ESC101 : Fundamental of computing

Exercises on recursion **6 November, 2008**

Note : I am providing the solution of the problems below. But these solutions will be of little use if you just read them without actualy attempting the problem sincerely first. there may be multiple solutions for a problem. In case you face difficulty, discuss it during the extra class this week end.

1. Generate all strings of length n consisting of 0s and 1s without any two consecutive 1s.
   Solution : Easy.

```
class pattern_without_11
{
        public static void pattern(int n, char ch, String S)
        {
             if(n==0) System.out.println(S);
             else
             {
                  pattern(n-1,'0', S+'0');
                  if(ch=='0') pattern(n-1,'1',S+'1');
             }
       }
       public static void main(String args[])
       {
             int n = Integer.parseInt(args[0]);
             pattern(n,'0',"");
       }
}
```

2. There is a $n \times n$ grid. You start from bottom left corner and you have to go to top right corner. At each stage you may move either one step up or one step right. Design a program to enumerate all paths to reach top right corner. You may use string "up" and "right" to express the move done in one step.

   Solution : Each path has exactly n "up" and n "right" steps. Therefore, this problem is similar to the problem of computing all strings with n bars and m stars

3. **Combination with repetition allowed** Generate all combinations of length L using characters from set A, with repetition allowed. For example, for A=$\{a, b, c\}$ and $L = 2$, the corresponding combinations are

   aa
   ab
   bc
   bb
   ac
   cc

   Solution :

```
class combination_with_repeat
{
    //This method prints all combinations whose String representation
    //(WHERE ORDER AMONG CHARACTERS DOES NOT MATTER) consists of S
    //concatenated with L characters from {A[i],A[i+1],...}
    //with any character possibly appearing multiple times
    public static void CombS(char[] A, int i, int L, String S)
    {
        if(L==0) System.out.println(S);
        else
        {
            CombS(A,i,L-1,S+A[i]);
            if(i<A.length-1)
                CombS(A,i+1,L,S);
        }
    }
```

```
    //This method is used jst for better readability and understandability
    //Otherwise we could have directly called CombS(A,0,L,"") from main;
    public static void Combination_with_repeat(char[] A, int i, int L)
    {
        CombS(A,i,L,"");
    }

    public static void main(String args[])
    {
        char[] A = new char[args[0].length()];
        for(int i = 0; i<A.length; i = i+1)
            A[i] = args[0].charAt(i);
        int L = Integer.parseInt(args[1]);
        System.out.println("The set of combinations_with_repeat are :");
        Combination_with_repeat(A,0,L);
    }
}
```

4. **Partitions of a positive integer**

   (a) Given a positive integer n, print all permutations of positive integers such that
   1. their sum is n
   2. the value of the numbers in the permutation is non-decreasing

   Example :

```
The non-decreasing permutations of integers summing up to 6 are :
[-6-]
[-1-5-]
[-1-1-4-]
[-1-1-1-3-]
[-1-1-1-1-2-]
[-1-1-1-1-1-1-]
[-1-1-2-2-]
[-1-2-3-]
[-2-4-]
[-2-2-2-]
[-3-3-]
```

   Solution:

```
class partition_a
{    public static void Partition_a_S(int n, int i, String S)
     {
          if(n==0) System.out.println("[-"+S+"]");
          else
          {
               Partition_a_S(0,n,S+n+"-");
               for(int j = i; n-j>=j; j=j+1)
               Partition_a_S(n-j,j,S+j+"-");
          }
     }


     public static void main(String args[])
     {
          int n = Integer.parseInt(args[0]);
          System.out.print("The non-decreasing permutations of ");
          System.out.println("integers summing up to "+n+" are : ");
          Partition_a_S(n,1,"");
     }
}
```

   (b) Solve the above problem but with the change that the value of the numbers in the permutations are strictly increasing.
   Example:

```
Example : n = 6
[-6-]
[-1-5-]
[-1-2-3-]
[-2-4-]
```

   Solution:

```
class partition_b
{    public static void Partition_b_S(int n, int i, String S)
     {
          if(n==0) System.out.println("[-"+S+"]");
          else
          {
               Partition_b_S(0,n,S+n+"-");
               for(int j = i+1; n-j>j; j=j+1)
               Partition_b_S(n-j,j,S+j+"-");
          }
     }


     public static void main(String args[])
     {
          int n = Integer.parseInt(args[0]);
          System.out.print("The increasing permutations of ");
          System.out.println("integers summing up to "+n+" are : ");
          Partition_b_S(n,0,"");
     }
}
```

(c) For a positive integer n, generate all permutations of 1s and 2s which sum upto n. **Solution : easy, do it yourself**

(d) Given a positive integer n, print all those sets of positive integers whose sum is n. **Solution : same as part (b) since each integer in the partition must be distinct.**

5. We know that an expression involving parenthesis is valid if for each right parenthesis there is a unique left parenthesis to match. Generate all valid expressions consisting of n left parenthesis and n right parenthesis. For example, for $n = 3$, there are five valid expressions :

```
{}{}{}
{{}}{}
{}{{}}
{{{}}}
{{}{}}
```

Hint : Observe that you need a permutation of n {'s and n }'s such that for each prefix, the number of left parenthesis should be at least equal to the number of right parenthesis. Make use of this observation and suitably modify the problem on the patterns with n bars and n stars to achieve the solution.
Solution :

```
class balanced_parenthesis
{
    public static void balanced(int n, int m, String S)
    {
        if(m==0) System.out.println(S);
        else
        {
                // we can always add ``{`` if it is possible
                if(n>0) balanced(n-1,m,S+"{");

                // we can add ``}'' only if there are more ``{'' in prefix S than ``}''
                if(n<m) balanced(n,m-1,S+"}");
        }
    }

    public static void main(String args[])
    {
        int n = Integer.parseInt(args[0]);
        balanced(n,n,"");
    }
}
```

6. **Permutations of a string with repeated characters** Given a string of length n with possibly repeated characters, print all permutations of length L. For example, if string is abac, and $L = 2$ then the set to be enumerated has following strings :

```
aa
```

```
ab
ac
ba
bc
ca
cb
```

If you have understood the problem of generating permutation of string which does not have repetitions, you should have no problem extending it to this problem. In case, you still are unable to do so, we can discuss it in extra class this weekend. But first make sincere effort from your side.