CMSC 132:
Object-Oriented Programming II

Algorithm Strategies

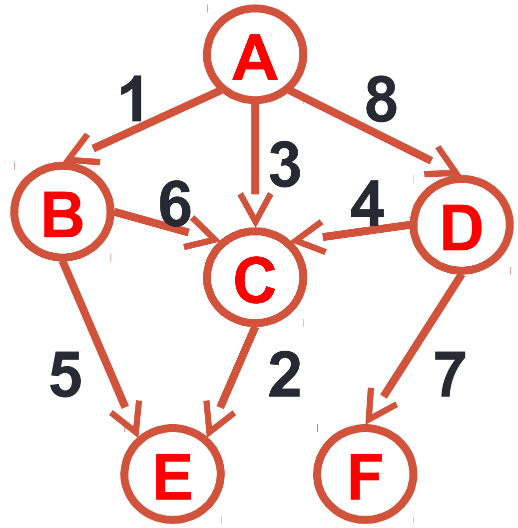
Department of Computer Science
University of Maryland, College Park

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**

Problem Type

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



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

Recursive Algorithm

- Based on reapplying algorithm to subproblem
- Approach
  - Solves base case(s) directly
  - Recurs with a simpler subproblem
  - May need to combine solution(s) to subproblems

Backtracking Algorithm

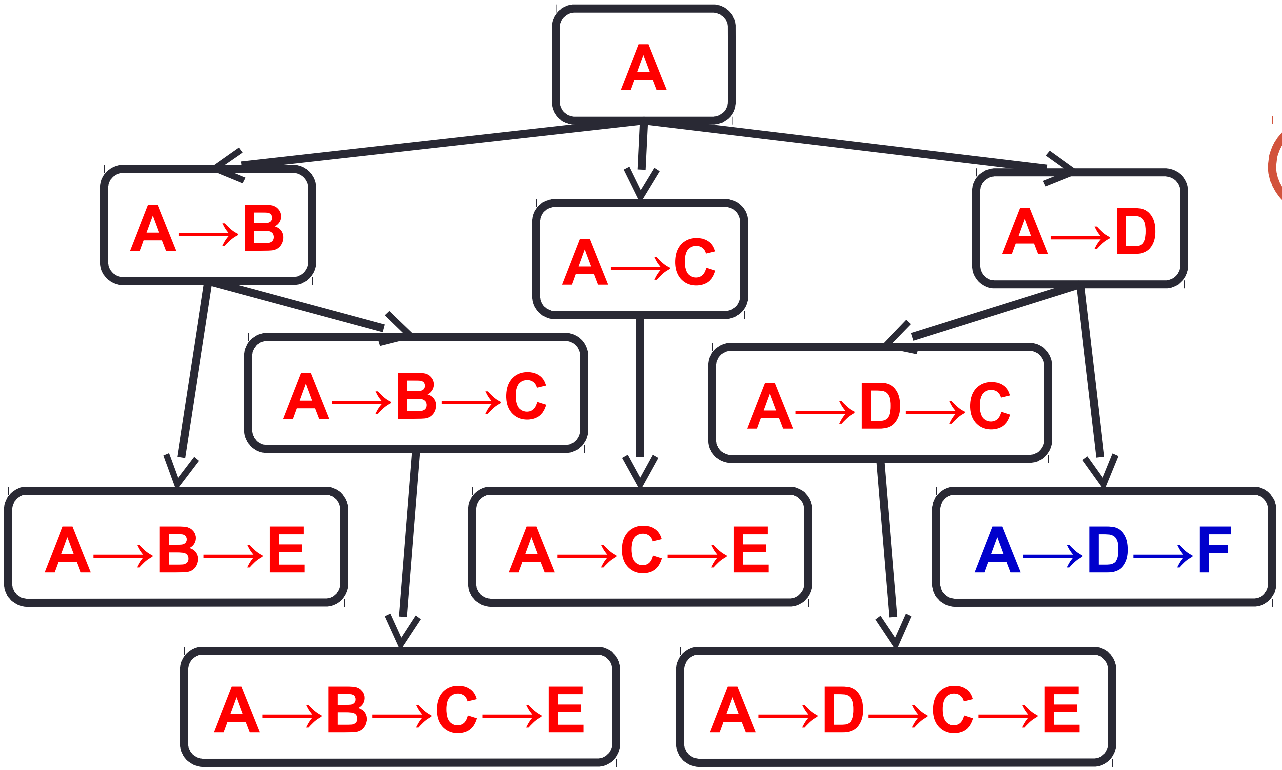
- Based on depth-first recursive search
- Approach
  - Tests whether solution has been found
  - If found solution, return it
  - Else for each choice that can be made
    - Make that choice
    - Recur
    - If recursion returns a solution, return it
  - If no choices remain, return failure
- Tree of alternatives → search tree

