Computer Science

Dr. Y. Liow (August 26, 2022)

Contents

110 Backtracking search **4000**
110.1 Backtrack 4001
110.2 Pseudocode: recursion 4005
110.3 Comparing backtrack search and depth-first search 4007
110.4 The knight's tour problem 4008
110.5 The knight's tour solution: recursive backtrack 4012
110.6 Pseudocode: no recursion 4048
110.6.1 Goal test 4051
110.6.2 Next sibling 4051
110.6.3 Difference between backtrack and depth-first search 4053
110.7 Knight's tour solution: non-recursive backtrack 4054
110.8 Knight's tour variants 4055
110.9 n-queens problem 4056
110.10 Magic square problem 4057
110.11 Sudoku problem 4058

Chapter 110

Backtracking search

File: chap.tex

110.1 Backtrack

TODO: cleanup

A search algorithm usually builds a solution in steps and has to try options at each step before finding a goal. For each option, the algorithm might need to try more options. Etc. There are times when a bad choice was made and your algorithm is stuck. In that case, the algorithm has to undo a choice made earlier. Therefore you can think of a complete solution as being made up of partial solutions. There are times when you made a wrong choice. In that case, the partial solution might be wrong and has to be corrected. That's why such a partial solution is called a partial candidate solution.

For instance say from where you are, want to get your copy of Lord of the Ring, you need to "get up, walk right 3 feet, turn left 90 degrees, stretch arm out by 2 feet, lower arm 1 foot, grasp book with hand.". Then the following are what I meant by partial solutions:

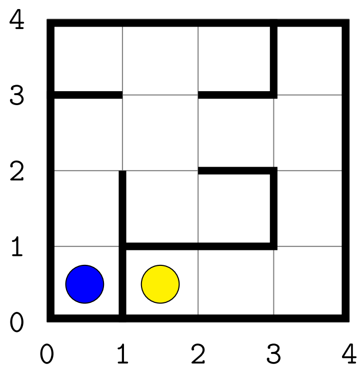
1. Do nothing
2. Get up.
3. Get up, walk right 3 feet.
4. Get up, walk right 3 feet, turn left 90 degrees.
5. etc.

There are times when you have several options, some of which will be wrong and some are right (maybe more than one). Sometimes the fact that one option is wrong will only be known after making several options. Sometimes there are more than one way to solve the problem, i.e., more than one sequence of options can solve the problem.

But .... the important thing is that the solution is made up of a sequence of choices.

Of course there must be a way to check that a sequence of choices is complete – at that point you have a solution. If the solution is not complete and at that point there are no options, then you need to go backward and unto a choice. That's called **backtracking**. backtracking

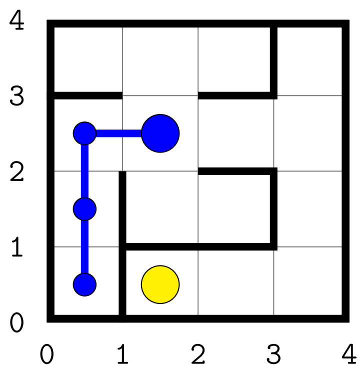
Let me give you another example, one that's more graphical. Say you're in a maze starting at the bottom left (the blue dot) and you are to look for the gold (the yellow dot):



The solution is your footsteps in the maze leading up to the gold (the yellow dot). So at this point the partial candidate solution is just

$$(0.5, 0.5)$$

Suppose you have reached this point :

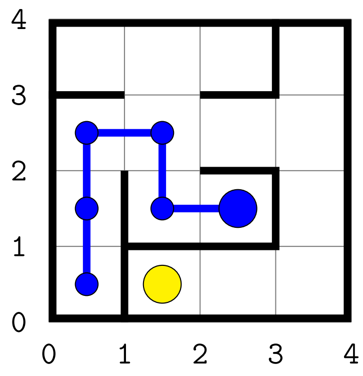


(The blue line represents your search path for the gold, or rather the dots on the blue line forms you search path.) Now you have three choices for going forward.

The candidate solution up to this point is made up of the smaller blue dots and the large blue dot (your footsteps):

$$(0.5, 0.5), (0.5, 1.5), (0.5, 2.5), (1.5, 2.5)$$

At this point you can go north, south or east. If you pick south, you will ultimately end up here:

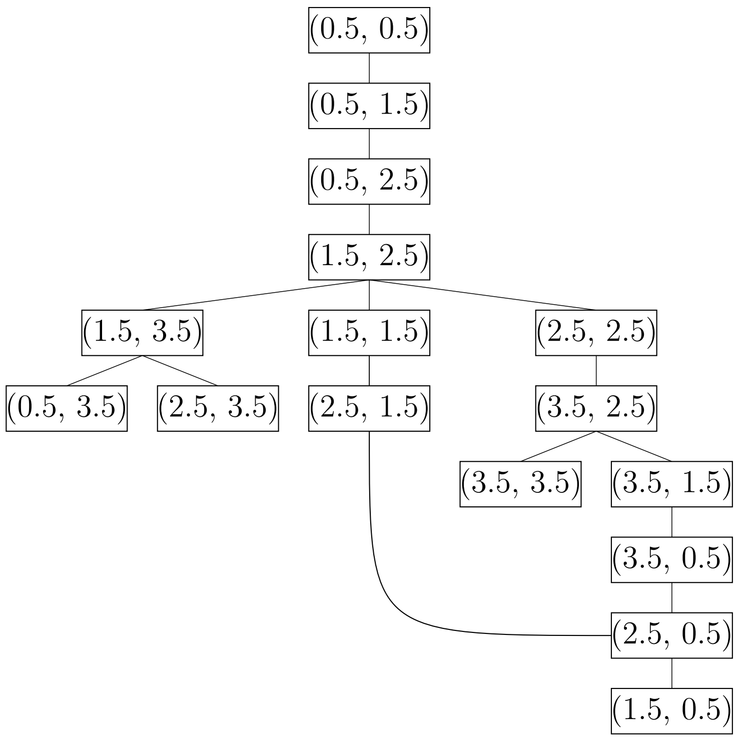


with this candidate solution:

$$(0.5), (0.5), (0.5, 1.5), (0.5, 2.5), (1.5, 2.5), (1.5, 1.5), (2.5, 1.5)$$

and you're stuck. You would have to go back twice and try other options..

You should realize that all the available steps in the maze forms a tree in the obvious way:



However, and this is the important point, backtrack search only create a single path in the tree during the computation of a search from your starting place to the gold.

You should recognize the function calls as forming a *depth-first search* of a solution.

Note that at each point in the maze, you should be able to query all the valid "directions" you can move in. But note that in solving the above maze, you clearly need to distinguish between "options for moving forward" and "the option that moves you backward". The moving backward is backtracking and is handled by returning a failure in the recursive function call. Therefore the backtrack search must select valid "forward" options.

110.2 Pseudocode: recursion

Here's sort of a template (i.e. general idea) of backtracking search:

```
backtrack_search(solution):
    if solution is complete:
        return true
    else:
        for each option at this point:
            flag = backtrack_search(solution + option)
            if flag: return true
        return false
```

Here `solution` means a partial candidate solution. By `solution + option`, I mean adding `option` to `solution` to get a new partial solution.

As you can see, it involves recursion since `backtrack_search` calls itself.

In the above, the function returns true if a solution can be found. If the parameter passing is by reference, then if the return value is true, `solution` will be the solution.

Another thing to note is that in the above pseudocode, it's assumed that we construct `solution + option1`. If the return value is false, we try `solution + option2`, etc. Now, the construction of `solution + option1`, `solution + option2`, etc., could be costly. Frequently, a single extensible object `solution` is used. (For instance an array.) In that case, extend `solution` by "adding" `option` to it and if the return flag is false, we have to "remove" `option` from `solution`. This prevents the creation of too many objects. So here's the new pseudocode:

```
backtrack_search(solution):
    if solution is complete:
        return true
    else:
        for each option at this point:
            extend solution by adding option
            flag = backtrack_search(solution)
            if flag: return true
            remove option from solution
        return false
```

The above gives the general template of a backtrack search. I don't mean to say that, for instance, the else part must have a for-loop. It can be a while-loop instead:

```
backtrack_search(solution):
    if solution is complete:
        return true
    else:
        let x be the first option at this point
        while x is a valid option:
            extend solution by adding x
            flag = backtrack_search(solution)
            if flag: return true
            remove x from solution
            let x = next option at this point
        return false
```

110.3 Comparing backtrack search and depth-first search

Notice that backtracking is very similar to DFS. The only difference is that in the case of DFS, all children of a computational state are generated and placed in a structure (stack) whereas for backtracking, only one child is examined. Therefore if the tree of function call has a depth of d and a branching factor of b, DFS will hold at worse $1 + bd$ computational states. On the other hand, backtracking will have d computational states kept in the function call stack.

