ESc101 : Fundamental of Computing

I Semester 2008-09

Lecture 31+32

- Solving problems recursively
- Examples

Implementation of Recursion involves "method calling itself". So it is essential that you FULLY understand the invocation of methods from Lecture 30 to really understand recursion and its examples

Computing power x^n with fewer multiplications

Domain : x is a positive real number, and n is a non-negative integer.

$$x^n = \begin{cases} 1 & \text{if } n = 0. \\ x^{\lfloor \frac{n}{2} \rfloor} \times x^{\lfloor \frac{n}{2} \rfloor} & \text{if } n \neq 0 \text{ and } n \text{ is even.} \\ x^{\lfloor \frac{n}{2} \rfloor} \times x^{\lfloor \frac{n}{2} \rfloor} \times x & \text{if } n \neq 0 \text{ and } n \text{ is odd.} \end{cases}$$

How to use this formulation for minimizing multiplications ?

Computing power x^n with fewer multiplications

```
public static long power(double x, int n)
{ if(n==0)
      return 1;
  else
  {
      double temp = power(x,n/2);
      if(n%2==0)
         return temp*temp;
      else
         return temp*temp*x;
  }
}
```

Computing all patterns (strings) formed by n |'s and m *'s

Domain : n, m are non-negative integers.

Let **Pattern**(n, m) be the set of all strings formed by n |'s and m *'s.

How to express **Pattern**(n, m) recursively ?

we generalize the definition of Pattern()

Let **PatternS**(n, m, S) be the set of all strings of the form S+P where P is the string containing n $|$'s and m *'s.

It is easy to observe that

$$\textbf{Pattern}(n, m) = \textbf{PatternS}(n, m, \text{""});$$

How to express **PatternS**(n, m, S) recursively ?

Recursive formulation of PatternS(n, m, S)

$$\textbf{PatternS}(n,m,S) = \begin{cases} S & \text{if } n = m = 0 \\ \textbf{PatternS}(n-1, 0, S +' \,|') & \text{if } n \neq 0, m = 0 \\ \textbf{PatternS}(0, m-1, S +' *') & \text{if } n = 0, m \neq 0 \\ \textbf{PatternS}(n-1, m, S +' \,|') \bigcup & \\ \textbf{PatternS}(n, m-1, S +' *') & \text{if } n \neq 0, m \neq 0 \end{cases}$$

Recursive method for computing PatternS(n, m, S)

```
public static long PatternS(int n, int m, String S)
{ if(n==0 && m==0)
      System.out.println(S);
  else
  {
      if(n!=0) PatternS(n-1,m,S+'|');
      if(m!=0) PatternS(n,m-1,S+'*');
  }
}
```

All combinations of length L formed by characters from set A

Comb(A,L) : all combinations of length L formed from set A

Domain : A is a set and L is a non-negative integer and $|A| \geq L$.

Example : For A = $\{$a,b,c,d$\}$, L=2, the solution is :
Comb(A,L) :

ab
ac
ad
bc
bd
cd

Note : order does not matter, i.e., ab=ba

What is recursive formulation of Comb(A,L) ?

Let us first generalize Comb to CombS

$S \cap A = \emptyset$

Combs(A,L,S) : All strings formed by concatenating S with L characters selected from A **where order does not matter among characters**.

It can be seen that

$$\textbf{Comb}(A, L) = \textbf{CombS}(A, L, \text{""})$$

Recursive formulation of CombS(A,L,S) : Two trivial cases

1. What is **CombS**(A,L,S) if L=0 ?

   Answer = ...

2. What is **CombS**(A,L,S) if $L > 0$ but $|A| = L$?

   Answer = ...

Recursive formulation of CombS(A,L,S) : Two trivial cases

1. What is **CombS**(A,L,S) if L=0 ?

   Answer = S

2. What is **CombS**(A,L,S) if $L > 0$ but $|A| = L$,

   Answer = ...

Recursive formulation of CombS(A,L,S) : Two trivial cases

1. What is **CombS**(A,L,S) if L=0 ?

   Answer = S

2. What is **CombS**(A,L,S) if L$>$0 but $|$A$|$=L ?

   Answer = **CombS**(A$\setminus\{x\}$,L-1,S+x) where $x \in A$.

Recursive formulation of CombS(A,L,S) : when L$\neq$0 and $|$A$|$ $>$L

Consider any $x \in A$.

CombS(A,L,S) consists of two disjoint groups.

