ESc101 : Fundamental of Computing

I Semester 2008-09

Lecture 33

- Clarifying doubts from previous lecture
- Proving correctness of a recursive method
- More examples of recursion
- Input/Output (if time permits)

Note I have tried to simplify these slides after the lecture. I hope you will go through these slides and attempt the exercises given in file recur_exercise.pdf. Extra class is on Sunday 10:00 AM in CS101.

Understanding recursion requires only two basic tools

- The control flow when a method calls another method (may be itself) :
  Lecture 30

- Mathematical Induction

How did you prove ?

- For all natural number n,

$$\sum_{0 \leq i \leq n} i^3 = \frac{n^2(n+1)^2}{4}$$

- ...
- more complicated assertions which you might have proved before coming to IIT.
- ...

How did you prove ?

- For all natural number n,

$$\sum_{0 \leq i \leq n} i^3 = \frac{n^2(n+1)^2}{4}$$

- ...
- more complicated problems which you might have proved before coming to IIT.
- ...

Principle of Mathematical Induction

Principle of Mathematical Induction

Let $\mathcal{P}(n)$ be a statement defined as function of integer n.

If the following assertions hold

1. $\mathcal{P}(n)$ is true for some $n = n_0$.
2. If $\mathcal{P}(i)$ is true for any $i \geq n_0$, then $\mathcal{P}(i+1)$ is also true.

Principle of Mathematical Induction

Let $\mathcal{P}(n)$ be a statement defined as function of integer n.

If the following assertions hold

1. $\mathcal{P}(n)$ is true for some $n = n_0$.
2. If $\mathcal{P}(i)$ is true for any $i \geq n_0$, then $\mathcal{P}(i+1)$ is also true.

We can conclude that $\mathcal{P}(n)$ **is true for all** $n > n_0$**.**

Principle of Mathematical Induction

Let $\mathcal{P}(n)$ be a statement defined as function of integer n.

If the following assertions hold

1. $\mathcal{P}(n)$ is true for some $n = n_0$.
2. If $\mathcal{P}(i)$ is true for any $i \geq n_0$, then $\mathcal{P}(i+1)$ is also true.

We can conclude that $\mathcal{P}(n)$ **is true** for all $n > n_0$.

Observe the similarity between induction and recursive formulation of a problem

Problem solved in last class

Enumerate the elements of the following sets.

1. **Pattern**(n, m): the set of all strings formed by n $|$'s and m *'s

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

3. **Permute**(A, L): all permutations of length L formed using characters from set A)

Partially solved in last class

Steps used in solving the problems

- understand the domain of the problem and the set to be enumerated
- To express the set recursively/inductively ?

Doubt 1 : Why do we have to extend

- **Pattern**(n, m) to **PatternS**(n, m, S) ?
- **Comb**(A, L) to **CombS**(A, L, S) ?
- **Permutation**(A, L) to **PermutationS**(A, L, S) ?
- **Partition**(n) to **PartitionS**(n, S) ?

Can we express Pattern(n, m) recursively ?

- **Pattern**(0,0) = $\{$“”$\}$
- for $n > 0, m > 0$, **Pattern**(n, m) : ??

Can we express Pattern(n, m) recursively ?

- **Pattern**(0,0) = $\{$“”$\}$
- for $n > 0, m > 0$, **Pattern**(n, m) is union of two sets :

Can we express Pattern(n, m) recursively ?

- **Pattern**(0,0) = $\{$“”$\}$
- for $n > 0, m > 0$, **Pattern**(n, m) is union of two sets :

  set of strings of the form : '$|$' followed by **Pattern**($n-1, m$).

  set of strings of the form : '*' followed by **Pattern**($n, m-1$).

Can we express Pattern(n, m) recursively ?

- **Pattern**(0,0) = $\{$“”$\}$
- for $n > 0, m > 0$, **Pattern**(n, m) is union of two sets :

  set of strings of the form : ’$|$’ followed by **Pattern**($n-1, m$).

  set of strings of the form : ’*’ followed by **Pattern**($n, m-1$).

Not exact recursive formulation !!

So 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

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

PatternS(n, m, S) can be expressed recursively quite easily

Recursive formulation of PatternS(n, m, S)

for $n > 0, m > 0$,

PatternS$(n, m, S) =$ **PatternS**$(n-1, m, S+'|') \bigcup$ **PatternS**$(n, m-1, S+'*')$

Conslusion :

we extend

- **Pattern**(n, m) to **PatternS**(n, m, S)
- **Comb**(A, L) to **CombS**(A, L, S)
- **Permutation**(A, L) to **PermutationS**(A, L, S)
- **Partition**(n) to **PartitionS**(n, S)

... ?? ...

