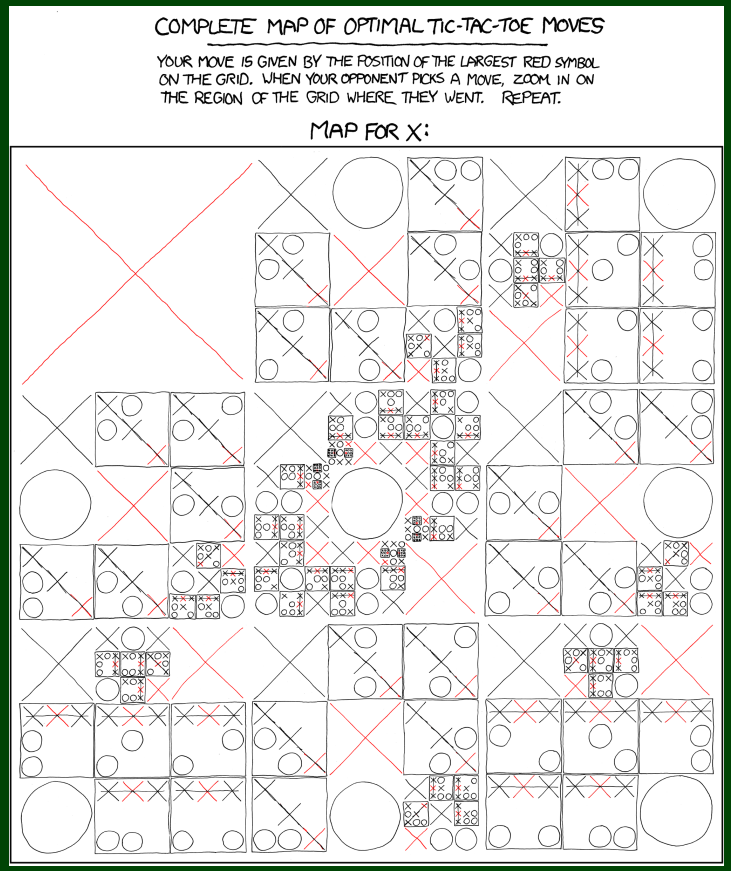
Recursive Backtracking



Outline

Definition (Recursive Backtracking)

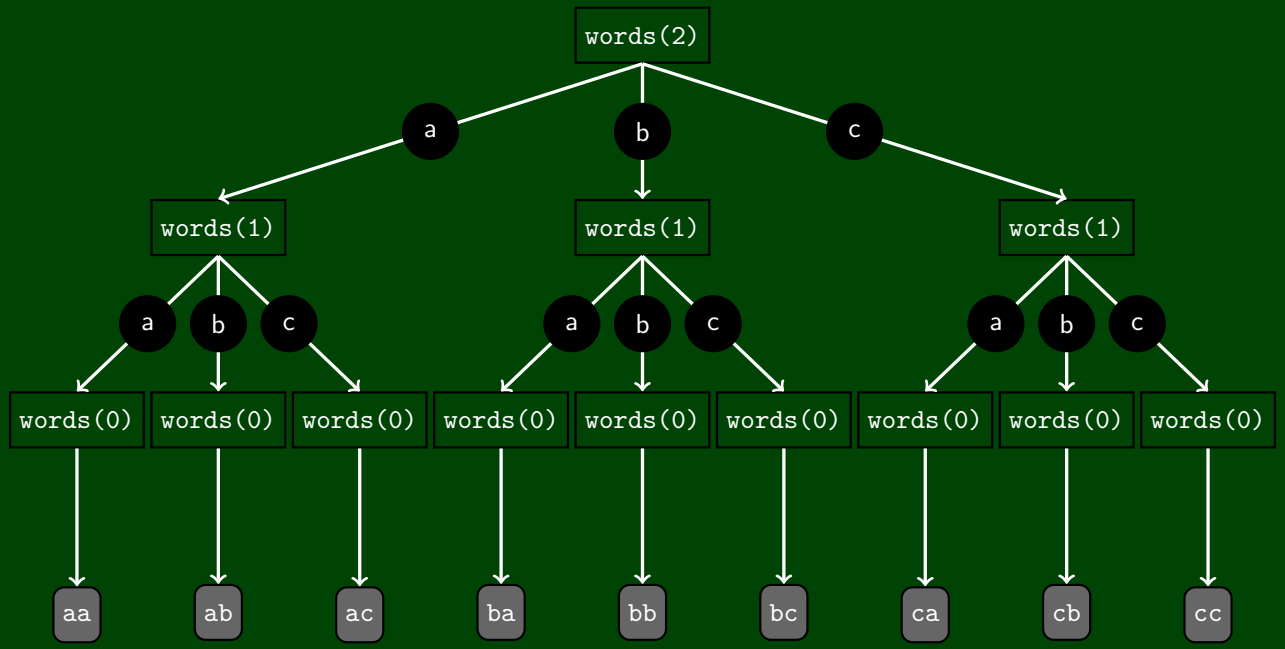
Recursive Backtracking is an attempt to find solution(s) by building up partial solutions and abandoning them if they don't work.

