HUST

ĐẠI HỌC BÁCH KHOA HÀ NỘI

HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

ONE LOVE. ONE FUTURE.

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

FUNDAMENTALS OF OPTIMIZATION

Modelling

ONE LOVE. ONE FUTURE.

Outline

- Modelling a Combinatorial Optimization Problem
  - Combinatorial Optimization Problem
  - N-Queen problem
  - Sudoku problem
  - Balanced Class Teacher Assignment Problem
  - Class Allocation Problem
  - Traveling Salesman Problem
  - Exercise
- Backtracking algorithm

Combinatorial Optimization Problem

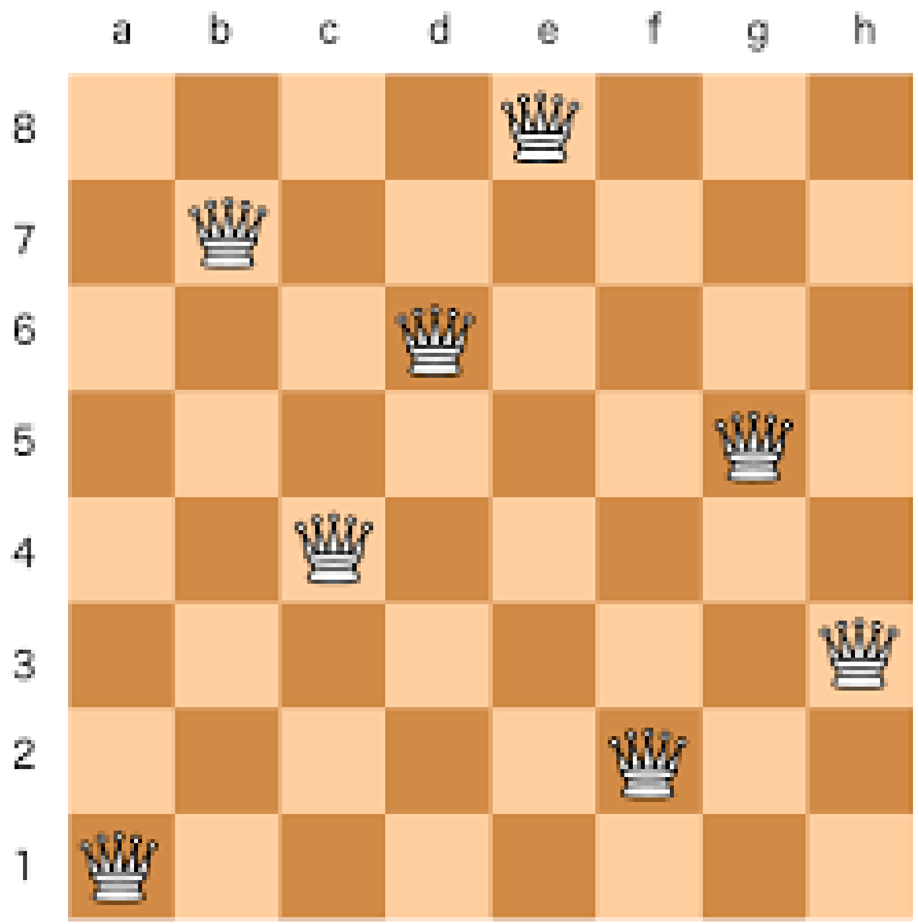
Find a solution (usually a combinatorial configuration) that satisfies a given set of constraints while simultaneously optimizing one or more specified objective functions.

- **Constraint Satisfaction Problem (CSP) = (X, D, C)**
  - $X = \{x_1, \dots, x_n\}$, set of variables
  - $D = \{D_1, \dots, D_n\}$, domains of the variables
  - $C = \{C_1, \dots, C_k\}$, set of constraints
- **Combinatorial Optimization Problem (COP) = (X, D, C, f)**
  - $X = \{x_1, \dots, x_n\}$, set of variables
  - $D = \{D_1, \dots, D_n\}$, domains of the variables
  - $C = \{C_1, \dots, C_k\}$, set of constraints
  - f: objective function

Example: Constraint Satisfaction Problem

Problem N-Queen, $CSP = (X, D, C)$

- Variables: $X = \{x_1, \dots, x_n\}$, in which x_i is the row of the queen in column $i, \forall\, i \in \{1, \dots, n\}$
- Domains: $D_i = D(x_i) = \{1, \dots, n\}, \forall i \in \{1, \dots, n\}$
- Constraints: For all pair (i, j), $1 \le i < j \le n$:
  - $x_i \neq x_j$
  - $x_i + i \neq x_j + j$
  - $x_i - i \neq x_j - j$



