PH502: Scientific Programming Concepts

Irish Centre for High End Computing (ICHEC)

September 23, 2020

Overview

- In this lecture we will continue to examine algorithms.
- We have already mentioned that speed and memory usage can be correlated inversely.
- However also some designs are simpler to code but maybe inefficient.

Designing Algorithms

- Insertion sort uses an *Incremental approach*: Sort array $A[1, ..., j-1]$, insert $A[j]$, get the sorted array $A[1, ..., j]$.
- Design Techniques:
  - Brute-Force
  - Divide and Conquer
  - Greedy Method
  - Backtracking
  - Dynamic Programming

Brute-Force

ICHEC
Irish Centre for High-End Computing

- Trivial naive method that tries every possible solution and check which is the best one
- Straightforward and easiest approach to apply
- Useful for solving small size problems, should always reach the solution.
- Not always efficient
- Very useful when writing a test routine to check the correctness of more efficient algorithms
- Examples:
  - Iterative Algorithm for Factorial
  - Linear search
  - Selection sort
  - Bubble sort

Divide and Conquer

ICHEC
Irish Centre for High-End Computing

- Break up problem into smaller parts of the same problem until each is small enough to be easily solved.
- Solve each part recursively.
- Combine solutions to sub-problems into overall solution.
- It is important that the subproblems are independent.
- Examples:
  - Recursive Algorithm for Factorial
  - Binary Search
  - Merge Sort

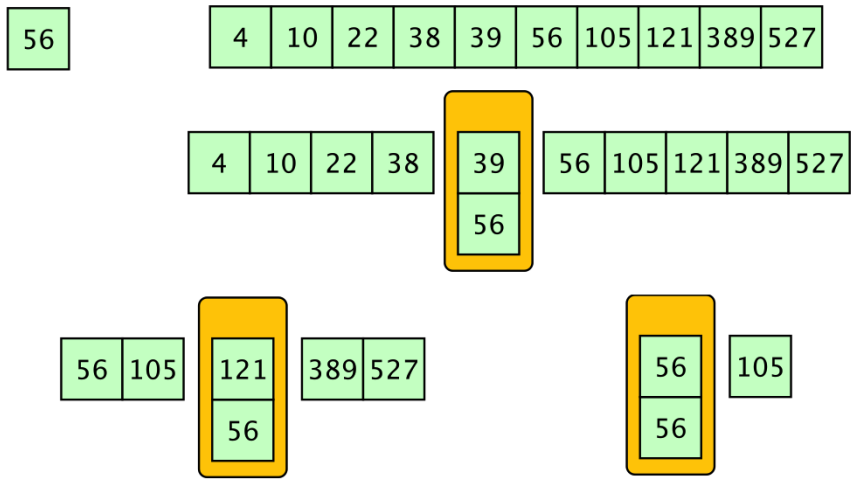
Example: Binary Search

Input: A list of integers sorted in ascending order $A = \{a_1, a_2, ..., a_n\}$, k

Output: True is the integer is found; False otherwise.

Procedure: Select the middle element on the list and compare; if not found, discard half of the list where the element is definitely not placed.

```
begin=1, end=n
while begin ≤ end do
    mid=⌊((begin+end)/2)⌋
    if k=A[mid] then
        return TRUE
    else if k<A[mid] then
        end=mid-1
    else
        end=mid+1
    end if
end while
return FALSE
```