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)

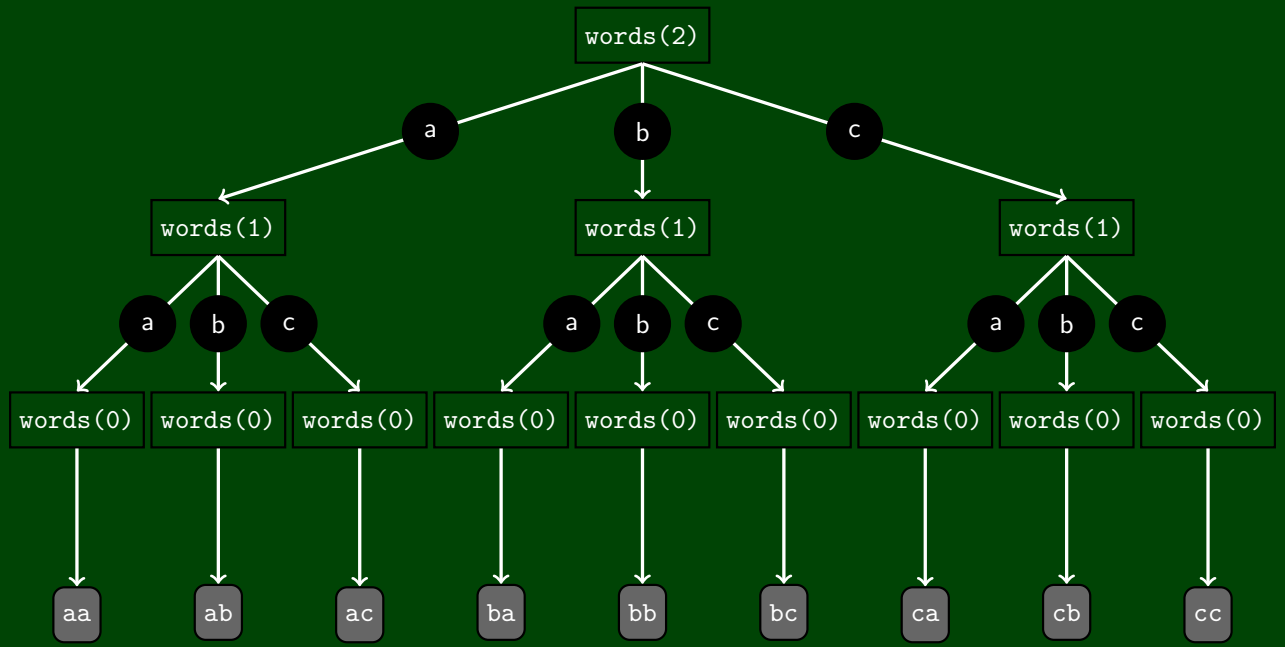
All Words

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



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).

