CS-204: Design and Analysis of Algorithms

220001078, 220001079, 220001080

April 5, 2024

1 Dynamic Programming

1.1 Inductive Definition

Inductively, dynamic programming breaks down problems into joint or disjoint subproblems for efficient solutions.

2 Insertion Sort

```
insertion_sort(array[0 to n]):
    if n > 0:
        insertion_sort(array[0 to n-1])
        insert A[n] in array
```

2.1 Example:

Let's say we have an array A that is already sorted till its n-1 elements (n being the number of elements in the array): $[1, 2, 3, 5]$. Thus, we have a recursive call for Insertion Sort.

3 Fibonacci Numbers

3.1 Naive Approach

The naive approach to calculating Fibonacci numbers involves recursion. The Fibonacci sequence is defined recursively as follows:

$$F(n) = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F(n-1) + F(n-2) & \text{otherwise} \end{cases}$$

Here's the implementation in code:

```
fibonacci_naive(n):
    if n <= 1:
        return n
    else:
        return fibonacci_naive(n-1) + fibonacci_naive(n-2)
```

However, this approach has exponential time complexity, making it highly inefficient for large values of n.

3.2 Dynamic Programming Approach

Dynamic programming can be used to optimize the calculation of Fibonacci numbers. We can store the previously computed Fibonacci numbers to avoid redundant calculations. Here's the implementation using dynamic programming:

```
fibonacci_dp(n):
    fib [0] = 0
    fib[1] = 1
    for i = 2 to n:
        fib[i] = fib[i - 1] + fib[i - 2]
    return fib[n]
```

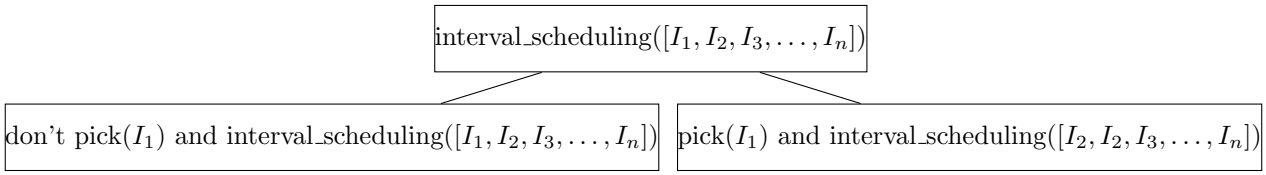
This approach has a time complexity of $O(n)$, making it much more efficient than the naive recursive approach.

4 Interval Scheduling Problem

Given a set of intervals $[I_1, I_2, I_3, \ldots, I_n]$, the overlapping intervals problem aims to find the maximum number of tasks that can be per that overlap each other, considering only one task can happen at once.

4.1 Naive Approach

The naive approach involves checking every pair of intervals for overlap and counting the maximum number of independent tasks. The time complexity of this approach is quadratic.



4.2 Dynamic Programming Approach

We can sort the intervals based on their start times and then use a dynamic programming approach to find the maximum number of non-overlapping intervals.

5 Longest Common Word Problem

Given two strings X and Y, the longest common word problem aims to find the length of the longest common subarray between X and Y. For example, we have BISECT and TRISECT two words. The length of the longest common word in the above problem is 5 and it is ISECT.

5.1 Naive Approach

The naive approach involves generating all possible subsequences of both strings and finding the longest common subsequence. The time complexity of this approach is $\mathrm{O}(nm^2 + mn^2)$.

5.2 Dynamic Programming Approach

Dynamic programming can be used to optimize the calculation of the longest common subsequence. We can create a matrix dp where $dp[i][j]$ represents the length of the longest common subsequence of the substrings $X[0:i]$ and $Y[0:j]$. We fill in the matrix iteratively using the following recurrence relation:

$$dp[i][j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ dp[i-1][j-1] + 1 & \text{if } X[i] = Y[j] \\ \max(dp[i-1][j], dp[i][j-1]) & \text{otherwise} \end{cases}$$

The time complexity of this approach is $O(mn)$, where m and n are the lengths of strings X and Y respectively. The space complexity is also $O(mn)$.