Example: Constraint Satisfaction Problem

Sudoku Problem, CSP = (X, D, C)

- **Variables**: $X = \{x_{1,1}, \dots, x_{9,9}\}$, where $x_{i,j}$ is the value in cell (i, j), $\forall\ i, j \in \{1,2, \dots, 9\}$
- **Domain**: $D\left(x_{i,j}\right) = \{1, \dots, 9\}, \forall i, j \in \{1,2, \dots, 9\}$
- **Constraints**:
  - The numbers in each column are pairwise distinct:
  $x_{i_1 j} \neq x_{i_2 j}$ for all $1 \leq i_1 < i_2 \leq 9$, $1 \leq j \leq 9$
  - The numbers in each row are pairwise distinct:
  $x_{j i_1} \neq x_{j i_2}$ for all $1 \leq i_1 < i_2 \leq 9$, $1 \leq j \leq 9$
  - The numbers in each 3x3 subgrid are pairwise distinct:
  $x_{3i+i_1,3j+j_1} \neq x_{3i+i_2,3j+j_2}$ for all $0 \leq i, j \leq 2$, $1 \leq i_1, i_2, j_1, j_2 \leq$ satisfying $(i_1, j_1) \neq (i_2, j_2)$

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

Balanced Class Teacher Assignment

- There are n classes labeled $1, 2, \ldots, n$ that have already been scheduled in a timetable, and they need to be assigned to m teachers labeled $1,2, \ldots, m$
- Each class i has $T(i)$, a list of teachers who can teach it ($i = \{1, \ldots, n\}$), and $c(i)$, the number of credits for the subject of that class.
- Since the timetable has been pre-arranged, there exists a set Q of pairs of classes (i, j) that are scheduled at the same time (these two classes cannot be assigned to the same teacher).
- Find an assignment of classes to teachers such that the maximum total number of credits assigned to any one teacher is minimized.

Balanced Class Teacher Assignment

- Example

Class	0	1	2	3	4	5	6	7	8	9	10	11	12
Credit	3	3	4	3	4	3	3	3	4	3	3	4	4

Teacher	List of classes that the teach can teach
0	0, 2, 3, 4, 8, 10
1	0, 1, 3, 5, 6, 7, 8
2	1, 2, 3, 7, 9, 11, 12

List Q

0	2
0	4
0	8
1	4
1	10
3	7
3	9
5	11
5	12
6	8
6	12

9

Balanced Class Teacher Assignment

- Example

Class	0	1	2	3	4	5	6	7	8	9	10	11	12
Credit	3	3	4	3	4	3	3	3	4	3	3	4	4

Teacher	List of classes that the teach can teach
0	0, 2, 3, 4, 8, 10
1	0, 1, 3, 5, 6, 7, 8
2	1, 2, 3, 7, 9, 11, 12

List Q

0	2
0	4
0	8
1	4
1	10
3	7
3	9
5	11
5	12
6	8
6	12

Assignment solution

Teacher	List of classes assigned to the teacher	Total credits
0	2, 4, 8, 10	15
1	0, 1, 3, 5, 6	15
2	7, 9, 11, 12	14

10

Balanced Class Teacher Assignment

- Input:
  - Set of classes: $S = \{1, \dots, n\}$
  - Set of teachers: $T = \{1, \dots, m\}$
  - Set of conflict classes: $Q = \{(i_1, j_1), \dots, (i_k, j_k) | i_1, \dots, i_k, j_1, \dots, j_k \in C, i_t \neq j_t \ \forall t \in \{1, \dots, k\}\}$
  - Set of teachers who can give some class $T = \{T_1, \dots, T_n\}$, in which T_i is the set of teachers who can give the class i ($i \in \{1, \dots, n\}$)
  - For each class i ($i \in \{1, \dots, n\}), c(i)$ is its number of credit ($c(i) \in N$)
- Variables:
  - Binary variable x_{ij} ($i \in C, j \in T$) is equal to 1 if the class i is assigned to the teacher j, otherwise the value of this variable is equal to 0;
  - Integral variable $maxcredit$ represents the maximum number of credits for a teacher;
- Domains of variables:
  - $D(x_{ij}) = \{0, 1\}, \forall\, i \in C, j \in T$
  - $D(maxcredit) = \{0, \dots, \sum_{i \in C} c(i)\}$
- Constraints:
  - Each class is assigned to one teacher $\sum_{j \in T_i} x_{ij} = 1, \ \forall\, i \in C$
  - Teacher is not assigned to a class that he cannot teach $x_{ij} = 0, \forall\, i \in C, j \notin T_i$
  - Teacher cannot give two classes in the conflict set $x_{i_1 j} + x_{i_2 j} \leq 1, \forall\, j \in T, (i_1, i_2) \in Q$
  - Relation between variable $maxcredit$ and workload of teacher $\sum_{i \in C} c(i) x_{ij} \leq maxcredit\, , \forall\, j \in T$
- Objective: $Minimize\ maxcredit$