110.4 The knight's tour problem

knight's tour

The following is a very famous problem called the **knight's tour**. You're given a 3×4 chessboard and you're told to put a knight at one square of your choice and then make the knight move from one square to another until all the squares in the chessboard are touched by your knight exactly once. I start here:

The length of the solution is 0. At this point I have 12 options. I start with this as my first option:

1			

Of course this is only a partial candidate, not a complete solution. If at some point, the candidate solution that starts at $(0, 0)$ cannot be extended, I will have to try the next square, i.e., start at $(0, 1)$.

With the above, I move on to extension my solution by one more step. I have two choices:

1			
		?	
	?		

Say I pick this:

1			
		2	

At this point, you can and should attempt to solve this problem on your own. It shouldn't take you more than 20 minutes. If you do make a mistake, just undo your choices until you have an valid option – i.e., you just backtrack. Once your knight has travelled the whole 3×4 board, we say that you have a **knight's tour** of the board. Keep track of all your options and all your

knight's tour

backtracks (if any).

Turn the page only when you have completed your knight's tour.

SOLUTION.

Here's the solution:

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

□

Now suppose that you *have* to start with this:

1			
	2		

Is it possible to complete the above to a complete knight's tour? Note all the options and all the backtracks before turning the page.

SOLUTION.

It turns out that it you have to start with this:

1			
	2		

then you *cannot* have a complete knight's tour. □

110.5 The knight's tour solution: recursive backtrack

Now we need to write the program to solve the $m \times n$ knight's tour problem. There are many ways to describe this problem:

1. We can have a 2d array to describe the progress of the solution such as

1			
	2		

   say the blanks are numbered with zeros.

2. Same as above. But note that in order to move forward, of course I would need to search for the maximum number to know what was the square the knight was on. So besides the 2d array, we can also include a (row,column) to indicate the last square.

1			
	2		

```
lastrow = 2, lastcol = 1
```

3. We can use a 2d array just to tell us if a square is available or not, i.e., a 2d array of 0s and 1s (or boolean values):

1			
	1		

   and then use an array to describe the path taken by the knight:

$$[(0,0),(2,1)]$$

4. We use an array to describe the path taken by the knight:

$$[(0,0),(2,1)]$$

No array at all – you'll see why having an array is helpful even though it costs memory.

5. Etc.

I'll use the third method from above.

Note that a partial candidate solution is a path of 2 integers. When I add an option to the solution, I'm adding 2 integers to the path at one end. When I remove the option, I'm removing the the last of the 2 integers placed in the path. Remember: We need both "add" and "remove" operations on the partial candidate solution. So at this point, the path is like a stack.

Another thing we need to do is to check that when we look at the options, we don't go backward. If you think about it, going backwards would mean that we're going to the (row, column) value *below* the stack. Right? So our path is not exactly a stack. (Of course if we insist on using a pure stack, then to see the value below the stack, we can pop the stack, look at the resulting top, push the old top back. But that's a waste of time.)

We also need to know when the partial candidate solution becomes a complete solution. For an m-by-n grid, that means that the length of the path is mn. So that means that we also need to keep track length of the path.

Once we have found a solution, we want to print an m-by-n array with the step numbers 1, 2, 3, ... (This depends on the requirements of the problem.)

I can use `std::vector` of, say `Step` where `Step` is a class where the objects have two integer values representing the row and column index values. But to make my solution as easy as possible (so that someone with CISS240 can read it), I'm going to use

- an array of values. Each value is made up of two integers: row and column index values. Since I'm using a fixed size array, I have to fix the the (maximum) array size. I will have two constants `MAXROWS` and `MAXCOLS` set to, say, 100,i.e., I'm going to assume that the largest board size is 100-by-100. My program will ask the user for `rows` and `cols` which will be the board size
- a length variable

For the solution below, instead of creating a new partial candidate solution each time I make a move, I'm going to pass a single partial candidate solution around by reference.

There are three things used for computation:

1. A 2d int array where a 0 at (r, c) means that (r, c) is available, otherwise a 1 is at (r, c). The row size of the array is `ROWS`; the column size is `COLS`.
2. A fixed–size array of row-column indices to indicate the steps taken by a knight.
3. A length variable, `pathlen`, for the above array.

The value returned by the function is a boolean: if the boolean returned is `false`, then it was returned from due to a backtrack and if it's `true`, then a solution was found. In the case of a return value of `false`, I have to remove the option that was taken.

