Computer Science 385
Design and Analysis of Algorithms
Siena College
Spring 2018

Lab 11: Backtracking

Due: Start of your next lab session

You will be assigned a partner to work with on this lab. Only one submission per group is needed.

Group members: ____

Learning goals:

1. to gain a deeper understanding of the complexity of the n-Queens problem
2. to see and modify and implementation of a backtracking algorithm
3. to learn about additional problems that can be solved with a backtracking approach

Getting Set Up

You will receive an email with the link to follow to set up your GitHub repository, which will be named `backtracking-lab-yourgitname`, for this Lab. One member of the group should follow the link to set up the repository on GitHub, then that person should email the instructor with the other group members' GitHub usernames so they can be granted access. This will allow all members of the group to clone the repository and commit and push changes to the origin on GitHub.

Submitting

Once all written items are initialed to indicate completion, turn in one copy of this handout. Be sure names of all group members are clearly on the first page.

Your submission requires that all required deliverables are committed and pushed to the master for your repository's origin on GitHub. That's it! If you see everything you intend to submit when you visit your repository's page on GitHub, you're set.

Grading

Score: $\boxed{}$ / 100

n-Queens

In class, we discussed the n-Queens problem at length, focusing on a backtracking solution. A Java program that implements the pseudocode from class is in your repository. Compile and run it. It takes n as a command-line parameter.

? Question 1:
Run the program for $n = 4$. What final configuration do you get? (4 points)

This obviously incorrect answer arises because the `legalMove` method is incomplete.

? Question 2:
Complete the `legalMove` method. (12 points)

? Question 3:
Run the program again for $n = 4$. What final configuration do you get? (4 points)

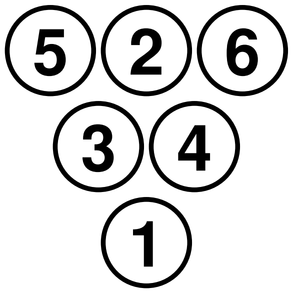
Modify the program so it runs all cases from $n = 1$ to $n = 32$. Be sure to reset the timer and the recursive call counter before each iteration.

? Question 4:
Run your modified version of the program. Demonstrate your results and include them in your repository's `README.md` file. (15 points)

Question 5:
Find a value k where the solution for $n = k$ is found in less time and fewer nontrivial recursive calls than it took for the solution to $n = k - 1$. Explain why the backtracking solution to n-Queens can sometimes find a solution to $n = k$ without first having to have found the solution for $n = k - 1$. (10 points)

The Billiard Ball Problem

The *billiard ball problem* consists of arranging a triangle of numbered billiard balls such that the resulting layout has the following arithmetic property: every ball below the top level is the absolute value of the difference between two balls immediately above it. For instance, the following is a solution to the problem when the top row consists of three balls (which requires a total of six balls).



Note that the balls are numbered from 1 to 6 in this case.

For the problems with four and five balls in the top row, there are a total of 10 and 15 balls, respectively.

You can read more about the problem in Martin Gardner's *Penrose Tiles to Trapdoor Ciphers:*

And the Return of Dr Matrix (Cambridge Press 1997) on pages 119-120. (`http://books.google.com/books?id=8-FlYl6-ML8C&lpg=PP1&pg=PA119#v=onepage&q&f=false`)

As far as I can tell, there are no known solutions for problems larger than five balls in the top row.

A Backtracking Approach

A backtracking algorithm, very similar to that for n-Queens problem, is one approach to solving this problem.

The idea is that we attempt to build up a candidate solution by adding balls to the top row. Each time a ball is added, we make sure we have not broken any of the rules (in this case, there is just one: there are no repeated numbers in the triangle generated by that top row, including the top row itself). If we have not yet broken a rule, we either have found a solution (if the top row now contains the desired number of balls) or we have a partial candidate solution and we should add another ball. Any time we generate a candidate solution that does violate the rule, we have hit a dead end, so we undo the most recent addition and try the next option. If we ever backtrack all the way to the beginning and have run out of options for our first move, we know no solution exists.

For example, consider a backtracking solution to the problem where there are 2 balls in the top row, and we choose numbers from the largest to the smallest each time we reach a decision point. (Note that this is a good strategy to get a solution more quickly, as larger numbers will tend to be in the top row.)

We start with an empty solution, and we are ready to add the first ball to the top row. Since we are trying numbers from the largest to smallest, we start with 3:

```
(3)
```

This is a legal configuration: there are no repeated digits when we expand this out (in fact, there is no expansion needed for a single ball). So we accept this as a partial solution and move on, trying to add a ball to the second position. The first ball we attempt to place at this position is the highest numbered, 3:

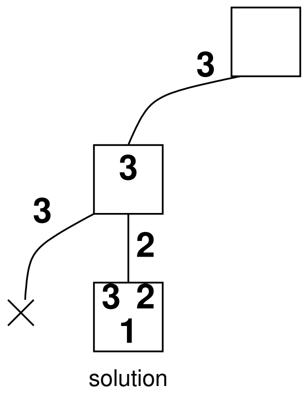
```
(3 3)        which expands to        (3 3)
                                      (0)
```

This is not a legal configuration: it includes two 3's. So we backtrack and erase our last move, and instead try the next option, which is to use the 2 ball:

```
(3 2)        which expands to        (3 2)
                                      (1)
```

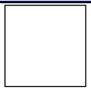
This is a legal solution: no repeats. Plus, we now have filled the top row, so our solution is complete.

We can display this in the format of a state space search diagram, like we did for n-Queens.



Question 6:

Show the state space search diagram for a backtracking solution to the problem with two balls in the top row if we instead chose balls for each position in increasing instead of decreasing numerical order. (15 points)



Now, let's consider the start of the procedure for the much more interesting (and much longer) backtracking computation of a solution to the problem with three balls in the top row. Note that here, we have a total of six balls.

Our first move is to place the largest number into the top row.

```
(6)
```

This is again legal, so we continue by adding a second number.

```
(6 6)
```

This contains a duplicate, so we backtrack and try a 5 in the second position.

```
(6 5)
 (1)
```

This is legal, so we accept the 5 for now, and start working on the third ball. We begin, as before, with the highest numbered ball and work our way down if we encounter illegal moves.

```
(6 5 6)
 (1 1)
  (0)
```

This has duplicates, so it is not legal. In fact, all of our choices for the third ball will result in illegal configurations here:

```
(6 5 6)    (6 5 5)    (6 5 4)    (6 5 3)    (6 5 2)    (6 5 1)
 (1 1)      (1 0)      (1 1)      (1 2)      (1 3)      (1 4)
  (0)        (1)        (0)        (1)        (2)        (3)
```

So this means 5 in the second position of the top row was a dead end. And we backtrack, and try a 4 there instead:

```
(6 4)
 (2)
```

So far so good here, so we move on trying each ball in the 3rd position of the top row...

Question 7:
On the back of a page in this packet, draw the state space search diagram for the 6-ball solution that will result from the above procedure. Note that this will not necessarily lead to the sample solution pictured earlier in this document. (20 points)

Subset-Sum Problem

Read Levitin p. 427–428, which describes the subset-sum problem.

? **Question 8:**
For each non-solution leaf in Figure 12.4 on p. 428 of Levitin, explain where the numbers in the condition come from and why that means the search can be cut off at that point. (5 points)

? **Question 9:**
Draw a complete state-space tree for a backtracking approach to the subset-sum problem as applied to the example in Levitin Exercise 12.1.3 (a), p. 431. (15 points)