DYNAMIC PROGRAMMING

Introduction

Simple recursive algorithms I

- A simple recursive algorithm:
  - Solves the base cases directly
  - Recurs with a simpler subproblem
  - Does some extra work to convert the solution to the simpler subproblem into a solution to the given problem
- I call these "simple" because several of the other algorithm types are inherently recursive

Divide and Conquer

- A divide and conquer algorithm consists of two parts:
  - Divide the problem into smaller subproblems of the same type, and solve these subproblems recursively
  - Combine the solutions to the subproblems into a solution to the original problem
- Traditionally, an algorithm is only called "divide and conquer" if it contains at least two recursive calls

Greedy algorithms

- An optimization problem is one in which you want to find, not just *a* solution, but the *best* solution
- A "greedy algorithm" sometimes works well for optimization problems
- A greedy algorithm works in phases: At each phase:
  - You take the best you can get right now, without regard for future consequences
  - You hope that by choosing a *local* optimum at each step, you will end up at a *global* optimum

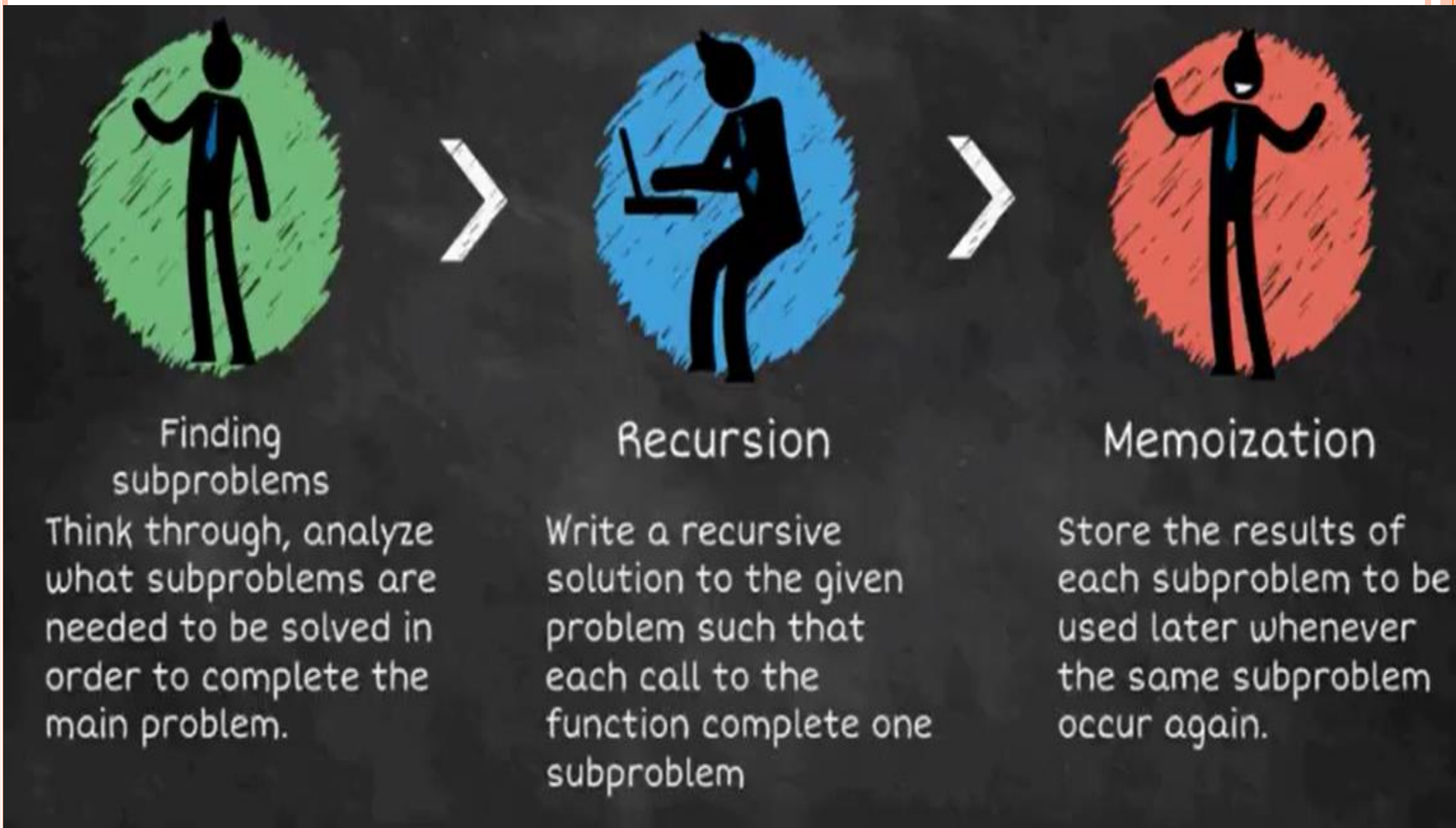
Example: Counting money

- Suppose you want to count out a certain amount of money, using the fewest possible bills and coins
