Data structure and Algorithms

Recursive algorithms

Thanh-Hai Tran

Electronics and Computer Engineering
School of Electronics and Telecommunications

Hanoi University of Science and Technology
1 Dai Co Viet - Hanoi - Vietnam

Outline

- **Basic concepts**
  - Recursion (sự đệ quy)
  - Recursive algorithms (giải thuật đệ quy)
    - Structure of recursive algorithms
    - Operation of recursive algorithms
- **Recursive procedures (thủ tục đệ quy)**
  - Concepts
  - Structure of recursive procedure
  - Operation
  - Implementation principles
  - Cancellation of recursion (Khử đệ quy)
- **Backtracking algorithms**

Outline

- **Examples of recursive algorithms**
  - Searching in linked lists
  - Problem of Hanoi tower (Bài toán Tháp Hà Nội)
  - Problem of 8 queens (Bài toán 8 con hậu)

Basic concept

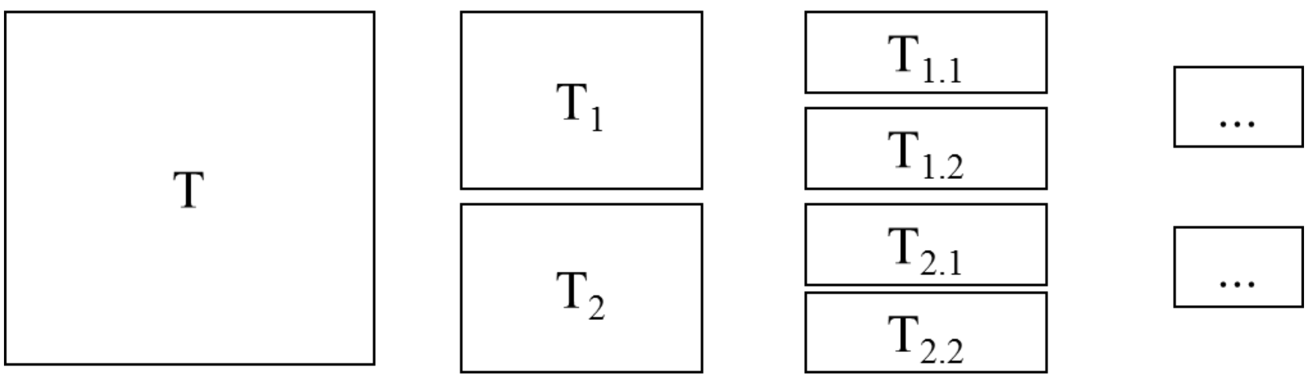
- **Recursion by examples:**
  - **String: is defined as:**
    - Rule 1: 1 char = String
    - Rule 2: String = 1 char + (sub) String
  - **Natural number: a natural number N is defined as:**
    - Rule 1: 1 is a natural number
    - Rule 2: x is a natural number if (x-1) is a natural number
  - **N! = 1.2. … .N can be defined as:**
    - Rule 1: 0! = 1
    - Rule 2: N! = N (N-1)!
  - **Definition of a linear list:**
    - Rule 1: `L` = empty is a linear list
    - Rule 2: If L_{n-1} is a linear list of $n-1$ length, then the construt $L_n = \langle a, L_{n-1} \rangle$ which means element `a` is before L_{n-1}, is also a linear list.

Basic concepts

- **Recursive definition (định nghĩa đệ quy): the way defining an object that based on other similar (usually smaller) objects. A recursive definition includes 2 parts:**
  - *Basic rule (Quy tắc cơ sở)*: also called basic case where a special case of target object is defined directly.
  - *Inductive rule (Quy tắc đệ quy)*: where the object is defined indirectly based on other similar objects
- **Recursive property (tính chất đệ quy): an object has recursive property if it can be defined recursively. It means that the object contains other similar objects.**

Basic concepts

- **Recursive algorithm (Giải thuật đệ quy)**
  - Exp 1: Searching for a given word `w` in a dictionary `T`
- **Main idea of algorithm:**
  - `T` is partitioned into 2 equal sub-dictionaries T_1 and T_2.
  - Searching for `w` in T_1 and T_2
  - The process continues until the final sub-dictionary has only one page, and the search can be done directly.