A complete solution is found when all the cells of the array is taken, i.e., when `pathlen` is `ROWS * COLS`.

```
#include <iostream>
#include <iomanip>

const int ROWS = 5;
const int COLS = 5;


void pprint(int path[ROWS * COLS][2],
            int & pathlen)
{
    int x[ROWS][COLS] = {{0}};
    for (int i = 0; i < pathlen; i++)
    {
        int r = path[i][0];
        int c = path[i][1];
        x[r][c] = i + 1;
    }
    for (int i = 0; i < COLS; i++) std::cout << "+--";
    std::cout << "+\n";
    for (int r = 0; r < ROWS; r++)
    {
        for (int c = 0; c < COLS; c++)
        {
            std::cout << '|';
            if (x[r][c] == 0)
                std::cout << std::setw(2) << ' ';
            else
                std::cout << std::setw(2) << x[r][c];
        }
        std::cout << '|' << std::endl;
        for (int i = 0; i < COLS; i++) std::cout << "+--";
        std::cout << "+\n";
    }
}


bool solve(int x[ROWS][COLS],
           int path[ROWS * COLS][2],
           int & pathlen)
{
    pprint(path, pathlen);

    if (pathlen == ROWS * COLS)
    {
        return true; // SUCCESS case
    }
    else
    {
        // compute options
        if (pathlen == 0)
        {
            // Path is empty: we can start anywhere
            for (int r = 0; r < ROWS; r++)
            {
                for (int c = 0; c < COLS; c++)
                {
                    x[r][c] = 1; pathlen = 1;
                    path[0][0] = r; path[0][1] = c;
                    bool flag = solve(x, path, pathlen);
                    if (flag)
                    {
                        return true;
                    }
                    // undo
                    x[r][c] = 0; pathlen = 0;
                }
            }
        }
        else
        {
            int r = path[pathlen - 1][0], c = path[pathlen - 1][1];

            // There are 8 options (some outside the board)
            int moves[8][2] = {{-2, -1},
                               {-2, +1},
                               {-1, -2},
                               {-1, +2},
                               {+1, -2},
                               {+1, +2},
                               {+2, -1},
                               {+2, +1}};
            for (int i = 0; i < 8; i++)
            {
                int newr = r + moves[i][0];
                int newc = c + moves[i][1];
                if (0 <= newr && newr < ROWS &&
                    0 <= newc && newc < COLS)
                {
                    if (x[newr][newc] == 0)
                    {
                        // CASE: (newr, newc) not occupied
                        path[pathlen][0] = newr;
                        path[pathlen][1] = newc;
                        x[newr][newc] = pathlen + 1;
                        pathlen++;
                        bool flag = solve(x, path, pathlen);
                        if (flag)
                        {
                            return true;
                        }

                        // CASE: (newr, newc) not occupied
                        // Backtrack
                        x[newr][newc] = 0;
                        pathlen--;
                        std::cout << "BACKTRACK!\n";
                        pprint(path, pathlen);
                    }
                    else
                    {
                        // The current cell is occupied.
                    }
                }
            }
        }
    }
    return false;
}

int main()
```

```
{
    int x[ROWS][COLS] = {{0}};
    int path[ROWS * COLS][2];
    int pathlen = 0;

    bool flag = solve(x, path, pathlen);
    if (flag)
    {
        std::cout << "SUCCESS!" << std::endl;
        for (int i = 0; i < ROWS * COLS; i++)
        {
            std::cout << path[i][0] << ',' << path[i][1]
                      << '\n';
        }
    }
    else
    {
        std::cout << "FAILURE!\n";
    }

    return 0;
}
```

```
+--+--+--+
|  |  |  |
+--+--+--+
|  |  |  |
+--+--+--+
|  |  |  |
+--+--+--+
+--+--+--+
| 1|  |  |
+--+--+--+
|  |  |  |
+--+--+--+
|  |  |  |
+--+--+--+
+--+--+--+
| 1|  |  |
+--+--+--+
|  |  | 2|
+--+--+--+
|  |  |  |
+--+--+--+
+--+--+--+
| 1|  |  |
+--+--+--+
|  |  | 2|
+--+--+--+
| 3|  |  |
+--+--+--+
+--+--+--+
| 1| 4|  |
+--+--+--+
|  |  | 2|
+--+--+--+
| 3|  |  |
+--+--+--+
+--+--+--+
| 1| 4|  |
+--+--+--+
|  |  | 2|
+--+--+--+
| 3|  | 5|
+--+--+--+
+--+--+--+
| 1| 4|  |
+--+--+--+
| 6|  | 2|
+--+--+--+
| 3|  | 5|
+--+--+--+
+--+--+--+
| 1| 4| 7|
+--+--+--+
| 6|  | 2|
+--+--+--+
| 3|  | 5|
+--+--+--+
+--+--+--+
| 1| 4| 7|
+--+--+--+
| 6|  | 2|
+--+--+--+
| 3| 8| 5|
+--+--+--+
BACKTRACK!
+--+--+--+
| 1| 4| 7|
+--+--+--+
| 6|  | 2|
+--+--+--+
| 3|  | 5|
+--+--+--+
BACKTRACK!
+--+--+--+
| 1| 4|  |
+--+--+--+
| 6|  | 2|
+--+--+--+
| 3|  | 5|
+--+--+--+
BACKTRACK!
+--+--+--+
| 1| 4|  |
+--+--+--+
|  |  | 2|
+--+--+--+
| 3|  | 5|
+--+--+--+
BACKTRACK!
+--+--+--+
| 1| 4|  |
+--+--+--+
|  |  | 2|
+--+--+--+
| 3|  |  |
+--+--+--+
BACKTRACK!
+--+--+--+
| 1|  |  |
+--+--+--+
|  |  | 2|
+--+--+--+
| 3|  |  |
+--+--+--+
BACKTRACK!
+--+--+--+
| 1|  |  |
+--+--+--+
|  |  | 2|
+--+--+--+
|  |  |  |
+--+--+--+
BACKTRACK!
+--+--+--+
| 1|  |  |
+--+--+--+
|  |  |  |
+--+--+--+
|  |  |  |
+--+--+--+
+--+--+--+
| 1|  |  |
+--+--+--+
|  |  |  |
+--+--+--+
|  | 2|  |
+--+--+--+
+--+--+--+
| 1|  | 3|
+--+--+--+
|  |  |  |
+--+--+--+
|  | 2|  |
+--+--+--+
+--+--+--+
| 1|  | 3|
+--+--+--+
| 4|  |  |
+--+--+--+
|  | 2|  |
+--+--+--+
+--+--+--+
| 1|  | 3|
+--+--+--+
| 4|  |  |
+--+--+--+
|  | 2| 5|
+--+--+--+
+--+--+--+
| 1| 6| 3|
+--+--+--+
| 4|  |  |
+--+--+--+
|  | 2| 5|
+--+--+--+
+--+--+--+
| 1| 6| 3|
+--+--+--+
| 4|  |  |
+--+--+--+
| 7| 2| 5|
+--+--+--+
+--+--+--+
| 1| 6| 3|
+--+--+--+
| 4|  | 8|
+--+--+--+
| 7| 2| 5|
+--+--+--+
BACKTRACK!
+--+--+--+
| 1| 6| 3|
+--+--+--+
| 4|  |  |
+--+--+--+
| 7| 2| 5|
+--+--+--+
BACKTRACK!
+--+--+--+
| 1| 6| 3|
+--+--+--+
| 4|  |  |
+--+--+--+
|  | 2| 5|
+--+--+--+
BACKTRACK!
+--+--+--+
| 1|  | 3|
+--+--+--+
| 4|  |  |
+--+--+--+
|  | 2| 5|
+--+--+--+
BACKTRACK!
+--+--+--+
| 1|  | 3|
+--+--+--+
| 4|  |  |
+--+--+--+
|  | 2|  |
+--+--+--+
BACKTRACK!
+--+--+--+
| 1|  | 3|
+--+--+--+
|  |  |  |
+--+--+--+
|  | 2|  |
+--+--+--+
BACKTRACK!
+--+--+--+
| 1|  |  |
+--+--+--+
|  |  |  |
+--+--+--+
|  | 2|  |
+--+--+--+
BACKTRACK!
+--+--+--+
| 1|  |  |
+--+--+--+
|  |  |  |
+--+--+--+
|  |  |  |
+--+--+--+
+--+--+--+
|  | 1|  |
+--+--+--+
|  |  |  |
+--+--+--+
|  |  |  |
+--+--+--+
+--+--+--+
|  | 1|  |
+--+--+--+
|  |  |  |
+--+--+--+
| 2|  |  |
+--+--+--+
+--+--+--+
|  | 1|  |
+--+--+--+
|  |  | 3|
+--+--+--+
| 2|  |  |
+--+--+--+
+--+--+--+
| 4| 1|  |
+--+--+--+
|  |  | 3|
+--+--+--+
| 2|  |  |
+--+--+--+
+--+--+--+
| 4| 1|  |
+--+--+--+
|  |  | 3|
+--+--+--+
| 2| 5|  |
+--+--+--+
+--+--+--+
| 4| 1| 6|
+--+--+--+
|  |  | 3|
+--+--+--+
| 2| 5|  |
+--+--+--+
+--+--+--+
| 4| 1| 6|
+--+--+--+
| 7|  | 3|
+--+--+--+
| 2| 5|  |
+--+--+--+
+--+--+--+
| 4| 1| 6|
+--+--+--+
| 7|  | 3|
+--+--+--+
| 2| 5| 8|
+--+--+--+
BACKTRACK!
+--+--+--+
| 4| 1| 6|
+--+--+--+
| 7|  | 3|
+--+--+--+
| 2| 5|  |
+--+--+--+
BACKTRACK!
+--+--+--+
| 4| 1| 6|
+--+--+--+
|  |  | 3|
+--+--+--+
| 2| 5|  |
+--+--+--+
BACKTRACK!
+--+--+--+
| 4| 1|  |
+--+--+--+
|  |  | 3|
+--+--+--+
| 2| 5|  |
+--+--+--+
BACKTRACK!
+--+--+--+
| 4| 1|  |
+--+--+--+
|  |  | 3|
+--+--+--+
| 2|  |  |
+--+--+--+
BACKTRACK!
+--+--+--+
|  | 1|  |
+--+--+--+
|  |  | 3|
+--+--+--+
| 2|  |  |
+--+--+--+
BACKTRACK!
+--+--+--+
|  | 1|  |
+--+--+--+
|  |  |  |
+--+--+--+
| 2|  |  |
+--+--+--+
BACKTRACK!
+--+--+--+
|  | 1|  |
+--+--+--+
|  |  |  |
+--+--+--+
|  |  |  |
+--+--+--+
+--+--+--+
|  | 1|  |
+--+--+--+
|  |  |  |
+--+--+--+
|  |  | 2|
+--+--+--+
+--+--+--+
|  | 1|  |
