Algorithm Design Techniques

- Greedy
  - Shortest path, minimum spanning tree, ...
- Divide and Conquer
  - Divide the problem into smaller subproblems, solve them, and combine into the overall solution
  - Often done recursively
  - Quick sort, merge sort are great examples
- Dynamic Programming
  - Brute force through all possible solutions, storing solutions to subproblems to avoid repeat computation
- Backtracking
  - A clever form of exhaustive search

Dynamic Programming: Idea

- Divide a bigger problem into many smaller subproblems
- If the number of subproblems grows exponentially, a recursive solution may have an exponential running time 😞
- Dynamic programming to the rescue! 😊
- Often an individual subproblem may occur many times!
  - Store the results of subproblems in a table and re-use them instead of recomputing them
  - Technique called memoization

Fibonacci Sequence: Recursive

- The fibonacci sequence is a very famous number sequence
  - 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...
- The next number is found by adding up the two numbers before it.
- Recursive solution:
  ```java
  fib(int n) {
    if (n == 1 || n == 2) {
      return 1
    }
    return fib(n - 2) + fib(n - 1)
  }
  ```
- Exponential running time!
  - A lot of repeated computation

Fibonacci Sequence: memoized

```java
fib(int n):  
results = Map();  # Empty mapping container.  
results.put(1, 1)  
results.put(2, 1)  
return fibHelper(n, results)

fibHelper(int n, Map results):  
if (!results.contains(n)):  
  results.put(n, fibHelper(n-2)+fibHelper(n-1))  
return results.get(n)
```

Now each call of fib(x) only gets computed once for each x!
Comments

- Dynamic programming relies on working “from the bottom up” and saving the results of solving simpler problems
  - These solutions to simpler problems are then used to compute the solution to more complex problems
- Dynamic programming solutions can often be quite complex and tricky
- Dynamic programming is used for optimization problems, especially ones that would otherwise take exponential time
  - Only problems that satisfy the principle of optimality are suitable for dynamic programming solutions
  - i.e. the subsolutions of an optimal solution of the problem are themselves optimal solutions for their subproblems
- Since exponential time is unacceptable for all but the smallest problems, dynamic programming is sometimes essential

Algorithm Design Techniques

- Greedy
  - Shortest path, minimum spanning tree, …
- Divide and Conquer
  - Divide the problem into smaller subproblems, solve them, and combine into the overall solution
  - Often done recursively
  - Quick sort, merge sort are great examples
- Dynamic Programming
  - Brute force through all possible solutions, storing solutions to subproblems to avoid repeat computation
- Backtracking
  - A clever form of exhaustive search

Backtracking: Idea

- Backtracking is a technique used to solve problems with a large search space, by systematically trying and eliminating possibilities.
- A standard example of backtracking would be going through a maze.
  - At some point, you might have two options of which direction to go:

Backtracking

- One strategy would be to try going through Portion A of the maze. If you get stuck before you find your way out, then you “backtrack” to the junction.
  - At this point in time you know that Portion A will NOT lead you out of the maze, so you then start searching in Portion B

Backtracking (animation)
Backtracking

- Dealing with the maze:
  - From your start point, you will iterate through each possible starting move.
  - From there, you recursively move forward.
  - If you ever get stuck, the recursion takes you back to where you were, and you try the next possible move.

- Make sure you don’t try too many possibilities,
  - Mark which locations in the maze have been visited already so that no location in the maze gets visited twice.
  - (If a place has already been visited, there is no point in trying to reach the end of the maze from there again.

Backtracking: The 8 queens problem

- Find an arrangement of 8 queens on a single chess board such that no two queens are attacking one another.
- In chess, queens can move all the way down any row, column or diagonal (so long as no pieces are in the way).
  - Due to the first two restrictions, it’s clear that each row and column of the board will have exactly one queen.

Backtracking – 8 queens Analysis

- Another possible brute-force algorithm is generate all possible permutations of the numbers 1 through 8 (there are 8! = 40,320).
  - Use the elements of each permutation as possible positions in which to place a queen on each row.
  - Reject those boards with diagonal attacking positions.

- The backtracking algorithm does a bit better
  - Constructs the search tree by considering one row of the board at a time, eliminating most non-solution board positions at a very early stage in their construction.
  - Because it rejects row and diagonal attacks even on incomplete boards, it examines only 15,720 possible queen placements.

  - 15,720 is still a lot of possibilities to consider
    - Sometimes we have no other choice but to do the best we can 😊

Algorithm Design Techniques

- Greedy
  - Shortest path, minimum spanning tree, …

- Divide and Conquer
  - Divide the problem into smaller subproblems, solve them, and combine into the overall solution
  - Often done recursively
  - Quick sort, merge sort are great examples

- Dynamic Programming
  - Brute force through all possible solutions, storing solutions to subproblems to avoid repeat computation

- Backtracking
  - A clever form of exhaustive search