CAPSTONE PROJECT PRESENTATION

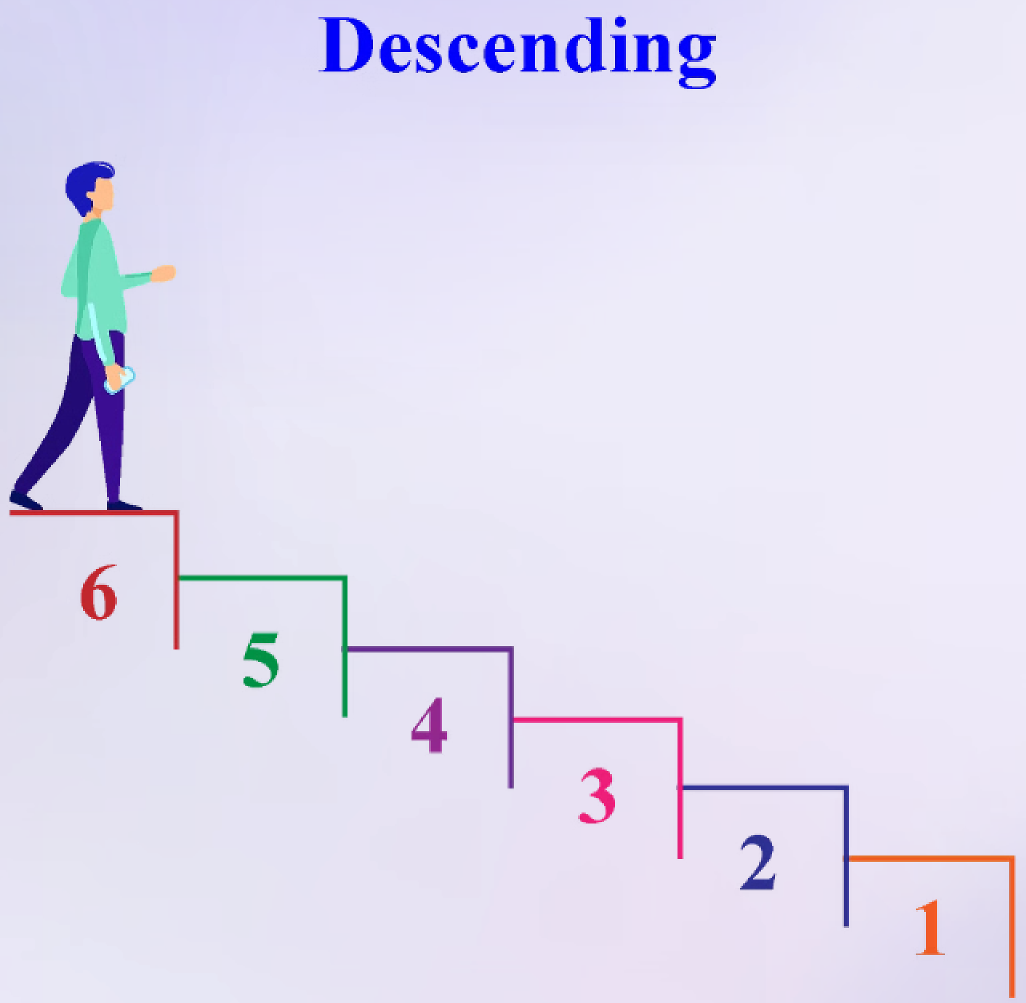
NAME : R . Gnana Prakash Reddy
REG NO : 192211740

CSA0656 – Design And Analysis Of Algorithms

Guided By
Dr.R.Dhanalakshmi Mam

Introduction to String Splitting and Backtracking

String splitting is a fundamental task in computer science. It involves involves dividing a string into smaller substrings based on specific specific criteria. Backtracking is a powerful algorithmic technique that technique that systematically explores possible solutions by trying trying different choices and backtracking when a choice leads to a dead to a dead end. This presentation delves into the intricate world of string of string splitting using backtracking, focusing on a challenging problem challenging problem involving descending consecutive values.



Problem Statement: Splitting a String into into Descending Consecutive Values

The problem involves splitting a given string into substrings, each representing a consecutive integer value. integer value. These values must be in descending order. For example, the string "9876543210" can be split can be split into "987" "654" "321" "0". This task presents unique challenges, as we need to consider all consider all possible splits while ensuring the resulting values are consecutive and descending.

String Input

The input is a string of digits.

Splitting Criteria

The string must be split into substrings that represent consecutive integers.

Descending Order

The consecutive integer values must be in descending order.

Backtracking Approach: Exploring Possible Splits

The backtracking approach systematically explores all possible splits of the string. It starts with an empty split, and at each step, it considers adding the next digit to the current substring or starting a new substring. The algorithm backtracks when a split leads to an invalid sequence, ensuring that only valid splits are considered.

