CSE 143

Recursive backtracking
Exercise: Dice rolls

- Write a method `diceRoll` that accepts an integer parameter representing a number of 6-sided dice to roll, and output all possible arrangements of values that could appear on the dice.

```
diceRoll(2);
```

```
diceRoll(3);
```

<table>
<thead>
<tr>
<th>[1, 1]</th>
<th>[3, 1]</th>
<th>[5, 1]</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1, 2]</td>
<td>[3, 2]</td>
<td>[5, 2]</td>
</tr>
<tr>
<td>[1, 3]</td>
<td>[3, 3]</td>
<td>[5, 3]</td>
</tr>
<tr>
<td>[1, 4]</td>
<td>[3, 4]</td>
<td>[5, 4]</td>
</tr>
<tr>
<td>[1, 5]</td>
<td>[3, 5]</td>
<td>[5, 5]</td>
</tr>
<tr>
<td>[1, 6]</td>
<td>[3, 6]</td>
<td>[5, 6]</td>
</tr>
<tr>
<td>[2, 1]</td>
<td>[4, 1]</td>
<td>[6, 1]</td>
</tr>
<tr>
<td>[2, 2]</td>
<td>[4, 2]</td>
<td>[6, 2]</td>
</tr>
<tr>
<td>[2, 3]</td>
<td>[4, 3]</td>
<td>[6, 3]</td>
</tr>
<tr>
<td>[2, 4]</td>
<td>[4, 4]</td>
<td>[6, 4]</td>
</tr>
<tr>
<td>[2, 5]</td>
<td>[4, 5]</td>
<td>[6, 5]</td>
</tr>
<tr>
<td>[2, 6]</td>
<td>[4, 6]</td>
<td>[6, 6]</td>
</tr>
</tbody>
</table>

...
Examining the problem

- We want to generate all possible sequences of values.
  
  for (each possible first die value):
    for (each possible second die value):
      for (each possible third die value):
        ...
      print!

- This is called a **depth-first search**

- How can we completely explore such a large search space?
Exercise: Dice roll sum

- Write a method `diceSum` similar to `diceRoll`, but it also accepts a desired sum and prints only arrangements that add up to exactly that sum.

```java
diceSum(2, 7);  // [1, 6], [2, 5], [3, 4], [4, 3], [5, 2], [6, 1]
diceSum(3, 7);  // [1, 1, 5], [1, 2, 4], [1, 3, 3], [1, 4, 2], [1, 5, 1], [2, 1, 4], [2, 2, 3], [2, 3, 2], [2, 4, 1], [3, 1, 3], [3, 2, 2], [3, 3, 1], [4, 1, 2], [4, 2, 1], [5, 1, 1]
```
Consider all paths?

<table>
<thead>
<tr>
<th>chosen</th>
<th>available</th>
<th>desired sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>3 dice</td>
<td>5</td>
</tr>
</tbody>
</table>

```
1 2 dice
2 2 dice
3 2 dice
4 2 dice
5 2 dice
6 2 dice
1, 1 1 die
1, 2 1 die
1, 3 1 die
1, 4 1 die
1, 5 1 die
1, 6 1 die
1, 1, 1
1, 1, 2
1, 1, 3
1, 1, 4
1, 1, 5
1, 1, 6
1, 6, 1
1, 6, 2
...```
New decision tree

<table>
<thead>
<tr>
<th>chosen</th>
<th>available</th>
<th>desired sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>3 dice</td>
<td>5</td>
</tr>
</tbody>
</table>

![Decision Tree Diagram]
Backtracking

- **backtracking**: Finding solution(s) by trying partial solutions and then abandoning them if they are not suitable.
  - a "brute force" algorithmic technique (tries all paths)
  - often implemented recursively

Applications:
- producing all permutations of a set of values
- parsing languages
- games: anagrams, crosswords, word jumbles, 8 queens
- combinatorics and logic programming
Backtracking algorithms

A general pseudo-code algorithm for backtracking problems:

Explore(\texttt{choices}):

- if there are no more \texttt{choices} to make: stop.
- else:
  - Make a single choice \texttt{C}.
  - Explore the remaining \texttt{choices}.
  - Un-make choice \texttt{C}, if necessary. (backtrack!)
Backtracking strategies

When solving a backtracking problem, ask these questions:

1. What are the "choices" in this problem?
   - What is the "base case"? (How do I know when I'm out of choices?)

2. How do I "make" a choice?
   - Do I need to create additional variables to remember my choices?
   - Do I need to modify the values of existing variables?

3. How do I explore the rest of the choices?
   - Do I need to remove the made choice from the list of choices?

4. Once I'm done exploring, what should I do?

5. How do I "un-make" a choice?
Exercise: Combinations

- Write a method `combinations` that accepts a string `s` and an integer `k` as parameters and outputs all possible `k`-letter words that can be formed from unique letters in that string. The arrangements may be output in any order.

- Example:
  
  `combinations("GOOGLE", 3)` outputs the sequence of lines at right.

- To simplify the problem, you may assume that the string `s` contains at least `k` unique characters.
Initial attempt