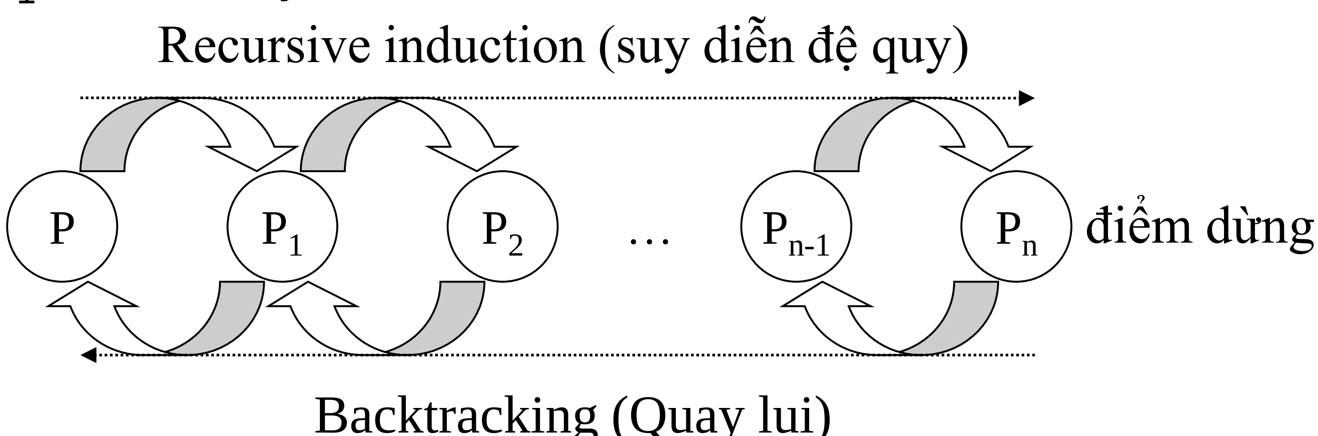


Basic concepts

- **Recursive algorithm**
  - Exp 2: Calculation of Fibonacci series:
    - F(n) = 1 for n <= 2 (F(1) = F(2) =1);
    - F(n) = F(n-1) + F(n-2) for n > 2;

Basic concepts

- **Recursive algorithm (Giải thuật đệ quy)**
  - *Concept*: Given a problem P, an algorithm for P is called **recursive** if it has following form:
    - For solving P, we need to solve problem P_1 that is similar and smaller than P. It means that if P_1 is solved, than P is also solved.
    - Similarly, for solving P1, we find another similar and smaller problem P2 needs to be solved. The process continues until we find problem P_n that is similar and smaller P_{n-1}.
    - P_n is small enough that it can be directly solved.
    - After resolving Pn, we will solve all generated problems P_{n-1}, P_{n-2}, …, P_1, and finally P.



Recursive algorithms

- **Exp 3: Find the minimal element of a given series with N elements $a_1, a_2, \ldots, a_N$**
  - If $N=1$ then $min=a_1$;
  - Else the series is divided into 2 sub-series:

    $L_1 = a_1, a_2, \ldots, a_m$ and $L_2 = a_{m+1}, a_{m+2}, \ldots, a_N$
    with $m = (1+N)$ DIV 2.
    Find min_1 in L_1 and min_2 in L_2.
    Compare min_1 and min_2 to find the minimal.

Recursive procedure

- **Concepts:**
  - *Recursive procedure:* is a simple and effective tool (sub-program) supported in most programming languages to implement recursive algorithms.
  - In C/C++ it is called *recursive function*.
  - Recursive procedure/function: is one that contains at least one *recursive call* (call to itself) in its body.

Recursive procedures

- **Examples**
  - Exp 6: Calculation of n!
    - 0! = 1
    - n! = n (n-1)!

```
int Fact (int n){
    if  (n <= 1) return 1;
    else return n * Fact (n-1);
}
```

recursive calls

  - Recall exp 2: Calculation of series Fibonacci
    - F (1) = 1 if n <= 2;
    - F (n) = F (n-1) + F (n-2) if n > 2;

```
int Fibo1 (int n) {
        if (n <= 2) return 1;
        else return (Fibo1 (n-1) + Fibo1 (n-2));
}
```

Recursive procedures

- **Structure of recursive procedure/function: In C/C++, a recursive function has the form:**

```
void  P(A) {
    if (A==A0)
         Base case
    else {             //recursive case
        Q1();
        P(A1);         //recursive call
        Q2();
        P(A2);         //recursive call
        …
    }
}
```

- **Structure of recursive procedure/function:**
  - *Header of function*: In header a recursive function must have at least one parameter. Besides of normal role of input/output, a parameter in recursive function can express the size of the function.
  - *Body of function*: including 2 branches that corresponds to 2 cases of the algorithm:
    - Base branch (Nhánh cơ sở): containing necessary statements for implementing the base case.
    - Recursive branch (Nhánh đệ quy): containing at least one recursive call.

