CSA0695- Design and Analysis of Algorithms for Open Addressing Techniques

Minimizing Job Completion Time

Imagine you have a list of jobs, each with its own duration, and you need to complete them all as quickly as possible. This seemingly simple problem can be surprisingly tricky, especially when considering dependencies between jobs.

By:
D TEJA (192210626)

Supervisor:
Dr R Dhanalakshmi

Defining the Problem

Given a set of jobs with their durations, and a list of dependencies (some jobs must be completed before others), we aim to find the minimum time required to complete all jobs.

1 Objective

Minimize the total completion time.

2 Constraints

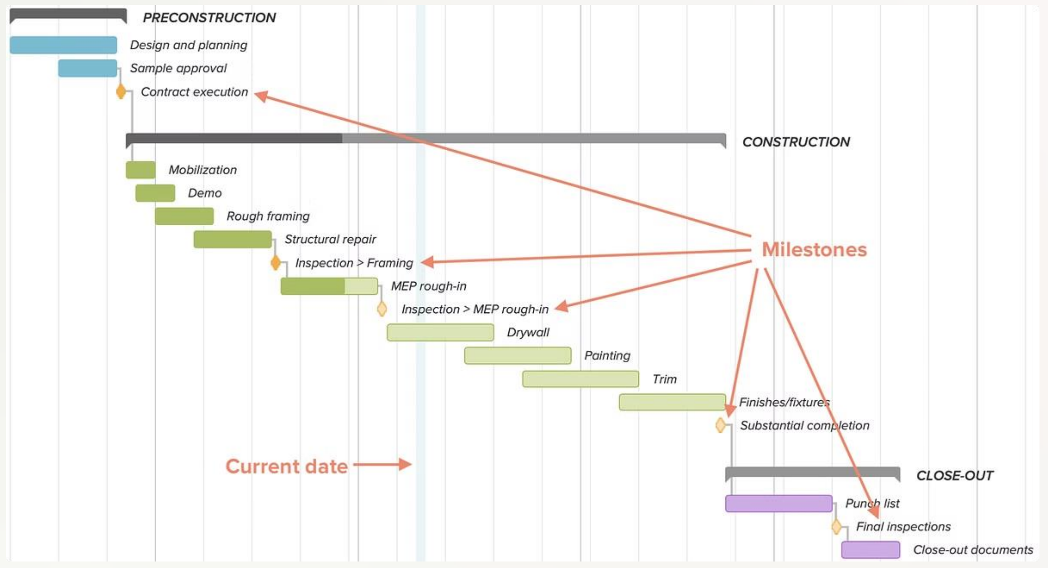
Dependencies must be respected, and each job can be done only once.

3 Inputs

Job durations and a dependency graph.

4 Output

The minimum completion time.



Brute Force: A Straightforward but Inefficient Approach

One way to approach this problem is by trying all possible job orderings, calculating the completion time for each, and finally choosing the ordering that yields the minimum time.

Advantages

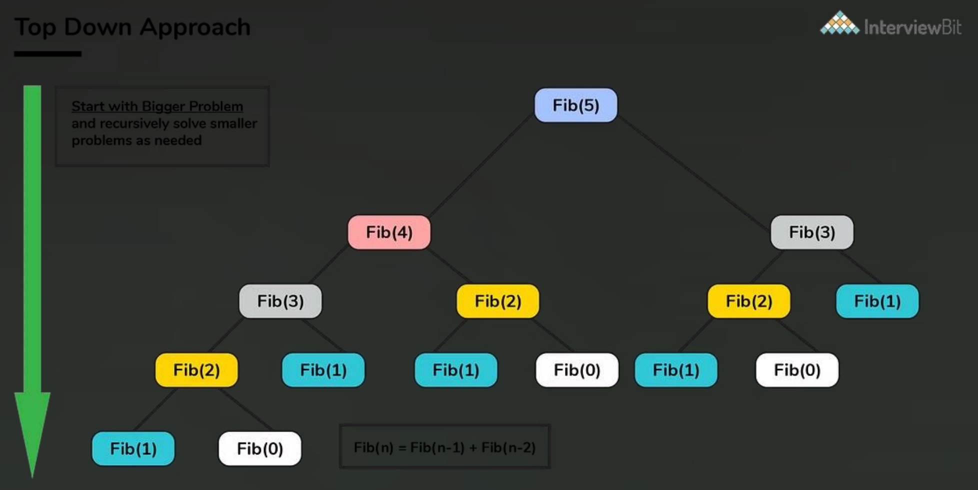
Simple to understand and implement.

