CSC 211: Computer Programming
(Recursive) Backtracking

Michael Conti

Department of Computer Science and Statistics
University of Rhode Island

Spring 2024

Images and material from web.stanford.edu

Administrative Announcements

- **A03 due 03/28**
- **MC05 due 03/21**
- **Exam# 02 - 04/02**

Recursion Reminder

- Problem solving technique in which we solve a task by reducing it to smaller tasks (of the same kind)
  - then use same approach to solve the smaller tasks
- Technically, a recursive function is one that **calls itself**
- General form:
  - **base case**
    - solution for a **trivial case**
    - it can be used to stop the recursion (prevents "*stack overflow*")
    - every recursive algorithm needs at least one base case
  - **recursive call(s)**
    - divide problem into **smaller instance(s)** of the **same structure**

Recursion Reminder

- Recursive Checklist:
  - **Find what information we need to keep track of.** What inputs/outputs are needed to solve the problem at each step?
  - **Find our base case(s).** What are the simplest (nonrecursive) instance(s) of this problem?
  - **Find our recursive step.** How can this problem be solved in terms of one or more simpler instances of the same problem that lead to a base case?
  - **Ensure every input is handled.** Do we cover all possible cases? Do we need to handle errors?

Recursion Reminder

- Recursive Checklist:
  - ✓ **Find what information we need to keep track of.** What inputs/outputs are needed to solve the problem at each step?
  - ✓ **Find our base case(s).** What are the simplest (nonrecursive) instance(s) of this problem?
  - ✓ **Find our recursive step.** How can this problem be solved in terms of one or more simpler instances of the same problem that lead to a base case?
  - ✓ **Ensure every input is handled.** Do we cover all possible cases? Do we need to handle errors?

Backtracking

- Write a recursive function printAllBinary that accepts an integer number of digits and prints all binary numbers that have exactly that many digits, in ascending order, one per line

printAllBinary(2);

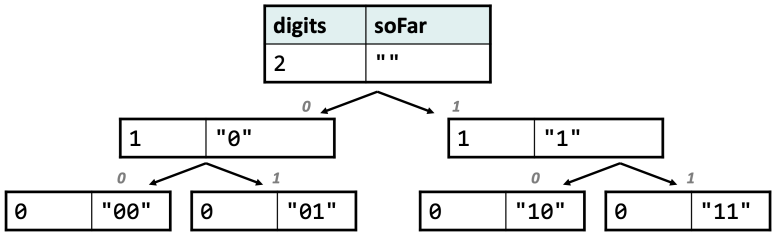
```
00
01
10
11
```

printAllBinary(3);

```
000
001
010
011
100
101
110
111
```


Decision Trees

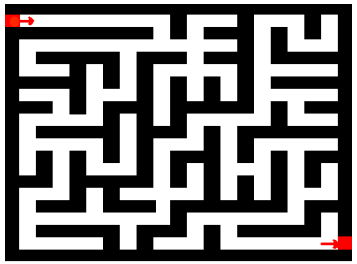
printAllBinary(2);



- This kind of diagram is called a **call tree** or **decision tree**
- Think of each call as a choice or decision made by the algorithm:
  - Should I choose 0 as the next digit?
  - Should I choose 1 as the next digit?
- The idea is to try every permutation. For every position, there are 2 options, either '0' or '1'. **Backtracking** can be used in this approach to try every possibility or permutation to generate the correct set of strings.

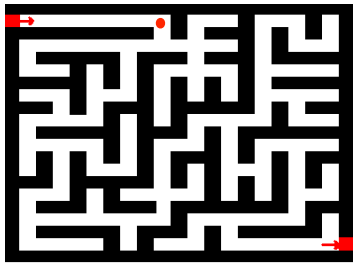
Backtracking

- **Recursive Backtracking**: using recursion to explore solutions to a problem and abandoning them if they are not suitable
-



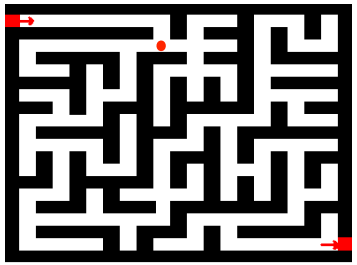
Backtracking

- **Recursive Backtracking**: using recursion to explore solutions to a problem and abandoning them if they are not suitable
-



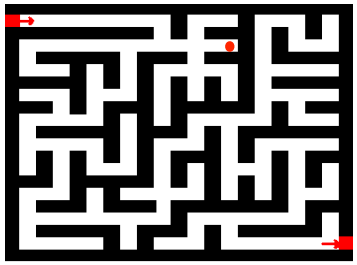
Backtracking

- **Recursive Backtracking**: using recursion to explore solutions to a problem and abandoning them if they are not suitable
-



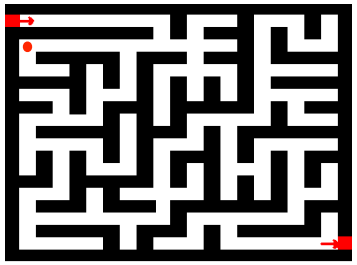
Backtracking

- **Recursive Backtracking**: using recursion to explore solutions to a problem and abandoning them if they are not suitable
-



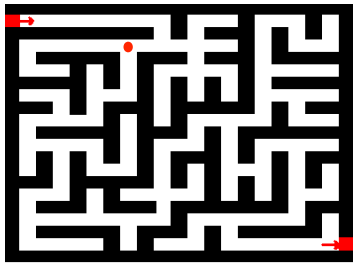
Backtracking

- **Recursive Backtracking**: using recursion to explore solutions to a problem and abandoning them if they are not suitable
-