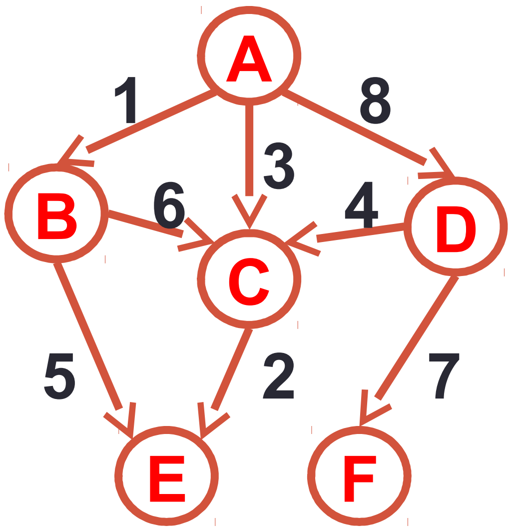
Backtracking Algorithm - Reachability

- Find path in graph from **A** to **F**
  - Start with currentNode = A
  - If currentNode has edge to F, return path
  - 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

Backtracking Algorithm – Path Finding

- Search tree (path A to F)





Backtracking Algorithm – Map Coloring

- Color a map using four colors so adjacent regions do not share the same color.
- Coloring map of countries
  - 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
- Map from wikipedia http://upload.wikimedia.org/wikipedia/commons/thumb/a/a5/Map_of_USA_with_state_names.svg/650px-Map_of_USA_with_state_names.svg.png

Divide and Conquer

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

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

Dynamic Programming Algorithm

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

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 □ exponential time O(2n)

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 → O(n) time
  - Since solving fibonacci(n-2) is just looking up value

Dynamic Programming – Shortest Path

Djikstra's Shortest Path Algorithm

S = ∅

C[X] = 0

C[Y] = ∞ 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] + cost of (K,M))