- **There are 2 types of recursive calls:**
  - Direct recursive call (calling to itself): the called function contains the call.
  - Indirect recursive call (Gọi gián tiếp):
    Example:

```
void A(n)  {
        ...
        B(p);
        ...
}
```

```
void B(m){
        ...
        A(k);
        ...
}
```

Operation of recursive procedure

```
int Fact (int n){
   if  (n <= 1) return 1;
   else return n * Fact (n-1);
}

void main(){
   int n = 3;
   int f = Fact(n);//operating process starts here

}
```

Operation of recursive procedure

- **Running process of a RP includes 2 stages:**
  - Recursive call stage (Giai đoạn gọi đệ quy):
    - Starting from the first call, the recursive branch will be repeated until the base branch arrives. In this stage, many intermediate results generated by intermediate calls need to be saved, because they will be used in the next stage.
    - The next stage (backtracking) will be triggered automatically.
  - Backtracking stage (Giai đoạn quay lui):
    - In this stage, all intermediate generated calls will be run one by one, and in reversed order.
- **Problem:** how intermediate calls and results can be saved in fisrt stage (recursive call) to be processed in the second stage (backtracking)?

Operation of recursive procedure

- **Implementation method: The saving and processing of recursive calls in 2 operating stages have following features:**
  - Last In, First Out (LIFO):
    - The processing order of recursive calls in second stage is reversed to the order in first stage. Therefore LIFO structure (stack) is a suitable choice.
  - Homogeneity (Tính đồng nhất):
    - Structure of recursive calls are the same, so homogenous stack can be used.
- **In reality, most programming languages (including C/C++) have already supported/implemented recursive calls and procedures.**

Operation of recursive procedure

(1) **Initialize Stack: S = empty;**

(2) **Recursive calls stage (Giai đoạn gọi đệ quy):**

- In this stage, intermediate recursive calls and results will be pushed into stack.

(3) **Backtracking stage (Giai đoạn quay lui)**

- Recursive calls and results saved in the stack will be popped to be processed until the stack is empty.

Operation of recursive procedure

- **Exp: operation of recursive function Fact(n) with n=3 (calculation of 3!).**

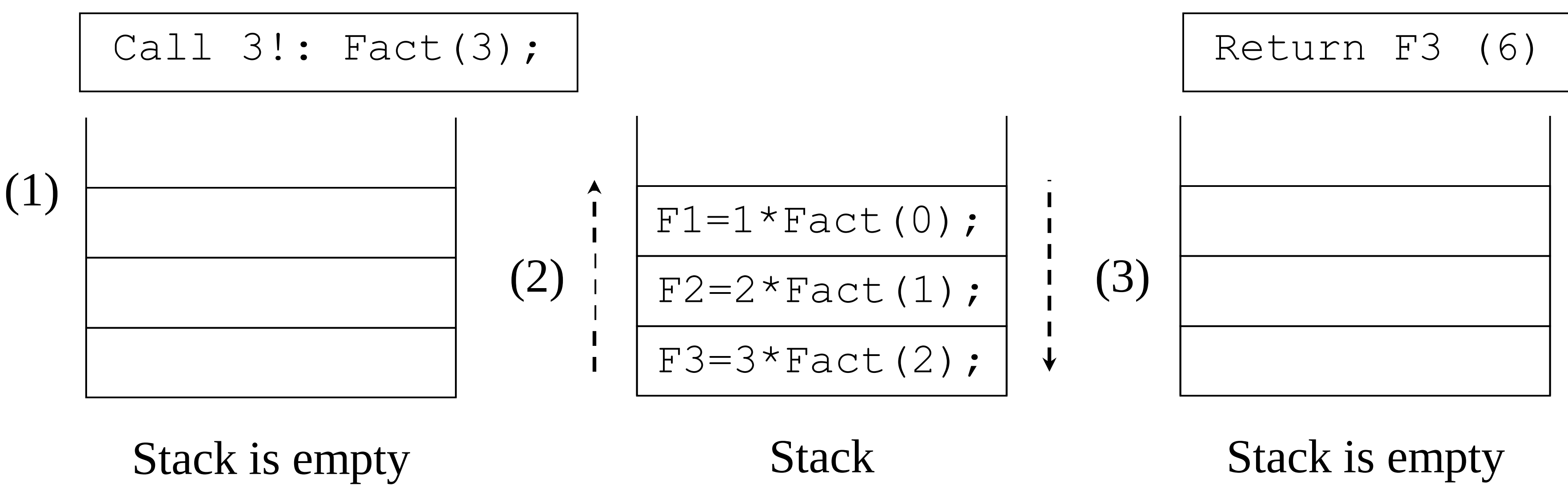
Recursive algorithm (recall)