- A greedy algorithm would do this would be: At each step, take the largest possible bill or coin that does not overshoot
  - Example: To make $6.39, you can choose:
    - a $5 bill
    - a $1 bill, to make $6
    - a 25¢ coin, to make $6.25
    - A 10¢ coin, to make $6.35
    - four 1¢ coins, to make $6.39
- For US money, the greedy algorithm always gives the optimum solution

Dynamic Programming

- Dynamic Programming(DP) is a method for solving complex problems by breaking them down into simpler sub problems".
- Also refer to as "Smart Recursion" or "Intelligent brute-force"
- For example DP is like building a wall brick by brick. Here each brick is a sub-problem and building a wall is a complex problem.
- In DP we will not repeat a sub problem we memorize the result

To write a Dynamic Programming solution



Dynamic programming algorithms

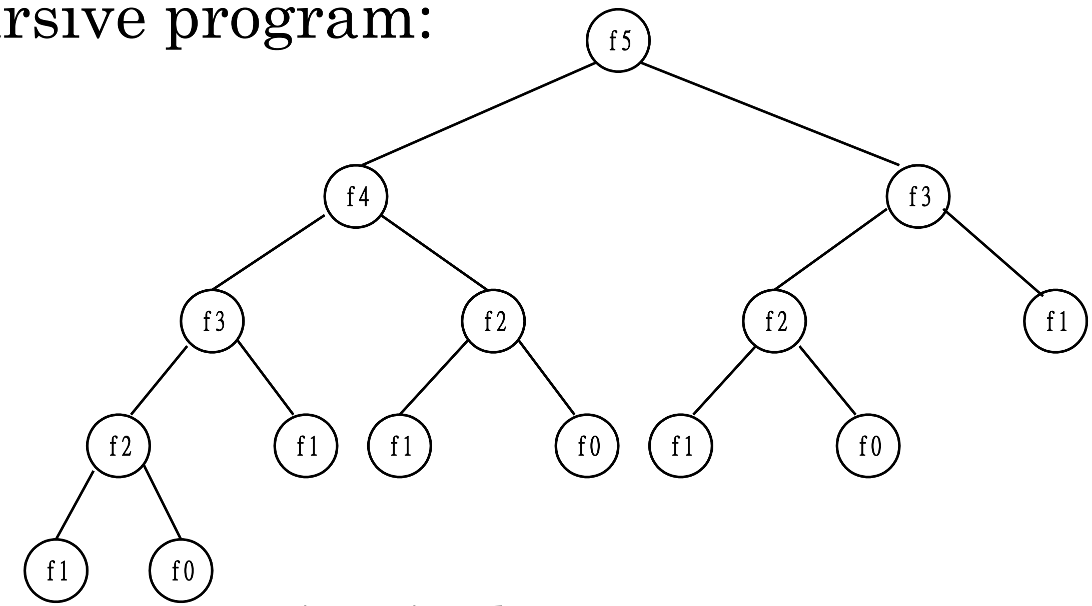
- A dynamic programming algorithm remembers past results and uses them to find new results
- Dynamic programming is generally used for optimization problems
  - Multiple solutions exist, need to find the “best” one
  - Requires “optimal substructure” and “overlapping subproblems”
    - Optimal substructure: Optimal solution contains optimal solutions to subproblems
    - Overlapping subproblems: Solutions to subproblems can be stored and reused in a bottom-up fashion
- This differs from Divide and Conquer, where subproblems generally need not overlap

