n-queens

① define board with 0, i=0, j=q-1

② if (arr[i][j] == 0){
    arr[i][j] = q;
    set row, column & diagonals for (q) to 1
    q++;}
else i++;
if (i == n && j = q-1){
    set row, column & diagonals for (q-1) to 0
    replace (q-1) with -1
    q--;}

③ print board repeat ② while (q != n+1)

magic sq

① define init pos of n=1 at
i = 0, j = N//2

② repeat while (n < n^2) {
    arr[i][j] = n;
    n+1;
    nexti = (i-1) % N;
    nextj = (j+1) % N;
    if (arr[ni][nj] == 0){
        i+ = 1;}
    else
        i, j = nexti, nextj
    }

③ Print arr[][]

BFS

(1) Start with root node
(2) repeat {while (child_node == UNIQUE)
generate child-node of parent
if (child_node == GOAL) {GOAL!;}
else
update open & closed list
(3) Backtrack to parent node of current-
~~left~~ leaf node
find next - child-node of node and
goto (2)

DFS (1) START with root node
(2) apply all possible moves to generate child-node
if (child_node == GOAL) {GOAL!!;}
else update open & close list
(3) find leftmost node & generate child-node
if (chil- - = GOA) — - -
else
update list
backtrack to root node find next LM node

conditions

② FUNCTION search (initial){
open.add(initial)
while (open != EMPTY){
if (open.top == GOAL){
GOAL!}
else
{
find_successor()of node
for all successors of node → {
find best successor based
on cost + heuristic
update openlist
}
③ print path

Hill Climb

① define root node as init state
② repeat while (node(h) != 0 || local_maxima
{
~~c = 0~~; apply all moves to gen.
~~if (node(h)~~ child node
get heuristic
for all child nodes →
c = 0
if (child_node(h) < parent(h){
c = 1;
parent = child;}
if (child_node(h) == 0){
GOAL;}
if (c == 0){L.M;}

DATE : / /

MINIMAX

(1) FUNCTION miniman (board, player) {
if (terminal (board) {
return utility (board, player)
available_moves = get_moves (board)
for all available_moves {
play move
miniman () opponent, store score
undo move
if player = curr_player
return (move, score) highest
else
return (move, score) lowest

MAP COLORING

① Define a graph, add map elements as vertex and link all neighbors

② while (OPEN[] != EMPTY){
    generate all neighbors of node
    remove used colors
    assign available color for i

③ if (OPEN[] == EMPTY){
    GOTO ④}
else { COLOR NOT POSSIBLE ;}

④ print color sequence.

CROSSWORD PUZZLE

```
function solve(board, across_words, down_words)
{
  for each row in board{
     len = len of longest whitespace

     for eword in across{
       if len(word) == {
         board.replace(longest whitespace
}  }  }

     for each column in board {
       len = length of longest whitespace

       for word in down_word{
         if (len(word) == len{
           board.replace(longest
```

Forwarded

Crossword Puzzle

```
function solve(board, across_words,
down_words) {
    for each row in board {
        len = length of longest whitespace
in that row

        for word in across_words {
            if len(word) == len{
                board.replace(longest
whitespace, word)
            }
        }
    }

    for each column in board {
        len = length of longest whitespace
in that column

        for word in down_words {
            if len(word) == len{
                board.replace(longest
whitespace, word)
            }
        }
    }
}
```

10:26 am

Forwarded

```
crypt(int map){
    bool mapped = False;
    if(map == len(uniqueChars)){
        if(satisfiyConstraint(Mapping))
            return sol(Mapping);
    }

    for(i=0;i<10;i++){
        if i not used:{
            Mapping[map] = i;
            used add i;
            crypt(map+1);
            mapped = True;
        }
    }
    if(not mapped){
        used remove Mapping[map-1];
        Mapping[map-1] = -1;
        crypt(map-1);
    }
}
```

10:26 am