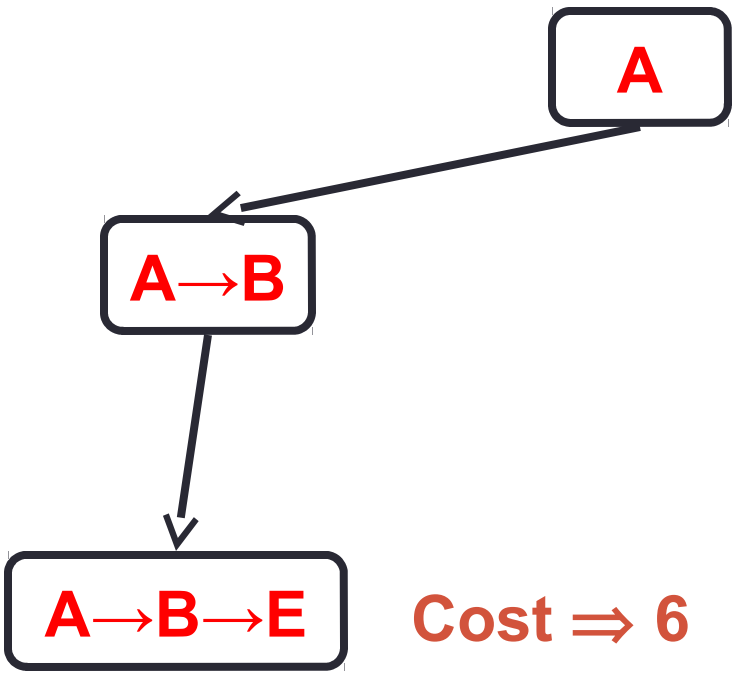
Stores results of smaller subproblems

Greedy Algorithm

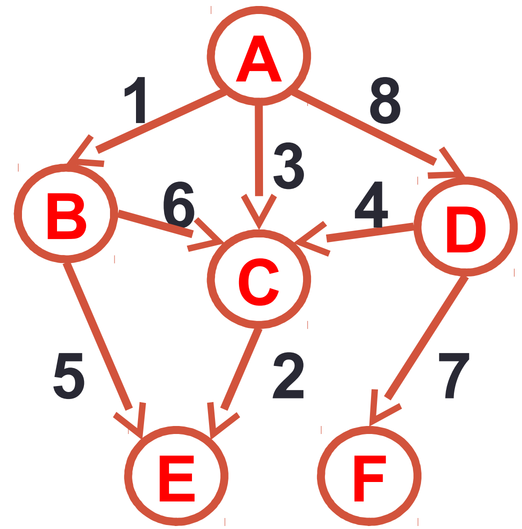
- Based on trying best current (local) choice
- Approach
  - At each step of algorithm
  - Choose best local solution
- Avoid backtracking, exponential time O(2n)
- Hope local optimum lead to global optimum
- Example: Coin System
  - Coins – 30 20 15 1
  - Find minimum number of coins for 40
  - Greedy Algorithm fails

Greedy Algorithm – Shortest Path

- Example (Shortest Path from A to E)
  - Choose lowest-cost neighbor



Cost ⇒ 6



- Does not yield overall (global) shortest path

Greedy Algorithm – MST

Kruskal's Minimal Spanning Tree Algorithm

sort edges by weight (from least to most)