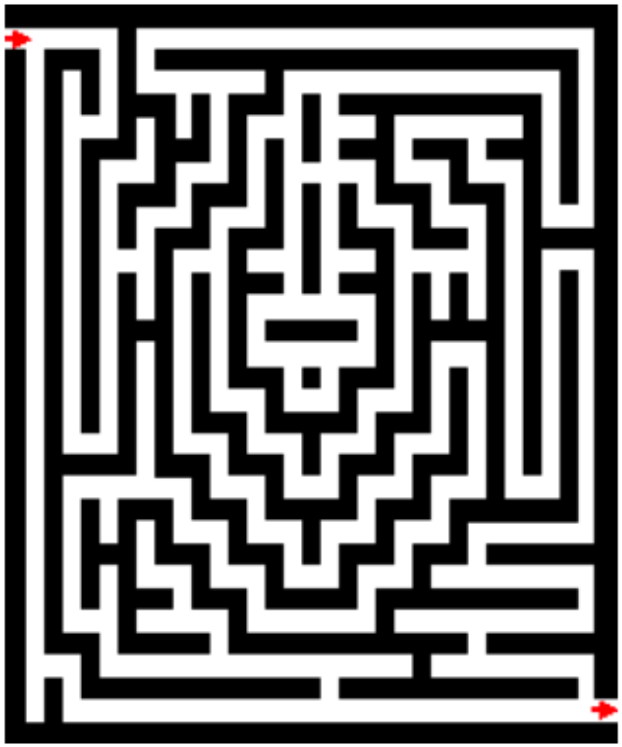
Greedy Method

- The solution is constructed through a sequence of steps. At each step we choose the locally optimal solution.
- Always makes the choice that looks best at the moment and adds it to the current subsolution.
- Simple and straightforward, Easy to invent, easy to implement and most of the time quite efficient
- Many problems cannot be solved correctly by greedy approach
- Mainly used to solve optimization problems
- Examples:
  - Job scheduling problems
  - The Knapsack problem: A thief robbing a store and can carry a maximal weight of w into their knapsack. There are n items and i^{th} item weigh w_i and is worth v_i dollars. What items should thief take?
  - The coin exchange problem: Pay money back to customer using fewest number of coins

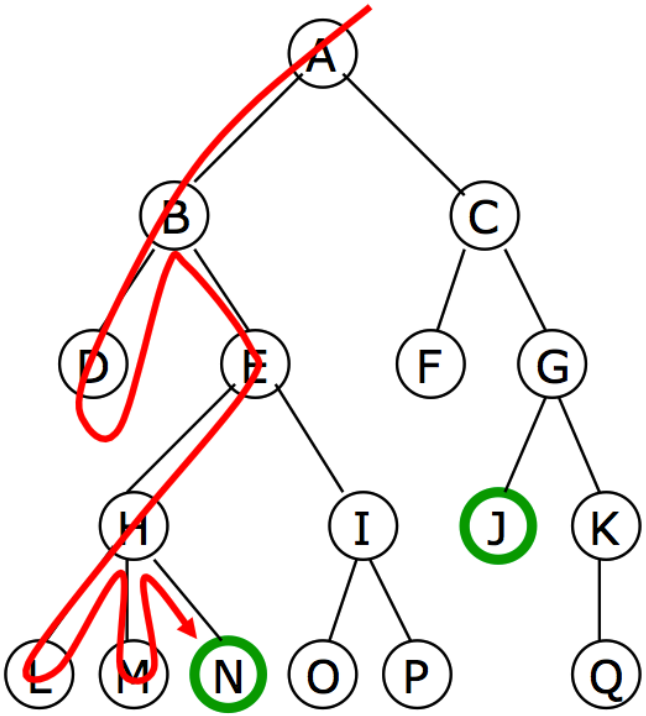
Backtracking

- A choice taken may be wrong, in which case the prospective solution is "backtracked" to undo the mistake
- A methodical way of trying out various sequences of decisions, until you find one that "works"
- Example: Maze Path Finder

while not at destination **do**
  Choose a path X
  if You have been in X **then**
    Choose another path Y
    X←Y
  end if
  Move to X
end while



Example: Depth-First Searching



- It is a combination of brute-force and backtracking.
- It starts at the root and explores nodes from there, looking for a goal node
- It explores a path all the way to a leaf before backtracking and exploring another path
- Node are explored in the order A B D E H L M **N** I O P C F G **J**

Dynamic Programming

- An inverse divide-and-conquer approach
- The smallest sub-problems are firstly solved and their solutions cached;
- The final solution arises from the solutions to each sub-problem
- A bottom-up approach
- Difference from the classical Divide and Conquer is subproblems are solved only once and solutions are stored for reuse.
- Applications: Bioinformatics, Control theory, Information theory, Operations research, ...
- Examples:
  - Viterbi Algorithm for Markov Models
  - Bellman-Ford for finding shortest path in networks
  - The Manhattan Tourist Problem

Example: Fibonacci Sequence

- **Definition:** The first two elements in the sequence are respectively 0 and 1. Every single element is the sum of the previous two elements in the sequence.

$$F_n = F_{n-1} + F_{n-2}, F_0 = 0, F_1 = 1$$

Dynamic Programming:

```
DynFib(n)
Table[0]=0
Table[1]=1
for i=2 to n do
   Table[i]=Table[i-1]+Table[i-2]
end for
return Table[n]
```

Divide and Conquer:

```
DivConFib(n)
if n >= 2 then
   return DivConFib(n-1)+DivConFib(n-2)
end if
return
```