Recursive induction

```
3! = 3 x 2!
2! = 2 x 1!
1! = 1 x 0!
0! = 1:
```
base case

Backtracking

Operation of recursive function Fact()



Recursive vs Non-recursive procedures

- **Recursive procedures**
  - Advantages:
    - Shorter
    - Easier to understand
  - Disadvantages:
    - More complex
    - More requirement of memory
    - Stack overflow error may happen

Cancellation of recursion

- *Concept*: *cancellation of recursion* **(sự khử đệ quy) means that recursive algorithm is replaced by non-recursive algorithm, or recursive calls is replaced by other implementation.**

  → Recursive calls are killed (cancelled)

- **Exp 1:**

```
int Fact (int n){
   if  (n <= 1) return 1;
   else return n * Fact (n-1);
}
```

```
int Fact (int n){
   if  (n <= 1) return 1;
   else {
      int x=1;
      for (int i=2;i<=n;i++)
         x*=i;
      return x;
    }
}
```

Cancellation of recursion

- **Ex 2**

```
int Fibo1(int n){
    if (n <= 2) return 1;
    else
        return (Fibo1 (n-1) +
            Fibo1 (n-2));
}
```

```
int  Fibo2(int n)  {
    int i, f, f1, f2;
    f1 = 1 ;
    f2 = 1 ;
    if  (n <= 2)  f = 1;
    else
        for (i = 3;i<=n;i++){
            f = f1 + f2;
            f1 = f2;
            f2 = f;
        }
    return f;
}
```

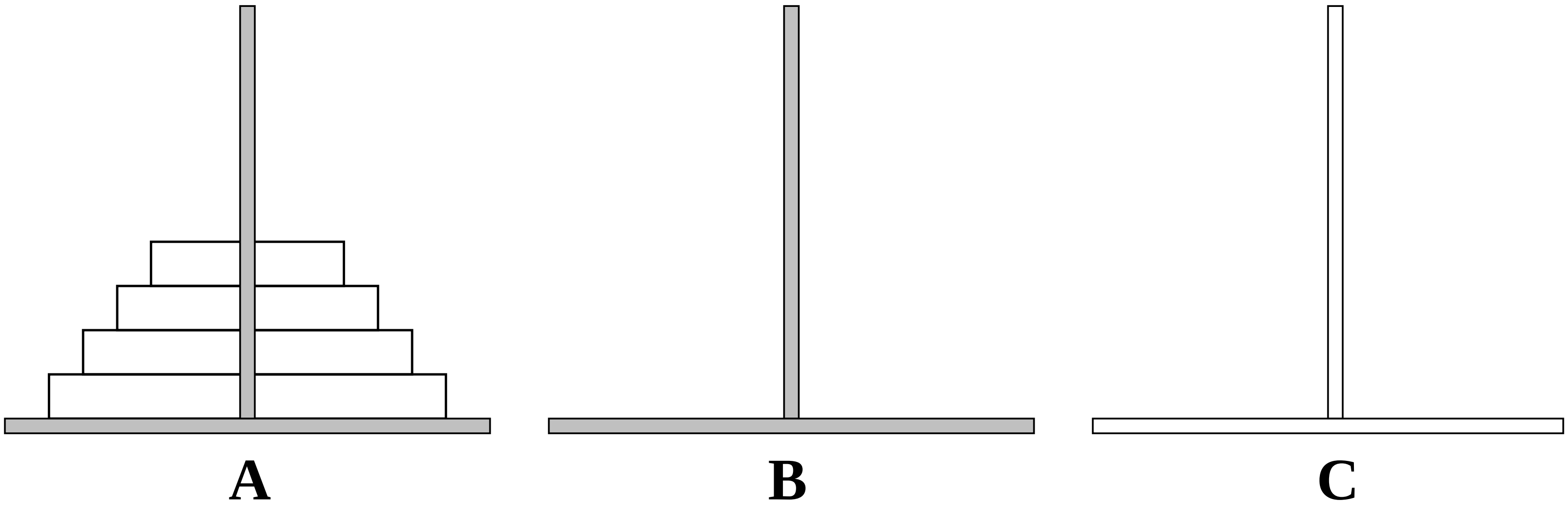
Examples of recursive algorithms & procedures