```java
public static void combinations(String s, int length) {
    combinations(s, "", length);
}

private static void combinations(String s, String chosen, int length) {
    if (length == 0) {
        System.out.println(chosen); // base case: no choices left
    } else {
        for (int i = 0; i < s.length(); i++) {
            String ch = s.substring(i, i + 1);
            if (!chosen.contains(ch)) {
                String rest = s.substring(0, i) + s.substring(i + 1);
                combinations(rest, chosen + ch, length - 1);
            }
        }
    }
}
```

- Problem: Prints same string multiple times.
public static void combinations(String s, int length) {
    Set<String> all = new TreeSet<String>();
    combinations(s, "", all, length);
    for (String comb : all) {
        System.out.println(comb);
    }
}

private static void combinations(String s, String chosen,
    Set<String> all, int length) {
    if (length == 0) {
        all.add(chosen); // base case: no choices left
    } else {
        for (int i = 0; i < s.length(); i++) {
            String ch = s.substring(i, i + 1);
            if (!chosen.contains(ch)) {
                String rest = s.substring(0, i) + s.substring(i + 1);
                combinations(rest, chosen + ch, all, length - 1);
            }
        }
    }
}
Exercise: Permutations

- Write a method `permute` that accepts a string as a parameter and outputs all possible rearrangements of the letters in that string. The arrangements may be output in any order.

- Example:
  ```java
  permute("TEAM")
  ```
  outputs the following sequence of lines:

<table>
<thead>
<tr>
<th>TEAM</th>
<th>ATEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEMA</td>
<td>ATME</td>
</tr>
<tr>
<td>TAEM</td>
<td>AETM</td>
</tr>
<tr>
<td>TAME</td>
<td>AEMT</td>
</tr>
<tr>
<td>TMEA</td>
<td>AMTE</td>
</tr>
<tr>
<td>TMAE</td>
<td>AMET</td>
</tr>
<tr>
<td>ETAM</td>
<td>MTEA</td>
</tr>
<tr>
<td>ETMA</td>
<td>MTAE</td>
</tr>
<tr>
<td>EATM</td>
<td>META</td>
</tr>
<tr>
<td>EAMT</td>
<td>MEAT</td>
</tr>
<tr>
<td>EMTA</td>
<td>MATE</td>
</tr>
<tr>
<td>EMAT</td>
<td>MAET</td>
</tr>
</tbody>
</table>
Decision tree

<table>
<thead>
<tr>
<th>chosen</th>
<th>available</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEAM</td>
<td></td>
</tr>
</tbody>
</table>

TEAM

TEAM

TEAM

TEAM

TEAM

TEAM

TEAM

TEAM

TEAM

TEAM

TEAM
Exercise solution

// Outputs all permutations of the given string.
public static void permute(String s) {
    permute(s, "");
}

private static void permute(String s, String chosen) {
    if (s.length() == 0) {
        // base case: no choices left to be made
        System.out.println(chosen);
    } else {
        // recursive case: choose each possible next letter
        for (int i = 0; i < s.length(); i++) {
            char c = s.charAt(i);
            // choose
            s = s.substring(0, i) + s.substring(i + 1);
            chosen += c;
            permute(s, chosen); // explore
            s = s.substring(0, i) + c + s.substring(i);
            chosen = chosen.substring(0, chosen.length() - 1); // un-choose
        }
    }
}