+--+--+--+
| 3|  |  |
+--+--+--+
|  |  | 2|
+--+--+--+
+--+--+--+
|  |  1| 4|
+--+--+--+
| 3|  |  |
+--+--+--+
|  |  | 2|
+--+--+--+
+--+--+--+
|  |  1| 4|
+--+--+--+
| 3|  |  |
+--+--+--+
|  | 5| 2|
+--+--+--+
+--+--+--+
| 6| 1| 4|
+--+--+--+
| 3|  |  |
+--+--+--+
|  | 5| 2|
+--+--+--+
+--+--+--+
| 6| 1| 4|
+--+--+--+
| 3|  | 7|
+--+--+--+
|  | 5| 2|
+--+--+--+
+--+--+--+
| 6| 1| 4|
+--+--+--+
| 3|  | 7|
+--+--+--+
| 8| 5| 2|
+--+--+--+
BACKTRACK!
+--+--+--+
| 6| 1| 4|
+--+--+--+
| 3|  | 7|
+--+--+--+
|  | 5| 2|
+--+--+--+
BACKTRACK!
+--+--+--+
| 6| 1| 4|
+--+--+--+
| 3|  |  |
+--+--+--+
|  | 5| 2|
+--+--+--+
BACKTRACK!
+--+--+--+
|  | 1| 4|
+--+--+--+
| 3|  |  |
+--+--+--+
|  | 5| 2|
+--+--+--+
BACKTRACK!
+--+--+--+
|  | 1| 4|
+--+--+--+
| 3|  |  |
+--+--+--+
|  |  | 2|
+--+--+--+
BACKTRACK!
+--+--+--+
|  | 1|  |
+--+--+--+
| 3|  |  |
+--+--+--+
|  |  | 2|
+--+--+--+
BACKTRACK!
+--+--+--+
|  | 1|  |
+--+--+--+
|  |  |  |
+--+--+--+
|  |  | 2|
+--+--+--+
BACKTRACK!
+--+--+--+
|  | 1|  |
+--+--+--+
|  |  |  |
+--+--+--+
|  |  |  |
+--+--+--+
+--+--+--+
|  |  | 1|
+--+--+--+
|  |  |  |
+--+--+--+
|  |  |  |
+--+--+--+
+--+--+--+
|  |  | 1|
+--+--+--+
| 2|  |  |
+--+--+--+
|  |  |  |
+--+--+--+
+--+--+--+
|  |  | 1|
+--+--+--+
| 2|  |  |
+--+--+--+
|  |  | 3|
+--+--+--+
+--+--+--+
|  | 4| 1|
+--+--+--+
| 2|  |  |
+--+--+--+
|  |  | 3|
+--+--+--+
+--+--+--+
|  | 4| 1|
+--+--+--+
| 2|  |  |
+--+--+--+
| 5|  | 3|
+--+--+--+
+--+--+--+
|  | 4| 1|
+--+--+--+
| 2|  | 6|
+--+--+--+
| 5|  | 3|
+--+--+--+
+--+--+--+
| 7| 4| 1|
+--+--+--+
| 2|  | 6|
+--+--+--+
| 5|  | 3|
+--+--+--+
+--+--+--+
| 7| 4| 1|
+--+--+--+
| 2|  | 6|
+--+--+--+
| 5| 8| 3|
+--+--+--+
BACKTRACK!
+--+--+--+
| 7| 4| 1|
+--+--+--+
| 2|  | 6|
+--+--+--+
| 5|  | 3|
+--+--+--+
BACKTRACK!
+--+--+--+
|  | 4| 1|
+--+--+--+
| 2|  | 6|
+--+--+--+
| 5|  | 3|
+--+--+--+
BACKTRACK!
+--+--+--+
|  | 4| 1|
+--+--+--+
| 2|  |  |
+--+--+--+
| 5|  | 3|
+--+--+--+
BACKTRACK!
+--+--+--+
|  | 4| 1|
+--+--+--+
| 2|  |  |
+--+--+--+
|  |  | 3|
+--+--+--+
BACKTRACK!
+--+--+--+
|  |  | 1|
+--+--+--+
| 2|  |  |
+--+--+--+
|  |  | 3|
+--+--+--+
BACKTRACK!
+--+--+--+
|  |  | 1|
+--+--+--+
| 2|  |  |
+--+--+--+
|  |  |  |
+--+--+--+
BACKTRACK!
+--+--+--+
|  |  | 1|
+--+--+--+
|  |  |  |
+--+--+--+
|  |  |  |
+--+--+--+
+--+--+--+
|  |  | 1|
+--+--+--+
|  |  |  |
+--+--+--+
|  | 2|  |
+--+--+--+
+--+--+--+
| 3|  | 1|
+--+--+--+
|  |  |  |
+--+--+--+
|  | 2|  |
+--+--+--+
+--+--+--+
| 3|  | 1|
+--+--+--+
|  |  | 4|
+--+--+--+
|  | 2|  |
+--+--+--+
+--+--+--+
| 3|  | 1|
+--+--+--+
|  |  | 4|
+--+--+--+
| 5| 2|  |
+--+--+--+
+--+--+--+
| 3| 6| 1|
+--+--+--+
|  |  | 4|
+--+--+--+
| 5| 2|  |
+--+--+--+
+--+--+--+
| 3| 6| 1|
+--+--+--+
|  |  | 4|
+--+--+--+
| 5| 2| 7|
+--+--+--+
+--+--+--+
| 3| 6| 1|
+--+--+--+
| 8|  | 4|
+--+--+--+
| 5| 2| 7|
+--+--+--+
BACKTRACK!
+--+--+--+
| 3| 6| 1|
+--+--+--+
|  |  | 4|
+--+--+--+
| 5| 2| 7|
+--+--+--+
BACKTRACK!
+--+--+--+
| 3| 6| 1|
+--+--+--+
|  |  | 4|
+--+--+--+
| 5| 2|  |
+--+--+--+
BACKTRACK!
+--+--+--+
| 3|  | 1|
+--+--+--+
|  |  | 4|
+--+--+--+
| 5| 2|  |
+--+--+--+
BACKTRACK!
+--+--+--+
| 3|  | 1|
+--+--+--+
|  |  | 4|
+--+--+--+
|  | 2|  |
+--+--+--+
BACKTRACK!
+--+--+--+
| 3|  | 1|
+--+--+--+
|  |  |  |
+--+--+--+
|  | 2|  |
+--+--+--+
BACKTRACK!
+--+--+--+
|  |  | 1|
+--+--+--+
|  |  |  |
+--+--+--+
|  | 2|  |
+--+--+--+
BACKTRACK!
+--+--+--+
|  |  | 1|
+--+--+--+
|  |  |  |
+--+--+--+
|  |  |  |
+--+--+--+
+--+--+--+
|  |  |  |
+--+--+--+
| 1|  |  |
+--+--+--+
|  |  |  |
+--+--+--+
+--+--+--+
|  |  | 2|
+--+--+--+
| 1|  |  |
+--+--+--+
|  |  |  |
+--+--+--+
+--+--+--+
|  |  | 2|
+--+--+--+
| 1|  |  |
+--+--+--+
|  | 3|  |
+--+--+--+
+--+--+--+
| 4|  | 2|
+--+--+--+
| 1|  |  |
+--+--+--+
|  | 3|  |
+--+--+--+
+--+--+--+
| 4|  | 2|
+--+--+--+
| 1|  | 5|
+--+--+--+
|  | 3|  |
+--+--+--+
+--+--+--+
| 4|  | 2|
+--+--+--+
| 1|  | 5|
+--+--+--+
| 6| 3|  |
+--+--+--+
+--+--+--+
| 4| 7| 2|
+--+--+--+
| 1|  | 5|
+--+--+--+
| 6| 3|  |
+--+--+--+
+--+--+--+
| 4| 7| 2|
+--+--+--+
| 1|  | 5|
+--+--+--+
| 6| 3| 8|
+--+--+--+
BACKTRACK!
+--+--+--+
| 4| 7| 2|
+--+--+--+
| 1|  | 5|
+--+--+--+
| 6| 3|  |
+--+--+--+
BACKTRACK!
+--+--+--+
| 4|  | 2|
+--+--+--+
| 1|  | 5|
+--+--+--+
| 6| 3|  |
+--+--+--+
BACKTRACK!
+--+--+--+
| 4|  | 2|
+--+--+--+
| 1|  | 5|
+--+--+--+
|  | 3|  |
+--+--+--+
BACKTRACK!
+--+--+--+
| 4|  | 2|
+--+--+--+
| 1|  |  |
+--+--+--+
|  | 3|  |
+--+--+--+
BACKTRACK!
+--+--+--+
|  |  | 2|
+--+--+--+
| 1|  |  |
+--+--+--+
|  | 3|  |
+--+--+--+
BACKTRACK!
+--+--+--+
|  |  | 2|
+--+--+--+
| 1|  |  |
+--+--+--+
|  |  |  |
+--+--+--+
BACKTRACK!
+--+--+--+
|  |  |  |
+--+--+--+
| 1|  |  |
+--+--+--+
|  |  |  |
+--+--+--+
+--+--+--+
|  |  |  |
+--+--+--+
| 1|  |  |
+--+--+--+
|  |  | 2|
+--+--+--+
+--+--+--+
|  | 3|  |
+--+--+--+
| 1|  |  |
+--+--+--+
|  |  | 2|
+--+--+--+
+--+--+--+
|  | 3|  |
+--+--+--+
| 1|  |  |
+--+--+--+
| 4|  | 2|
+--+--+--+
+--+--+--+
|  | 3|  |
+--+--+--+
| 1|  | 5|
+--+--+--+
| 4|  | 2|
+--+--+--+
+--+--+--+
| 6| 3|  |
+--+--+--+
| 1|  | 5|
+--+--+--+
| 4|  | 2|
+--+--+--+
+--+--+--+
| 6| 3|  |
+--+--+--+
| 1|  | 5|
+--+--+--+
| 4| 7| 2|
+--+--+--+
+--+--+--+
| 6| 3| 8|
+--+--+--+
| 1|  | 5|
+--+--+--+
| 4| 7| 2|
+--+--+--+
BACKTRACK!
+--+--+--+
| 6| 3|  |
+--+--+--+
| 1|  | 5|
+--+--+--+
| 4| 7| 2|
+--+--+--+
BACKTRACK!
+--+--+--+
| 6| 3|  |
+--+--+--+
| 1|  | 5|
+--+--+--+
| 4|  | 2|
+--+--+--+
BACKTRACK!
+--+--+--+
|  | 3|  |
+--+--+--+
| 1|  | 5|
+--+--+--+
| 4|  | 2|
+--+--+--+
BACKTRACK!
+--+--+--+
|  | 3|  |
+--+--+--+
| 1|  |  |
+--+--+--+
| 4|  | 2|
+--+--+--+
BACKTRACK!
+--+--+--+
|  | 3|  |
+--+--+--+
| 1|  |  |
+--+--+--+
|  |  | 2|
+--+--+--+
BACKTRACK!
+--+--+--+
|  |  |  |
+--+--+--+
| 1|  |  |
+--+--+--+
|  |  | 2|
+--+--+--+
BACKTRACK!
+--+--+--+
|  |  |  |
+--+--+--+
| 1|  |  |
+--+--+--+
|  |  |  |
+--+--+--+
+--+--+--+
|  |  |  |
+--+--+--+
|  | 1|  |
+--+--+--+
|  |  |  |
+--+--+--+
+--+--+--+
|  |  |  |
+--+--+--+
|  |  | 1|
+--+--+--+
|  |  |  |
+--+--+--+
+--+--+--+
| 2|  |  |
+--+--+--+
|  |  | 1|
+--+--+--+
|  |  |  |
+--+--+--+
+--+--+--+
| 2|  |  |
+--+--+--+
|  |  | 1|
+--+--+--+
|  | 3|  |
+--+--+--+
+--+--+--+
| 2|  | 4|
+--+--+--+
|  |  | 1|
+--+--+--+
|  | 3|  |
+--+--+--+
+--+--+--+
| 2|  | 4|
+--+--+--+
| 5|  | 1|
+--+--+--+
|  | 3|  |
+--+--+--+
+--+--+--+
| 2|  | 4|
+--+--+--+
| 5|  | 1|
+--+--+--+
|  | 3| 6|
+--+--+--+
+--+--+--+
| 2| 7| 4|
+--+--+--+
| 5|  | 1|
+--+--+--+
|  | 3| 6|
+--+--+--+
+--+--+--+
| 2| 7| 4|
+--+--+--+
| 5|  | 1|
+--+--+--+
| 8| 3| 6|
+--+--+--+
BACKTRACK!
+--+--+--+
| 2| 7| 4|
+--+--+--+
| 5|  | 1|
+--+--+--+
|  | 3| 6|
+--+--+--+
BACKTRACK!
+--+--+--+
| 2|  | 4|
+--+--+--+
| 5|  | 1|
+--+--+--+
|  | 3| 6|
+--+--+--+
BACKTRACK!
+--+--+--+
| 2|  | 4|
+--+--+--+
| 5|  | 1|
+--+--+--+
|  | 3|  |
+--+--+--+
BACKTRACK!
+--+--+--+
| 2|  | 4|
+--+--+--+
|  |  | 1|
+--+--+--+
|  | 3|  |
+--+--+--+
BACKTRACK!
+--+--+--+
| 2|  |  |
+--+--+--+
|  |  | 1|
+--+--+--+
|  | 3|  |
+--+--+--+
BACKTRACK!
+--+--+--+
| 2|  |  |
+--+--+--+
|  |  | 1|
+--+--+--+
|  |  |  |
+--+--+--+
BACKTRACK!
+--+--+--+
|  |  |  |
+--+--+--+
|  |  | 1|
+--+--+--+
|  |  |  |
+--+--+--+
+--+--+--+
|  |  |  |
+--+--+--+
|  |  | 1|
+--+--+--+
| 2|  |  |
+--+--+--+
+--+--+--+
|  | 3|  |
+--+--+--+
|  |  | 1|
+--+--+--+
| 2|  |  |
+--+--+--+
+--+--+--+
|  | 3|  |
+--+--+--+
|  |  | 1|
+--+--+--+
| 2|  | 4|
+--+--+--+
+--+--+--+
|  | 3|  |
+--+--+--+
| 5|  | 1|
+--+--+--+
| 2|  | 4|
+--+--+--+
+--+--+--+
|  | 3| 6|
+--+--+--+
| 5|  | 1|
+--+--+--+
| 2|  | 4|
+--+--+--+
+--+--+--+
|  | 3| 6|
+--+--+--+
| 5|  | 1|
+--+--+--+
| 2| 7| 4|
+--+--+--+
+--+--+--+
| 8| 3| 6|
+--+--+--+
| 5|  | 1|
+--+--+--+
| 2| 7| 4|
+--+--+--+
BACKTRACK!
+--+--+--+
|  | 3| 6|
+--+--+--+
| 5|  | 1|
+--+--+--+