- **Exp 1: Searching in linked lists:**
  - Write a recursive function that search for an element k in linked list H which points to head of the list. If found, the function returns the pointer pointing to found element. Otherwise, it returns NULL.

```
PNode Search (Item k, PNode H) {
   if (H == NULL) return NULL;
   else
       if  (H->info == k) return H;
       else  return Search (k; H->next) ;
}
```

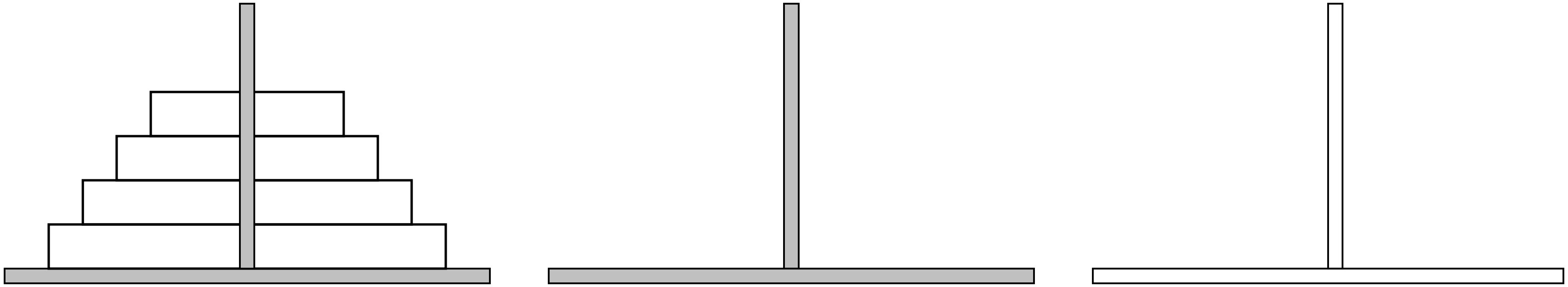
Examples of recursive algorithms & procedures

- **Exp 2: Problem of Hanoi tower (Bài toán Tháp Hà Nội)**
  - Problem: there are N disks with different sizes and 3 towers A, B and C. At the beginning, N disks lie on each others (smaller one on top of bigger one) at tower A. You are required to move N disks from tower A to tower B with following conditions:
    - Each time only one disk can be moved
    - Bigger disk is never allowed to lie on top of smaller one.
    - Only one intermediate tower C can be used



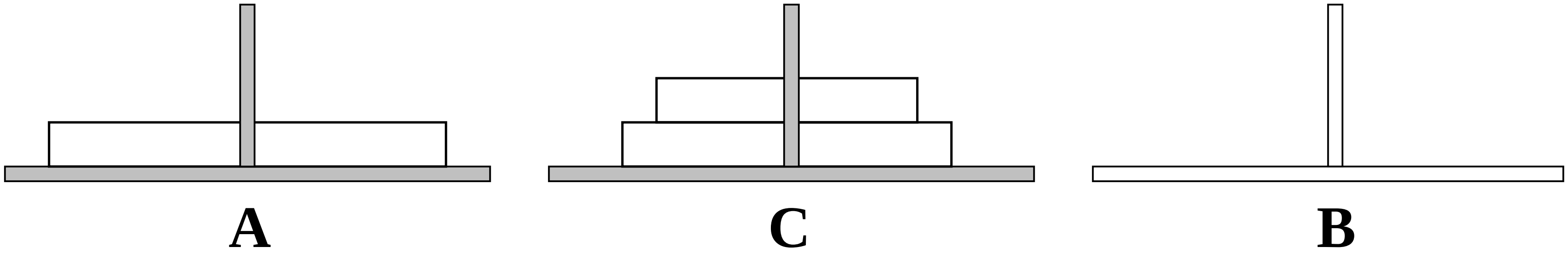
Hanoi tower (Bài toán Tháp Hà Nội)

