Data Structures and Algorithms (3)

Instructor: Ming Zhang

Textbook Authors: Ming Zhang, Tengjiao Wang and Haiyan Zhao

Higher Education Press, 2008.6 (the "Eleventh Five-Year" national planning textbook)

https://courses.edx.org/courses/PekingX/04830050x/2T2014/

Chapter 3 Stacks and Queues

- Stacks
- Applications of stacks
  - Implementation of Recursion using Stacks
- Queues

Transformation from recursion to non-recursion

- The principle of recursive function
- Transformation of recursion
- The non recursive function after optimization

(2) Transformation of recursion

Method of transform recursion to non-recursion

$$fu(n) = \begin{cases} n+1 & \text{when } n<2 \\ fu(\lfloor n/2 \rfloor) * fu(\lfloor n/4 \rfloor) & n \geq 2 \end{cases}$$

- Direct transformation method

1.Set a working stack to record the current working record
2. Set t+2 statement label
3. Increase non recursive entrance
4. Replace the i-th (i = 1, …, t)recursion rule
5. Add statement： “goto label t+1” at all the Recursive entrance
6. The format of the statement labeled t+1
7. Rewrite the recursion in circulation and nest
8. Optimization

rd=2: n=0 f=? u1=? u2=?
rd=1: n=3 f=? u1=2 u2=?
rd=3: n=7 f=? u1=? u2=?

(2) Transformation of recursion

1. Set a working stack to record the working record

- All the parameters and local variables that occur in the function must be replaced by the corresponding data members in the stack
    - Return statement label domain (t+2 value)
    - Parameter of the function(parameter value, reference type)
    - Local variable

```
typedef struct elem { // ADT of  stacks
    int rd;              // return the label of the statement
    Datatypeofp1 p1;           // parameter of the function
    …
    Datatypeofpm pm;
    Datatypeofq1 q1;           // local variable
    …
    Datatypeofqn qn;
} ELEM;
```

2. Set t+2 statement label

- label *0* ： The first executable statement
- label t+1 ：set at the end of the function body
- label i (1<=i<=t)： the ith return place of the recursion

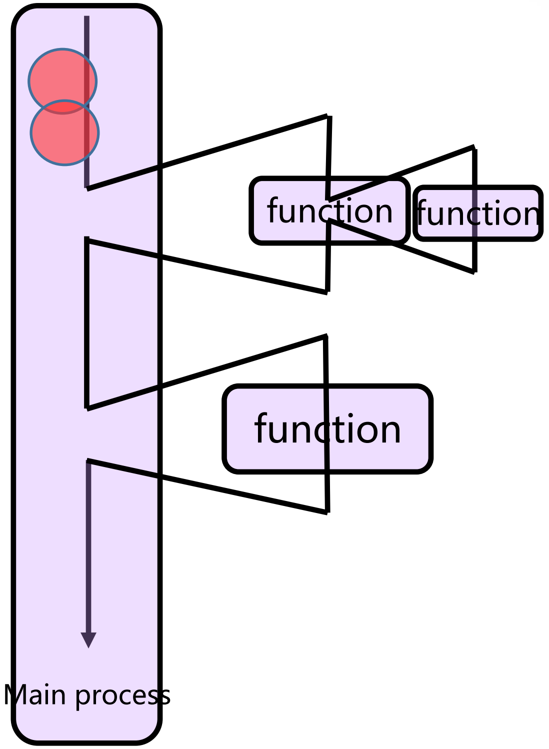
3. Increase non recursive entrance

// push

S.push(t+1 , p1, … , pm , q1 , …qn);

(2) Transformation of recursion

4. Replace the ith (i = 1, …, t)recursion rule



- Suppose the ith (i=1, …, t) recursive call statement is : recf(a1, a2, …,am) ;
- Then replace it with the following statement :

```
        S.push(i, a1, …, am) ;  // Push the actual  parameter
        goto label 0 ;
        …..
  label i :  x = S.top() ; S.pop();
```

/* pop , and assign some value of X to the working record of stack top S.top()— It is equivalent to send the value of reference type parameter back to the local variable*/

(2) Transformation of recursion

5. Add statement at all the Recursive entrance

- goto label *t+1*;

6. The format of the statement labeled t+1

```
switch ((x=S.top ()).rd) {
    case 0 :    goto label 0;
                break;
    case 1 : goto label 1;
                    break;
    ……….
    case t+1 : item = S.top(); S.pop(); // return
               break;
    default : break;
}
```

7. Rewrite the recursion in circulation and nest

- For recursion in circulation , you can rewrite it into circulation of goto type which is equivalent to it
- For nested recursion call

