

# Algorithm Strategies

---



**Fawzi Emad**  
**Chau-Wen Tseng**

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**
  - Satisfying    ⇒ find any satisfactory solution
  - Optimization ⇒ find **best** solutions (vs. cost metric)

## 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
  1. Solves **base case(s)** directly
  2. Recurs with a simpler subproblem
  3. May need to convert solution(s) to subproblems

## Recursive Algorithm – Examples

- **To count elements in list**
  - If list is empty, return 0
  - Else skip 1<sup>st</sup> element and **recur** on remainder of list
  - Add 1 to result
- **To find element in list**
  - If list is empty, return false
  - Else if first element in list is given value, return true
  - Else skip 1<sup>st</sup> element and **recur** on remainder of list

## 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
- **Some times called “search tree”**

## Backtracking Algorithm – Example

- Find path through maze
  - Start at beginning of maze
  - If at exit, return true
  - Else for each step from current location
    - Recursively find path
    - Return with first successful step
    - Return false if all steps fail

## Backtracking Algorithm – Example

- 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

## 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**

## Divide and Conquer – Examples

- **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
  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
  3. Combine solutions into to solve original problem
  4. 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  $\Rightarrow$  exponential time  $O(2^n)$

## Dynamic Programming – Example

- 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

## Dynamic Programming – Example

### 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

## 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

## Greedy Algorithm – Example

### 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

## 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 – Example

- 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  $O(2^n)$

## Branch and Bound Algorithm

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

## Branch and Bound – Example

- 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  $O(2^n)$

## 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 Algorithm – Example

- 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**