- Those combinations in which **x** is present.
- Those combinations in which **x** is **not** present.

Recursive formulation of CombS(A,L,S) : when L$\neq$0 and $|$A$|$ $>$L

Consider any $x \in A$.

CombS(A,L,S) consists of two disjoint groups.

- Those combinations in which **x** is present.

$$\textbf{Comb}(A\backslash\{x\}, L-1, S+x)$$

- Those combinations in which **x** is **not** present.

Recursive formulation of CombS(A,L,S) : when L$\neq$0 and $|$A$|$ $>$L

Consider any $x \in A$.

CombS(A,L,S) consists of two disjoint groups.

- Those combinations in which **x** is present.

$$\textbf{Comb}(A\backslash\{x\}, L-1, S+x)$$

- Those combinations in which **x** is **not** present.

$$\textbf{Comb}(A\backslash\{x\}, L, S)$$

Complete recursive formulation of CombS(A,L,S)

Let $x \in A$.

CombS(A,L,S) =

$$
= \begin{cases}
\text{S} & \text{if } L = 0 \\
\textbf{CombS}(A \backslash \{x\}, L-1, S +' x') & \text{if } L > 0 \text{ and } |A| = L. \\
\textbf{CombS}(A \backslash \{x\}, L-1, S +' x') \bigcup & \\
\textbf{CombS}(A \backslash \{x\}, L, S) & \text{if } L > 0 \text{ and } |A| > L.
\end{cases}
$$

Complete recursive formulation of CombS(A,L,S) with A as array

CombS(A, i, L, S) = All strings formed by concatenating S with L characters selected from $\{A[i], A[i+1], ...\}$ **where order does not matter among characters**.

Recall in the above definition, we assume that S does not have any character from $\{A[i], A[i+1], ...\}$.

Complete recursive formulation of CombS(A,L,S) with A as array

CombS(A, i, L, S) =

$$= \begin{cases} \text{S} & \text{if } L = 0 \\ \textbf{CombS}(A, i+1, L-1, S+A[i]) & \text{if } L > 0 \text{ and } A.length - i = L. \\ \textbf{CombS}(A, i+1, L-1, S+A[i]) \bigcup & \\ \textbf{CombS}(A, i+1, L, S) & \text{if } L > 0 \text{ and } A.length - i > L. \end{cases}$$

Recursive method for CombS(A,L,S)

```
public static void CombS(char[] A, int i, int L, String S)
{   int current_size_of_A= A.length-i;
    if(L==0) System.out.println(S);
    else
    {
        CombS(A,i+1,L-1,S+A[i]);
        if(current_size_of_A>L)
        CombS(A,i+1,L,S);
    }
}
```

Permutation of L characters chosen from set A

Domain : L is non-negative integer, A is a set of character with $|A| \geq L$.

Perm(A, L) : the set of all strings of length L whose characters belong to A.

Aim : To design a program to compute **Perm**(A, L)

Extension of Perm to PermS

PermS(A, L, S) : the set of all strings with S as prefix and followed by a permutation of L characters from A.

It can be seen that

$$\textbf{Perm}(A, L) = \textbf{PermS}(A, L, \text{``''})$$

Recursive formulation of PermS(A,L,S)

$$\mathbf{PermS}(A, L, S) = \left\{ \begin{array}{l} \\ \\ \end{array} \right.$$

Recursive formulation of PermS(A,L,S)

$$\mathbf{PermS}(A, L, S) = \begin{cases} S & \text{if } L = 0 \\ \\ \bigcup_{x \in A} \textbf{PermS}(A \backslash \{x\}, L - 1, S +' x') & \text{if } L > 0 \end{cases}$$

Recursive formulation of PermS when A is given as array

PermS(A, i, L, S) : the set of all strings with S as prefix and followed by a permutation of L characters from $\{A[i], A[i+1], ...\}$.

Recursive formulation of PermS when A is given as array

How to express subset of those strings from **PermS**(A, i, L, S) which begin with $A[i]$?

.... ?? ....

Recursive formulation of PermS when A is given as array

How to express subset of those strings from **PermS**(A, i, L, S) which begin with $A[i]$?

$$\textbf{PermS}(A, i+1, L-1, S+A[i])$$

Recursive formulation of PermS when A is given as array

How to express subset of those strings from **PermS**(A, i, L, S) which begin with $A[j], j > i$?

.... ?? ....

.... ?? ....

Please make sincere attempt to understand and answer the above question

Nice recursive exercises will be posted today