  For example , recf (… recf())

  Change it into :

  $exmp_1$ = recf ( );

  $exmp_2$ = recf ($exmp_1$);

  …

  $exmp_k$ = recf ($exmp_{k-1}$)

  Then solve it use the rule 4

8. Optimization

- Further optimization
  - Remove redundant push and pop operation
  - According to the flow chart to find the corresponding cyclic structure, thereby eliminating the goto statement

(2) Transformation of recursion

Definition of data structure

$$fu(n) = \begin{cases} n+1 & \text{when } n<2 \\ fu(\lfloor n/2 \rfloor) * fu(\lfloor n/4 \rfloor) & n \geq 2 \end{cases}$$

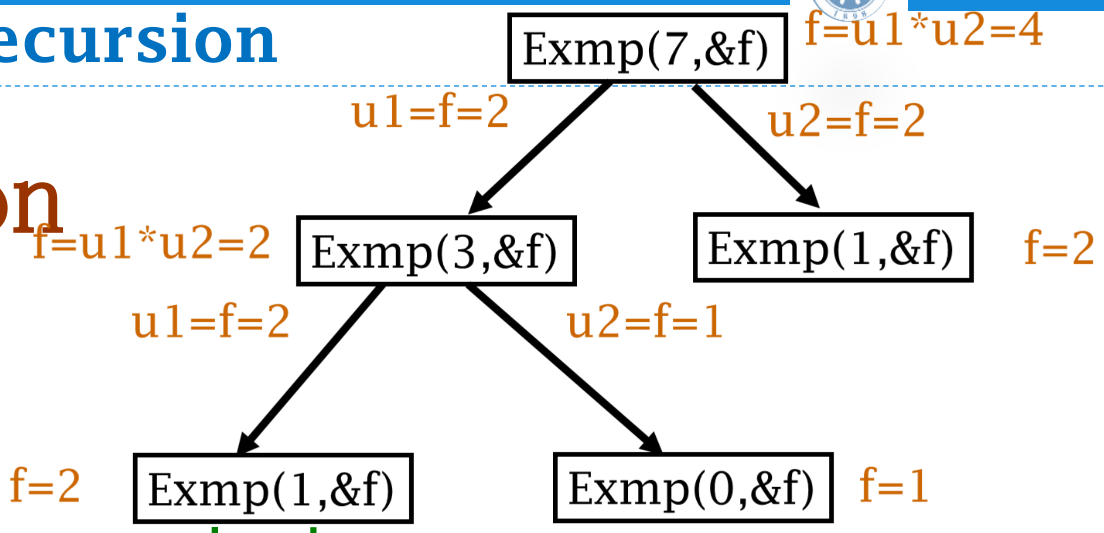
```
typedef struct elem {
    int rd, pn, pf, q1, q2;
} ELEM;
class nonrec {
    private:
        stack <ELEM> S;
    public:
        nonrec(void) { }      // constructor
        void replace1(int n, int& f);
};
```

rd=2: n=0 f=? u1=? u2=?

rd=1: n=3 f=? u1=2 u2=?

rd=3: n=7 f=? u1=? u2=?

(2) Transformation of recursion



Entrance of recursion

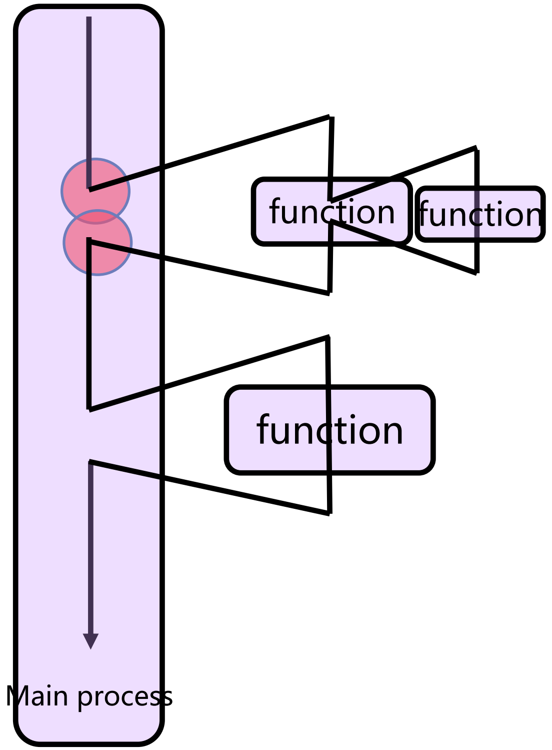
```
void nonrec::replace1(int n, int& f)  {
   ELEM x, tmp
   x.rd = 3;    x.pn = n;
   S.push(x);    // pushed into the stack bottom as lookout
label0: if ((x = S.top()).pn < 2)  {
         S.pop( );
         x.pf = x.pn + 1;
         S.push(x);
         goto label3;
      }
```