- Some simple cases
  - 1 disk: A → B
  - 2 disks: A → C, A → B, C → B
- General case: movement process is follows:
  1. Move `N-1` disks from A to C
  2. Move `1` disk from A to B
  3. Move `N-1` disks from C to B

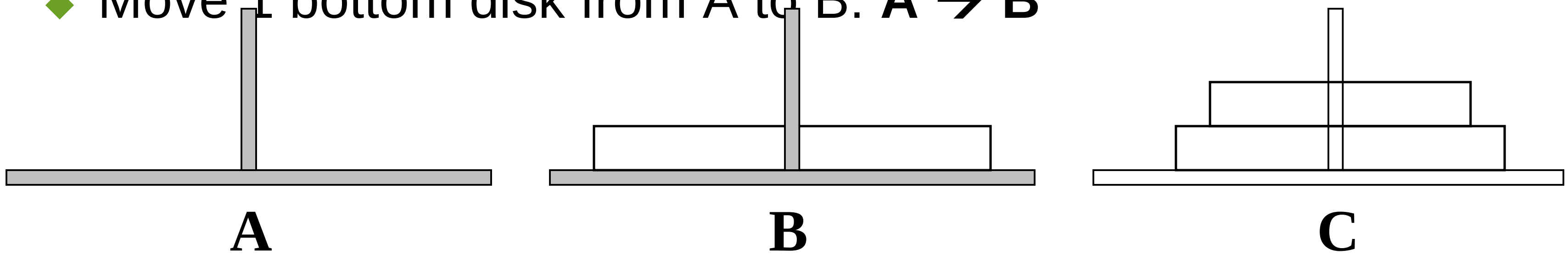


Hanoi tower (Bài toán Tháp Hà Nội)

- **For example with 3 disks:**
  - Move 2 top disks from A to C: **A → B, A → C, B → C**



  - Move 1 bottom disk from A to B: **A → B**



  - Move 2 disks from C to B: **C → A, C → B, A → B**

Hanoi tower (Bài toán Tháp Hà Nội)

- **Function structure:**
  - Base case: `N=1` disk: A → B
  - Recursive case: `N>1`:
    1. Move `N-1` disks from A to C
    2. Move `1` disk from A to B
    3. Move `N-1` disks from C to B

```
void  TowerHN (int n, Tower A, B, C){
   if (n == 1) Transfer (A, B);
   else {
      TowerHN (n-1, A, C, B);
      TowerHN (1, A, B, C);
      TowerHN (n-1, C, B, A);
   }
}
```

Backtracking Algorithms

- Ideas of backtracking algorithm (Ý tưởng giải thuật)
- Implementation methods (Cách cài đặt)

Backtracking Algorithms

- **Ideas of backtracking algorithm:**
  - *Problem*: given a set of N variables $x_1, x_2, \ldots, x_N$ with N corresponding value domains $D_1, D_2, \ldots, D_N$. Find a set of N variables with suitable values that satisfies given condition K.
  - *Ideas of Backtracking Algorithm*:
    - Set vector $V_i = <x_1, x_2, \ldots, x_i>$, with $i=1..N$.
    - We need to find vector V_N satisfying condition K, with $x_i \in D_i$ and $i=1..N$.
    - The algorithm includes N steps, each one aims to find one suitable variable in vector V.

Ideas of backtracking algorithm

Recursive algorithm for backtracking as follows:

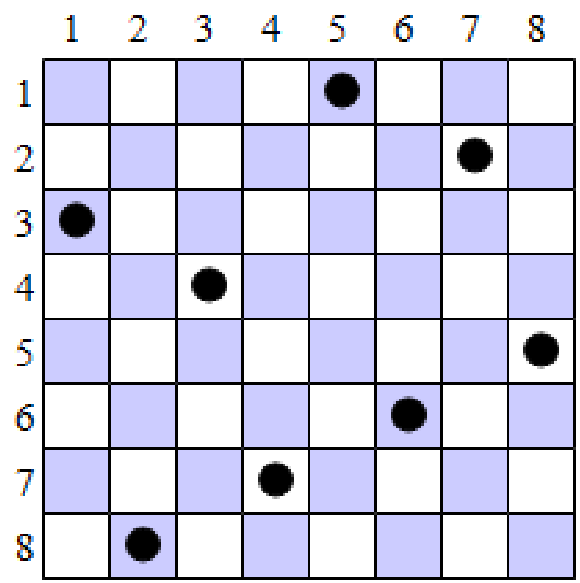
- **Suppose that first i steps have been done, it means that we found V_i=<x_1, x_2,…, x_i> satisfies K. If i=N the algorithm finishes, it returns V_N. Otherwise, next step continues with two exclusive cases:**
  - If found suitable variable, means that $\exists\ p \in D_{i+1}$ so that with $x_{i+1}=p$ then V_{i+1} satisfies K, set $x_{i+1}=p$. It repeats for next step (i+1) (recursive induction).
  - Otherwise, means that $\forall p \in D_{i+1}$ we all have V_{i+1} not satisfied K. We have to backtrack to step i to try next suitable value of x_i (current x_i is suitable). If not found and i=1, the algorithm terminates without any suitable vector.

Example of backtracking

- **Problem of 8 queens (Bài toán 8 con hậu)**

  Find an arrangement of 8 queens on chessboard so that not any two queens check each other. This problem can be generalized for N queens.

- **One of 92 solutions:**



- ***Ideas of Algorithm***:
  - Chessboard can be considered as an 2D array a_{NxN}, (N=8 in reality);
  - Suppose that N queens are: h_1, h_2,…, h_N, and queen h_i will be placed in row *i*, with *i=1..N*.
  - We need to choose suitable column of each queen so that they do not check each other.

- **The algorithm consists of `N` steps, each one will find a suitable position (column) for one queen as follows:**
  - Suppose that at step i (first step $i=1$), i queens $h_1, h_2, \ldots, h_i$ have been placed in first i rows and this arrangement satisfies the condition (with $1 \le i \le N$). If $i=N$ the algorithm finishes and returns the result. Otherwise it continues with next step.
  - At step $i+1$, it finds a suitable column at row $i+1$ in order to put the queen h_{i+1}. Two cases may happen:
    - If a suitable column j is found, then put queen h_{i+1} at that position j. Go to the next step $i+1$.
    - Otherwise if all N columns are not suitable, it must go back to step i and try to find next suitable column (backtracking) of h_i (recall that h_i is already in a suitable position). If no next suitable position exists for h_i and $i=1$ then it terminates without any solution.

Backtracking Algorithms

- **Implementation methods: there are 2 methods:**
  - Method 1: Using recursive procedures (thủ tục đệ quy)
  - Method 2: Using non-recursive procedures (thủ tục không đệ quy)

Backtracking Algorithms
Implementation methods

- **Method 1: Using recursive procedures**

```
void  RBacktrack(vector V, int step)
{
   if (step == N) {
      V is a solution;
   }
   else
      while (exists xi in Di and V+{xi} satisfies K)
      {
         Di = Di \ {xi} ;
         V = V+ {xi};
         RBacktrack(V, step+1);
         V = V \ {xi};  //prepare before backtrack
      }
}
```

Backtracking Algorithms
Implementation methods

- **Method 2: Using non-recursive procedures (loops)**

```
int  NBacktrack ()
{
    int  i = 1;
    int tong = 0;             //total number of solutions
    vector  V = ∅ ;           //V is empty
    do {
        while (exists x_i in D_i and V+{x_i} satisfies K)
        {
            D_i  = D_i  \ {x_i}  ;
            V = V+ {x_i};
            if  (i == N)  {
            V is a solution;
            tong++;
        }
        else  i++;
        }
        V = V \ {x_i};
        i-- ;  //Backtrack
    }
    while  (i >= 1);
    return  tong;
}
```

Exercises

1. **Find recursive algorithm and function that sum up a series of N numbers.**
2. **Find recursive algorithm and function that look for minimal element in a series of N numbers.**
3. **Write a recursive function for insertion sort algorithm.**
4. **(Subset sum problem): Given a series A of N numbers and a number K, is there a group of numbers of A that sums exactly to K?**

References

- Recursive algorithm – slide – Nguyen Thanh Binh – Data structure and Algorithm