| 2| 7| 4|
+--+--+--+
BACKTRACK!
+--+--+--+
|  | 3| 6|
+--+--+--+
| 5|  | 1|
+--+--+--+
| 2|  | 4|
+--+--+--+
BACKTRACK!
+--+--+--+
|  | 3|  |
+--+--+--+
| 5|  | 1|
+--+--+--+
| 2|  | 4|
+--+--+--+
BACKTRACK!
+--+--+--+
|  | 3|  |
+--+--+--+
|  |  | 1|
+--+--+--+
| 2|  | 4|
+--+--+--+
BACKTRACK!
+--+--+--+
|  | 3|  |
+--+--+--+
|  |  | 1|
+--+--+--+
| 2|  |  |
+--+--+--+
BACKTRACK!
+--+--+--+
|  |  |  |
+--+--+--+
|  |  | 1|
+--+--+--+
| 2|  |  |
+--+--+--+
BACKTRACK!
+--+--+--+
|  |  |  |
+--+--+--+
|  |  | 1|
+--+--+--+
|  |  |  |
+--+--+--+
+--+--+--+
|  |  |  |
+--+--+--+
|  |  |  |
+--+--+--+
| 1|  |  |
+--+--+--+
+--+--+--+
|  | 2|  |
+--+--+--+
|  |  |  |
+--+--+--+
| 1|  |  |
+--+--+--+
+--+--+--+
|  | 2|  |
+--+--+--+
|  |  |  |
+--+--+--+
| 1|  | 3|
+--+--+--+
+--+--+--+
|  | 2|  |
+--+--+--+
| 4|  |  |
+--+--+--+
| 1|  | 3|
+--+--+--+
+--+--+--+
|  | 2| 5|
+--+--+--+
| 4|  |  |
+--+--+--+
| 1|  | 3|
+--+--+--+
+--+--+--+
|  | 2| 5|
+--+--+--+
| 4|  |  |
+--+--+--+
| 1| 6| 3|
+--+--+--+
+--+--+--+
| 7| 2| 5|
+--+--+--+
| 4|  |  |
+--+--+--+
| 1| 6| 3|
+--+--+--+
+--+--+--+
| 7| 2| 5|
+--+--+--+
| 4|  | 8|
+--+--+--+
| 1| 6| 3|
+--+--+--+
BACKTRACK!
+--+--+--+
| 7| 2| 5|
+--+--+--+
| 4|  |  |
+--+--+--+
| 1| 6| 3|
+--+--+--+
BACKTRACK!
+--+--+--+
|  | 2| 5|
+--+--+--+
| 4|  |  |
+--+--+--+
| 1| 6| 3|
+--+--+--+
BACKTRACK!
+--+--+--+
|  | 2| 5|
+--+--+--+
| 4|  |  |
+--+--+--+
| 1|  | 3|
+--+--+--+
BACKTRACK!
+--+--+--+
|  | 2|  |
+--+--+--+
| 4|  |  |
+--+--+--+
| 1|  | 3|
+--+--+--+
BACKTRACK!
+--+--+--+
|  | 2|  |
+--+--+--+
|  |  |  |
+--+--+--+
| 1|  | 3|
+--+--+--+
BACKTRACK!
+--+--+--+
|  | 2|  |
+--+--+--+
|  |  |  |
+--+--+--+
| 1|  |  |
+--+--+--+
BACKTRACK!
+--+--+--+
|  |  |  |
+--+--+--+
|  |  |  |
+--+--+--+
| 1|  |  |
+--+--+--+
+--+--+--+
|  |  |  |
+--+--+--+
|  |  | 2|
+--+--+--+
| 1|  |  |
+--+--+--+
+--+--+--+
| 3|  |  |
+--+--+--+
|  |  | 2|
+--+--+--+
| 1|  |  |
+--+--+--+
+--+--+--+
| 3|  |  |
+--+--+--+
|  |  | 2|
+--+--+--+
| 1| 4|  |
+--+--+--+
+--+--+--+
| 3|  | 5|
+--+--+--+
|  |  | 2|
+--+--+--+
| 1| 4|  |
+--+--+--+
+--+--+--+
| 3|  | 5|
+--+--+--+
| 6|  | 2|
+--+--+--+
| 1| 4|  |
+--+--+--+
+--+--+--+
| 3|  | 5|
+--+--+--+
| 6|  | 2|
+--+--+--+
| 1| 4| 7|
+--+--+--+
+--+--+--+
| 3| 8| 5|
+--+--+--+
| 6|  | 2|
+--+--+--+
| 1| 4| 7|
+--+--+--+
BACKTRACK!
+--+--+--+
| 3|  | 5|
+--+--+--+
| 6|  | 2|
+--+--+--+
| 1| 4| 7|
+--+--+--+
BACKTRACK!
+--+--+--+
| 3|  | 5|
+--+--+--+
| 6|  | 2|
+--+--+--+
| 1| 4|  |
+--+--+--+
BACKTRACK!
+--+--+--+
| 3|  | 5|
+--+--+--+
|  |  | 2|
+--+--+--+
| 1| 4|  |
+--+--+--+
BACKTRACK!
+--+--+--+
| 3|  |  |
+--+--+--+
|  |  | 2|
+--+--+--+
| 1| 4|  |
+--+--+--+
BACKTRACK!
+--+--+--+
| 3|  |  |
+--+--+--+
|  |  | 2|
+--+--+--+
| 1|  |  |
+--+--+--+
BACKTRACK!
+--+--+--+
|  |  |  |
+--+--+--+
|  |  | 2|
+--+--+--+
| 1|  |  |
+--+--+--+
BACKTRACK!
+--+--+--+
|  |  |  |
+--+--+--+
|  |  |  |
+--+--+--+
| 1|  |  |
+--+--+--+
+--+--+--+
|  |  |  |
+--+--+--+
|  |  |  |
+--+--+--+
|  | 1|  |
+--+--+--+
+--+--+--+
| 2|  |  |
+--+--+--+
|  |  |  |
+--+--+--+
|  | 1|  |
+--+--+--+
+--+--+--+
| 2|  |  |
+--+--+--+
|  |  | 3|
+--+--+--+
|  | 1|  |
+--+--+--+
+--+--+--+
| 2|  |  |
+--+--+--+
|  |  | 3|
+--+--+--+
| 4| 1|  |
+--+--+--+
+--+--+--+
| 2| 5|  |
+--+--+--+
|  |  | 3|
+--+--+--+
| 4| 1|  |
+--+--+--+
+--+--+--+
| 2| 5|  |
+--+--+--+
|  |  | 3|
+--+--+--+
| 4| 1| 6|
+--+--+--+
+--+--+--+
| 2| 5|  |
+--+--+--+
| 7|  | 3|
+--+--+--+
| 4| 1| 6|
+--+--+--+
+--+--+--+
| 2| 5| 8|
+--+--+--+
| 7|  | 3|
+--+--+--+
| 4| 1| 6|
+--+--+--+
BACKTRACK!
+--+--+--+
| 2| 5|  |
+--+--+--+
| 7|  | 3|
+--+--+--+
| 4| 1| 6|
+--+--+--+
BACKTRACK!
+--+--+--+
| 2| 5|  |
+--+--+--+
|  |  | 3|
+--+--+--+
| 4| 1| 6|
+--+--+--+
BACKTRACK!
+--+--+--+
| 2| 5|  |
+--+--+--+
|  |  | 3|
+--+--+--+
| 4| 1|  |
+--+--+--+
BACKTRACK!
+--+--+--+
| 2|  |  |
+--+--+--+
|  |  | 3|
+--+--+--+
| 4| 1|  |
+--+--+--+
BACKTRACK!
+--+--+--+
| 2|  |  |
+--+--+--+
|  |  | 3|
+--+--+--+
|  | 1|  |
+--+--+--+
BACKTRACK!
+--+--+--+
| 2|  |  |
+--+--+--+
|  |  |  |
+--+--+--+
|  | 1|  |
+--+--+--+
BACKTRACK!
+--+--+--+
|  |  |  |
+--+--+--+
|  |  |  |
+--+--+--+
|  | 1|  |
+--+--+--+
+--+--+--+
|  |  | 2|
+--+--+--+
|  |  |  |
+--+--+--+
|  | 1|  |
+--+--+--+
+--+--+--+
|  |  | 2|
+--+--+--+
| 3|  |  |
+--+--+--+
|  | 1|  |
+--+--+--+
+--+--+--+
|  |  | 2|
+--+--+--+
| 3|  |  |
+--+--+--+
|  | 1| 4|
+--+--+--+
+--+--+--+
|  | 5| 2|
+--+--+--+
| 3|  |  |
+--+--+--+
|  | 1| 4|
+--+--+--+
+--+--+--+
|  | 5| 2|
+--+--+--+
| 3|  |  |
+--+--+--+
| 6| 1| 4|
+--+--+--+
+--+--+--+
|  | 5| 2|
+--+--+--+
| 3|  | 7|
+--+--+--+
| 6| 1| 4|
+--+--+--+
+--+--+--+
| 8| 5| 2|
+--+--+--+
| 3|  | 7|
+--+--+--+
| 6| 1| 4|
+--+--+--+
BACKTRACK!
+--+--+--+
|  | 5| 2|
+--+--+--+
| 3|  | 7|
+--+--+--+
| 6| 1| 4|
+--+--+--+
BACKTRACK!
+--+--+--+
|  | 5| 2|
+--+--+--+
| 3|  |  |
+--+--+--+
| 6| 1| 4|
+--+--+--+
BACKTRACK!
+--+--+--+
|  | 5| 2|
+--+--+--+
| 3|  |  |
+--+--+--+
|  | 1| 4|
+--+--+--+
BACKTRACK!
+--+--+--+
|  |  | 2|
+--+--+--+
| 3|  |  |
+--+--+--+
|  | 1| 4|
+--+--+--+
BACKTRACK!
+--+--+--+
|  |  | 2|
+--+--+--+
| 3|  |  |
+--+--+--+
|  | 1|  |
+--+--+--+
BACKTRACK!
+--+--+--+
|  |  | 2|
+--+--+--+
|  |  |  |
+--+--+--+
|  | 1|  |
+--+--+--+
BACKTRACK!
+--+--+--+
|  |  |  |
+--+--+--+
|  |  |  |
+--+--+--+
|  | 1|  |
+--+--+--+
+--+--+--+
|  |  |  |
+--+--+--+
|  |  |  |
+--+--+--+
|  |  | 1|
+--+--+--+
+--+--+--+
|  | 2|  |
+--+--+--+
|  |  |  |
+--+--+--+
|  |  | 1|
+--+--+--+
+--+--+--+
|  | 2|  |
+--+--+--+
|  |  |  |
+--+--+--+
| 3|  | 1|
+--+--+--+
+--+--+--+
|  | 2|  |
+--+--+--+
|  |  | 4|
+--+--+--+
| 3|  | 1|
+--+--+--+
+--+--+--+
| 5| 2|  |
+--+--+--+
|  |  | 4|
+--+--+--+
| 3|  | 1|
+--+--+--+
+--+--+--+
| 5| 2|  |
+--+--+--+
|  |  | 4|
+--+--+--+
| 3| 6| 1|
+--+--+--+
+--+--+--+
| 5| 2| 7|
+--+--+--+
|  |  | 4|
+--+--+--+
| 3| 6| 1|
+--+--+--+
+--+--+--+
| 5| 2| 7|
+--+--+--+
| 8|  | 4|
+--+--+--+
| 3| 6| 1|
+--+--+--+
BACKTRACK!
+--+--+--+
| 5| 2| 7|
+--+--+--+
|  |  | 4|
+--+--+--+
| 3| 6| 1|
+--+--+--+
BACKTRACK!
+--+--+--+
| 5| 2|  |
+--+--+--+
|  |  | 4|
+--+--+--+
| 3| 6| 1|
+--+--+--+
BACKTRACK!
+--+--+--+
| 5| 2|  |
+--+--+--+
|  |  | 4|
+--+--+--+
| 3|  | 1|
+--+--+--+
BACKTRACK!
+--+--+--+
|  | 2|  |
+--+--+--+
|  |  | 4|
+--+--+--+
| 3|  | 1|
+--+--+--+
BACKTRACK!
+--+--+--+
|  | 2|  |
+--+--+--+
|  |  |  |
+--+--+--+
| 3|  | 1|
+--+--+--+
BACKTRACK!
+--+--+--+
|  | 2|  |
+--+--+--+
|  |  |  |
+--+--+--+
|  |  | 1|
+--+--+--+
BACKTRACK!
+--+--+--+
|  |  |  |
+--+--+--+
|  |  |  |
+--+--+--+
|  |  | 1|
+--+--+--+
+--+--+--+
|  |  |  |
+--+--+--+
| 2|  |  |
+--+--+--+
|  |  | 1|
+--+--+--+
+--+--+--+
|  |  | 3|
+--+--+--+
| 2|  |  |
+--+--+--+
|  |  | 1|
+--+--+--+
+--+--+--+
|  |  | 3|
+--+--+--+
| 2|  |  |
+--+--+--+
|  | 4| 1|
+--+--+--+
+--+--+--+
| 5|  | 3|
+--+--+--+
| 2|  |  |
+--+--+--+
|  | 4| 1|
+--+--+--+
+--+--+--+
| 5|  | 3|
+--+--+--+
| 2|  | 6|
+--+--+--+
|  | 4| 1|
+--+--+--+
+--+--+--+
| 5|  | 3|
+--+--+--+
| 2|  | 6|
+--+--+--+
| 7| 4| 1|
+--+--+--+
+--+--+--+
| 5| 8| 3|
+--+--+--+
| 2|  | 6|
+--+--+--+
| 7| 4| 1|
+--+--+--+
BACKTRACK!
```

```
+--+--+--+
| 5|  | 3|
+--+--+--+
| 2|  | 6|
+--+--+--+
| 7| 4| 1|
+--+--+--+
BACKTRACK!
+--+--+--+
| 5|  | 3|
+--+--+--+
| 2|  | 6|
+--+--+--+
|  | 4| 1|
+--+--+--+
BACKTRACK!
+--+--+--+
| 5|  | 3|
+--+--+--+
| 2|  |  |
+--+--+--+
|  | 4| 1|
+--+--+--+
BACKTRACK!
+--+--+--+
|  |  | 3|
+--+--+--+
| 2|  |  |
+--+--+--+
|  | 4| 1|
+--+--+--+
BACKTRACK!
+--+--+--+
|  |  | 3|
+--+--+--+
| 2|  |  |
+--+--+--+
|  |  | 1|
+--+--+--+
BACKTRACK!
+--+--+--+
|  |  |  |
+--+--+--+
| 2|  |  |
+--+--+--+
|  |  | 1|
+--+--+--+
BACKTRACK!
+--+--+--+
|  |  |  |
+--+--+--+
|  |  |  |
+--+--+--+
|  |  | 1|
+--+--+--+
FAILURE!
```