Class Allocation Problem

- n classes labeled by $1, 2, \ldots, n$ need to be allocated in p semesters $1, 2, \ldots p$. Each class i has a credit value of $c(i)$, and its prerequisite conditions are defined by a set Q of pairs (i, j), where subject i must be taken before j. Given the constant $\alpha, \beta, \delta, \gamma$, it is necessary to determine an allocation plan that satisfies the following:
  - The total number of classes assigned to each semester must be greater than or equal to α and less than or equal to β.
  - The total number of credits of the classes assigned to each semester must be greater than or equal to δ and less than or equal to γ
  - For each pair $(i, j) \in Q$, class i must be scheduled in a semester prior to the semester in which class j is scheduled.
- Objective: The maximum number of credits in any one semester must be minimized.

Class Allocation Problem

- Example

Class	1	2	3	4	5	6	7	8	9	10	11	12
Number of credits	2	1	2	1	3	2	1	3	2	3	1	3

3 ≤ Number of classes in each semester ≤ 3
5 ≤ Number of credits in each semester ≤ 7

Set Q

2	1
6	9
5	6
5	8
4	11
6	12
2	7
3	10
5	7
8	11
4	12

Class Allocation Problem

- Example

Class	1	2	3	4	5	6	7	8	9	10	11	12
Number of credits	2	1	2	1	3	2	1	3	2	3	1	3

Set Q

2	1
6	9
5	6
5	8
4	11
6	12
2	7
3	10
5	7
8	11
4	12

$3 \leq$ Number of classes in each semester ≤ 3

$5 \leq$ Number of credits in each semester ≤ 7

Allocation solution

Semester	1	2	3	4
List of classes	2, 5, 3	1, 6,10	4,7,8	9,11,12

Class Allocation Problem

- Input:
  - Set of classes: $C = \{1, \dots, n\}$
  - Set of semesters: $S = \{1, \dots, p\}$
  - Set of precedent classes: $Q = \{(i_1, j_1), \dots, (i_k, j_k) | i_1, \dots, i_k, j_1, \dots, j_k \in C, i_t \neq j_t \; \forall t \in \{1, \dots, k\}\}$
  - Constant $\alpha, \beta, \delta, \gamma$
- Variables:
  - Binary variable x_{ij} ($i \in C, j \in S$) is equal to 1 if the class i is assigned to the semester j, otherwise the value of this variable is equal to 0;
  - Integral variable $maxcredit$ represents the maximum number of credits for a semester
- Constraints:
  - Every class is allocated to some semester $\sum_{j \in S} x_{ij} = 1, \forall i \in C$
  - Number of classes in a semester must be in a range of $[\alpha, \beta]$, it means that $\alpha \leq \sum_{i \in C} x_{ij} \leq \beta, \forall j \in S$
  - Number of credits in a semester must be in a range of $[\delta, \gamma]$, it means that $\delta \leq \sum_{i \in C} c(i) x_{ij} \leq \gamma$
  - For each pair $(i_1, i_2) \in Q$, class i_1 must be scheduled in a semester prior to the semester in which class i_2 is scheduled $\sum_{j \in S} j x_{i_1 j} < \sum_{j \in S} j \, x_{i_2 j}, \forall (i_1, i_2) \in Q$
  - Relation between variable $maxcredit$ and workload in a semester $\sum_{i \in C} c(i) x_{ij} \leq maxcredit, \forall j \in S$
- Objective: $Minimize \; maxcredit$

Traveling Salesman Problem

- A traveler starts from city 1 and needs to visit cities $2, 3, \ldots, n$, passing through each city exactly once before returning to the starting city. The cost of traveling from city i to city j is $c(i, j)$. Calculate the plan for the traveler that results in the minimum total cost.