Therefore,

we extend

- **Pattern**(n, m) to **PatternS**(n, m, S)
- **Comb**(A, L) to **CombS**(A, L, S)
- **Permutation**(A, L) to **PermutationS**(A, L, S)
- **Partition**(n) to **PartitionS**(n, S)

.. **to express the sets exactly in an inductive/recursive manner**

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+'*');
  }
}
```

What is the guarantee that this is method is correct ?

Proof of correctness is based on mathematical induction

Proof that the method PatternS(n,m,S) is correct

What is the inductive assertion ?

Proof that the method PatternS(n,m,S) is correct

The inductive assertion is :

$\mathcal{P}$**(k) :**

The method PatternS(n, m, S) for all nonnegative integers n, m with $n + m = k$ and any string S will print all strings of the form S+P where P is the string containing n $|$'s and m *'s.

Proving that $\mathcal{P}(k)$ is true for all $k \geq 0$

Base Case : It is easy to conclude that $\mathcal{P}(0)$ is true.

Induction step : We have to prove $\mathcal{P}(k)$ for $k > 0$ given that $\mathcal{P}(k-1)$ holds.

The Proof uses the principle of mathematical induction and uses

- the description of the method PatternS(n, m, S)
- the recursive formulation of the set **PatternS**(n,m,S).
  (We use bold letters to distinguish set from the method)

Note : The detailed proof has been provided in the file **inductive_proof.pdf** available on the website.

All combinations of length L formed by characters from set A

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) : 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.

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);
    }
}
```

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);
    }
}
```

What is proof of correctness ?

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);
    }
}
```

What is inductive Assertion : $\mathcal{P}(k)$?

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);
    }
}
```

$\mathcal{P}(k)$: (try to prove similar to the proof of PatternS(n,m,S))

For all arrays A, nonnegative integers i, L such that $A.length - i + L = k$, and any string S, the method CombS(A, i, L, S) prints All strings formed by concatenating S with L characters selected from $\{A[i], A[i+1], ...\}$ **where order does not matter among characters**.

Permutation of L characters chosen from set A

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

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

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

Extension of Permute to PermuteS

PermuteS(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

$$\textbf{Permute}(A, L) = \textbf{PermuteS}(A, L, \text{“”})$$

Recursive formulation of PermuteS(A,L,S)

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

Recursive formulation of PermuteS(A,L,S)

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

Recursive formulation of PermuteS when A is given as array

PermuteS(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 PermuteS when A is given as array

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

.... ?? ....

Recursive formulation of PermuteS when A is given as array

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

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

Recursive formulation of PermuteS when A is given as array

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

.... ?? ....

.... ?? ....

Please make sincere attempt to understand and answer the above question

Recursive formulation of PermuteS when A is given as array

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

PermuteS($A, i+1, L-1, S+A[i]$)

after we have swapped $A[i]$ and $A[j]$;

Recursive formulation of PermuteS when A is given as array

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

PermuteS($A, i+1, L-1, S+A[i]$)

after we have swapped $A[i]$ and $A[j]$;

Mathematically it is correct, but we have to be cautious during implementation because ...

Recursive formulation of PermuteS when A is given as array

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

PermuteS($A, i+1, L-1, S+A[i]$)
after we have swapped $A[i]$ and $A[j]$;

Mathematically it is correct, but we have to be cautious during implementation because ...

here we are changing the contents of array A.

The recursive method for enumerating PermuteS(A,i,L,S)

```
public static void PermuteS(char[] A, int i, int L, String S)
{   if(L==0) System.out.println(S);
    else
       for(int j = i; j<A.length; j = j+1)
       {
           swap(A,i,j);
           PermuteS(A,i+1,L-1,S+A[i]);

       }
}
```

The recursive method for enumerating PermuteS(A,i,L,S)

```
public static void PermuteS(char[] A, int i, int L, String S)
{   if(L==0) System.out.println(S);
    else
       for(int j = i; j<A.length; j = j+1)
       {
           swap(A,i,j);
           PermuteS(A,i+1,L-1,S+A[i]);

       }
}
```

Inducive Assertion : $\mathcal{P}(k)$:

- For any array A, nonnegative integers i, L with $A.length - i + L = k$ and any string S, the method PermuteS(A, i, L, S) prints all strings of the form S+P where P is a permutation of L characters chosen from $\{A[i], A[i+1], ...\}$.

The recursive method for enumerating PermuteS(A,i,L,S)

