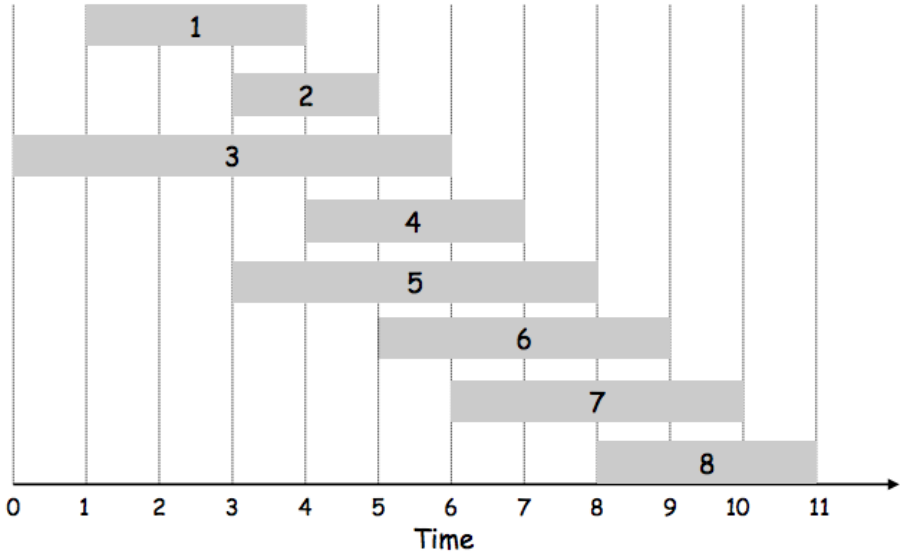
Missouri University of Science & Technology — Department of Computer Science
Spring 2023 — **CS 2500: Algorithms**

Homework 3: Advanced Design Techniques

Instructor: *Sid Nadendla* — **Due:** *April 10, 2023*

Problem 1: Task Selection *25 points*

Consider a total of n jobs. Let the i^{th} job be designated with a start time s_i, a finish time f_i and a net value v_i. Two jobs are said to be *compatible* if they do not overlap in time. Your goal is to find the subset of mutually compatible jobs that have the maximum total value. Present the following four stages of your design approach to this problem:



1. Model the above problem as a multi-stage decision problem, identify the state and decision variables, define the state transitions and derive the Bellman equation.
2. Using the Bellman equation, write a pseudocode to compute the optimal solution using dynamic programming approach.
3. Write down the pseudocode for the greedy solution to this problem.
4. Implement in Python, both the dynamic programming and greedy solutions to this problem and compare the value of the solutions returned for random inputs when there are a total of $n = 10$ jobs.

Problem 2: String Edit Problem *25 points*

The *string edit* problem is to find the cheapest way to modify two strings so that they are the same. The permitted operations are *deletions*, *insertions* and *substitutions*.

Example: Consider two strings: ALKHWARIZMI and ALGORITHM. We need to perform the following sequence of operations in order to modify ALKHWARIZMI into ALGORITHM:

- Substitute K with G
- Substitute H with O
- Delete W
- Delete A
- Replace Z with T
- Insert H
- Delete I

Let the two strings be denoted as $a_1 a_2 \cdots a_m$ and $b_1 b_2 \cdots b_n$, where each a_i and each b_j are characters in the set S. If s_i and s_j are any two characters in S, let

- the cost of deleting s_i = $D_i > 0$
- the cost of inserting s_i = $I_i > 0$
- the cost of substituting s_i with s_j = $C_{ij} \geq 0$.

Assume $C_{ij} = C_{ji}$ for all i, j and $C_{ij} = 0$ if and only if $i = j$.

Then, present the following four stages of your design approach to this problem:

1. Model the above problem as a multi-stage decision problem, identify the state and decision variables, define the state transitions and derive the Bellman equation.
2. Using the Bellman equation, write a pseudocode to compute the optimal solution using dynamic programming approach.
3. Write down the pseudocode for the greedy solution to this problem.
4. Implement in Python, both the dynamic programming and greedy solutions to this problem and compare the value of the solutions returned for random pairs of strings.

Problem 3: Palindromic Subsequences *25 points*

A subsequence is *palindromic* if it is the same whether read from left to right, or right to left. For instance, the sequence $ACGTGTCAAAATCG$ has many palindromic subsequences, including $ACGCA$ and $AAAA$. On the other hand, ACT is a subsequence that is not palindromic.

Devise an algorithm that takes a sequence $x[1:n]$ of length n, and returns the length of the longest palindromic subsequence.

Note: Its run time should be $O(n^2)$.

Problem 4: Polygon Triangulation *25 points*

Consider a convex polygon $\mathcal{P}$ on n vertices in a plane. A *triangulation* of $\mathcal{P}$ is a collection of $n-3$ diagonals of $\mathcal{P}$ such that no two diagonals intersect (except possibly at their end-points). Notice that a triangulation splits the polygon's interior into $n-2$ disjoint triangles. Let the cost of triangulation be the sum of the lengths of the diagonals in it.

Then, give the pseudocode for the optimal algorithm for finding the triangulation that minimizes the cost.

Problem 5: Huffman Coding *(Extra Credit: 5 points)*

Given a benchmark sequence as an input, write a Python function to construct the Huffman tree. Using this tree, develop an encoder to convert any input string into a binary sequence, and a decoder to convert any binary sequence back into the original input sequence.

For your convenience, the following Python base structure is provided to help develop your code:

```
from collections import Counter

class Node():
    def __init__(self, freq=None, left=None, right=None):
        self.set_children(left, right)
        self.freq = freq
        self.set_frequency()

    def set_children(self, left, right):
        self.left = left
        self.right = right

    def get_children(self):
        return self.left, self.right

    def set_frequency(self):
        if self.left and self.right:
            self.freq = self.left.freq + self.right.freq

    def get_frequency(self):
        return self.freq

    def __str__(self):
        return f"left:{self.left},right:{self.right}"


def huffman_tree(string):

    # Compute the occurrence frequency of all symbols in the input string
    symb_freq = Counter(string)

    # Sort the symbols according to their frequency of occurrence
    symb_freq = sorted(symb_freq.items(), key=lambda x: x[1])

    """
    TODO:   Function to construct Huffman tree
            depending on symbol frequency using Node()
            and return the root node
    """
    pass


def huffman_encoder(input_string, tree_root_node):
    """
```

```
    TODO:   Function to encode the input string using huffman_tree()

    Suggestion: Label left children with '0' and right children with '1'
    """
    pass


def huffman_decoder(binary_string, tree_root_node):
    """
    TODO:   Function to decode a binary sequence constructed using huffman_encoder()
    """
    pass


if __name__ == '__main__':
    benchmark_string = 'AABCACAAABCBABBAABAAAACACBBABCBABBACB'
    root = huffman_tree(benchmark_string)
    test_string = 'ABCAAB'
    encoded_sequence = huffman_encoder(test_string, root)
    decoded_sequence = huffman_decoder(encoded_sequence, root)
    print(f"Input:{test_string}")
    print(f"Decoded Sequence:{decoded_sequence}")
    print("TEST:PASS" if test_string == decoded_sequence else "TEST:FAIL")
```