1. **Start with Empty Split**

   Initially, there are no substrings.

2. **Add Digit to Current Substring**

   The current digit is added to the last substring.

3. **Start a New Substring**

   A new substring is created with the current digit.

4. **Check Validity**

   The split is checked to ensure consecutive and descending values.

Defining the Backtracking Algorithm

The backtracking algorithm can be defined recursively. It takes the input string, the current split, and the current index as arguments. It explores all possible splits by trying two options at each step: adding the current digit to the last substring or starting a new substring. The algorithm recursively calls itself with the updated split and index for each option. The base case is reached when the end of the string is reached.

Base Case

When the end of the string is reached, the algorithm checks if the split is valid. If so, it adds the split to the list of solutions. If not, it backtracks.

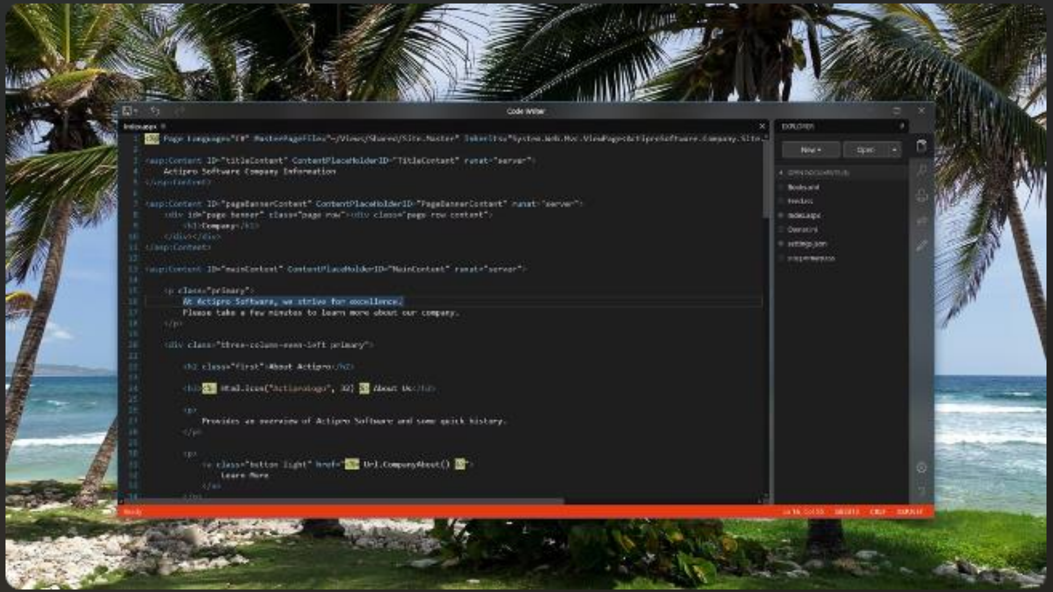
Recursive Steps

For each digit in the string, the algorithm explores two options: add it to the current substring or start a new substring. The algorithm recursively calls itself with the updated split and index for each option.

Implementing the Backtracking Function

The backtracking algorithm can be implemented using a recursive function. The function takes the input string, the current split, and the current index as arguments. It explores all possible splits by trying two options at each step: adding the current digit to the last substring or starting a new substring. The function recursively calls itself with the updated split and index for each option. The base case is reached when the end of the string is reached.

Parameter	Description
string	The input string of digits.
split	The current list of substrings.
index	The current index in the string.



```
<%@ Page Language="C#" MasterPageFile="~/Views/Shared/Site.Master" Inherits="System.Web.Mvc.ViewPage<ActiproSoftware.Company.Site...

<asp:Content ID="titleContent" ContentPlaceHolderID="TitleContent" runat="server">
    Actipro Software Company Information
</asp:Content>

<asp:Content ID="pageBannerContent" ContentPlaceHolderID="PageBannerContent" runat="server">
    <div id="page banner" class="page row"><div class="page row content">
        <h1>Company</h1>
    </div></div>
</asp:Content>

<asp:Content ID="mainContent" ContentPlaceHolderID="MainContent" runat="server">

    <p class="primary">
        At Actipro Software, we strive for excellence.
        Please take a few minutes to learn more about our company.
    </p>

    <div class="three-column-even-left primary">

        <h2 class="first">About Actipro</h2>

        <h3><%= Html.Icon("ActiproLogo", 32) %> About Us</h3>

        <p>
            Provides an overview of Actipro Software and some quick history.
        </p>

        <p>
            <a class="button light" href="<%= Url.CompanyAbout() %>">
                Learn More
            </a>
```

Handling Edge Cases and Constraints

It's crucial to handle edge cases and constraints to ensure the algorithm's robustness. For example, the algorithm should check for invalid inputs, such as empty strings or strings containing non-digit characters. It should also handle cases where no valid splits exist, returning an empty list of solutions. Additionally, constraints like maximum substring length or minimum value can be implemented to refine the algorithm's behavior.

