Recursive Backtracking

Outline

1 Words & Permutations

2 Solving Mazes

Words & Permutations

All Words

Find all length \( n \) strings made up of a's, b's, and c's.

To do this, we build up partial solutions as follows:

- The only length 0 string is ""; so, we’re done.
- Otherwise, the three choices are a, b, and c:
  - Make the choice letter
  - Find all solutions with one fewer letter recursively.
  - Unmake the choice (to continue looking).

Recursive Backtracking

Definition (Recursive Backtracking)

Recursive Backtracking is an attempt to find solution(s) by building up partial solutions and abandoning them if they don’t work.

Recursive Backtracking Strategy

- If we found a solution, stop looking (e.g. return)
- Otherwise for each possible choice \( c \)
  - Make the choice \( c \)
  - Recursively continue to make choices
  - Un-make the choice \( c \) (if we got back here, it means we need to continue looking)

All Words Solution

```java
1 private static void words(int length) {
2     String[] choices = {"a", "b", "c", "d"};
3     // The empty string is the only word of length 0
4     if (length == 0) {
5         print();
6     }
7     else {
8         // Try appending each possible choice to our partial word.
9         for (String choice : choices) {
10             choose(choice); // Add the choice
11             words(length - 1); // Recurse on the rest
12             unchoose(); // undo the choice
13         }
14     }
15 }
```
Accumulators

1 private static void words(String acc, int length) {
2 String[] choices = {"a", "b", "c", "d"};
3 // The empty string is the only word of length 0
4 if (length == 0) {
5 print();
6 }
7 else {
8 for (String choice : choices) {
9 acc += choice;
10 words(acc, length - 1);
11 acc = acc.substring(0, acc.length() - 1);
12 }
13 }
14 }

Recursion Reminder

Solving Recursion Problems
- Figure out what the pieces of the problem are.
- What is the base case? (the smallest possible piece of the problem)
- Solve one piece of the problem and recurse on the rest.

Solving a Maze

Solving a maze is a lot like paintbucket. What is the difference?

Instead of filling everything in, we want to stop at dead ends!

If you were in a maze, how would you solve it?
- Try a direction.
- Every time you go in a direction, draw an X on the ground.
- If you hit a dead end, go back until you can go in another direction.

This is recursive backtracking!

1 public boolean canSolveMaze(int x, int y) {
2 if (isGoal(x, y)) {
3 return true;
4 }
5 else if (inBounds(x, y) && isPassage(panel)) {
6 return solveMaze(x - 1, y) ||
7 solveMaze(x, y - 1) ||
8 solveMaze(x, y + 1) ||
9 solveMaze(x + 1, y);
10 }
11 return false;
12 }

Recursive Backtracking Tips!

- The most important part is figuring out what the choices are.
- It can help to draw out a tree of choices
- Make sure to undo your choices after the recursive call.
- You will still always have a base case.