Traveling Salesman Problem

- Input:
  - n number of cities
  - $c(i,j)$ traveling cost from the city i to the city j
- Variables:
  - Binary variable x_{ij} with $i,j \in \{1,\dots,n\}, i \neq j$ is equal to 1 if the traveler go from the city i to the city j in the optimal plan, otherwise the value of this variable is equal to 0;
- Constraints:
  - For each city, the traveler goes in once and goes out once

$$\sum_{j \in \{1,\dots,n\}} x_{ij} = \sum_{j \in \{1,\dots,n\}} x_{ji} = 1, \forall i \in \{1,\dots,n\}$$

  - No subtour

$$\sum_{i,j \in S, i \neq j} x_{ij} \leq |S| - 1, \forall S \subset \{1,\dots,n\}$$

- Objective: $Minimize \ \sum_{i,j \in \{1,\dots,n\}, i \neq j} c(i,j) x_{ij}$

Exercises

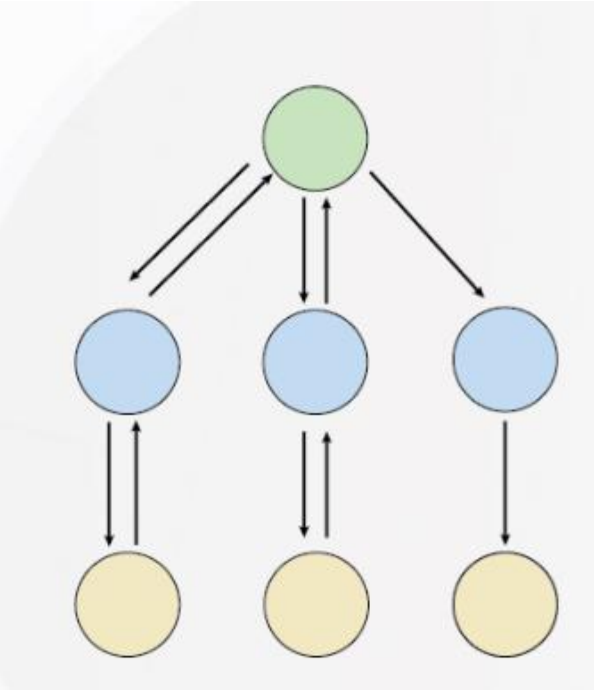
- Modelling the problem in your mini-project

Outline

- Modelling a Combinatorial Optimization Problem
  - Combinatorial Optimization Problem
  - N-Queen problem
  - Sudoku problem
  - Balanced Class Teacher Assignment Problem
  - Class Allocation Problem
  - Traveling Salesman Problem
  - Exercise
- **Backtracking algorithm**

Backtracking Algorithm

- Backtracking is a problem-solving algorithmic technique that involves finding a solution incrementally by trying **different options** and **undoing** them if they lead to a **dead end**. It is commonly used in situations where you need to explore multiple possibilities to solve a problem, like searching for a path in a maze or solving puzzles like Sudoku. When a dead end is reached, the algorithm backtracks to the previous decision point and explores a different path until a solution is found or all possibilities have been exhausted.