Here's a solution to the 5-by-5 knight tours problem:

```
+--+--+--+--+--+
|  |  |  |  |  |
+--+--+--+--+--+
|  |  |  |  |  |
+--+--+--+--+--+
|  |  |  |  |  |
+--+--+--+--+--+
|  |  |  |  |  |
+--+--+--+--+--+
|  |  |  |  |  |
+--+--+--+--+--+
+--+--+--+--+--+
| 1|  |  |  |  |
+--+--+--+--+--+
|  |  |  |  |  |
+--+--+--+--+--+
|  |  |  |  |  |
+--+--+--+--+--+
|  |  |  |  |  |
+--+--+--+--+--+
|  |  |  |  |  |
+--+--+--+--+--+
+--+--+--+--+--+
| 1|  |  |  |  |
+--+--+--+--+--+
|  |  | 2|  |  |
+--+--+--+--+--+
|  |  |  |  |  |
+--+--+--+--+--+
|  |  |  |  |  |
+--+--+--+--+--+
|  |  |  |  |  |
+--+--+--+--+--+
+--+--+--+--+--+
| 1|  |  |  | 3|
+--+--+--+--+--+
|  |  | 2|  |  |
+--+--+--+--+--+
|  |  |  |  |  |
+--+--+--+--+--+
|  |  |  |  |  |
+--+--+--+--+--+
|  |  |  |  |  |
+--+--+--+--+--+
... snipped ...
+--+--+--+--+--+
| 1|18| 5|10| 3|
+--+--+--+--+--+
| 6|11| 2|19|14|
+--+--+--+--+--+
|17|  |13| 4| 9|
+--+--+--+--+--+
|12| 7|20|15|  |
+--+--+--+--+--+
|  |16|  | 8|  |
+--+--+--+--+--+
+--+--+--+--+--+
| 1|18| 5|10| 3|
+--+--+--+--+--+
| 6|11| 2|19|14|
+--+--+--+--+--+
|17|  |13| 4| 9|
+--+--+--+--+--+
|12| 7|20|15|  |
+--+--+--+--+--+
|  |16|  | 8|21|
+--+--+--+--+--+
BACKTRACK!
+--+--+--+--+--+
| 1|18| 5|10| 3|
+--+--+--+--+--+
| 6|11| 2|19|14|
+--+--+--+--+--+
|17|  |13| 4| 9|
+--+--+--+--+--+
|12| 7|20|15|  |
+--+--+--+--+--+
|  |16|  | 8|  |
+--+--+--+--+--+
BACKTRACK!
+--+--+--+--+--+
| 1|18| 5|10| 3|
+--+--+--+--+--+
| 6|11| 2|19|14|
+--+--+--+--+--+
|17|  |13| 4| 9|
+--+--+--+--+--+
|12| 7|  |15|  |
+--+--+--+--+--+
|  |16|  | 8|  |
+--+--+--+--+--+
+--+--+--+--+--+
| 1|18| 5|10| 3|
+--+--+--+--+--+
| 6|11| 2|19|14|
+--+--+--+--+--+
|17|  |13| 4| 9|
+--+--+--+--+--+
|12| 7|  |15|20|
+--+--+--+--+--+
|  |16|  | 8|  |
+--+--+--+--+--+
+--+--+--+--+--+
| 1|18| 5|10| 3|
+--+--+--+--+--+
| 6|11| 2|19|14|
+--+--+--+--+--+
|17|  |13| 4| 9|
+--+--+--+--+--+
|12| 7|  |15|20|
+--+--+--+--+--+
|  |16|21| 8|  |
+--+--+--+--+--+
+--+--+--+--+--+
| 1|18| 5|10| 3|
+--+--+--+--+--+
| 6|11| 2|19|14|
+--+--+--+--+--+
|17|22|13| 4| 9|
+--+--+--+--+--+
|12| 7|  |15|20|
+--+--+--+--+--+
|  |16|21| 8|  |
+--+--+--+--+--+
+--+--+--+--+--+
| 1|18| 5|10| 3|
+--+--+--+--+--+
| 6|11| 2|19|14|
+--+--+--+--+--+
|17|22|13| 4| 9|
+--+--+--+--+--+
|12| 7|  |15|20|
+--+--+--+--+--+
|23|16|21| 8|  |
+--+--+--+--+--+
+--+--+--+--+--+
| 1|18| 5|10| 3|
+--+--+--+--+--+
| 6|11| 2|19|14|
+--+--+--+--+--+
|17|22|13| 4| 9|
+--+--+--+--+--+
|12| 7|24|15|20|
+--+--+--+--+--+
|23|16|21| 8|  |
+--+--+--+--+--+
+--+--+--+--+--+
| 1|18| 5|10| 3|
+--+--+--+--+--+
| 6|11| 2|19|14|
+--+--+--+--+--+
|17|22|13| 4| 9|
+--+--+--+--+--+
|12| 7|24|15|20|
+--+--+--+--+--+
|23|16|21| 8|25|
+--+--+--+--+--+
SUCCESS!
0,0
1,2
0,4
2,3
0,2
1,0
3,1
4,3
2,4
0,3
1,1
3,0
2,2
1,4
3,3
4,1
2,0
0,1
1,3
3,4
4,2
2,1
4,0
3,2
4,4
```