(2) Transformation of recursion

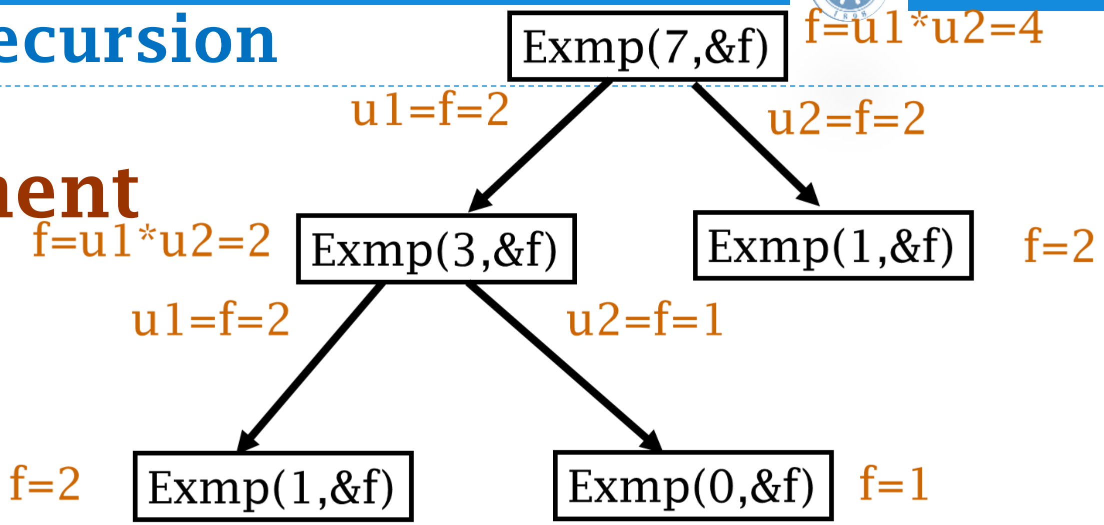
The first recursion statement

$$fu(n) = \begin{cases} n+1 & \text{when } n<2 \\ fu(\lfloor n/2 \rfloor) * fu(\lfloor n/4 \rfloor) & n \geq 2 \end{cases}$$

```
    x.rd = 1;     // the first recursion
    x.pn = (int)(x.pn/2);
     S.push(x);
    goto label0;
label1: tmp = S.top(); S.pop();
    x = S.top(); S.pop();
    x.q1 = tmp.pf; // modify u1=pf
    S.push(x);
```



