GALGOTIAS UNIVERSITY

Solving the Traveling Salesman Problem with Dynamic Programming

Session No.: 35
Course Name: Design and Analysis of Algorithms
Course Code: R1UC407B
Instructor Name: ANKIT VERMA
Duration: 50 minutes
Date of Conduction of Class:

GALGOTIAS UNIVERSITY

Quick Recap of Previous Session 34

- **Instructions:**
  - Let's refresh our memory with a quick Wooclap quiz on the last session's topic: Dynamic Programming and the Longest Common Subsequence Problem.
- Please take 5 minutes to answer the quiz questions.

How to participate?

1. Go to wooclap.com
2. Enter the event code in t top banner

GALGOTIAS UNIVERSITY

Beyond Brute Force and Greedy

We've explored brute force, greedy techniques, and divide and conquer.
Now, let's tackle a classic optimization problem with dynamic programming.
Let's brainstorm some ideas and applications using a Wooclap word cloud.

How to participate?

1. Go to wooclap.com
2. Enter the event co
top banner

Learning Outcomes:
By the end of this session, you will be able t

Learning Outcome 1:

Define the Traveling Salesman Problem and its relevance.

Learning Outcome 2:

Apply the dynamic programming algorithm to solve TSP and analyze the algorithm's time and space complexity.

GALGOTIAS UNIVERSITY

Session Outline

1 Travelling Salesman Prob

2 Dynamic Programming

3 TSP using Dynamic Progra

4 Time Complexity

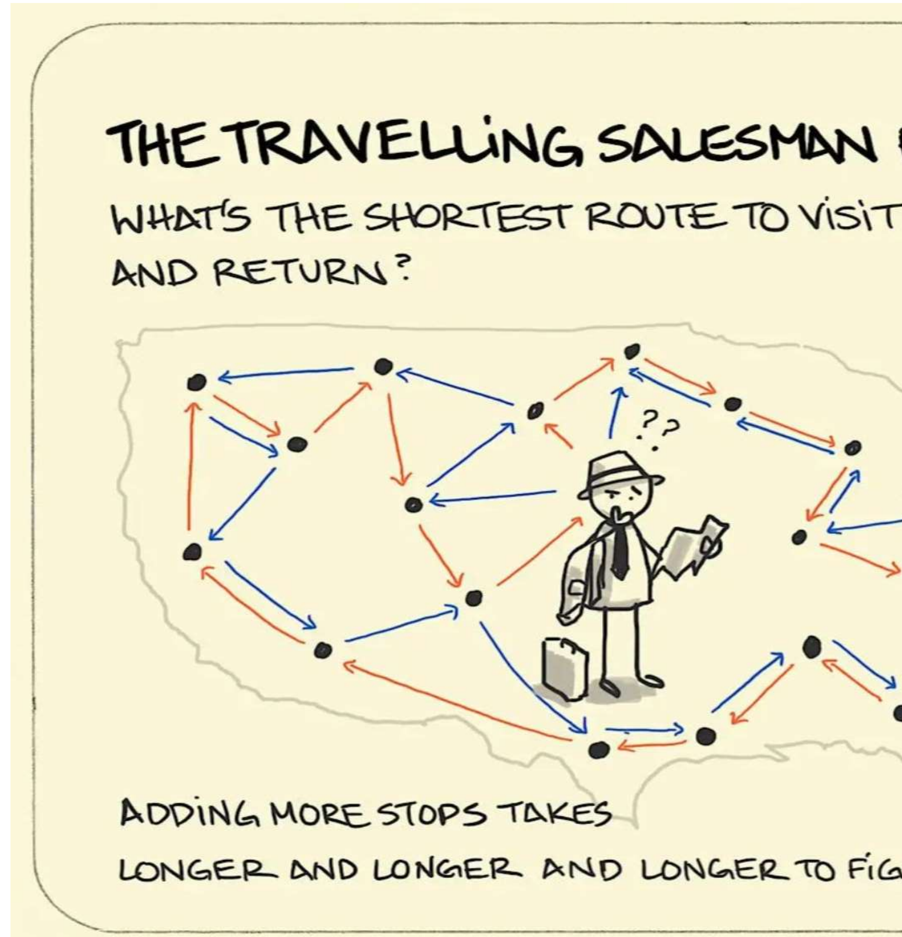
5 Space Complexity

6 Conclusion

What is the Traveling Salesman Problem

Formal Definition:
"Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?"