File: backtrack-non-recursive.tex

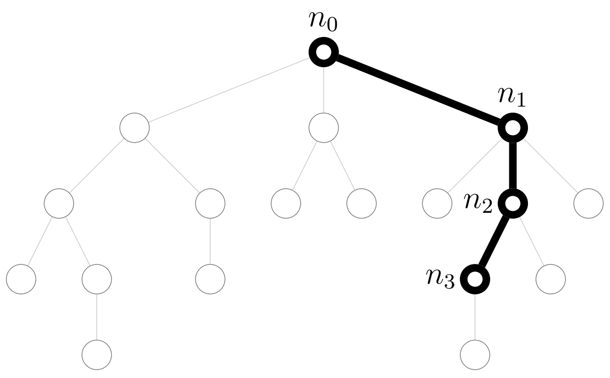
110.6 Pseudocode: no recursion

Here's the recursive backtrack search again:

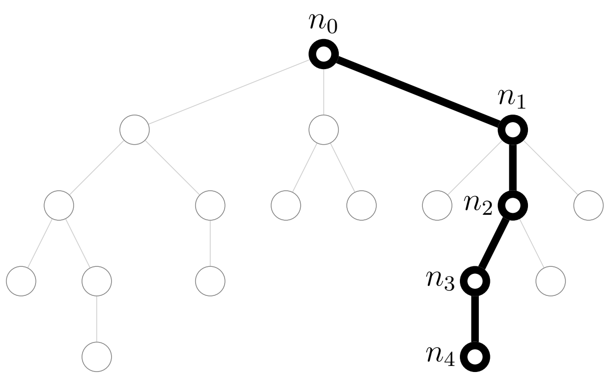
```
backtrack_search(solution):
    if solution is complete:
        return true
    else:
        for each option at this point:
            extend solution by adding option
            flag = backtrack_search(solution)
            if flag: return true
            remove option from solution
        return false
```

Of course it's possible to rewrite the above so that recursion is not used. Let's think carefully about backtrack search so that we can translate that into a procedure that does not require recursion.