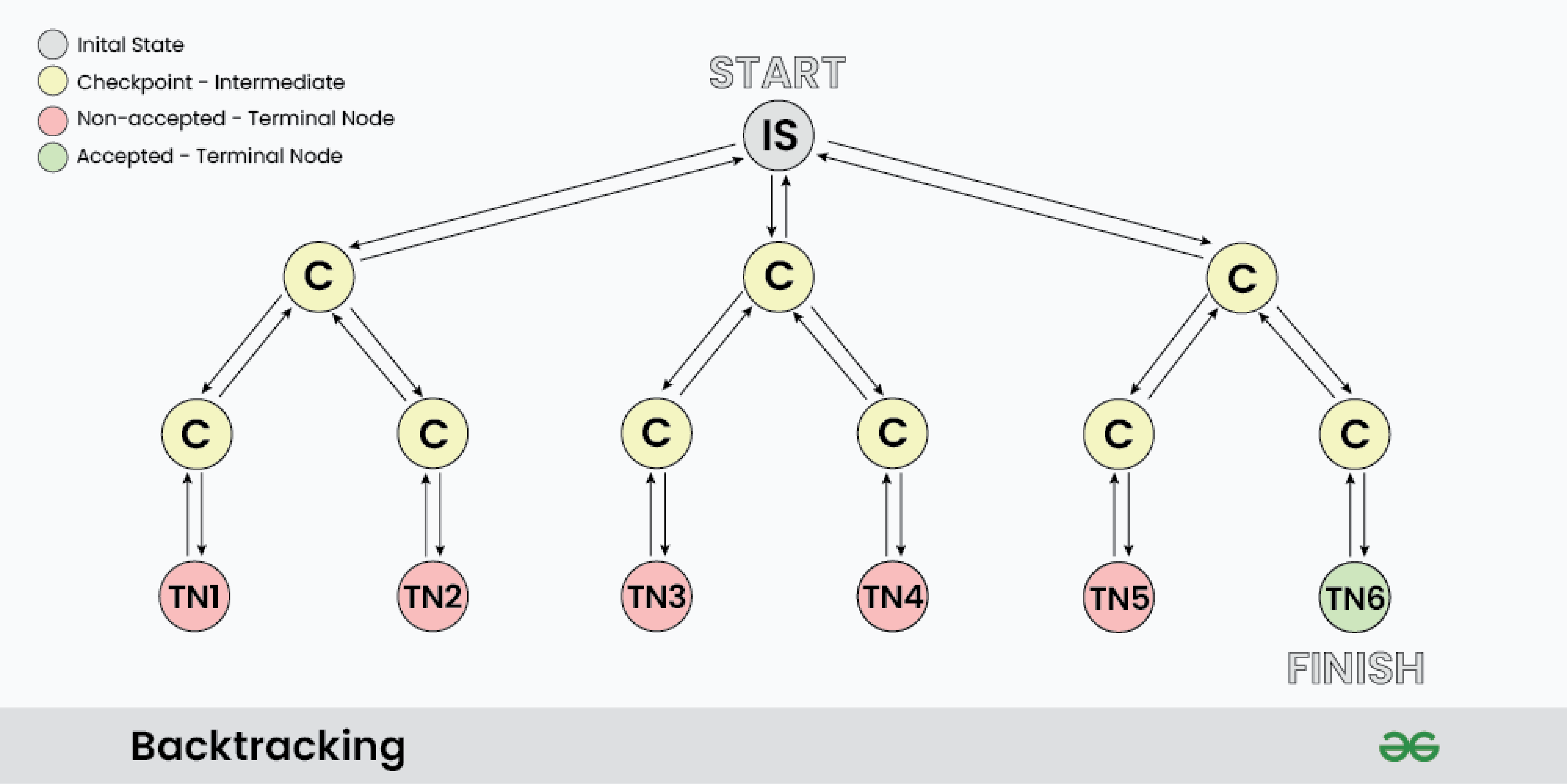


- Backtracking can be defined as a general algorithmic technique that considers searching every possible combination in order to solve a computational problem.

Backtracking Algorithm

- **Candidate**: A candidate is a potential choice or element that can be added to the current solution.
- **Solution**: The solution is a valid and complete configuration that satisfies all problem constraints.
- **Partial Solution**: A partial solution is an intermediate or incomplete configuration being constructed during the backtracking process.
- **Decision Space**: The decision space is the set of all possible candidates or choices at each decision point.
- **Decision Point**: A decision point is a specific step in the algorithm where a candidate is chosen and added to the partial solution.
- **Feasible Solution**: A feasible solution is a partial or complete solution that adheres to all constraints.
- **Dead End**: A dead end occurs when a partial solution cannot be extended without violating constraints.
- **Backtrack**: Backtracking involves undoing previous decisions and returning to a prior decision point.
- **Search Space**: The search space includes all possible combinations of candidates and choices.
- **Optimal Solution**: In optimization problems, the optimal solution is the best possible solution.

Backtracking Algorithm



Recursive Technique for Backtracking Algorithm

```
TRY(k)
  Begin
    Foreach v in A_k
      if check(v,k) /* Check for feasibility of assigning v to x_k  */
        Begin
          x_k = v;
          [Update data structure D]
          if(k = n) save a feasible solution;
          else TRY(k+1);
          [Recovery D]
        End
  End
Main()
Begin
  TRY(1);
End
```

Generate binary string

```python
n = 3
x = [-1] * n
def Try(k):
    if k == n:
        print(x)
    else:
        for i in range(2):
            x[k] = i
            Try(k+1)

Try(0)
```

```
[0, 0, 0]
[0, 0, 1]
[0, 1, 0]
[0, 1, 1]
[1, 0, 0]
[1, 0, 1]
[1, 1, 0]
[1, 1, 1]
```

Generate permutation of a set

```python
n = 3
x = [-1] * n
visited = [False] * (n)

def Try(k):
    if k == n:
        print(x)
        return

    for i in range(0, n):
        if visited[i] == True:
            continue

        x[k] = i
        visited[i] = True
        Try(k+1)
        visited[i] = False

Try(0)
```

```
[0, 1, 2]
[0, 2, 1]
[1, 0, 2]
[1, 2, 0]
[2, 0, 1]
[2, 1, 0]
```

HUST

hust.edu.vn fb.com/dhbkhn

THANK YOU !