tree = ∅

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

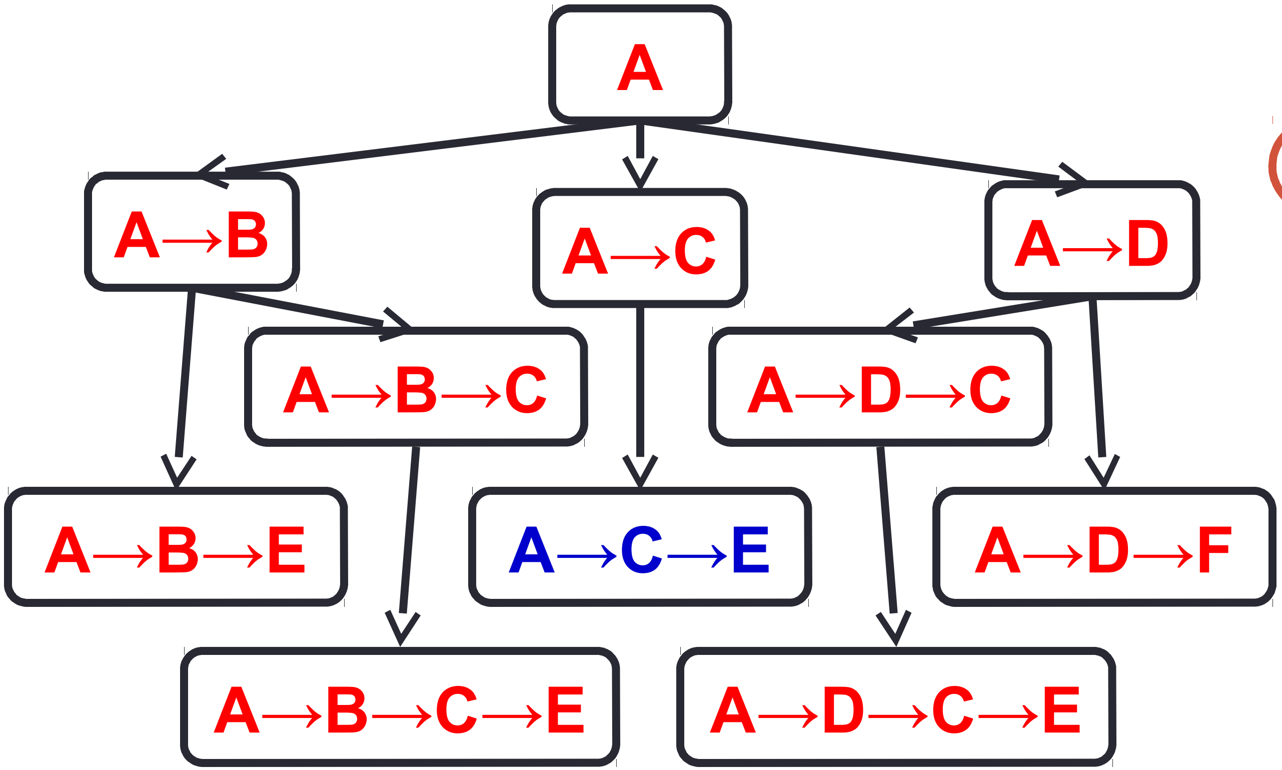
Picks best local solution at each step

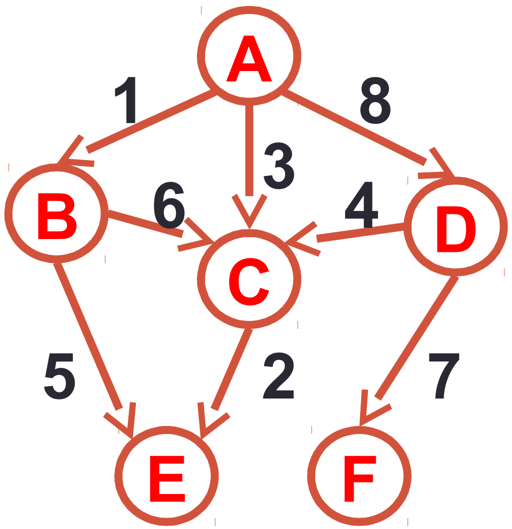
Brute Force Algorithm

- 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

Brute Force Algorithm – Shortest Path

- Example (From A to E)





Brute Force Algorithm – TSP

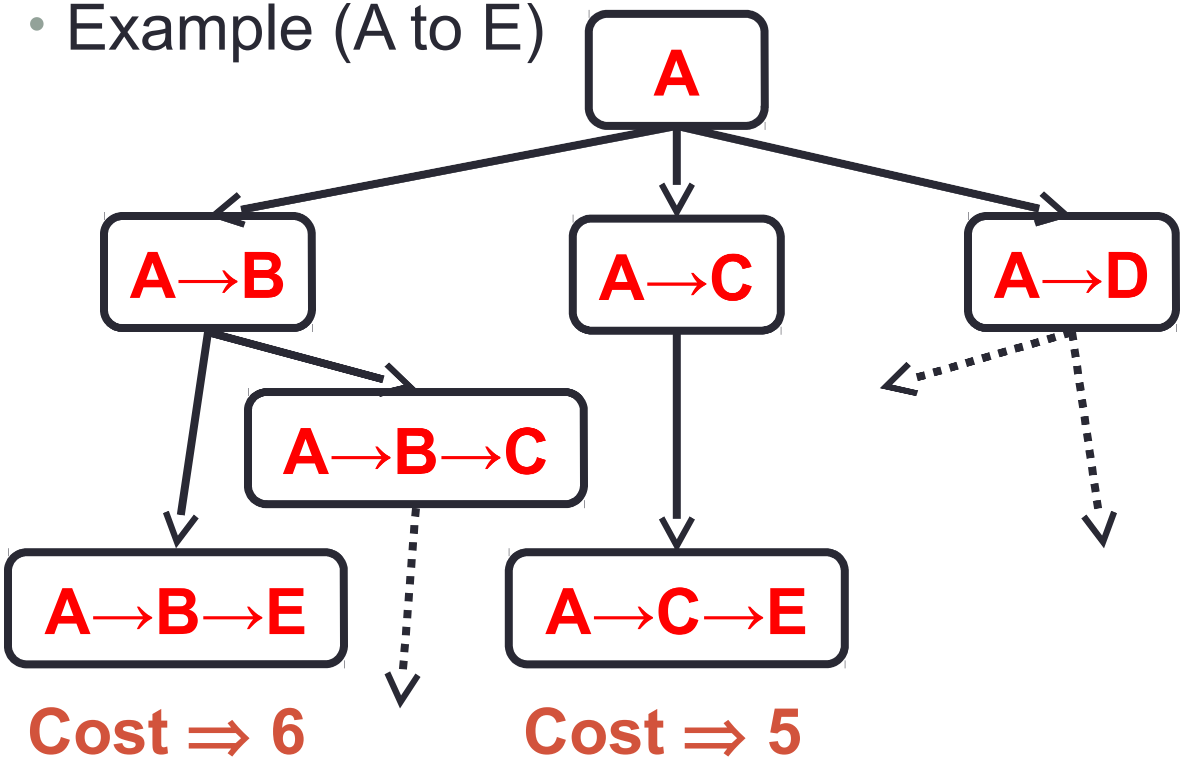
- Traveling Salesman Problem (TSP)
  - Given weighted undirected graph (map of cities)
  - Find lowest cost path visiting all nodes (cities) once
  - No known polynomial-time general solution
- Brute force approach
  - Find all possible paths using recursive backtracking
  - Calculate cost of each path
  - Return lowest cost path
  - Complexity O(n!)