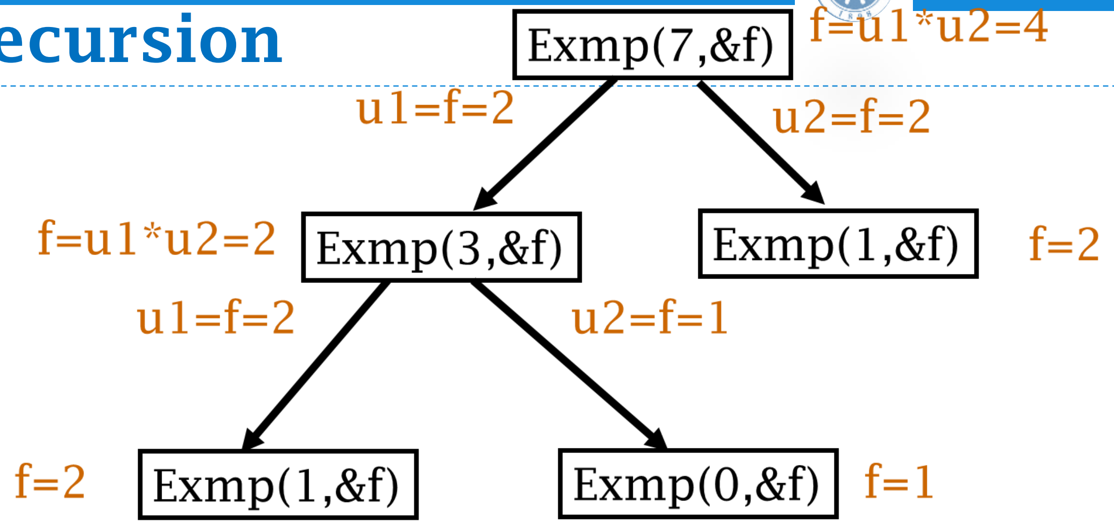
(2) Transformation of recursion



The second recursion statement

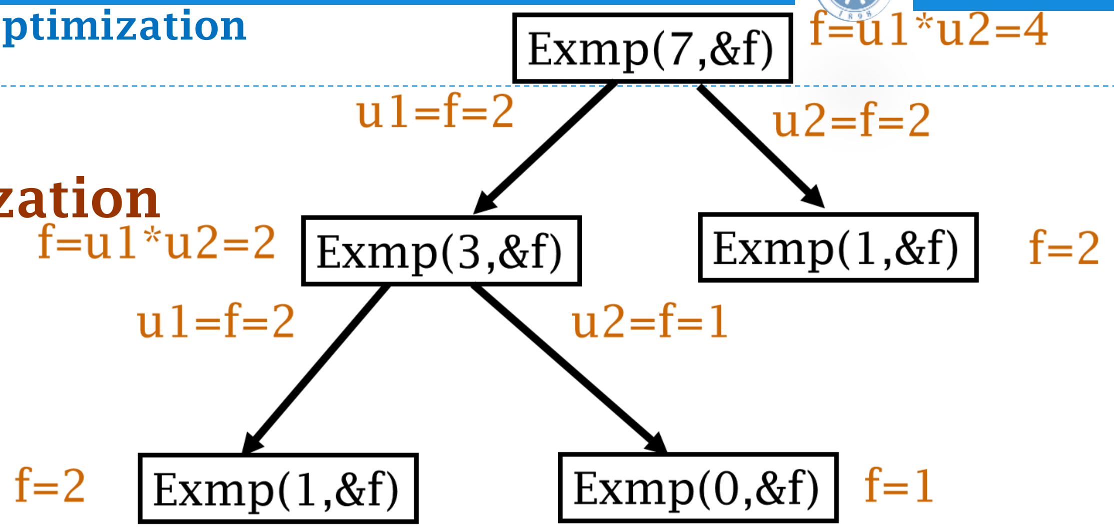
```
    x.pn = (int)(x.pn/4);
    x.rd = 2;
    S.push(x);
    goto label0;
label2: tmp = S.top(); S.pop();
    x = S.top(); S.pop();
    x.q2 = tmp.pf;
    x.pf = x.q1 * x.q2;
    S.push(x);
```

(2) Transformation of recursion




```
label3: x = S.top();
    switch(x.rd)  {
      case 1 : goto label1;
               break;
      case 2 : goto label2;
               break;
      case 3 : tmp = S.top(); S.pop();
               f = tmp.pf;          //finish calculating
               break;
      default : cerr << "error label number in stack";
                break;
    }
}
```

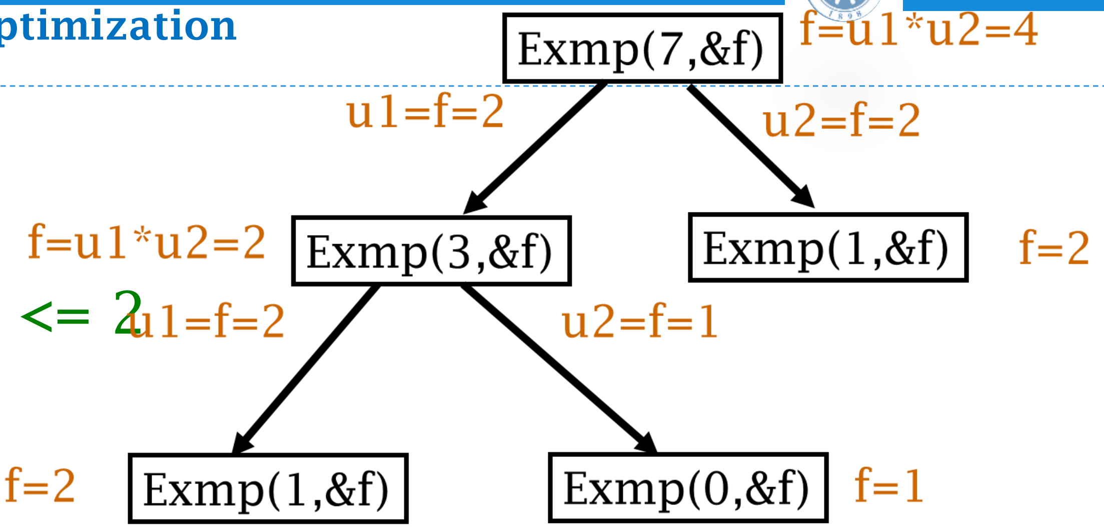
(3) The non recursive function after optimization