Disadvantages

Extremely inefficient for large sets of jobs. Time complexity grows factorially with the number of jobs.

Dynamic Programming: A More Efficient Approach

Dynamic programming allows us to break down the problem into smaller, overlapping subproblems, solving each subproblem only once and storing the results to avoid redundant calculations.



1

1. Base Case

Define the minimum completion time for each individual job as its duration.

2

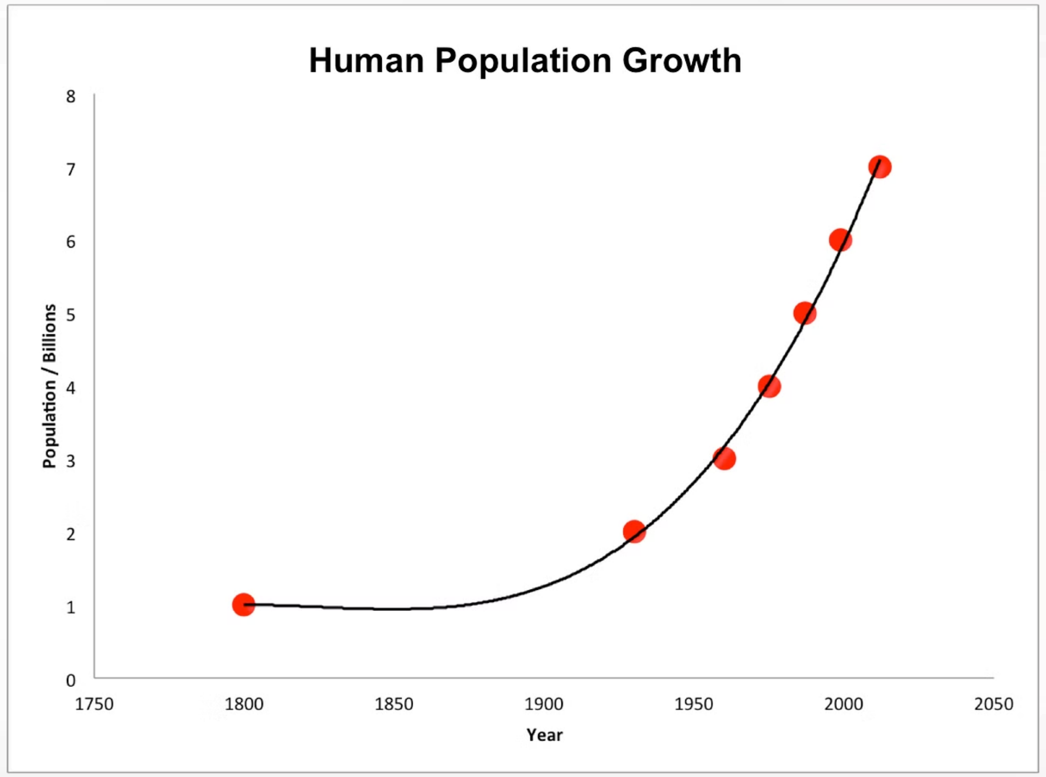
2. Recursive Step

For a job with dependencies, calculate the minimum time by adding its duration to the minimum completion time of its predecessors.

3

3. Memoization

Store the calculated minimum completion times to avoid recalculations.

