Algorithm Design and Analysis

วิชาบังคับก่อน: 204251 และ 206281
ผู้สอน: ตอน 1 ผศ. เบญจมาศ ปัญญางาม เรียน ห้อง 100
ตอน 2 ผศ. ดร.จักริน ชวชาติ เรียน ห้อง 209

บทที่ 4
การแก้ปัญหาความสัมพันธ์แบบรีเคอร์เรนซ์
(Solving recurrence relations)
Part I

Solving recurrence relations

Contents

Recursion Review : Why recursion?

- Recursion comes directly from Mathematics, where there are many examples of expressions written in terms of themselves.
- In general,
  - Recursive code is generally shorter and easier to write than an iterative code
  - Loops are also converted into recursive functions when they are compiled or interpreted
- Recursion is most useful for tasks that can be defined in terms of similar subtasks, for examples, sorting, search traversal,....

Recursion Review : Format of a Recursive Function

A recursive function consists of 2 main parts:

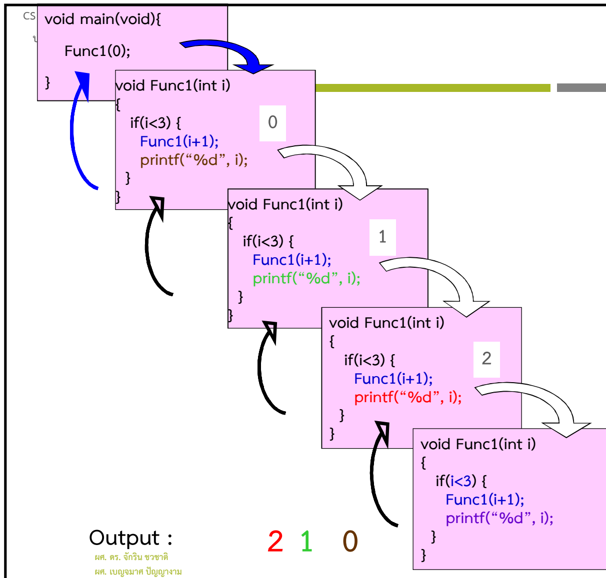
Base Case: The base case is where all further calls to the same function stop, meaning that it does not make any subsequent recursive calls.

Recursive Case: The recursive case is where the function calls itself repeatedly until it reaches the base case.

```
if(test for base case)
      return some base case value
else if(test for another base case)
      return some other base case value
else //recursive case
      return (some work and then recursive call)
```

Recursion Review : Recursion and memory

- The function solves a task by calling itself multiple times
  - Each time, a copy of the local variables and parameters for that function, as well as the return address, are pushed onto the stack memory.
  - When the function returns, the local variables, parameters and return addresses are popped from the stack frame.
- It's important : make sure that every function call eventually hits the base case in order to avoid infinite recursion.




```
#include<stdio.h>
void Func1(int i)
{
    if(i<3) {
        Func1(i+1);
        printf("%d", i);
    }
}
void main(void)
{
    Func1(0);
}
```


Recursion Review : Recursive Algorithm Example

- To design the Recursive function

1) Design recurrence relation for the problem solved

A recurrence is an equation or inequality that describes a function in terms of its value on smaller inputs.

2) Write Recursive Function for solving the problem by using the designed Recurrence Relation (1)

Recursion Review : Recursive Algorithm Example

- There are many problems whose solution can be defined recursively. Example: *n factorial*

Ex. 5! = 5 * 4 * 3 * 2 * 1
4! = 4 * 3 * 2 * 1
3! = 3 * 2 * 1
2! = 2 * 1
และ 1! = 1 , 0! = 1

so that ➔

5! = 5 * 4!
4! = 4 * 3!
3! = 3 * 2!
2! = 2 * 1!
1! = 1 , 0! = 1 ← base case

Use this base case as a condition that stop the recursion

1. Define Recurrence Relation for calculating n factorial (n!) by writing the FAC(n) function as the following equations

$$FAC(n) = \begin{cases} 1 & \text{if } n = 0 \text{ or } n = 1 \\ n * FAC(n-1) & \text{if } n > 1 \end{cases}$$

← base case

Recursion Review : Recursive Algorithm Example

Examples : n factorial (n!)

```
#include <stdio.h>

long FAC(int n)
{
   if (n <= 1)
       return 1;
   else
       return n*FAC(n-1);
}

void main()
{
   int num;

   printf("Enter number  (n) : ");  scanf("%d",&num);
   printf("n! = %ld", FAC(num));
}
```

