Principles of Recursive Program Design

28

Designing Recursive Algorithms

DESIGN & ALALYSIS OF ALGORITHM

- Find the key step.
  - How can this problem be divided into parts?
  - How will the key step in the middle be done?
  - Avoid ending with multitude of special cases.
- Find a stopping rule.
  - This stopping rule is usually the small case that is easy to handle without recursion.
- Outline your algorithm.
  - Combine the stopping rule and the key step, using an if statement to select between them.

ALG00S, javed@kent.edu

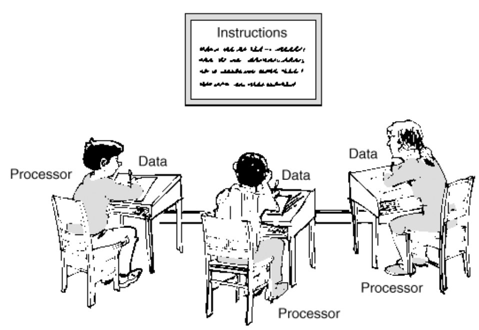
Designing Recursive Algorithms
(Continued..)

- Check termination.
  - Verify that the recursion will always terminate.
  - Be sure that your algorithm correctly handles extreme cases.
- Draw a recursion tree.
  - The height of the tree is closely related to the amount of memory that the program will require,
  - the total size of the tree relates to the number of times the key step will be done.

DESIGN &
ALALYSIS OF
ALGORITHM

Implementing Recursion

- **Implementation is separate from design.**
  - Implementation can be in any language.
- **Multiple Processors:**
  - Processes that take place simultaneously are called **concurrent**.
- **Single Processor:**
  - can use multiple storage areas with a single processor.
- **Re-entrant Programs**

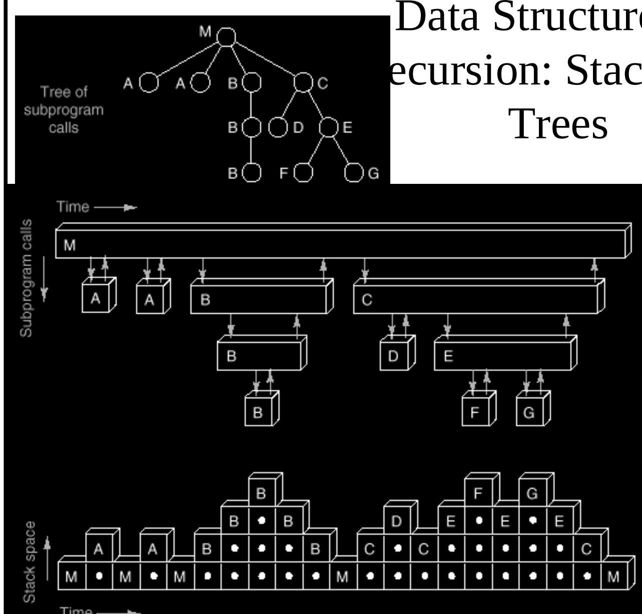


Improving Recursive Programs: The case of Tail Recursion

32

DESIGN & ALALYSIS OF ALGORITHM

Data Structure for Recursion: Stacks and Trees



Who creates/manages these stacks?

Tail Recursion

DESIGN & ALALYSIS OF ALGORITHM

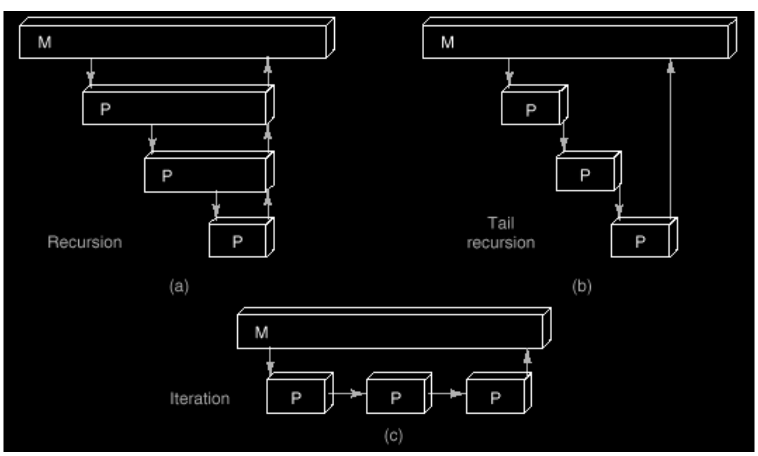
DEFINITION ***Tail recursion*** occurs when the last-executed statement of a function is a recursive call to itself.

If the last-executed statement of a function is a recursive call to the function itself, then this call can be eliminated by reassigning the calling parameters to the values specified in the recursive call, and then repeating the whole function.

ALG00S, javed@kent.edu
Javed I. Khan@1999

Removal of Tail Recursion

DESIGN & ALALYSIS OF ALGORITHM



ALG00S, javed@kent.edu
Javed I. Khan@1999

Can Tower of Hanoi be simplified?

```
/* Move: moves count disks from start to finish using temp
for temporary storage. */

void Move(int count, int start, int finish, int temp)
{
    if (count > 0) {
        Move(count-1, start, temp, finish);
        printf("Move a disk from %d to %d.\n", start,
finish);
        Move(count-1, temp, finish, start);
    }
}
```


Iterative Tower of Hanoi

```
void Move(int count, int start, int finish, int temp)
{
    if (count > 0) {
        Move(count-1, start, temp, finish);
        printf("Move a disk from %d to %d.\n", start, finish);
        Move(count-1, temp, finish, start);
    }
}
```

```
void Move(int count, int start, int finish, int temp)
{
    int swap;   /* temporary storage to swap towers */
    while (count > 0) {
        Move(count - 1, start, temp, finish);
        printf("Move %d from %d to %d.\n", count, start, finish);
        count--;
        swap = start;
        start = temp;
        temp = swap;
    }
}
```

Can n-queen be simplified?