1 **Empty String**

Handle empty strings by returning an empty list of solutions.

2 **Non-Digit Characters**

Reject strings containing non-digit characters.

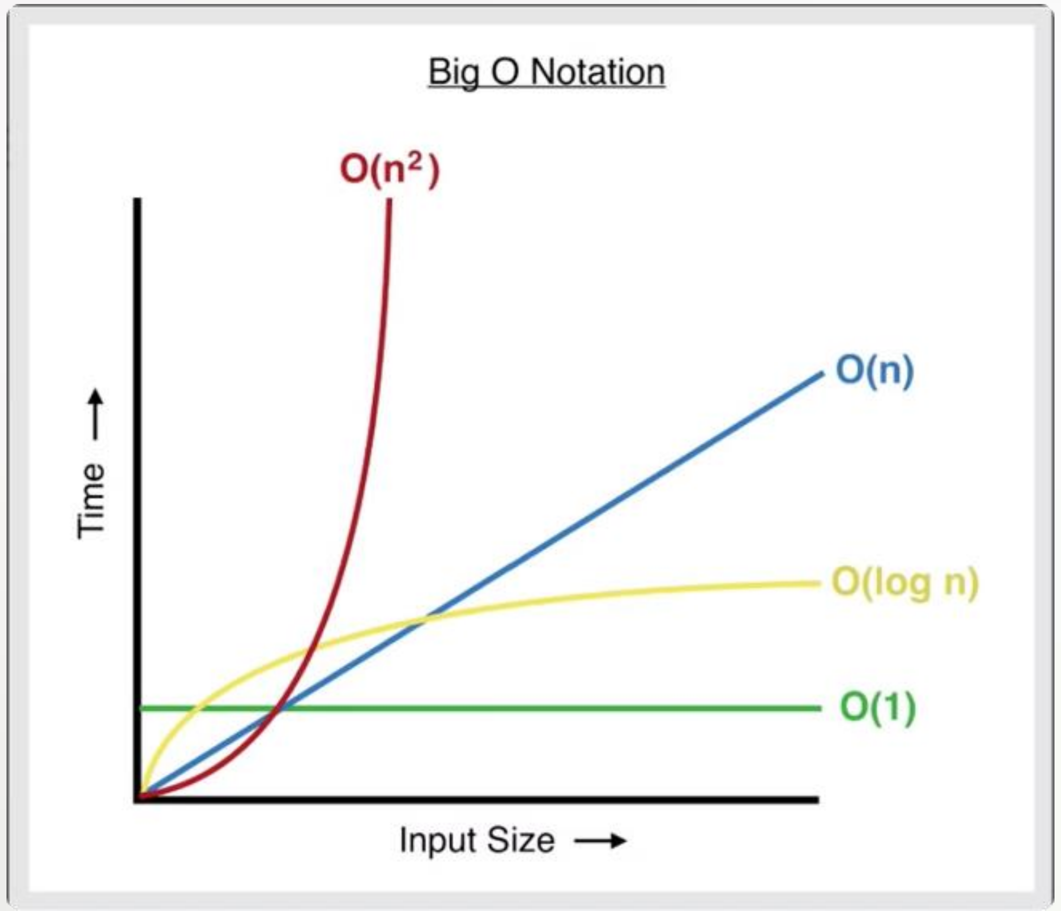
3 **Maximum Substring Length**

Implement a constraint to limit the length of each substring.

4 **Minimum Value**

Set a minimum value for the consecutive integers.

Time and Space Complexity Analysis



The time complexity of the backtracking algorithm is exponential, as it explores all possible splits of the string. In the worst case, it may need to explore 2^n splits, where n is the length of the string. The space complexity is mainly due to the recursive call stack, which can grow to a depth of n. However, the algorithm can be optimized to reduce the search space, improving the time complexity.

Time Complexity

Exponential: O(2^n) in the worst case.

Space Complexity

Linear: O(n) due to the recursive call stack.

Conclusion and Potential Applications

String splitting and backtracking are powerful tools for solving a wide range of problems in computer science. The ability to break down complex problems into smaller subproblems and systematically explore solutions is fundamental to many algorithms and applications. This specific problem, splitting a string into descending consecutive values, highlights the efficiency and flexibility of backtracking, opening doors to diverse applications in areas like data processing, code optimization, and puzzle solving.

1

Data Processing

String splitting and backtracking can be used to extract patterns and analyze data.

2

Code Optimization

These techniques can help optimize code by identifying redundant calculations and simplifying expressions.

3

Puzzle Solving

Backtracking is widely used in solving various puzzles, like Sudoku or crosswords.