

# CMSC 132: Object-Oriented Programming II

---



## Algorithm Strategies

Department of Computer Science  
University of Maryland, College Park

1

## General Concepts

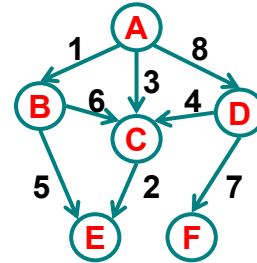
- **Algorithm strategy**
  - Approach to solving a problem
  - May combine several approaches
  
- **Algorithm structure**
  - Iterative      ⇒ execute action in loop
  - Recursive     ⇒ reapply action to subproblem(s)
  
- **Problem type**

2

## Problem Type

### ■ Satisfying

- Find any satisfactory solution
- Example → Find path from A to F



### ■ Optimization

- Find **best** solution (vs. cost metric)
- Example → Find **shortest** path from A to E

3

## Some Algorithm Strategies

- Recursive algorithms
- Backtracking algorithms
- Divide and conquer algorithms
- Dynamic programming algorithms
- Greedy algorithms
- Brute force algorithms
- Branch and bound algorithms
- Heuristic algorithms

4

## Recursive Algorithm

- Based on reapplying algorithm to subproblem
- Approach
  1. Solves **base case(s)** directly
  2. Recurs with a simpler subproblem
  3. May need to convert solution(s) to subproblems

5

## Backtracking Algorithm

- Based on **depth-first** recursive search
- Approach
  1. Tests whether solution has been found
  2. If found solution, return it
  3. Else for each choice that can be made
    - a) Make that choice
    - b) Recur
    - c) If recursion returns a solution, return it
  4. If no choices remain, return failure
- Tree of possible solutions → **search tree**

6

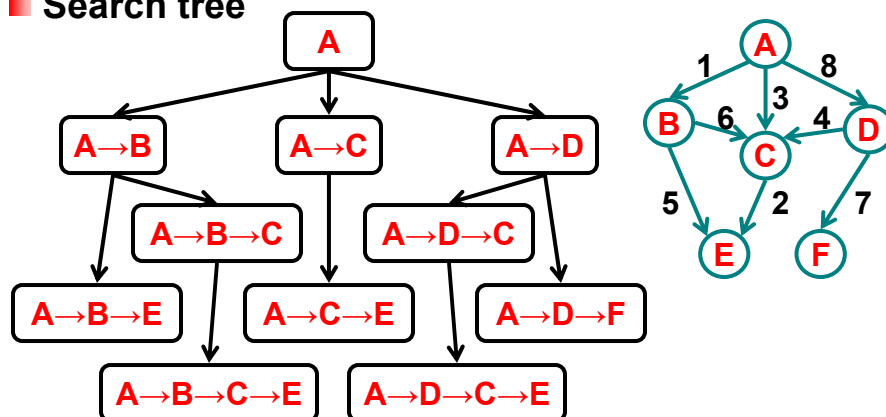
## Backtracking Algorithm – Reachability

- Find path in graph from A to F
  1. Start with currentNode = A
  2. If currentNode has edge to F, return path
  3. Else select neighbor node X for currentNode
    - Recursively find path from X to F
      - If path found, return path
      - Else repeat for different X
    - Return false if no path from any neighbor X

7

## Backtracking Algorithm – Path Finding

- Search tree



- Internal nodes → partial solutions
- Leaves → complete solutions

8

## Backtracking Algorithm – Map Coloring

- **Color a map with no more than four colors**
  - If all countries have been colored return success
  - Else for each color c of four colors and country n
    - If country n is not adjacent to a country that has been colored c
      - Color country n with color c
      - Recursively color country n+1
      - If successful, return success
  - Return failure

9

## Divide and Conquer

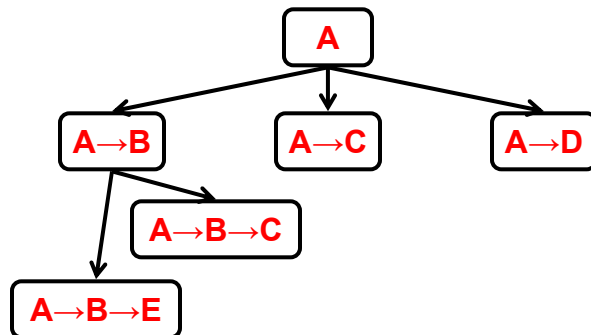
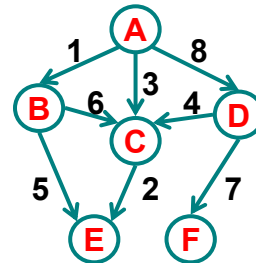
- Based on dividing problem into subproblems
- Approach
  1. Divide problem into smaller subproblems
    - Subproblems must be of same type
    - Subproblems do not need to overlap
  2. Solve each subproblem recursively
  3. Combine solutions to solve original problem
- Usually contains two or more recursive calls