Condition that stop the recursion

Calling itself

Example of function operations

FAC(4)
4 * FAC(3)
4 * 3 * FAC(2)
4 * 3 * 2*FAC(1)
4 * 3 * 2*1

Example of Output

Enter number (n) : 4
n!=24

Recursion Review : Recursive Algorithm Example

- Examples : Power(x,y) ➔ x^y

Ex. 2^4, x =2,y =4

2^4	=	2 * 2 * 2 * 2
2^3	=	2 * 2 * 2
2^2	=	2 * 2
2^1	=	2
2^0	=	1

So that ➔

2^4	=	$2 * 2^3$
2^3	=	$2 * 2^2$
2^2	=	$2 * 2^1$
2^1	=	$2 * 2^0$
2^0	=	1 ← base case

1. Define Recurrence Relation for calculating power(x,y) as following equations

Power(x,y) = { 1 if y = 0 ← base case; x * Power(x,y-1) if y > 0 }

Where x, y and results of power(x,y) are the integer numbers

Recursion Review : Recursive Algorithm Example

2. Write Recursive Function

from Recurrence Relation : Power(x,y) = { 1 if y = 0; x*Power(x,y-1) if y > 0 }

```
long power(int x, int y) {
    if (y == 0)  return(1);
    else  return(x*power(x, y-1));
}
void main() {
    int x, y;
    printf("Input x: ");scanf("%d", &x);
    printf("Input y: ");scanf("%d", &y);
    printf("Output : %ld", power(x, y));
}
```

condition that stop the recursion

Calling itself

Example of function operations

Input x: 3
Input y: 4
Output: 81

Recursion Review : Recursion vs. Iteration

Recursion	Iteration
Recursion ends when base case become true	Loop ends when control variables 's value satisfies the condition
Each function call consumes any extra memory space (stack)	No extra space In each iteration
Infinite recursion may cause stack overflow error (memory full)	infinite loops (repeat forever without stopping) uses CPU cycles (not create extra memory)
Many of the problems can be solved easily using recursion if you think recursively.	

- Any recursive algorithm can be expressed as an iterative algorithm, but you may need to keep an explicit stack.

```
int powerxy(int x,y) {
   long powxy = 1;

   while (y >= 1) {
     powxy *= X;
     --y;
   }
   return powxy;
}
```

```
long factorial(int n) {
   long fact = 1;

   while (n >= 1) {
       fact *= n;
       --n;
   }
   return fact;
}
```

Analyzing Recursive Algorithm : Time complexity

- An algorithm contains a recursive call
  - T(n), running time to solve problem of size n, is described by a recurrence

$$T(n)=\begin{cases} t_A & \text{Base case} \\ t_B + t_C & \text{Other} \end{cases}$$

1. t_A = the running time of the base condition (base case)
2. t_B = the running time of recursive call
3. t_C = the running time of operations that are done after/before the recursion calls (not t_A and t_B)

Analyzing Recursive Algorithm : Time complexity

Example1: recursive selection sort

Selection_Sort_Recursive(A)

1. if (n <=1) return; — O(1)
2. j= FindIndexMax(A[1..n]) — O(n)
3. swap(A,n,j); — O(1)
4. Selection_Sort_Recursive(A[1..n-1])

3) Not recursive call: t_C=O(n)

2) Recursive call : t_B =T(n-1)

1) t_A =T(1)=O(1)

$T(n)=t_B+ t_C$ = T(n-1)+O(n)

- An algorithm contains a recursive call
  - Running time : Described by a recurrence.

$$T(n)=\begin{cases} O(1) & \text{if } n=1 \\ \underbrace{T(n-1)}_{t_B} + \underbrace{O(n)}_{t_C} & \text{if } n>1 \end{cases}$$

Recurrence Relation of time complexity

Analyzing Recursive Algorithm : Time complexity

- Recurrence Relation of time complexity

$$T(n)=\begin{cases} O(1) & \text{if } n=1 \\ T(n-1) + O(n) & \text{if } n>1 \end{cases}$$

or

$$T(n)=\begin{cases} c & \text{if } n=1 \\ T(n-1) + cn & \text{if } n>1 \end{cases}$$

or

$$T(n)=\begin{cases} c & \text{if } n=1 \\ T(n-1) + n & \text{if } n>1 \end{cases}$$

or

$$T(n)=\begin{cases} 1 & \text{if } n=1 \\ T(n-1) + n & \text{if } n>1 \end{cases}$$

All in the same meaning

Analyzing Recursive Algorithm : Time complexity

Example2: recursive binary search

BinarySearchRecursive(A[left..right])

1. if (left > right) return(-1)
2. m=(left + right)/2
3. if x == A[m] return m;
4. If x < A[m]
5. return(BinarySearchRecursive (A[left..m-1]))
6. else
7. return(BinarySearchRecursive (A[m...right]))

1) t_A= T(1)=O(1)

3) ไม่ใช่ recursive call: t_C= O(1)

2) Recursive call : t_B=T(n/2)

$\therefore T(n) = t_B + t_C$= T(n/2)+O(1)