DESIGN &
ALALYSIS OF
ALGORITHM

```
void AddQueen(void)
{
    int col;    /* column being tried for the queen */

    queencount++;
    for (col = 0; col < BOARDSIZE; col++)
        if (colfree[col] && upfree[queencount + col] &&
            downfree[queencount - col + DOWNOFFSET]) {
            /* Put a queen in position (queencount, col).   */
            queencol[queencount] = col;
            colfree[col] = FALSE;
            upfree[queencount + col] = FALSE;
            downfree[queencount - col + DOWNOFFSET] = FALSE;
            if (queencount == BOARDSIZE-1)  /* termination condition
*/
                WriteBoard();
            else
                AddQueen();                 /* Proceed recursively.
*/
            colfree[col] = TRUE;    /* Now backtrack by removing the
queen. */
            upfree[queencount + col] = TRUE;
            downfree[queencount - col + DOWNOFFSET] = TRUE;
        }
    queencount--;
}
```

Flashback: AddQueen()

Demo:
Time Comparison of two N-Queen Solutions

`Queennr.exe`

DESIGN & ALALYSIS OF ALGORITHM

LECT-03, S-40
ALG00S, javed@kent.edu
Javed I. Khan@1999

Should we Always use Recursion?
Case: Factorials

- Recursive Version:

```
/* Factorial: recursive version.*/
int Factorial(int n)
{
    if (n == 0)
        return 1;
    else
        return n * Factorial(n-1);
}
```

Which one will work harder?

- Iterative Version?

```
/* Function: iterative version.*/
int Factorial(int n)
{
    int count, product;
    for (product = 1, count = 2; count <= n;
count++)
        product *= count;
    return product;
}
```

DESIGN & ALALYSIS OF ALGORITHM

LECT-03, S-41
ALG00S, javed@kent.edu
Javed I. Khan@1999

Fibonacci Numbers

- Finonacci Number

$$F_0 = 0$$
$$F_1 = 1$$
$$F_n = F_{n-1} + F_{n-1} \quad when \quad n \geq 2$$

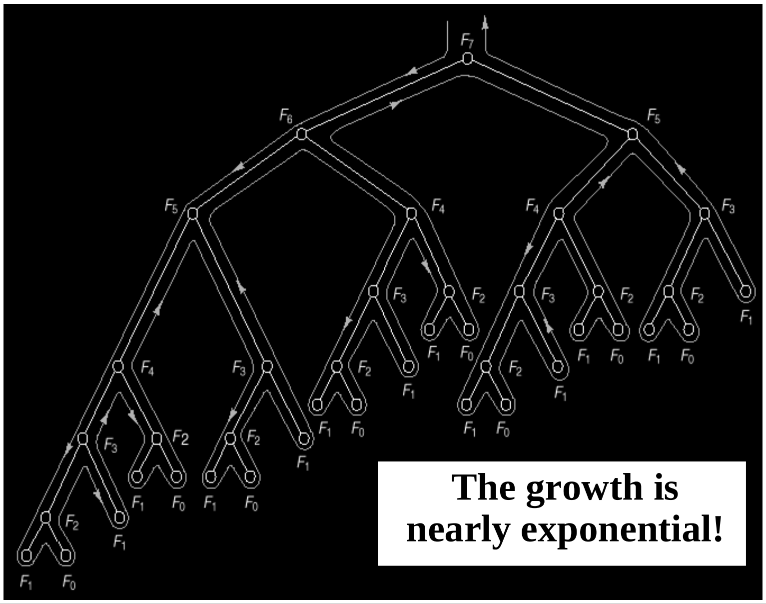
A Cute Solution

```
int Fibonacci(int n)
{
    if (n <= 0)
        return 0;
    else if (n == 1)
        return 1;
    else
        return Fibonacci(n-1) + Fibonacci(n-2);
}
```

Efficiency?

DESIGN & ALALYSIS OF ALGORITHM

Recursion Tree



LECT-03, S-44
ALG00S, javed@kent.edu
Javed I. Khan@1999

Iterative Fibonacci

```
 /* Fibonacci: iterative version.*/

int Fibonacci(int n)
{
    int i;
    int twoback;    /* second previous number, F_i-2    */
    int oneback;    /* previous number, F_i-1       */
    int current;    /* current number, F_i      */

    if (n <= 0)
        return 0;
    else if (n == 1)
        return 1;
    else {
        twoback = 0;
        oneback = 1;

        for (i = 2; i <= n; i++) {
            current = twoback + oneback;
            twoback = oneback;
            oneback = current;
        }
        return current;
    }
}
```

LECT-03, S-45
ALG00S, javed@kent.edu
Javed I. Khan@1999

Comparison: Iteration vs. Recursion

- Chain
- Duplicate Task
- Change Data Structures
- Recursion Removal

ALG00S, javed@kent.edu

Guidelines

DESIGN &
ALALYSIS OF
ALGORITHM

- If the recursion tree has a simple form, the iterative version may be better.
- If the recursion tree involves duplicate task, then data structures other than stacks will be appropriate.
- If the recursion tree appears quite bushy, with little duplicate tasks, then recursion is likely the natural solution.

ALG00S, javed@kent.edu
Javed I. Khan@1999