HUST

ĐẠI HỌC BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

ONE LOVE. ONE FUTURE.

DATA STRUCTURES AND ALGORITHSMS

ĐẠI HỌC
BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY
OF SCIENCE AND TECHNOLOGY

DATA STRUCTURES AND ALGORITHMS

WEEK 3: RECURSIVE BACKTRACKING

ONE LOVE. ONE FUTURE.

CONTENT

- General recursive backtracking diagram
- Listing binary strings
- Listing permutations
- Sudoku puzzles

GENERAL DIAGRAM

- Combination enumeration problem: List the sets x = (x[1], x[2],…, x[k], x[k+1],. . ., x[n]) with x[i] ∈ Ai , i = 1, 2, ..., n and satisfy the given set of constraints P.
- Example:
  - “The problem of listing a binary string of length n” leads to listing the sets x = (x[1], x[2],…, x[k], x[k+1],. . ., x [n]) with x[i] ? {0, 1}, i = 1, 2, ..., n
  - “The problem of listing a binary string of length n with an even number of 0 bits” leads to listing the sets x = (x[1], x[2],…, x[k], x[k +1],. . ., x[n]) with x[i] ? {0, 1}, i = 1, 2, ..., n and satisfying the constraint: number of elements x[i] = 0 with i = 1, 2, ..., n is an even number.
  - The backtracking algorithm allows solving combinatorial enumeration problems. There are two ways to implement the backtracking algorithm: recursive or non-recursive.

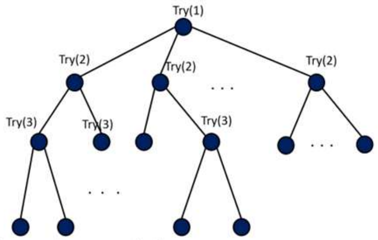
GENERAL DIAGRAM

- Combination enumeration problem: List the sets x = (x[1], x[2],…, x[k], x[k+1],. . ., x[n]) with x[i] ∈ Ai , i = 1, 2, ..., n and satisfy the given set of constraints P.
- The call to execute the backtracking recursive algorithm is: Try(1);
- If only need to find one solution, you need to find a way to terminate the nested recursive procedure calls generated by the Try(1) call after the first solution is recorded.
- If at the end of the algorithm we do not get any solution, it means the problem has no solution.

```
Try(k){// try values assignable to x[k]
  for v in candidates(k) do {
     if (check(v,k)) then {
        x[k] = v;
        [Update the data structure D]
        if (k == n) then solution();
        else Try(k+1);
        [Recover the data structure D]
     }
  }
}
```

GENERAL DIAGRAM

- Combination enumeration problem: List the sets x = (x[1], x[2],…, x[k], x[k+1],. . ., x[n]) with x[i] ∈ Ai , i = 1, 2, ..., n and satisfy the given set of constraints P.
- The call to execute the backtracking recursive algorithm is: Try(1);



```
Try(k){// try values assignable to x[k]
  for v in candidates(k) do {
     if (check(v,k)) then {
        x[k] = v;
        [Update the data structure D]
        if (k == n) then solution();
        else Try(k+1);
        [Recover the data structure D]
     }
  }
}
```

LISTING BINARY STRINGS

- Given a positive integer n ≥ 1. List all binary strings of length n in lexicographic order.
- For example: n = 3, we have binary strings of length 3 that need to be listed in the following order:
  - 000
  - 001
  - 010
  - 011
  - 100
  - 101
  - 110
  - 111

LISTING BINARY STRINGS

- Given a positive integer n ≥ 1. List all binary strings of length n in lexicographic order.
- Solution representation: each binary string is represented by the array (x[1], x[2], . . ., x[n]) in which x[k] ∈{0,1} is the k^{th} bit in binary string.

```
Try(k){// try values assignable to x[k]
  for v in candidates(k) do {
     if (check(v,k)) then {
        x[k] = v;
        [Update the data structure D]
        if (k == n) then solution();
        else Try(k+1);
        [Recover the data structure D]
     }
  }
}
```

Determine:
- candidates(k)
- check(v, k)

