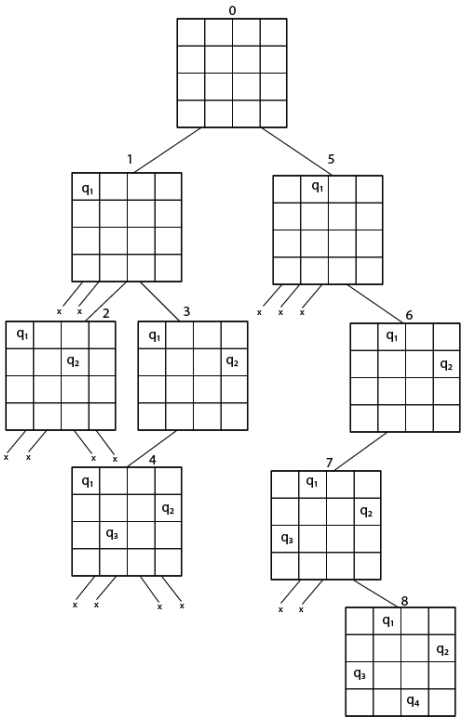
N бэрсний бодлого

Энэ бодлого нь шатрын n тооны бэрсүүдийг бие биенээ идэхгүй байхаар N x N хэмжээтэй шатрын хөлөгт байрлуулах юм. NQueens-ийн асуудлыг модоор хайлтын аргаар шийдвэрлэх жишээг авч үзье. Өргөнөөр хайх, гүнээр хайх гэсэн 2 функц дуудаж ажлуулна. Node, NQueensproblem гэсэн 2 классаас функуудаа дуудаж ажлуулна. Breadth функц нь deque өгөгдлийн төрөлтэй бол depth функц нь stack өгөгдлийн төрөлтэй. Хайлтын функуудаа модыг дүүртэл нь давтаж ажиллана. Breadth хайлт level level-ээр нь буюу эхний элемэнтээс зүүнээс нь нэмдэг бол Depth хайлт нь гүнээс нь буюу сүүлийн элемэнтээс баруун талаас нь нэмнэ.




```
class Problem:

    def __init__(self, initial, goal=None):
```

```
        self.initial = initial
        self.goal = goal

    def actions(self, state):
        raise NotImplementedError

    def result(self, state, action):
        raise NotImplementedError

    def goal_test(self, state):
        if isinstance(self.goal, list):
            return is_in(state, self.goal)
        else:
            return state == self.goal

    def path_cost(self, c, state1, action, state2):
        return c + 1

    def value(self, state):
        raise NotImplementedError



class Node:
    def __init__(self, state, parent=None, action=None, path_cost=0):
        self.state = state
        self.parent = parent
        self.action = action
        self.path_cost = path_cost
        self.depth = 0
        if parent:
            self.depth = parent.depth + 1

    def __repr__(self):
```

```
        return "<Node {}>".format(self.state)

    def __lt__(self, node):
        return self.state < node.state

    def expand(self, problem):
        return [self.child_node(problem, action)
                for action in problem.actions(self.state)]

    def child_node(self, problem, action):
        next_state = problem.result(self.state, action)
        next_node = Node(next_state, self, action, problem.path_cost(self.path_cost,
self.state, action, next_state))
        return next_node

    def solution(self):
       return [node.action for node in self.path()[1:]]

    def path(self):
        node, path_back = self, []
        while node:
            path_back.append(node)
            node = node.parent
        return list(reversed(path_back))


    def __eq__(self, other):
        return isinstance(other, Node) and self.state == other.state

    def __hash__(self):
        return hash(self.state)
```

```
import sys
from collections import deque
class NQueensProblem(Problem):
    #NxN хөлөг дээр нэг ч бие биедээ халдаагүй N бэрсийг
    #байрлуулах  асуудал. Төлөв нь N элементийн массив
    #хэлбэрээр илэрхийлэгддэг бөгөөд с-р оролтын r утга нь
    #багана с-д мөр r-д хатан байна гэсэн үг. -1 утга нь с-р
    #баганад ороогүй гэсэн үг юм.