Significance:
Logistics and Transportation: Optimizing delivery routes, minimizing transportation costs.
Circuit Design: Finding the shortest path for connecting components on a circuit board.
Other Applications: Planning, manufacturing, and DNA sequencing.



GALGOTIAS UNIVERSITY

Activity 1 - Wooclap Poll : Testing Your Understanding

How to participate?

1. Go to wooclap.com
2. Enter the event code in the top banner

The Need for Dynamic Programmi

Limitations of Previous Approaches:

Brute-force: Exhaustively trying all possible permutations of cities quickly becomes computa
infeasible as the number of cities (n) increases.
The time complexity is O(n!), making it impractical for larger problems.

Greedy Approach: While greedy algorithms can provide quick solutions, they don't guarante
optimal solution for TSP.
Greedy choices at each step might lead to a locally optimal solution that is not globally optim

Divide and Conquer: Although effective for some problems, divide and conquer doesn't eas
TSP due to the interconnectedness of the subproblems.
Dividing the TSP into smaller tours and combining them doesn't guarantee the shortest overa

The Need for Dynamic Programmi

Dynamic Programming as a Solution:

Dynamic programming overcomes these limitations by exploiting the problem's inherent struc
Optimal Substructure: An optimal solution to the TSP contains within it optimal solutions to subproblems.
For example, the shortest tour that includes a specific subset of cities must also contain the s paths between those cities.

Overlapping Subproblems: The same subproblems are encountered multiple times when e different tours.
Dynamic programming stores the solutions to these subproblems, avoiding redundant compu significantly improving efficiency.

GALGOTIAS UNIVERSITY

Activity 2 - Dynamic Programming for TSP : Building the Op
Tour

Step-by-Step Explanation:

1. Define the DP Table:
   - We'll use a 2D table dp[S][j], where:
   - S represents a subset of cities.
   - j represents the last city visited in the subset S.
   - dp[S][j] stores the minimum cost to visit all cities in the subset S, starting from city 1 and ending at city j.
2. Initialize Base Cases:
   - For all j != 1, dp[{1}][j] = cost[1][j], where cost[i][j] is the distance between city i and city j.
   - This means the cost to travel from city 1 to any other city j when only those two cities are in the subset is simply the direct distance between them.
3. Fill the Table:
   - We'll fill the table in increasing order of s
   - For each subset S of size greater th ending city j in S:
   - dp[S][j] = min(cost[k][j] + dp[S-{j}][k]) fo j.
     - This means we find the minimum considering all possible previous c S (excluding j itself), and adding th the cost of the optimal sub-tour end
4. Trace Back:
   - Start from the final entry in the table, dp represents the minimum cost of the com city 1.
   - To reconstruct the optimal tour, trace ba by identifying the previous city k that cost at each step.

Activity 2 - Dynamic Programming for TSP : Building the Op
Tour

- **Illustrative Example: Cities:** A, B, C, D
- **Cost Matrix:**

	A	B	C	D
A	0	10	15	20
B	10	0	35	25
C	15	35	0	30
D	20	25	30	0

DP Table:

S	j	dp[S][j]	Calculation
{A}	A	0	Base case
{A}	B	10	Base case (cost[A][B]
{A}	C	15	Base case (cost[A][C]
{A}	D	20	Base case (cost[A][D]
{AB}	B	20	min(cost[A][B] + dp[{A
{AC}	C	25	min(cost[A][C] + dp[{A
{AD}	D	30	min(cost[A][D] + dp[{A
{ABC}	B	40	min(cost[C][B] + dp[{A dp[{AC}][A]) = min(35
{ABC}	C	45	min(cost[B][C] + dp[{A dp[{AB}][A]) = min(35
...	...	...	...
{ABCD}	A	80	(Calculated based on

