

Flowersayo · 로그

Flowersayo · 방문 전

자료구조

자료구조

▼ 목록 보기

2/2

Flowersayo · 방문 전

자료구조

▼ 목록 보기

2/2

## Recursion vs Iteration

Flowersayo · 방문 전

자료구조

▼ 목록 보기

2/2

## Recursion

: Method that solves the problem by calling the algorithms (or function) back. A suitable method for circular definition.

Factorial computation

$$n! = \begin{cases} 1 & n=0 \\ n * (n-1)! & n \geq 1 \end{cases}$$

Fibonacci series

$$fib(n) = \begin{cases} 0 & \text{if } n=0 \\ 1 & \text{if } n=1 \\ fib(n-2) + fib(n-1) & \text{otherwise} \end{cases}$$

Binomial coefficient

$${}_nC_k = \begin{cases} 1 & n=0 \text{ or } n=k \\ {}_{n-1}C_{k-1} + {}_{n-1}C_k & \text{otherwise} \end{cases}$$

## Recursion principle

https://okpki.com/flowersayo/recursion-iteration

Recursion vs Iteration

21.9.15.9:21.9:48

```
int f1
double x = 1, y;
for (int i = 1; i <= n; i++) {
    y = x * y;
}
return y;
```

Time Complexity : O(N)

### 2. Iterative Implementation of Power Computation

double power\_iter(double x, int n) {

if (n == 0) return 1;

else if (n > 0) {

return power(x, n / 2);

else

return x \* power(x, n - 1 / 2);

}

Time Complexity : T(N) = T(N/2) + C → O(logN)

When n is even

$$power(x, n) = power(x^2, \frac{n}{2})$$

When n is odd

$$power(x, n) = x * power(x^2, \frac{n-1}{2}) = x * x^{n-1} = x^n$$

### 3. compare

https://okpki.com/flowersayo/recursion-iteration

Recursion vs Iteration

21.9.15.9:21.9:48

A B C

A<sub>1</sub> A<sub>2</sub> A<sub>3</sub> A<sub>4</sub> A<sub>5</sub> A<sub>6</sub> A<sub>7</sub> A<sub>8</sub> A<sub>9</sub> A<sub>10</sub> A<sub>11</sub> A<sub>12</sub> A<sub>13</sub> A<sub>14</sub> A<sub>15</sub> A<sub>16</sub> A<sub>17</sub> A<sub>18</sub> A<sub>19</sub> A<sub>20</sub> A<sub>21</sub> A<sub>22</sub> A<sub>23</sub> A<sub>24</sub> A<sub>25</sub> A<sub>26</sub> A<sub>27</sub> A<sub>28</sub> A<sub>29</sub> A<sub>30</sub> A<sub>31</sub> A<sub>32</sub> A<sub>33</sub> A<sub>34</sub> A<sub>35</sub> A<sub>36</sub> A<sub>37</sub> A<sub>38</sub> A<sub>39</sub> A<sub>40</sub> A<sub>41</sub> A<sub>42</sub> A<sub>43</sub> A<sub>44</sub> A<sub>45</sub> A<sub>46</sub> A<sub>47</sub> A<sub>48</sub> A<sub>49</sub> A<sub>50</sub> A<sub>51</sub> A<sub>52</sub> A<sub>53</sub> A<sub>54</sub> A<sub>55</sub> A<sub>56</sub> A<sub>57</sub> A<sub>58</sub> A<sub>59</sub> A<sub>60</sub> A<sub>61</sub> A<sub>62</sub> A<sub>63</sub> A<sub>64</sub> A<sub>65</sub> A<sub>66</sub> A<sub>67</sub> A<sub>68</sub> A<sub>69</sub> A<sub>70</sub> A<sub>71</sub> A<sub>72</sub> A<sub>73</sub> A<sub>74</sub> A<sub>75</sub> A<sub>76</sub> A<sub>77</sub> A<sub>78</sub> A<sub>79</sub> A<sub>80</sub> A<sub>81</sub> A<sub>82</sub> A<sub>83</sub> A<sub>84</sub> A<sub>85</sub> A<sub>86</sub> A<sub>87</sub> A<sub>88</sub> A<sub>89</sub> A<sub>90</sub> A<sub>91</sub> A<sub>92</sub> A<sub>93</sub> A<sub>94</sub> A<sub>95</sub> A<sub>96</sub> A<sub>97</sub> A<sub>98</sub> A<sub>99</sub> A<sub>100</sub>