10

## Divide and Conquer – Shortest Path

### ■ Example

1. Divide into subproblem for each neighboring node
2. Solve subproblems (recursively)
3. Combine by comparing costs



11

## Divide and Conquer – Sorting

### ■ Quicksort

- Partition array into **two parts** around pivot
- Recursively quicksort each part of array
- Concatenate solutions

### ■ Mergesort

- Partition array into **two parts**
- Recursively mergesort each half
- Merge two sorted arrays into single sorted array

12

## Dynamic Programming Algorithm

- Based on remembering past results
- Approach
  1. Divide problem into smaller subproblems
    - Subproblems must be of same type
    - Subproblems must **overlap**
  2. Solve each subproblem recursively
    - May simply look up solution (if previously solved)
  3. Combine solutions into to solve original problem
  4. Store solution to problem
- Generally applied to optimization problems

13

## Fibonacci Algorithm

- Fibonacci numbers
  - fibonacci(0) = 1
  - fibonacci(1) = 1
  - fibonacci(n) = fibonacci(n-1) + fibonacci(n-2)
- Recursive algorithm to calculate fibonacci(n)
  - If n is 0 or 1, return 1
  - Else compute fibonacci(n-1) and fibonacci(n-2)
  - Return their sum
- Simple algorithm  $\Rightarrow$  exponential time  $O(2^n)$

14

## Dynamic Programming – Fibonacci

- Dynamic programming version of fibonacci(n)
  - If n is 0 or 1, return 1
  - Else solve fibonacci(n-1) and fibonacci(n-2)
    - Look up value if previously computed
    - Else recursively compute
  - Find their sum and store
  - Return result
- Dynamic programming algorithm  $\Rightarrow O(n)$  time
  - Since solving fibonacci(n-2) is just looking up value

15

## Dynamic Programming – Shortest Path

### Dijkstra's Shortest Path Algorithm

$S = \emptyset$

$C[X] = 0$

$C[Y] = \infty$  for all other nodes

while ( not all nodes in S )

find node K not in S with smallest  $C[K]$

add K to S

for each node M not in S adjacent to K

$C[M] = \min ( C[M] , C[K] + \text{cost of } (K,M) )$

Stores results of  
smaller subproblems

16

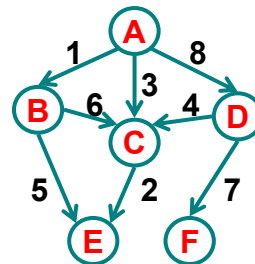
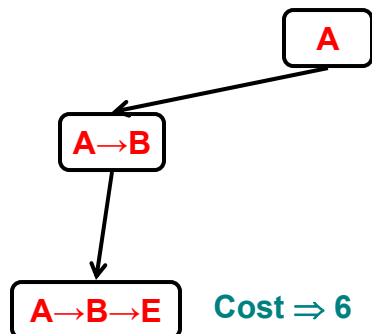
## Greedy Algorithm

- Based on trying best current (local) choice
- Approach
  - At each step of algorithm
  - Choose best local solution
- Avoid backtracking, exponential time  $O(2^n)$
- Hope local optimum lead to **global** optimum

17

## Greedy Algorithm – Shortest Path

- Example
  - Choose lowest-cost neighbor



- Does not yield overall (global) shortest path

18

## Greedy Algorithm – MST

### **Kruskal's Minimal Spanning Tree Algorithm**

sort edges by weight (from least to most)

tree =  $\emptyset$

for each edge (X,Y) **in order**

if it does not create a cycle

add (X,Y) to tree

stop when tree has N-1 edges



**Picks best  
local solution  
at each step**

19

## Brute Force Algorithm

■ Based on trying all possible solutions

■ Approach

■ Generate and evaluate possible solutions until

■ Satisfactory solution is found

■ Best solution is found (if can be determined)

■ All possible solutions found

■ Return best solution

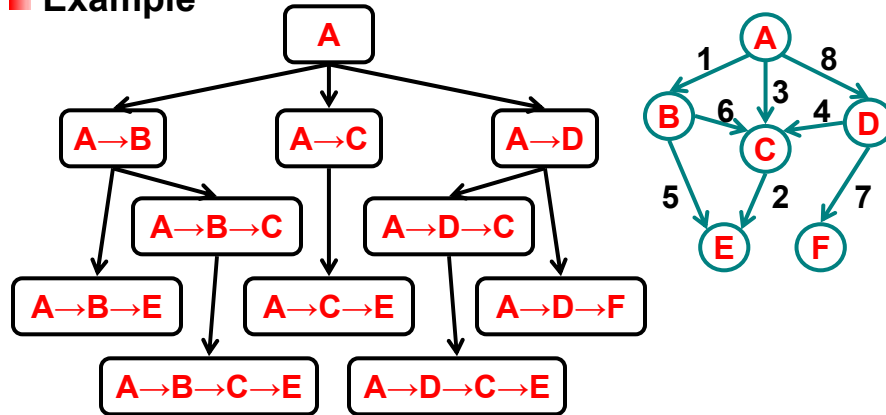
■ Return failure if no satisfactory solution