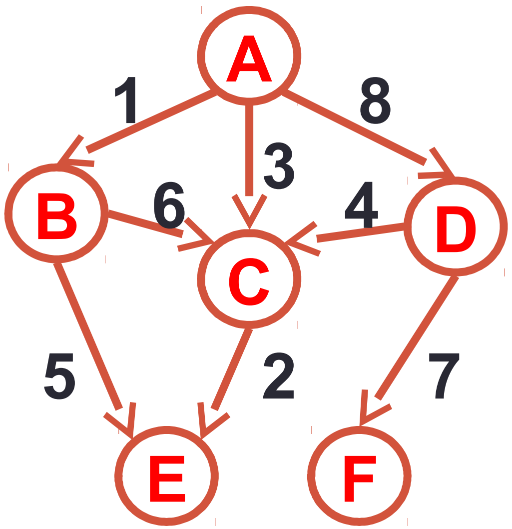
Branch and Bound Algorithm

- 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 $O(2n)$

Branch & Bound Alg. – Shortest Path

- Example (A to E)





- Starting with A →B →E

Branch and Bound – TSP

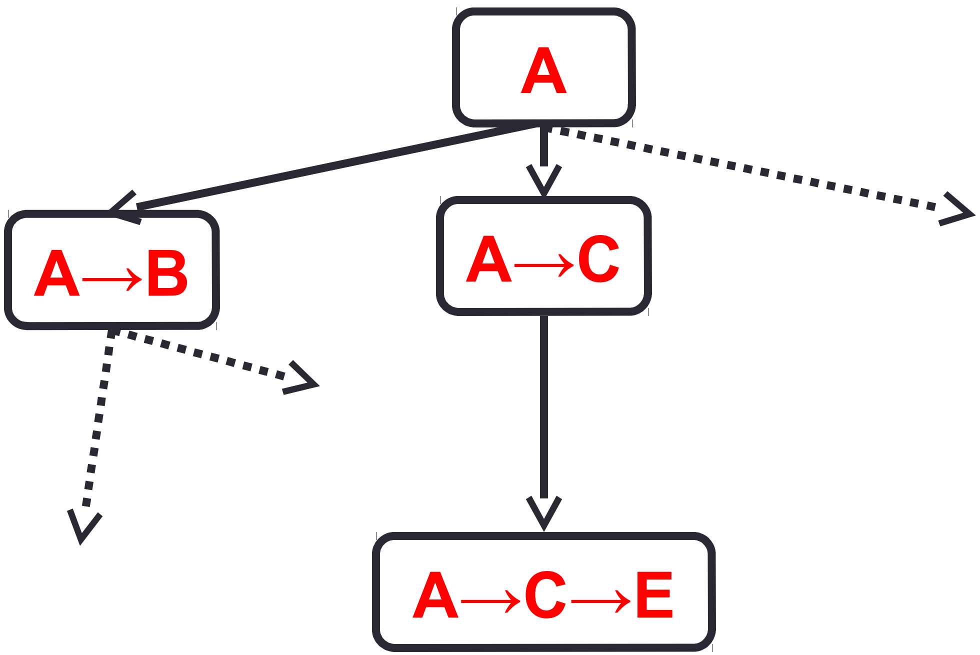
- Branch and bound algorithm for TSP
  - Find possible paths using recursive backtracking
  - Track cost of best current solution found
  - Stop searching path if cost > best current solution
  - Return lowest cost path
- If good solution found early, can reduce search
- May still require exponential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

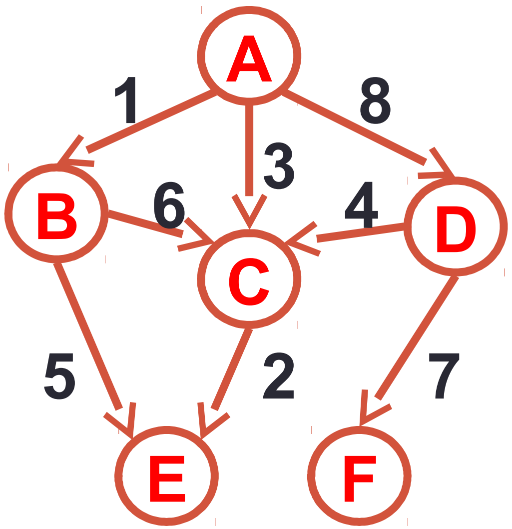
Heuristic Algorithm

- 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

Heuristic – Shortest Path

- Example (A to E)
  - Try only edges with cost < 5





- Worked…in this case

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

Summary

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