A B C

A<sub>1</sub> A<sub>2</sub> A<sub>3</sub> A<sub>4</sub> A<sub>5</sub> A<sub>6</sub> A<sub>7</sub> A<sub>8</sub> A<sub>9</sub> A<sub>10</sub> A<sub>11</sub> A<sub>12</sub> A<sub>13</sub> A<sub>14</sub> A<sub>15</sub> A<sub>16</sub> A<sub>17</sub> A<sub>18</sub> A<sub>19</sub> A<sub>20</sub> A<sub>21</sub> A<sub>22</sub> A<sub>23</sub> A<sub>24</sub> A<sub>25</sub> A<sub>26</sub> A<sub>27</sub> A<sub>28</sub> A<sub>29</sub> A<sub>30</sub> A<sub>31</sub> A<sub>32</sub> A<sub>33</sub> A<sub>34</sub> A<sub>35</sub> A<sub>36</sub> A<sub>37</sub> A<sub>38</sub> A<sub>39</sub> A<sub>40</sub> A<sub>41</sub> A<sub>42</sub> A<sub>43</sub> A<sub>44</sub> A<sub>45</sub> A<sub>46</sub> A<sub>47</sub> A<sub>48</sub> A<sub>49</sub> A<sub>50</sub> A<sub>51</sub> A<sub>52</sub> A<sub>53</sub> A<sub>54</sub> A<sub>55</sub> A<sub>56</sub> A<sub>57</sub> A<sub>58</sub> A<sub>59</sub> A<sub>60</sub> A<sub>61</sub> A<sub>62</sub> A<sub>63</sub> A<sub>64</sub> A<sub>65</sub> A<sub>66</sub> A<sub>67</sub> A<sub>68</sub> A<sub>69</sub> A<sub>70</sub> A<sub>71</sub> A<sub>72</sub> A<sub>73</sub> A<sub>74</sub> A<sub>75</sub> A<sub>76</sub> A<sub>77</sub> A<sub>78</sub> A<sub>79</sub> A<sub>80</sub> A<sub>81</sub> A<sub>82</sub> A<sub>83</sub> A<sub>84</sub> A<sub>85</sub> A<sub>86</sub> A<sub>87</sub> A<sub>88</sub> A<sub>89</sub> A<sub>90</sub> A<sub>91</sub> A<sub>92</sub> A<sub>93</sub> A<sub>94</sub> A<sub>95</sub> A<sub>96</sub> A<sub>97</sub> A<sub>98</sub> A<sub>99</sub> A<sub>100</sub>

Original problem

Move 4 discs A<sub>1</sub>, A<sub>2</sub>, A<sub>3</sub>, A<sub>4</sub> from A to C

Move 3 discs A<sub>1</sub>, A<sub>2</sub>, A<sub>3</sub> from A to B

Move 2 discs A<sub>1</sub>, A<sub>2</sub> from A to C

Move 1 disc A<sub>1</sub> from A to B

Note) T(n) = 2<sup>n</sup> - 1

### 1. recursive Implementation of Hanoi tower

void hanoi\_iter(int n, char from, char to, char by)

{

if (n == 1) printf("Move 1 disc from %c to %c\n", from, to);

else {

hanoi\_iter(n - 1, from, to, by);

printf("Move disc %d from %c to %c\n", n, from, to);

hanoi\_iter(n - 1, by, from, to);

}

}

Time Complexity : T(N) = 2T(N-1) + 1 = 2<sup>n</sup> - 1 → O(2<sup>n</sup>)

### 2. Iterative Implementation of Hanoi tower

### 3. compare

https://okpki.com/flowersayo/recursion-iteration

Recursion vs Iteration

21.9.15.9:21.9:48

https://okpki.com/flowersayo/recursion-iteration

21.9.15.9:21.9:48

https://okpki.com/flowersayo/recursion-iteration

21.9.15.9:21.9:48

## Divide-and-conquer

: Divide the problem into a set of sub problems. recursively **breaks down a problem** into two or more sub-problems of the same or related type, until these become simple enough to be **solved** directly.

1) Divide problem

Max in [2, 5, 3, 1]?  
then return 5

2) Divide problem

Max in [5, 3, 1]?  
then return 5

3) Divide problem

Max in [3, 1]?  
then return 3

4) Return 1

5) Compare 3 and 1,  
then return 3

6) Compare 5 and 3,  
then return 5

7) Compare 2 and 5,  
then return 5

## Recursion types

- tail recursion : **easy** implemented using iteration  
ex) return n\*factorial(n-1);
- head recursion : **difficult** to implement using iteration  
ex) return factorial(n-1) \* n;
- multi recursion : **difficult** to implement using iteration  
ex)

function(A, n)

{

//recursive call of 2-

function(A, n-1)

function(A, n-1)

}

## Recursion VS Iteration

- Recursion : using recursive calls  
pros - Good choice for recursive problems (**easy** to implement)  
cons - overhead of function calls → usually **slower** execution time