■ Generally most expensive approach

20

## Brute Force Algorithm – Shortest Path

### ■ Example



- Examines all paths in graph

21

## Brute Force Algorithm – TSP

### ■ Traveling Salesman Problem (TSP)

- Given weighted undirected graph (map of cities)
- Find lowest cost path visiting all nodes (cities) once
- No known polynomial-time general solution

### ■ Brute force approach

- Find all possible paths using recursive backtracking
- Calculate cost of each path
- Return lowest cost path

### ■ Requires exponential time $O(2^n)$

22

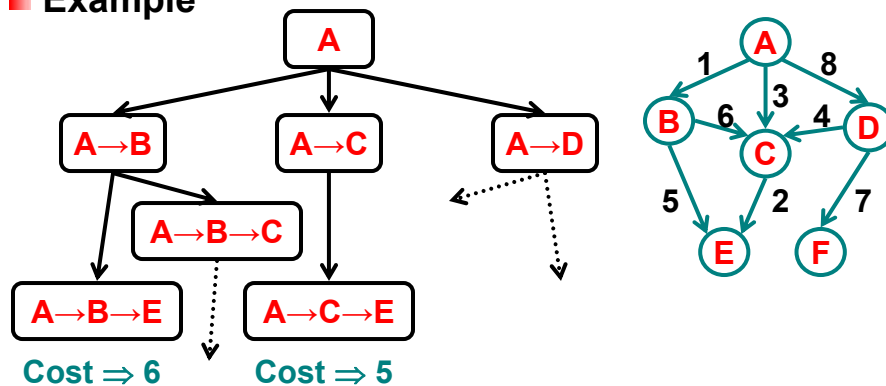
## Branch and Bound Algorithm

- Based on limiting search using current solution
- Approach
  - Track best current solution found
  - Eliminate (**prune**) partial solutions that can not improve upon best current solution
- Reduces amount of backtracking
  - Not guaranteed to avoid exponential time  $O(2^n)$

23

## Branch & Bound Alg. – Shortest Path

### ■ Example



- Pruned paths beginning with A→B→C & A→D

24

## Branch and Bound – TSP

- **Branch and bound algorithm for TSP**
  - Find possible paths using recursive backtracking
  - Track cost of best current solution found
  - Stop searching path if cost > best current solution
  - Return lowest cost path
- If good solution found early, can reduce search
- May still require exponential time  $O(2^n)$

25

## Heuristic Algorithm

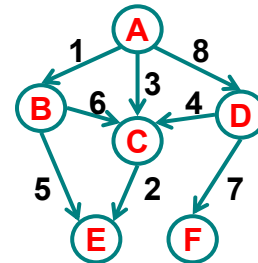
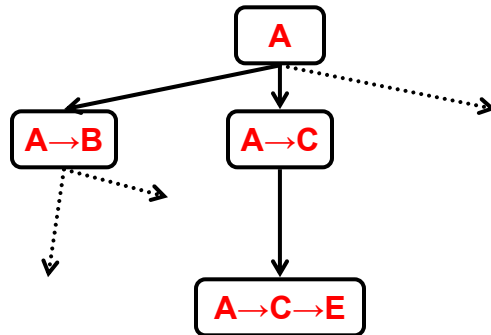
- Based on trying to guide search for solution
- Heuristic  $\Rightarrow$  “rule of thumb”
- Approach
  - Generate and evaluate possible solutions
    - Using “rule of thumb”
    - Stop if satisfactory solution is found
- Can reduce complexity
- Not guaranteed to yield best solution

26

## Heuristic – Shortest Path

### ■ Example

- Try only edges with cost < 5



- Worked...in this case

27

## Heuristic Algorithm – TSP

### ■ Heuristic algorithm for TSP

- Find possible paths using recursive backtracking
  - Search 2 lowest cost edges at each node first
- Calculate cost of each path
- Return lowest cost path from first 100 solutions
- Not guaranteed to find best solution
- Heuristics used frequently in real applications

28

## Summary

- **Wide range of strategies**
- **Choice depends on**
  - **Properties of problem**
  - **Expected problem size**
  - **Available resources**