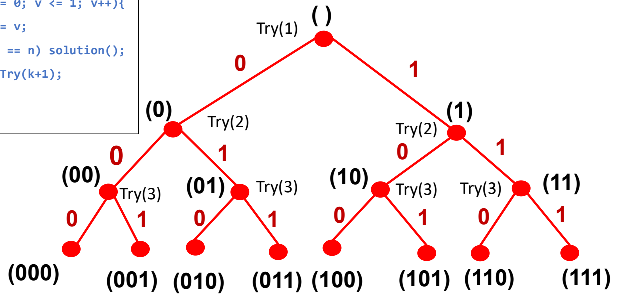
→

```
void Try(int k){
    for (int v = 0; v <= 1; v++){
        x[k] = v;
        if (k == n) solution();
        else Try(k+1);
    }
}
```

LISTING BINARY STRINGS

```
void Try(int k){
  for (int v = 0; v <= 1; v++){
        x[k] = v;
        if (k == n) solution();
        else Try(k+1);
  }
}
```

Try to list binary strings of length 3



LISTING PERMUTATIONS

- Given a positive integer n ≥ 1. List all permutations of n numbers 1, 2, ..., n in lexicographic order.
- For example: n = 3, we have the permutations of 1, 2, 3 in lexicographic order as follows :
  - (1, 2, 3)
  - (1, 3, 2)
  - (2, 1, 3)
  - (2, 3, 1)
  - (3, 1, 2)
  - (3, 2, 1)

LISTING PERMUTATIONS

- Solution representation: each permutation of n elements is represented by the array (x[1], x[2], . . ., x[n]) where :
  - x[k] ∈{1, 2,..., *n*} is the kth element in the permutation
  - x[k] ≠ x[1], x[2],... x[*k*-1], x[*k*+1], ...,x[*n*]

```
Try(k){// try values assignable to x[k]
  for v in candidates(k) do {
     if (check(v,k)) then {
        x[k] = v;
        [Update the data structure D]
        if (k == n) then solution();
        else Try(k+1);
        [Recover the data structure D]
     }
  }
}
```

Determine:
- candidates(k)
- check(v, k)

→

```
void Try(int k){
    for (int v = 1; v <= n; v++){
        if (check(v, k)) {
            x[k] = v;
            if (k == n) solution();
            else Try(k+1);
        }
    }
}
```

LISTING PERMUTATIONS

```
#include <stdio.h>
int n;
int x[100];
void solution(){
    for(int k = 1; k <= n; k++)
        printf("%d ",x[k]);
    printf("\n");
}
int check(int v, int k) {
  for (int i = 1; i <= k-1; i++)
    if (x[i] == v) return 0;
  return 1;
}
```

```
void Try(int k){
    for (int v = 1; v <= n; v++){
        if (check(v, k)) {
           x[k] = v;
           if (k == n) solution();
           else Try(k+1);
        }
    }
}
int main(){
   scanf("%d",&n);
   Try(1);
}
```

LISTING PERMUTATIONS

- Marking technique
  - used[v] = 1: v appear
  - used[v] = 0: v does not appear

```
try(k){//try values assignable to x[k]
  for v in candidates(k) do {
     if (check(v,k)) then {
        x[k] = v;
        [Update the data structure D]
        if (k == n) then solution();
        else try(k+1);
        [Recover the data structure D]
     }
  }
}
```

```
void Try(int k){
    for(int v = 1; v <= n; v++){
        if (used[v]==0){
            x[k] = v;
            used[v] = 1;
            if (k == n) solution();
            else Try(k+1);
            used[v] = 0;
        }
    }
}
int main(){
   scanf("%d",&n);
   for(int v = 1; v <= n; v++) used[v] = 0;
   Try(1);
}
```