FIBONACCI SEQUENCE

- Fibonacci sequence: 0 , 1 , 1 , 2 , 3 , 5 , 8 , 13 , 21 , …

  $F_i = i$ if $i \le 1$

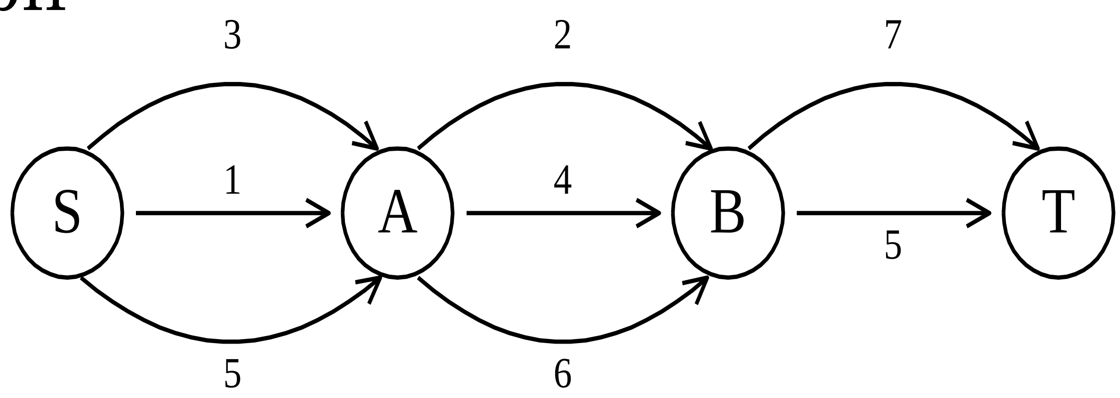
  $F_i = F_{i-1} + F_{i-2}$ if $i \ge 2$
- Solved by a recursive program:



- Much replicated computation is done.
- It should be solved by a simple loop.

The shortest path

- To find a shortest path in a multi-stage graph



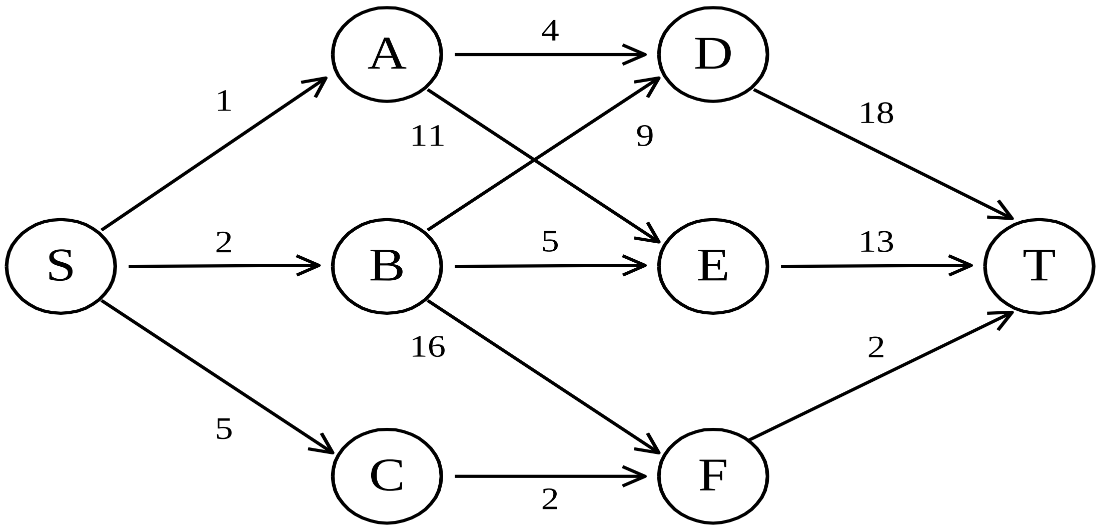
- Apply the greedy method :