Activity 2 - Dynamic Programming for TSP : Building the Op
Tour

- **Illustrative Example: Cities:** A, B, C, D
- **1. Cost Matrix:**

	A	B	C	D
A	0	10	15	20
B	10	0	35	25
C	15	35	0	30
D	20	25	30	0

2. DP Table:

S	j	dp[S][j]	Calculation
{A}	A	0	Base case
{A}	B	10	Base case (cost[A][B]
{A}	C	15	Base case (cost[A][C]
{A}	D	20	Base case (cost[A][D]
{AB}	B	20	min(cost[A][B] + dp[{A
{AC}	C	25	min(cost[A][C] + dp[{A
{AD}	D	30	min(cost[A][D] + dp[{A
{ABC}	B	40	min(cost[C][B] + dp[{A dp[{AC}][A]) = min(35
{ABC}	C	45	min(cost[B][C] + dp[{A dp[{AB}][A]) = min(35
...	...	...	...
{ABCD}	A	80	(Calculated based on

3. Trace Back:

- Starting from dp[ABCD][A] = 80, we find that the previous city that led to this minimum cost was city C (cost[C][A] + dp[ABC][C] = 15 + 65 = 80).
- From dp[ABC][C] = 65, the previous city was B (cost[B][C] + dp[AB][B] = 35 + 30 = 65).
- From dp[AB][B] = 30, the previous city was A (cost[A][B] + dp[A][A] = 10 + 20 = 30).
- Therefore, the optimal tour is A -> D -> B -> C -> A with a total cost of 80.

Activity 2 - Wooflash Flashcards : Mastering the Algorithm

Instructions:

Let's create Wooflash flashcards for key terms and steps in the dynamic programming algorithm for TSP

Key terms:

Subset: A smaller set of cities chosen from the complete set.

Cost/Distance: The distance between two cities, which contributes to the total tour cost.

Memoization: Storing the results of expensive function calls and returning the cached result when the same inputs occur again.

Base Case: The simplest subproblem that can be solved directly without recursion.

Recursive Case: A more complex subproblem that is solved by breaking it down into smaller subproblems and combining their solutions.

Steps:

Define the DP Table: Create a table to st subproblems.

Initialize Base Cases: Fill in the table wi simplest subproblems.

Fill the Table: Use a recursive approach solving subproblems based on the solutio subproblems.

Trace Back: Once the table is filled, trace table to reconstruct the optimal solution to problem.

"Exchange your flashcards with a partner learning."

Activity 2 – Reflection : Efficiency of Dynamic Programming

Comparison to Brute-Force:

- Brute-force explores all possible permutations, leading to a factorial time complexity of O(n!).
- Dynamic programming drastically reduces computations by storing and reusing solutions to overlapping subproblems.
- For example, with 10 cities, brute-force would require evaluating 3,628,800 permutations, while dynamic programming reduces this significantly.

Time and Space Complexit

Time Complexity: The dyna algorithm for TSP has a ti O(n^2 * 2^n). This is be through all possible subsets for each subset, it consi ending cities (n) and all cities (n).

Space Complexity: The sp O(n * 2^n) due to the size which stores the optimal s subset and ending city.

GALGOTIAS UNIVERSITY

Conclusion – Summary : Key Takeawa

- The Traveling Salesman Problem is a classic optimization problem with various real-worl applications.
- Dynamic programming is an efficient technique for solving TSP by breaking it into smalle subproblems.
- The dynamic programming algorithm for TSP has a time complexity of $O(n^2 * 2^n)$ and space complexity of $O(n * 2^n)$.

GALGOTIAS UNIVERSITY

Information about the next less

- In the next session, we will explore another application of dynamic programming:
  Finding the shortest path in Multistage Graphs.
- Please come prepared by reviewing the relevant sections on Multistage Graphs and dynamic programming in your textbook.