Recurrence Relation of time complexity ?

$$T(n)=\begin{cases} O(1) & \text{if } n=1 \\ T(n/2) + O(1) & \text{if } n>1 \end{cases}$$

Analyzing Recursive Algorithm : Practice

```
Merge_Sort(A,p,r)
If p < r then
    q= ⌊ (p+r)/2 ⌋
    Merge_Sort(A,p,q)
    Merge_Sort(A,q+1,r)
    Merge(A,p,q,r)
```

```
Merge(A,p,q,r)
i = p, j = q + 1, n = r - p + 1
for k = 1 to n
  if ((A[i]< A[j]) or (j > r)) and (i≤q)
        B[k] = A[i]
           i = i +1
     else
           B[k] = A[j]
           j = j +1
for k = 0 to n – 1
    A[p + k] = B[k]
```

Recurrence Relation of time complexity ?

Analyzing Recursive Algorithm : Practice

Power Problem ➔ Power(x,y) ➔ x^y

- Directly solving by repetition algorithm
- Soloving by recursive algorithm
  - Recurrence Relation of the problem

$$\text{Power}(x,y) = \begin{cases} 1 & \text{if } y = 0 \\ x*\text{Power}(x,y-1) & \text{if } y > 0 \end{cases}$$

```
long Power(int x,int y){
  if (y == 0)
     return(1)
  else
     return(x*power(x,y-1)
}
```

Recurrence Relation of time complexity?

Solving recurrence relations

3 different ways to solve recurrences

1. Iterative substitution method
   - Iteratively apply the recurrence equation to itself
   - and try to discover a pattern
2. Recursive tree
3. Master method

Iterative substitution method

- Start with the recurrence relation
- Repeated substitution until you see a pattern

Example: Merge sort analysis

$$T(n) = \begin{cases} c & \text{if } n = 1 \\ 2T(n/2)+cn & \text{if } n>1 \end{cases}$$

T(n) =2T(n/2)+cn   k=1

=2[2T(n/4)+cn/2]+cn

=4T(n/4)+cn+cn

=4T(n/4)+2cn   k=2, $4=2^2$

=4[2T(n/8)+cn/4]+2cn

=8T(n/8)+3cn   k=3, $8=2^3$

=2^kT(n/2^k)+kcn

set k=log n, so that 2^k=n

T(n) = nT(1)+ (log n) (cn)

= cn+c(n log n)= O(n log n)   using T(1)=c

Iterative substitution method

Example:
Recurrence equation $T(N) = 2T(\sqrt{N}) + 1$,T(2)=0

T(N) = $2T(N^{1/2})+1$
= $2(2T(N^{1/4}) +1)+1$
= $4T(N^{1/4}) + 2 + 1$
= $8T(N^{1/8})+ 4 + 2 + 1$
....
= $2^kT(N^{1/2^k})+ 2^{k-1}+ \dots +2^2+ 2^1+ 2^0$

$\log N^{1/2^k} = \log 2$
$1/2^k \log N = 1$
$\log N = 2^k$
$\log \log N = \log 2^k$
$\log \log N = k \log 2$
$\log \log N = k$

เมื่อ $N^{1/2^k} = 2$ ดังนั้น k = loglog N

T(N) = $2^{\log \log N} (T(2))+2^{\log \log N-1} + \dots +2^2+ 2^1 + 2^0$

= logN-1 (ใช้ T(2)=0 และ $\sum_{i=0}^{n} x^i = \frac{x^{n+1}-1}{x-1}$)

$(2^{\log \log N -1 + 1} - 1)/2-1 = 2^K-1$ และ $\log N = 2^k$

Solving recurrence relations : Math Review

- Math formulas for solving recurrence relations

Arithmetic series $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$

Geometric series $\sum_{i=0}^{n} x^i = \frac{x^{n+1}-1}{x-1}$ for real x > 1

Inverse harmonic series

$\sum_{k=0}^{\infty} x^k = \frac{1}{1-x}$ for $|x| < 1$

When x = 1/2 the series is

$1 + 1/2 + 1/4 + \dots = \frac{1}{1/2} = 2$

Iterative substitution method : Practice & Solution

1. To Solve Recurrence equation of T(n) for recursive binary search problem

$$T(n) = \begin{cases} O(1) & \text{if } n = 1 \\ T(n/2) + O(1) & \text{if } n > 1 \end{cases}$$

2. To solve recurrence equation of T(n) for recursive selection sort problem

$$T(n) = \begin{cases} O(1) & \text{if } n = 1 \\ T(n-1) + O(n) & \text{if } n > 1 \end{cases}$$

Solving recurrence relations : Practice

To solve each recurrence relation in practice sheet : Assignment#03 by using

1) Iterative substitution method OR Recursion-tree Method
2) Master Theorem