Chapter 5
Examples of Combinatorial Problems

ABSTRACT

Combinatorial problems are mathematical challenges that involve finding optimal solutions while satisfying specific constraints. The chapter explores various problems, including the Longest Common Subsequence (LCS), used in bioinformatics and text analysis, and clustering, which groups data based on similarities. It also covers the Minimum Spanning Tree (MST) problem, crucial for efficient networks, and the Traveling Salesman Problem (TSP), a classic optimization issue. Other problems discussed include the maximum flow in a graph, which is used for network and traffic optimization, and the maximum cut problem, which is applicable in social network analysis. The chapter also examines algorithms for maximum node independence in graphs, maximum coverage, stable matching (Gale-Shapley problem), and even soccer team formation strategies. Finally, it explores efficient algorithms for solving these problems, demonstrating their practical applications in various fields.

INTRODUCTION

Not everything that counts can be counted, and not everything that can be counted counts

Albert Einstein

Combinatorial challenges are, in essence, large-scale puzzles that require a successful outcome to adhere to a specific set of rules. Imagine, for instance, the task of assigning duties where each individual holds distinct preferences. Combinatorial methods facilitate the identification of the most effective solution. Practical instances abound; consider the intricacies of planning travel itineraries, assembling a football team's lineup, or curating a restaurant menu.

Numerous examples of complex combinatorial endeavours are currently active subjects of research within the mathematical community. Nonetheless, specific common characteristics stand out:

1. A finite set of elements can be people, objects, numbers, etc.
2. There are constraints or restrictions that the solutions must meet in order to satisfy certain conditions.
3. We try to find the best solution, which can be the one that maximizes or minimizes a specific function.

Here are some examples that refer to different areas:

The *Longest Common Sequence* (LCS) is a classic. It consists of finding the most extended sequence common to two (or more) strings while maintaining the relative order of the characters, but not necessarily the contiguity. The problem has several practical applications: in bioinformatics, it enables the comparison of DNA sequences and the identification of similarities between genes, while in text comparison, it is used to detect changes between different versions of a document.

The problem can be solved using several algorithmic techniques, including dynamic programming, which has a complexity of $O(n \quad m)$.

DOI: 10.4018/979-8-3373-3089-1.ch005

Program 25 longest common sequence

```
import random
import string
def generate_random_string(length):
# Generate a random string of uppercase letters
letters = string.ascii_uppercase
return ''.join(random.choice(letters) for _ in range(length))
def longest_common_subsequence(X, Y):
# Lengths of the two strings
n = len(X)
m = len(Y)
# Create a 2D table (n+1) x (m+1) initialized to zero for dynamic programming
dp = [[0] * (m + 1) for _ in range(n + 1)]
# Fill the dp table
for i in range(1, n + 1):
for j in range(1, m + 1):
# If the characters of the two strings match
if X[i - 1] == Y[j - 1]:
dp[i][j] = dp[i - 1][j - 1] + 1
else:
dp[i][j] = max(dp[i - 1][j], dp[i][j - 1])
# The length of the longest common subsequence is stored in dp[n][m]
length_lcs = dp[n][m]
# Reconstruct the longest common subsequence
lcs = []
i, j = n, m
while i > 0 and j > 0:
# If the characters match, they are part of the LCS
if X[i - 1] == Y[j - 1]:
lcs.append(X[i - 1])
i -= 1
j -= 1
# If they don't match, move in the direction with the higher value
elif dp[i - 1][j] >= dp[i][j - 1]:
i -= 1
else:
j -= 1
# Reverse the sequence since it was constructed backwards
lcs.reverse()
# Convert the list of characters to a string and return it along with the length
return length_lcs, ''.join(lcs)
# Generate two random strings
X = generate_random_string(20)
Y = generate_random_string(20)
# Compute the LCS
length, lcs = longest_common_subsequence(X, Y)
# Print the results
```

Related Content