    def __init__(self, N):
        super().__init__(tuple([-1] * N))
        self.N = N

    def actions(self, state):
        #Хамгийн зүүн талын хоосон баганад бүх зөрчилдөхгүй
        #мөрүүдийг туршиж үзээрэй
        if state[-1] != -1:
            return []  # Бүх баганууд дүүрсэн; эсвэл бэрсний тоо орж ирээгүй
        else:
            col = state.index(-1)
            return [row for row in range(self.N)
                    if not self.conflicted(state, row, col)]

    def result(self, state, row):
        #Дараагийн бэрсийг өгөгдсөн мөрөнд байрлуул.
        col = state.index(-1)
```

```
    new = list(state[:])
    new[col] = row
    return tuple(new)

def conflicted(self, state, row, col):
    #Бэрсийг (мөр, багана) байрлуулах нь ямар нэгэн
    #зүйлтэй зөрчилдөх үү?
    return any(self.conflict(row, col, state[c], c)
               for c in range(col))

def conflict(self, row1, col1, row2, col2):
    #Хоёр бэрсийг (row1, col1) болон (row2, col2)-д
    #байрлуулахад зөрчилдөх эсэхийг шалгана.
    return (row1 == row2 or  # ижил row
            col1 == col2 or  # ижил column
            row1 - col1 == row2 - col2 or  # ижил\ diagonal
            row1 + col1 == row2 + col2)  # ижил / diagonal

def goal_test(self, state):
    #Бүх багана дүүрсэн эсэхийг шалгана уу, зөрчилгүй
    if state[-1] == -1:
        return False
    return not any(self.conflicted(state, state[col], col)
                   for col in range(len(state)))

def h(self, node):
```

```
        #Өгөгдсөн зангилааны зөрчилтэй бэрсний тоог буцаах
        num_conflicts = 0
        for (r1, c1) in enumerate(node.state):
            for (r2, c2) in enumerate(node.state):
                if (r1, c1) != (r2, c2):
                    num_conflicts += self.conflict(r1, c1, r2, c2)

        return num_conflicts

def breadth_first_tree_search(problem):
    frontier = deque([Node(problem.initial)])  # FIFO queue

    while frontier:
        node = frontier.popleft()
        if problem.goal_test(node.state):
            return node
        frontier.extend(node.expand(problem))
    return None

def depth_first_tree_search(problem):
    #Эхлээд хайлтын модны хамгийн гүн цэгүүдийг хайж олох.
     #Зорилгоо олохын тулд асуудлын залгамж зангилааг
     #хайж олно.
    frontier = [Node(problem.initial)]  # Stack
    while frontier:
        node = frontier.pop()
```

```
        if problem.goal_test(node.state):
            return node
        frontier.extend(node.expand(problem))
    return None
```

Дээрх кодыг BFS хайлтаар хэрэгжүүлвэл дараах байдлаар ажлуулна. Жишээ нь 6 бэрсийг 6x6 хэмжээтэй хөлөгт байрлуулахад дараах байрлалуудад бэрсийг бие биенээ идэхгүй байхаар байрлуулж болно. Үүнд: (1,0), (3,1), (5,2), (0,3), (2,4), (4,5). Мөн гүнээр хайлтыг хэрэгжүүлж үзвэл (4,0), (2,1), (0,2), (5,3), (3,4), (1,5) гэсэн байрлалуудад бэрсийг бие биенээ идэхгүй байхаар байрлуулж болж байна. Бэрсийн бодлогын хувьд гүнээр хайх арга нь хамгийн тохиромжтой бөгөөд хугацаа олон орон зай бага шаардана.

```
gnode = breadth_first_tree_search(NQueensProblem(6))
print(gnode)
<Node (1, 3, 5, 0, 2, 4)>
print(Node.path(gnode))
[<Node (-1, -1, -1, -1, -1, -1)>, <Node (1, -1, -1, -1, -1, -1)>,
<Node (1, 3, -1, -1, -1, -1)>, <Node (1, 3, 5, -1, -1, -1)>, <Node
(1, 3, 5, 0, -1, -1)>, <Node (1, 3, 5, 0, 2, -1)>, <Node (1, 3, 5,
0, 2, 4)>]

gnode = breadth_first_tree_search(NQueensProblem(6))
print(gnode)
<Node (4, 2, 0, 5, 3, 1)>
```