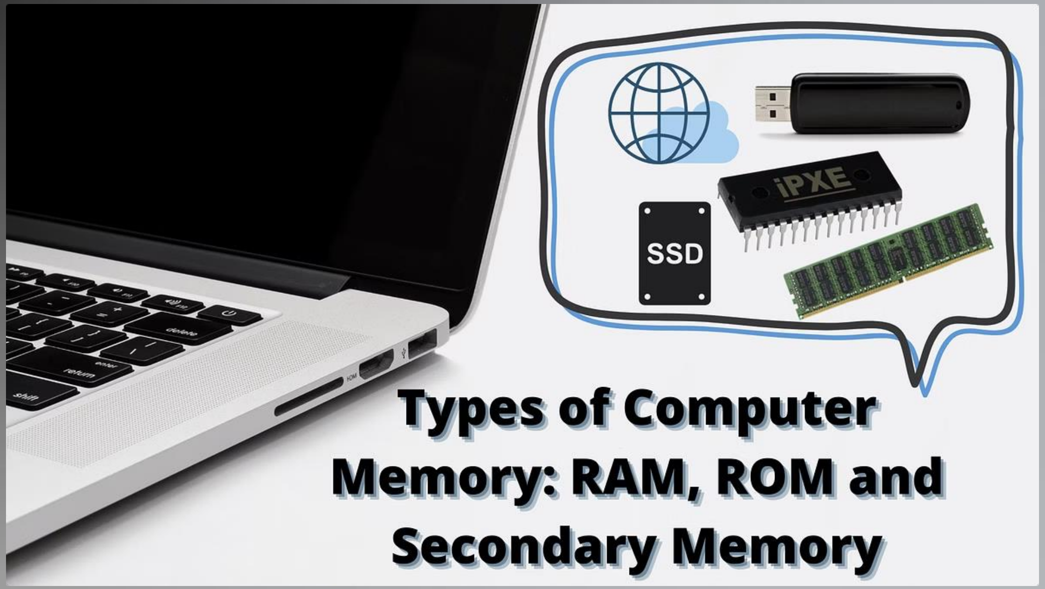


Time Complexity Analysis

The dynamic programming solution significantly improves the time complexity compared to the brute force approach.

Approach	Time Complexity
Brute Force	O(n!)
Dynamic Programming	O(n*m)

Where n is the number of jobs and m is the number of dependencies.



Space Complexity Analysis

The space complexity of the dynamic programming solution is determined by the storage required for the memoization table.

The memoization table has a size proportional to the number of jobs. Therefore, the space complexity is O(n), where n is the number of jobs.

Real-World Applications

The job scheduling problem finds practical applications in various domains.

Project Management

Optimizing the sequence of tasks in a project to minimize overall project duration.

Software Development

Scheduling the compilation and execution of software modules to reduce build times.

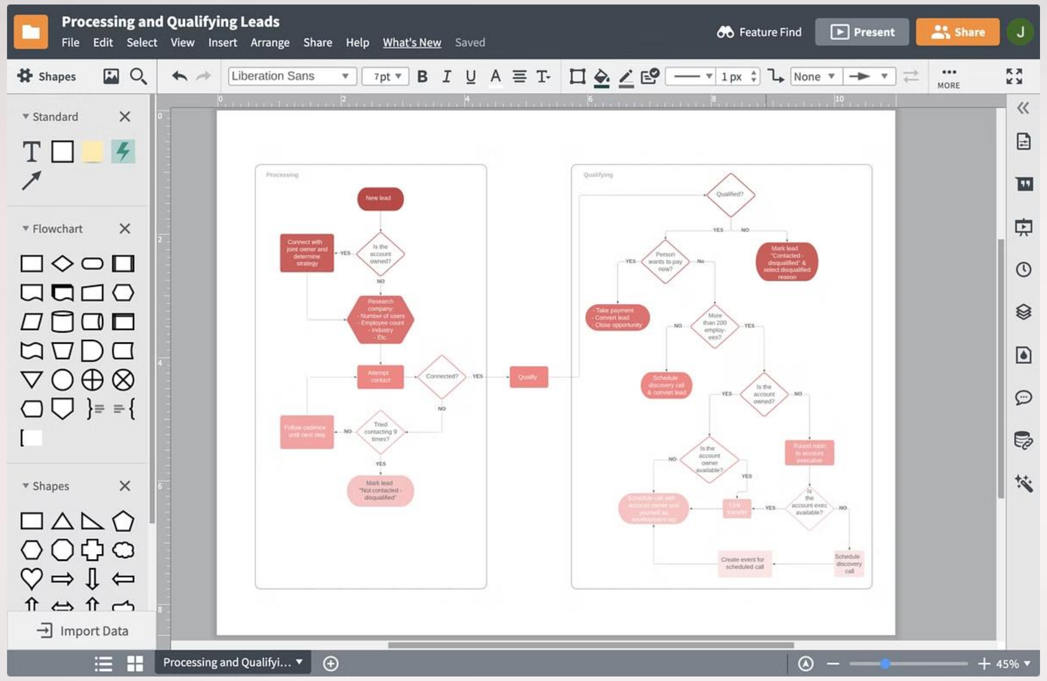
Network Routing

Finding the fastest path for data packets to traverse a network by scheduling data transfer operations.

Conclusion: The Power of Dynamic Programming

By leveraging the principles of dynamic programming, we can solve the job scheduling problem efficiently, finding the optimal solution in a reasonable amount of time. This demonstrates the power of dynamic programming in tackling complex problems by breaking them into smaller, manageable subproblems.

Dynamic programming is a valuable tool in the arsenal of computer scientists and engineers, enabling efficient solutions to various optimization problems.


Processing and Qualifying Leads
File Edit Select View Insert Arrange Share Help What's New Saved
Feature Find
Present
Share
Shapes
Standard
Flowchart
Import Data
Processing and Qualifyi...
45%