Recursion vs Iteration

21.9.15.9:21.9:48

	slow_power (iteration)	power (recursion)
Time complexity	O(n)	O(log <sub>2</sub> n)
Execution time	7.17 sec	0.47 sec

iteration is not good choice

## Fibonacci Series

### Fibonacci Series

Ex) 0,1,1,2,3,5,8,13,21,...

$$fib(n) = \begin{cases} 0 & n=0 \\ 1 & n=1 \\ fib(n-2) + fib(n-1) & \text{otherwise} \end{cases}$$

### 1. recursive Implementation of Fibonacci Series

int fib\_recursion(int n) {

if (n == 0) return 0;

if (n == 1) return 1;

return fib\_recursion(n - 1) + fib\_recursion(n - 2);

}

Time Complexity : T(N) < 2<sup>n</sup> (N-1) → O(2<sup>n</sup> N)

why ? for fib(n), maximum depth of tree is n-1, and tree must be not filled with leaf node fully.

## Binary Search

: when an array of ordered numbers is given, find an index k where a[k]=b.

### 1. recursive Implementation of Binary Search

int search\_iter(A, b, start, end)

{

if (start > end) return -1;

int mid = (start + end) / 2;

if (A[mid] == b)

return mid;

else if (A[mid] < b)

return search\_iter(A, b, mid + 1, end);

else

return search\_iter(A, b, start, mid - 1);

}

Time Complexity : O(N)

### 2. Iterative Implementation of Binary Search

int search\_iter(A, b, start, end)

{

if (start > end) return -1;

int mid = (start + end) / 2;

if (A[mid] == b)

return mid;

else if (A[mid] < b)

start = mid + 1;

else

end = mid - 1;

}

Time Complexity : T(N) = t(N/2) + C → O(logN)

### 3. compare

21.9.15.9:21.9:48

21.9.15.9:21.9:48

21.9.15.9:21.9:48

21.9.15.9:21.9:48

- Iteration : using **for** or **while** loop  
pros - fast execution time  
cons - programming can be often very **difficult** for recursive problems
- Then, what is best strategy?  
→ It **depends on problems**.

## Factorial

$$n! = \begin{cases} 1 & n=0 \\ n * (n-1)! & n \geq 1 \end{cases}$$

### 1. recursive Implementation of Factorial

int factorial\_iter(int n) {

if (n == 1) return 1;

else return n \* factorial\_iter(n - 1);

}

T(N) = T(N-1) + 1 → Time Complexity : O(N)

$$T(n) = T(n-1) + T(n-2) + 1$$

$$T(n) = \sum_{k=0}^{n-1} \binom{n-k-1}{k} = O(2^n)$$

### 2. Iterative Implementation of Fibonacci Series

: to get fib(n), calculating one by one by one from the first two terms .

int fib\_iter(int n) {

int temp, current = 1, last = 0; //the first two terms of Fibonacci sequence.

if (n < 2) return n;

else {

for (int i = 1; i <= n; i++) {

temp = current; //keep the last value.

current = last + current; //removed nth term.

last = temp; //save it for the next calculation.

}

return current;

}

Time Complexity : O(N)

### 3. compare

21.9.15.9:21.9:48

Recursion vs Iteration

21.9.15.9:21.9:48

실시간

EMHA CSE 20

이전 프로젝트

Data Structure

0개의 댓글

댓글을 작성하세요

댓글 작성

21.9.15.9:21.9:48

Recursion vs Iteration

21.9.15.9:21.9:48

factorial(3) = 3 \* factorial(2)  
= 3 \* 2 \* factorial(1)  
= 3 \* 2 \* 1  
= 6

Operation in ①, ②

1. Save the return address at system stack

2. Allocate parameters and local variables from system stack

3. Jump to the address of the called function

Operation in ③, ④

1. Call the return address from system stack

2. Go back to the call function

### 2. Iterative Implementation of Factorial

int factorial\_iter(int n) {

int mul = 1;

for (int i = 1; i <= n; i++) {

mul \*= i; // mul = (i-1) \* ... \* 2 \* 1;

}

return mul;

}

Time Complexity : O(N)

### 3. compare

## Power Computation

: Let's find n-squared value of x : x<sup>n</sup>

### 1. recursive Implementation of Power Computation

double power\_iter(double x, int n) { //x^n

https://okpki.com/flowersayo/recursion-iteration

Recursion vs Iteration

21.9.15.9:21.9:48

## Hanoi tower

: move n disc stacked on rod A to rod C.

For n discs

n-1 disc

1 disc

A B C

A B C

A B C

Using C as a temporary buffer. Move n-1 discs stacked in A to B.

Move the largest disc of A to C.

Using A as a temporary buffer, move n-1 discs in B to C.

## Divide and conquer