  the shortest path from S to T :

  $$1 + 2 + 5 = 8$$

THE SHORTEST PATH IN MULTISTAGE GRAPHS

- e.g.



- The greedy method can not be applied to this case: (S, A, D, T) 1+4+18 = 23.
- The real shortest path is:

  (S, C, F, T) 5+2+2 = 9.

INTRODUCTION

- Dynamic Programming is an algorithm design technique for ***optimization problems:*** often minimizing or maximizing.

- Solves problems by combining the solutions to subproblems that contain common sub-sub-problems.

- DP can be applied when the solution of a problem includes solutions to subproblems
- We need to find a recursive formula for the solution
- We can recursively solve subproblems, starting from the trivial case, and save their solutions in memory
- In the end we'll get the solution of the whole problem

STEPS

- Steps to Designing a Dynamic Programming Algorithm
- 1. Characterize optimal sub-structure
- 2. Recursively define the value of an optimal solution
- 3. Compute the value bottom up
- 4. (if needed) Construct an optimal solution

Diff. B/w Dynamic Programming and Divide & Conquer:

- **Divide-and-conquer** algorithms split a problem into separate subproblems, solve the subproblems, and combine the results for a solution to the original problem.
  - Example: Quicksort, Mergesort, Binary search
- **Divide-and-conquer** algorithms can be thought of as top-down algorithms

- **Dynamic Programming** split a problem into subproblems, some of which are common, solve the subproblems, and combine the results for a solution to the original problem.
  - Example: Matrix Chain Multiplication, Longest Common Subsequence
- **Dynamic programming** can be thought of as bottom-up

Diff. B/w Dynamic Programming and Divide & Conquer (Cont…):

- In divide and conquer, subproblems are independent.
- Divide & Conquer solutions are simple as compared to Dynamic programming .
- Divide & Conquer can be used for any kind of problems.
- Only one decision sequence is ever generated

- In Dynamic Programming , subproblems are not independent.
- Dynamic programming solutions can often be quite complex and tricky.
- Dynamic programming is generally used for Optimization Problems.
- Many decision sequences may be generated.

PRINCIPLE OF OPTIMALITY

- Principle of optimality: Suppose that in solving a problem, we have to make a sequence of decisions $D_1, D_2, \ldots, D_n$. If this sequence is optimal, then the last k decisions, $1 < k < n$ must be optimal.
- e.g. the shortest path problem

  If $i, i_1, i_2, \ldots, j$ is a shortest path from i to j, then $i_1, i_2, \ldots, j$ must be a shortest path from i_1 to j
- In summary, if a problem can be described by a multistage graph, then it can be solved by dynamic programming.

Dynamic programming

- Forward approach and backward approach:
  - Note that if the recurrence relations are formulated using the forward approach then the relations are solved backwards . i.e., beginning with the last decision
  - On the other hand if the relations are formulated using the backward approach, they are solved forwards.
- To solve a problem by using dynamic programming:
  - Find out the recurrence relations.
  - Represent the problem by a multistage graph.

- Multistage graph
- Computing a binomial coefficient
- Matrix-chain multiplication
- Longest Common Subsequence
- 0/1 Knapsack
- The Traveling Salesperson Problem
- Warshall’s algorithm for transitive closure
- Floyd’s algorithm for all-pairs shortest paths