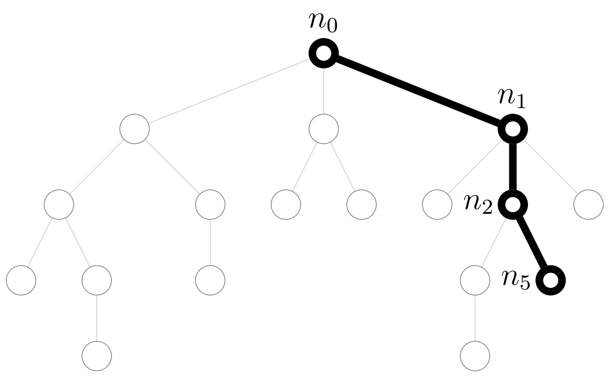
Suppose this is part of the search where the function call stack contains data along the path that is drawn in bold:



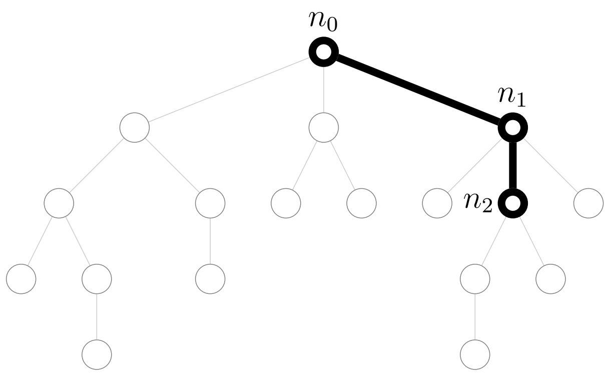
What does backtrack search really do? It creates paths in the above tree and tests if the path forms a valid solution. Of course backtrack search must create all possible paths starting from the root. Now, how does backtrack search creates these paths? Look at the above path. Backtrack search modifies the path by modifying the end of the path (the lowest point in the diagram). This endpoint is modified in three ways. Either the above path becomes



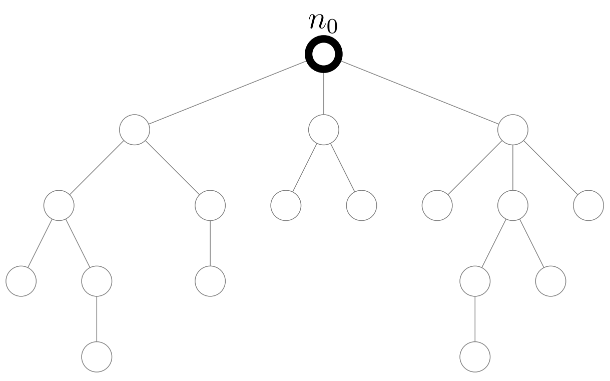
I'll call this the going *down case*. Or the above can be this:



which I will call the *go right case*. Or it can becomes this which I will call the *go up case*:



How are these three "operations on paths" use together to compute all paths? We start here:



The above is the initialization. At the beginning, we clearly must extend the path going down. We can for instance have a variable to tell us which "path extension direction" to use. For instance we can have a variable call `direction` and set it to `"down"` at this point.

We then look the end of the path, i.e., the node n_0 and based on the direction, we compute the first child of n_0. What will happen when we cannot go down? We perform go right on the path. Correct? And what if we can't go right? We go up! Once we execute a go up, we set the direction to go right.

```
ALGORITHM: NONRECURSIVE-BACKTRACK

    let s be an empty stack
    push the empty solution onto s
    perform_goal_test = FALSE

    direction = "down"

    while s is not empty:

        if perform_goal_test:
            if s is solution:
                return s
            perform_goal_test = FALSE

        let t be the top of stack s

        if direction is "down":
            if t has a child:
                push child of t onto s
                perform_goal_test = TRUE
            else:
                direction = "right"
        elif direction is "right":
            if t has a next sibling:
                replace t (top of stack) with next sibling of t
                perform_goal_test = TRUE
                direction = "down"
            else:
                direction = "up"
        elif direction is "up":
            pop s
            direction = "right"
```

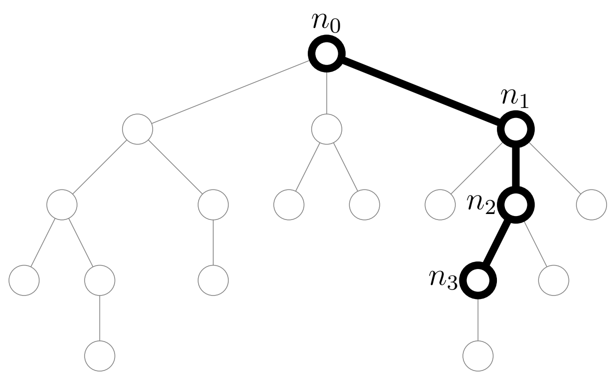
110.6.1 Goal test

Note that in the above pseudocode, we do not need to perform a goal test after a pop, i.e., when the direction is up. Also, we assume that the empty is not a valid solution and therefore we are not going to perform goal test on the empty solution.

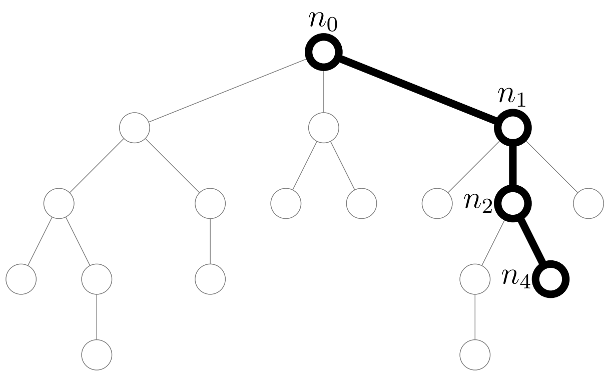
110.6.2 Next sibling

Note that you need to compute the next sibling of the node at the top of the stack. Sometimes, you will need to include extra data to help compute the next sibling. Suppose the siblings can be numbered 0, 1, 2, etc. (The first child is numbered 0.)

Look at this:



n_3 is child-0 of n_2. The next sibling of n_3 is n_4 in the following:



n_4 is child-1 of n_2.

Frequently, a child node is computed from the parent node and not from the previous sibling. For instance in the above diagram, it might be easier to compute n_4 from n_2 and the fact that n_4 is child-1. Now, it's usually possible to compute the child number from the node itself. For instance with node n_3 and its parent n_2, you can usually compute the fact that n_3 is child-0. But this might be a waste of time. Therefore if you have the following path:

$$[n_0, n_1, n_2, n_3]$$

you will also need to remember the child number. For instance, using the above diagram, the path can be

$$[(2, n_0), (1, n_1), (0, n_2)]$$

with n_3 in memory. Meaning to say for instance that node n_3 is child-0 of n_2, node n_2 is child-1 of n_1, etc. Note that n_3 is not in the stack.

Note the following

1. If the computation of next sibling of the top of stack requires both the top of stack the value below the top of stack, then in fact a (pure) stack should not be used for the path. In this case, an array or a doubly-linked list is better.

2. Sometimes it's possible to compute compute child-$(i + 1)$ from child-i without requiring the data of the parent. (In this case, the path can be a stack.)

We are assuming that the i–th child of the parent node can be derived from the parent and i. There are cases where the children are orgnized as a linked list so that the *next* child is obtained through, for instance, `parent.nextchild()`. It is easy to implement `parent.nextchild()` if we started off with `parent.child(i)`. If that's the case, the first child is obtained by `parent.firstchild()` (`parent.has_child()` can be used to detect if `parent` has at least one child; additionally if `parent` does not have a child, then `parent.firstchild()` will result in an exception being thrown.) Calling `parent.nextchild()` will result in an exception being thrown if there is no next child.

Should next child be derived from child or parent?

Note that we do not use

$$[(-1, n_0), (2, n_1), (1, n_2), (0, n_3)]$$

the node being processed, n_3 is in memory.

Sometimes, from the parent and the child, you can compute the next child quickly, therefore the i in (i, n_j) is not necessary.

110.6.3 Difference between backtrack and depth-first search

If you don't pay attention, you would think that backtracking search is the same as depth-first. They are similar but not the same!

In the case of depth-first search, all the children of a node is placed in the stack. For backtrack search, only one child is placed in the stack.

110.7 Knight's tour solution: non-recursive backtrack

110.8 Knight's tour variants

Exercise 110.8.1. Same as knight's tour except that on the board, some squares are marked as invalid. Find all longest knight's tours. □

Exercise 110.8.2. There are two knights: one white and one black. Places for them are fixed. Moves are made alternately. Find two knight's tour (one or white and one for black) that covers as many squares as possible. Or consider this as a game: make white move intelligently. □

110.9 n-queens problem

n–QUEENS PROBLEM. Place n queens on an n–by–n chessboard so that no two pairs are attacking each other.

n–Queens Problem

Note that two queens are attacking each other if they are on the same row or on the same column or on the same diagonal.

Exercise 110.9.1. Solve the n–queens problem using backtracking search. The program should prompt for n. □

110.10 Magic square problem

Magic Problem

Magic Square Problem. Will an n–by–n grid with the numbers $1, 2, 3, ..., n$ such that each number is used exactly one and on the resulting grid, the sum of each column, the sum of each row, and the sum of each diagonal are all the same. (In this case, a diagonal has n values. There are two diagonals.)

Exercise 110.10.1. Solve the magic square problem using backtracking search. □

110.11 Sudoku problem

Exercise 110.11.1. Solve the sudoku problem using backtracking search.
□

Index

n–Queens Problem, 4056

backtracking, 4001

knight's tour, 4008

Magic Square Problem, 4057