BÀI TẬP THỰC HÀNH

NHẬP MÔN TRÍ TUỆ NHÂN TẠO

TUẦN 2 – BÀI TOÁN *N* HẬU

GIẢI QUYẾT BẰNG THUẬT TOÁN DFS, BFS

Mô tả bài toán: bài toán *N* hậu là bài toán đặt *N* quân hậu lên bàn cờ *NxN* sao cho không có bất kỳ một xung đột nào giữa các quân hậu.

Input: một số nguyên *N* từ 4 đến 8

Output: vị trí đặt *N* quân hậu trên bàn cờ có kích thước *NxN*

Yêu cầu: giải bài toán bằng các thuật toán tìm kiếm BFS, DFS

Hướng dẫn:

> Cấu trúc dữ liệu sử dụng: ArrayList. Lý do: mỗi quân hậu phải được đặt trên một column riêng biệt thì mới tránh được sự xung đột cơ bản, do đó chỉ cần một danh sách ghi lại vị trí của quân hậu trên column tương ứng, không cần phải dùng mảng 2 chiều. Ví dụ:
> Input: 4
> Output: 1,3,0,2 tương ứng với hình vẽ sau:

		Q	
Q			
			Q
	Q		

Bước 1: Xây dựng cấu trúc dữ liệu Node

1. Tạo class Node với các thuộc tính:

```
Int n;
List<Integer> state;
List<Node> neighbours;
```

Trong đó n là số quân hậu cần được đặt, state là vị trí các quân hậu hiện tại trên bàn cờ, neighbours là danh sách các Node con của node đang xét

2. Viết hai constructor cho Node

```
public Node(int n)
{
```

```
        this.n=n;
        this.state=new ArrayList<>();
        this.neighbours=new ArrayList<>();

    }

    public Node(int n,List<Integer> state)
    {
        this.n=n;
        this.state=state;
        this.neighbours=new ArrayList<>();

    }
```

3. Viết hàm addNeighbours

```
Public void addNeighbours(Node neighbourNode)
{
        this.neighbours.add(neighbourNode);
}
```

4. Viết hàm isValid(List<Integer> state) để kiểm tra một state có hợp lệ hay không biết rằng chỉ có column cuối cùng trong state là chưa được kiểm tra, còn tất cả các column trước đều đã hợp lệ
   Gợi ý:
   + Nếu state.size()==1 đương nhiên hợp lệ (vì chỉ mới có 1 quân hậu trên bàn cờ, muốn đặt đâu cũng được)
   + Khi state.size()>1, chỉ cần kiểm tra xung đột của quân hậu cuối cùng với các quân trước nó
   + Kiểm tra xung đột theo hàng, theo đường chéo, không cần theo cột
5. Viết hàm `private List<Integer> place(int x)` để kiểm tra có thể đặt con hậu mới vào vị trí x của cột mới hay không
   Gợi ý:
   + Tạo một List<Integer> sao chép state hiện tại của Node
   + Thử thêm con hậu mới vào vị trí x của cột mới
   + Dùng hàm isValid() để kiểm tra xung đột của state mới, nếu không có xung đột thì trả về state mới, nếu có xung đột thì trả về null
6. Viết hàm `public List<Node>getNeighbours()` trả về danh sách các Node con của Node đang xét
   Gợi ý:
   + Nếu state.size()==n trảvề null
   + Xét tất cả các dòng x trongcột mới, nếu có thể thêm vào dòng nào mà không gây xung đột thì tạo một Node mới chứa state mới, add Node mới này vào danh sách neighbours của Node cũ

Bước 2:

Sử dụng lại DFS và BFS đã cài đặt ở tuần 1, chỉ sửa lại ở vị trí kiểm tra goal và cách in ra goal state

Bước 3: Xây dựng class Queens như sau

```
Public class Queens {
    Private int n;
```

```
        private Node goal;
        public Queens (int n)
        {
                this.n=n;
        }
        Public void dfs()
        {
                DFS dfs=new DFS();
                this.goal=dfs.dfsUsingStack(new Node(n), n);
        }
        Public void bfs()
        {
                BFS bfs=new BFS();
                this.goal=bfs.bfsUsingQueue(new Node(n), n);
        }
}
```

Bước 4:Viết hàm main trong class Main

```
Public static void main (String[] args)
{
        Queens q=new Queens(8);
        q.dfs();
        q.bfs();
}
```