisible by 3, ..., analogously up to $x_1, \ldots, x_9$. Find all such permutations.

Take as starting point the source code below. This contains a generic, recursive algorithm generating all permutations of a text string.

Consider the permutations of the string

[1] The additional exercice mentioned in the texts needs not be solved.

`123456789`

We are interested only in the permutations matching the requirement. In other words: you should use *pruning* ("avkjaering" in the lecture = pruning) such that the algorithms discontinues to look at the strings not satisfying the requirement. For instance: having generated the string `17`, stop generating extensions of that string, as this number is not divisible by 2.

Exercise 5 Solve the same exercise, but this time use the framework of Krogdahl & Maus to generate the permutation (cf. `AvskProg` at page 8).

Exercise 6 In the well-known *eight queens problem*, the challenge is to place eight queens on an `8 x 8` chessboard so that no queen can take another one, i.e., no *two* queens share the same *row*, *column*, or *diagonal*. The eight queens problem is an example of the more general *N-queens problem* of placing `n` queens on an `N x N` chessboard. Write an implementation to solve the *N-queens problem*, where $N \in \mathbb{N}$ and $N > 3$.

Listing 1: Code skeleton

```
public class Permutation{

  // use the entire string as the endingString
  // when called with one parameter
  public void permuteString(String s){
      permuteString("", s);
  }

  public void assignment1(){
      long t_0 = System.currentTimeMillis();

      permuteString("123456789");

      System.out.println("time used: "+(System.currentTimeMillis() - t_0)+" ms");
  }

  // recursive declaration of method permuteString
  public void permuteString(
          String beginningString, String endingString ){

      // base case: if string to permute is length less than or equal to
      // 1, just display this string concatenated with beginningString
      if ( endingString.length() <= 1 ){
          if(isDevidable(beginningString + endingString)){
              System.out.println( beginningString + endingString );
          }
      }else{ // recursion step: permute endingString

          // for each character in endingString
          for ( int i = 0; i < endingString.length(); i++ ){
              try{

                  // create new string to permute by eliminating the
                  // character at index i
                  String newEndString = endingString.substring( 0, i ) + endingString.substring( i

                  String newBeginString = beginningString + endingString.charAt( i );

                  // recursive call with a new string to permute
                  // and a beginning string to concatenate, which
                  // includes the character at index i

                  if( isDevidable( newBeginString)){
                      permuteString( newBeginString, newEndString );
                  }
```

```
                }catch ( StringIndexOutOfBoundsException exception ){
                    exception.printStackTrace();
                }
            }
        }
    }

    public boolean isDevidable(String s){

        // ignore test if string is not a number
        // also ignore for '1' length digits since
        // all numbers % 1 == 0

        if(! s.matches("^\\d+$") || s.length() == 1) return true;

        int sAsInt = Integer.parseInt(s);
        return (sAsInt % s.length() == 0);
    }


    public static void main(String[] args){
        Permutation p = new Permutation();
        for(String s: args){
            p.permuteString(s);
            System.out.println("----");
        }

        p.assignment1();
    }
}
```