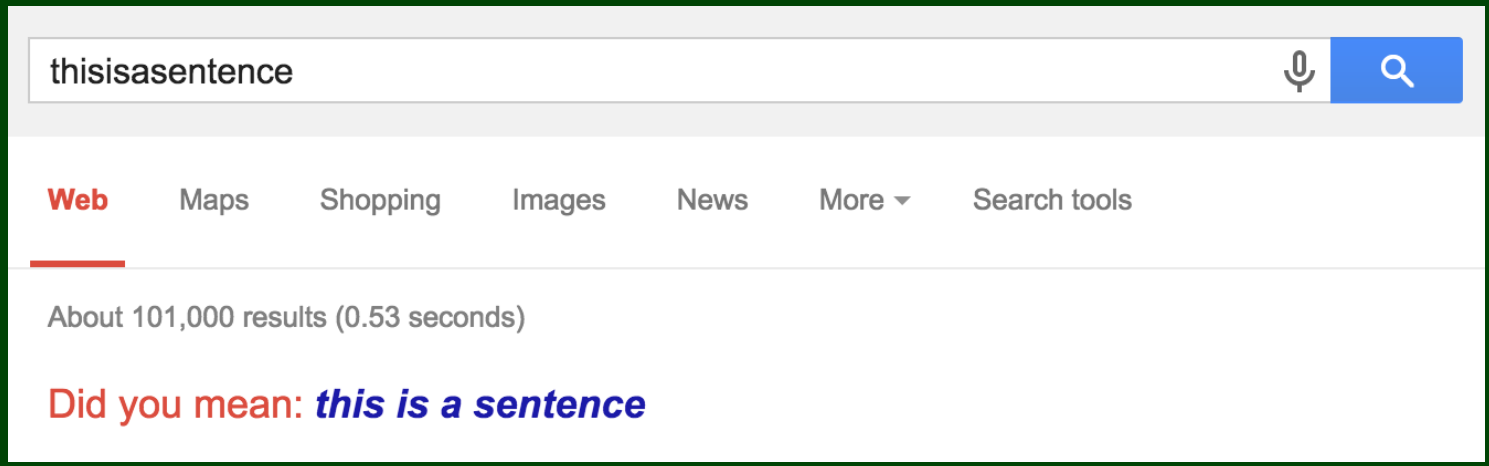
All Words Solution

```
 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  }
```

When you enter a query with no spaces like **thisisasentence** into Google:



It fixes it into **this is a sentence** using recursive backtracking.

Sentence Splitting

Given an input string, `sentence`, containing **no spaces**, write a method:

```
public static String splitSentence(String sentence)
```

that returns `sentence` split up into words.

Sentence Splitting

Given an input string, `sentence`, containing **no spaces**, write a method:

```
public static String splitSentence(String sentence)
```

that returns `sentence` split up into words.

To do recursive backtracking, we need to answer these questions:

- What are the choices we're making incrementally?
- How do we "undo" a choice?
- What are the base case(s)?

It helps to answer these questions for a particular input. So, pretend we're working with:

thisisasentence

Sentence Splitting

Given an input string, `sentence`, containing **no spaces**, write a method:

```
public static String splitSentence(String sentence)
```

that returns `sentence` split up into words.

To do recursive backtracking, we need to answer these questions:

- What are the choices we're making incrementally?
  ... which character to split at
- How do we "undo" a choice?
  ... re-combine a string by the char we split at
- What are the base case(s)?
  ... our left choice isn't a word **and** our right choice IS a word

It helps to answer these questions for a particular input. So, pretend we're working with:

thisisasentence

One More Important Choice

When doing recursive backtracking, we need to differentiate between:

- finding a result
- failing to find a result (e.g., backtracking)

One More Important Choice

When doing recursive backtracking, we need to differentiate between:

- finding a result
- failing to find a result (e.g., backtracking)

Generally, we do this by treating `null` as a failure. For example:

- On the input, "**thisisasentence**", none of the recursive calls should return "**thisis**", because it isn't a word.
- If we get down to an empty string, that would indicate a failure; so, we'd return `null`

Sentence Splitter Solution


```
 1  public String splitSentence(String sentence) {
 2     // The entire sentence is a dictionary word!
 3     if (words.contains(sentence)) {
 4        return sentence;
 5     }
 6
 7     // Try splitting at every character until we find one that works...
 8     for (int i = sentence.length() - 1; i > 0; i--){
 9        String left = sentence.substring(0, i);
10        String right = sentence.substring(i, sentence.length());
11
12        // If the left isn't a word, don't bother recursing.
13        // If it is, split the remainder of the sentence recursively.
14        if (words.contains(left)) {
15           right = splitSentence(right);
16           // Since the left was a word, if the right is also an answer,
17           // then we found an answer to the whole thing!
18           if (right != null) {
19              return left + " " + right;
20           }
21
22           // Undo our choice by going back to sentence
23        }
24     }
25     return null;
26  }
```