The non recursive function after optimization

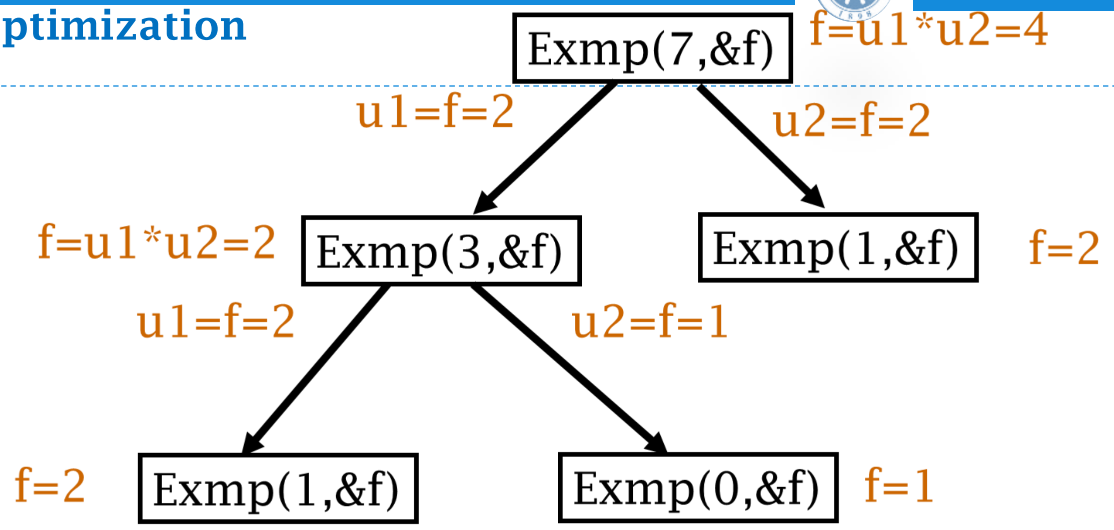
```
void nonrec::replace2(int n, int& f)  {
      ELEM x, tmp;
       // information of the entrance
      x.rd = 3;   x.pn = n;   S.push(x);
      do  {
        // go into the stack along the left side
         while ((x=S.top()).pn >= 2){
            x.rd = 1;
            x.pn = (int)(x.pn/2);
            S.push(x);
         }
```

(3) The non recursive function after optimization




```
x = S.top(); S.pop();   // initial entrance , n <= 2
x.pf = x.pn + 1;
S.push(x);
// If it is returned from the second recursion then rise
while ((x = S.top()).rd==2) {
  tmp = S.top(); S.pop();
  x = S.top(); S.pop();
  x.pf = x.q * tmp.pf;
  S.push(x);
}
```

(3) The non recursive function after optimization




```
        if ((x = S.topValue()).rd == 1) {
            tmp = S.top(); S.pop();
            x = S.top(); S.pop();
            x.q = tmp.pf;    S.push(x);
            tmp.rd = 2;  // enter the second recursion
            tmp.pn = (int)(x.pn/4);
            S.push(tmp);
        }
    } while ((x = S.top()).rd != 3);
    x = S.top(); S.pop();
    f = x.pf;
}
```

Performance experiment of transformation from recursion to non recursive

Comparison of quicksort (unit ms)

Method \ Scale	10000	100000	1000000	10000000
Quicksort with recursion	4.5	29.8	268.7	2946.7
Quicksort with non recursive fixed method	1.6	23.3	251.7	2786.1
Quicksort with non recursive unfixed method	1.6	20.2	248.5	2721.9
Sort in STL	4.8	59.5	629.8	7664.1

Note : testing environment
Intel Core Duo CPU T2350
Memory 512MB
Operating system Windows XP SP2
Programming environment Visual C++ 6.0

Performance experiment of transformation from recursion to non recursive

Scale of processing problems using recursion and non recursive method

- Evaluate f(x) by recursion:

```
int f(int x) {
    if (x==0) return 0;
    return f(x-1)+1;
}
```

  - Under the default settings, when x exceed **11,772** **,**the stack overflow may occur.

- Evaluate f(x) by non recursive method，the element in the stack record the current x and the return value
  - Under the default settings, when x exceed **32,375,567** , error may occur

Questions

- Use the direct transformation for …
  - The factorial function
  - 2-order Fibonacci function
  - Hanoi Tower algorithm

Data Structures and Algorithms

Thanks

the National Elaborate Course (Only available for IPs in China)
http://www.jpk.pku.edu.cn/pkujpk/course/sjjg/

Ming Zhang, Tengjiao Wang and Haiyan Zhao
Higher Education Press, 2008.6 (awarded as the "Eleventh Five-Year" national planning textbook)