```
public static void PermuteS(char[] A, int i, int L, String S)
{   if(L==0) System.out.println(S);
    else
       for(int j = i; j<A.length; j = j+1)
       {
           swap(A,i,j);
           PermuteS(A,i+1,L-1,S+A[i]);

       }
}
```

Inducive Assertion : $\mathcal{P}(k)$:

- For any arrays A, nonnegative integers i, L with $A.length - i + L = k$ and any string S, the method PermuteS(A, i, L, S) prints all strings of the form S+P where P is a permutation of L characters chosen from $\{A[i], A[i+1], ...\}$.

  The assertion is not true for the above method PermuteS(A, i, L, S).

The recursive method for enumerating PermuteS(A,i,L,S)

```
public static void PermuteS(char[] A, int i, int L, String S)
{   if(L==0) System.out.println(S);
    else
       for(int j = i; j<A.length; j = j+1)
       {
           swap(A,i,j);
           PermuteS(A,i+1,L-1,S+A[i]);

       }
}
```

Inducive Assertion : $\mathcal{P}(k)$:

- For any array A, nonnegative integers i, L with $A.length - i + L = k$ and any string S, the method PermuteS(A, i, L, S) prints all strings of the form S+P where P is a permutation of L characters chosen from $\{A[i], A[i+1], ...\}$.

  Reason : since contents of array changes in recursive calls

The recursive method for enumerating PermuteS(A,i,L,S)

```
public static void PermuteS(char[] A, int i, int L, String S)
{   if(L==0) System.out.println(S);
    else
       for(int j = i; j<A.length; j = j+1)
       {
           swap(A,i,j);
           PermuteS(A,i+1,L-1,S+A[i]);

       }
}
```

Inducive Assertion : $\mathcal{P}(k)$:

- For any array A, nonnegative integers i, L with $A.length - i + L = k$ and any string S, the method PermuteS(A, i, L, S) prints all strings of the form S+P where P is a permutation of L characters chosen from $\{A[i], A[i+1], \ldots\}$.

  Try execution on : : A=[a,b,c],i=0 and L = 3. It does not print any string starting with b

The recursive method for enumerating PermuteS(A,i,L,S)

```
public static void PermuteS(char[] A, int i, int L, String S)
{   if(L==0) System.out.println(S);
    else
       for(int j = i; j<A.length; j = j+1)
       {
           swap(A,i,j);
           PermuteS(A,i+1,L-1,S+A[i]);

       }
}
```

Inducive Assertion : $\mathcal{P}(k)$:

- For any array A, nonnegative integers i, L with $A.length - i + L = k$ and any string S, the method PermuteS(A, i, L, S) prints all strings of the form S+P where P is a permutation of L characters chosen from $\{A[i], A[i+1], ...\}$.

  Idea : change the above method and augment the assertion accordingly.

The recursive method for enumerating PermuteS(A,i,L,S)

```
public static void PermuteS(char[] A, int i, int L, String S)
{   if(L==0) System.out.println(S);
    else
       for(int j = i; j<A.length; j = j+1)
       {
           swap(A,i,j);
           PermuteS(A,i+1,L-1,S+A[i]);

       }
}
```

Inducive Assertion : $\mathcal{P}(k)$:

- For any array A, nonnegative integers i, L with $A.length - i + L = k$ and any string S, the method PermuteS(A, i, L, S) prints all strings of the form S+P where P is a permutation of L characters chosen from $\{A[i], A[i+1], ...\}$.
- The state of Array A is same before and after **PermuteS**(A, i, L, S).

The recursive method for enumerating PermuteS(A,i,L,S)

```
public static void PermuteS(char[] A, int i, int L, String S)
{   if(L==0) System.out.println(S);
    else
       for(int j = i; j<A.length; j = j+1)
       {
           swap(A,i,j);
           PermuteS(A,i+1,L-1,S+A[i]);
           swap(A,i,j);
       }
}
```

Inducive Assertion : $\mathcal{P}(k)$:

- For any array A, nonnegative integers i, L with $A.length - i + L = k$ and any string S, the method PermuteS(A, i, L, S) prints all strings of the form S+P where P is a permutation of L characters chosen from $\{A[i], A[i+1], ...\}$.
- The state of Array A is same before and after **PermuteS**(A, i, L, S).

Nice recursive exercises

Available in file recur_exercise.pdf on the course webpage.