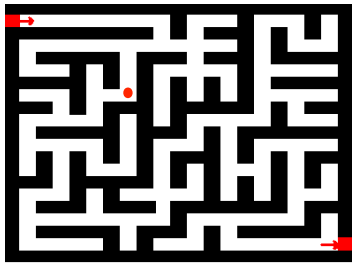
Backtracking

- **Recursive Backtracking**: using recursion to explore solutions to a problem and abandoning them if they are not suitable
-



Backtracking

- **Recursive Backtracking**: using recursion to explore solutions to a problem and abandoning them if they are not suitable
-



Backtracking

- Let's take a look at a problem similar to the binarySequence problem.
- Write a recursive function diceRoll that accepts an integer representing a number of 6-sided dice to roll, and output all possible permutations of values that could appear on the dice.

diceRoll(2)

{1,1}	{3, 1}	{5, 1}
{1, 2}	{3, 2}	{5, 2}
{1, 3}	{3, 3}	{5, 3}
{1, 4}	{3, 4}	{5, 4}
{1, 5}	{3, 5}	{5, 5}
{1, 6}	{3, 6}	{5, 6}
{2, 1}	{4, 1}	{6, 1}
{2, 2}	{4, 2}	{6, 2}
{2, 3}	{4, 3}	{6, 3}
{2, 4}	{4, 4}	{6, 4}
{2, 5}	{4, 5}	{6, 5}
{2, 6}	{4, 6}	{6, 6}

Backtracking

- Backtracking Checklist:
  - **Find what choice(s) we have at each step.** What different options are there for the next step?

    For each valid choice:
    - **Make it and explore recursively.** Pass the information for a choice to the next recursive call(s).
    - **Undo it after exploring.** Restore everything to the way it was before making this choice.
  - **Find our base case(s).** What should we do when we are out of decisions?

Backtracking

- Backtracking Checklist:
  - **Find what choice(s) we have at each step.** What different options are there for the next step?

    For each valid choice:
    - **Make it and explore recursively.** Pass the information for a choice to the next recursive call(s).
    - **Undo it after exploring.** Restore everything to the way it was before making this choice.

  - **Find our base case(s).** What should we do when we are out of decisions?

What die value should I choose next?

Backtracking

- Backtracking Checklist:
  - **Find what choice(s) we have at each step.** What different options are there for the next step?

    For each valid choice:
    - **Make it and explore recursively.** Pass the information for a choice to the next recursive call(s).
    - **Undo it after exploring.** Restore everything to the way it was before making this choice.

  - **Find our base case(s).** What should we do when we are out of decisions?

We need to communicate the dice chosen so far to the next recursive call

Backtracking

- Backtracking Checklist:
  - **Find what choice(s) we have at each step.** What different options are there for the next step?

    For each valid choice:
    - **Make it and explore recursively.** Pass the information for a choice to the next recursive call(s).
    - **Undo it after exploring.** Restore everything to the way it was before making this choice.

  - **Find our base case(s).** What should we do when we are out of decisions?

We need to be able to remove the die we added to our first roll so far

Backtracking

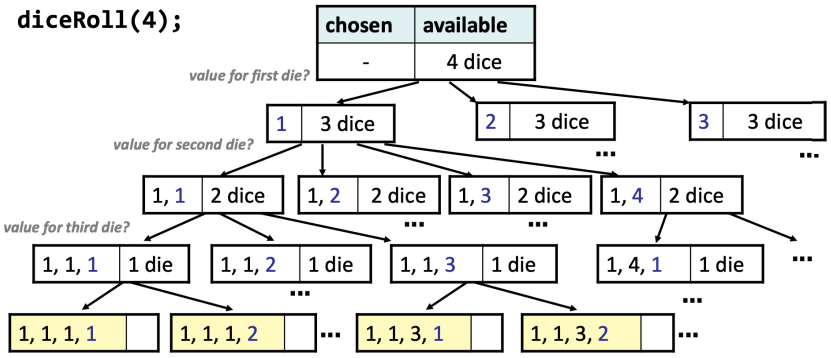
- Backtracking Checklist:
  - **Find what choice(s) we have at each step.** What different options are there for the next step?

    For each valid choice:
    - **Make it and explore recursively.** Pass the information for a choice to the next recursive call(s).
    - **Undo it after exploring.** Restore everything to the way it was before making this choice.

  - **Find our base case(s).** What should we do when we are out of decisions?

We have no dice left to choose, print them out

Backtracking



- Observations?
- This is a really big search space.
- Depending on approach, we can make wasteful decisions. Can we optimize it? Yes. Will we right now? No.

Backtracking

- Let's us write flexible code, allowing us to make a decision and "backtrack" if we need to

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

Backtracking

- **Pseudocode**

```
function diceRolls(dice, chosenArr):
        if dice == 0:
            Print current roll.
        else:
            // handle all roll values for a single die; let recursion do the rest.
            for each die value i in range [1..6]:
                choose that the current die will have value i
                // explore the remaining dice
                diceRolls(dice-1, chosenArr)
                un-choose (backtrack) the value I
```

- Write a recursive function diceRoll that accepts an integer representing a number of 6-sided dice to roll, and output all possible combinations of values that could appear on the dice.

** Need to keep track of our choices somehow

Code Demo