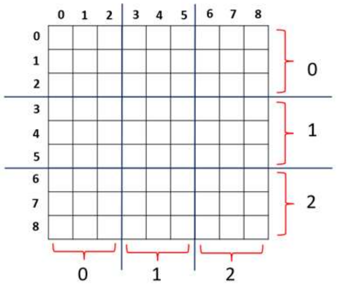
LISTING PERMUTATIONS

```
#include <stdio.h>
int n;
int x[100];
int used[100];
void solution(){
    for(int k = 1; k <= n; k++)
         printf("%d ",x[k]);
    printf("\n");
}
```

```
void Try(int k){
    for(int v = 1; v <= n; v++){
        if (used[v]==0){
            x[k] = v;
            used[v] = 1;
            if (k == n) solution();
            else Try(k+1);
            used[v] = 0;
        }
    }
}
int main(){
   scanf("%d",&n);
   for(int v = 1; v <= n; v++) used[v] = 0;
   Try(1);
}
```

SODOKU PUZZLES

- Problem statement:
  - Given a 9x9 square grid divided into 9 3x3 grids.
  - Rows and columns are numbered 0, 1, 2, ..., 8
  - List all the ways to fill in the numbers 1, 2. ..., 9 cells in a 9x9 square grid such that:
    - The numbers on each line are different,
    - The numbers on each column are different,
    - The numbers on each 3x3 grid are different.



SODOKU PUZZLES

- State the problem:
  - Given a 9x9 square grid divided into 9 3x3 grids.
  - Rows and columns are numbered 0, 1, 2, ..., 8
  - List all the ways to fill in the numbers 1, 2. ..., 9 cells in a 9x9 square grid such that:
    - The numbers on each line are different,
    - The numbers on each column are different,
    - The numbers on each 3x3 grid are different.

1	2	3	4	5	6	7	8	9
4	5	6	7	8	9	1	2	3
7	8	9	1	2	3	4	5	6
2	1	4	3	6	5	8	9	7
3	6	5	8	9	7	2	1	4
8	9	7	2	1	4	3	6	5
5	3	1	6	4	2	9	7	8
6	4	2	9	7	8	5	3	1
9	7	8	5	3	1	6	4	2

SODOKU PUZZLES

- Solution representation: X[i, j] is the numeric value filled in cell row i column j (i, j = 0, 1, 2, . . ., 8)
- Marking array:
  - markR[*r*, *v*] = 1: *v* appears in row *r* and markR[*r*, *v*] = 0 for others(*r* = 0, 1, ..., 8 và *v* = 1, 2, . . ., 9)
  - markC[*c*, *v*] = 1: v appears in column c and markC[*c*, *v*] = 0, for others (*c* = 0, 1, 2, . . ., 8 và *v* = 1, 2, . . ., 9)
  - markS[*i*, *j*, *v*] = 1: *v* appear in the grid 3x3 at row *i* and column *j* ,and markS[*i*, *j*, *v*] = 0 for others (*i*, *j* = 0, 1, 2 và *v* = 1, 2, . . ., 9)

1	2	3	4	5	6	7	8	9
4	5	6	7	8	9	1	2	3
7	8	9	1	2	3	4	5	6
2	1	4	3	6	5	8	9	7
3	6	5	8	9	7	2	1	4
8	9	7	2	1	4	3	6	5
5	3	1	6	4	2	9	7	8
6	4	2	9	7	8	5	3	1
9	7	8	5	3	1	6	4	2

SODOKU PUZZLES

- Order of browsing cells to test values: top to bottom and left to right
- Recursive function Try(r, c): tries the value for cell row r column c

```
check(v, r, c){
  if markR[r,v] = 1 then return 0;
  if markC[c,v] = 1 then return 0;
  if markS[r/3,c/3,v] = 1 then return 0;
  return 1;
}
```

```
Try(r, c){
  for v = 1 to 9 do {
    if (check(v,r,c)) then {
      X[r,c] = v;
      markR[r,v] = 1; markC[c,v] = 1; markS[r/3,c/3,v] = 1;
      if r = 8 and c = 8 then solution();
      else {
        if c = 8 then Try(r+1, 0); else Try(r, c+1);
      }
      markR[r,v] = 0; markC[c,v] = 0; markS[r/3,c/3,v] = 0;
    }
  }
}
```

HUST

hust.edu.vn fb.com/dhbkhn

THANK YOU !