CSE246: Algorithms(Section No.2)
[Fall-25]

Project Report

Travelling Puzzle Man

Submitted by:

Student ID	Student Name
2022-1-60-222	Farzia Hossain
2023-1-60-256	Fatema Akter Nishi
2023-3-60-157	Shimul Kumar
2023-3-60-166	Ibna Jiam

Submitted to –

Dr. Md. Tauhid Bin Iqbal

Assistant Professor

Department of Computer Science & Engineering

Date of Submission: 17-12-2025

Introduction:

The village **Gorithm Algo** is filled with problem solvers. There 2 people called **Maij** and **Lumihs** claim themselves best among all the villagers. But the villagers do not seem like they approve of the claim. They believe that the one person also known as "**The Puzzle Man**" is the best among all the neighboring villages. This made **Maij** and **Lumihs** anxious, in order to prove everyone wrong they proposed a competition where 3 of them will sit for an intensive contest. To make the contest even more fair they invited the wisest persons in the village **Dihut** and his subordinate **Zifan** to become the judge for the contest. The judges provide a set of rules, that includes:

- The contest will be divided into **4 rounds**.
- Each round the contestant will face **4 different problems**.

The judges will evaluate their script based on the **time complexity**. The lower the time complexity is the higher the mark they will get out of 10. With the aggregation of the marks of the 4 problems, the contestant with the most marks will win and will be entitled as "**The Dlog Medalist**".

Round 1:

To take the advantage of this opportunity the village chief admits his ever since problem with spacing in the village register book. From the very beginning of his era he included every villager's name without spacing. Good for him that each villager has a unique name. But every time he tries to search for a name, he is forced to match it letter by letter.

Suppose he wants to search for a name called "**ihsin**" now he would have to match it with the long list of names like "**maijlumihsdihutzifanaizrafametaf*ihsin***".

Now the contestant will have to come up with the most favorable solution so that the chief can easily search for a name from the village register book in a short period of time.

Round 2:

Seeing the stress-free expression on the village chief's face, **Aizraf**, the village electrician, stepped forward.
He said: "Chief, I've been suffering from this problem for years. Our village is growing, new houses are popping up everywhere as well as the electric poles but the wires are too

costly. We need electricity in every corner of the village. But our budget is constrained. What can we do now?”

To ensure the electricity of the whole village the contestant must provide solutions that will take less amount of expense as well as save the villagers from this problem.

```
Using default village (6 houses):
Houses: 0–5
Connections with costs:
  House 0 ↔ House 1 : 4 coins
  House 0 ↔ House 2 : 3 coins
  House 1 ↔ House 2 : 1 coins
  House 1 ↔ House 3 : 2 coins
  House 2 ↔ House 3 : 4 coins
  House 2 ↔ House 4 : 5 coins
  House 3 ↔ House 4 : 7 coins
  House 3 ↔ House 5 : 3 coins
  House 4 ↔ House 5 : 6 coins
```

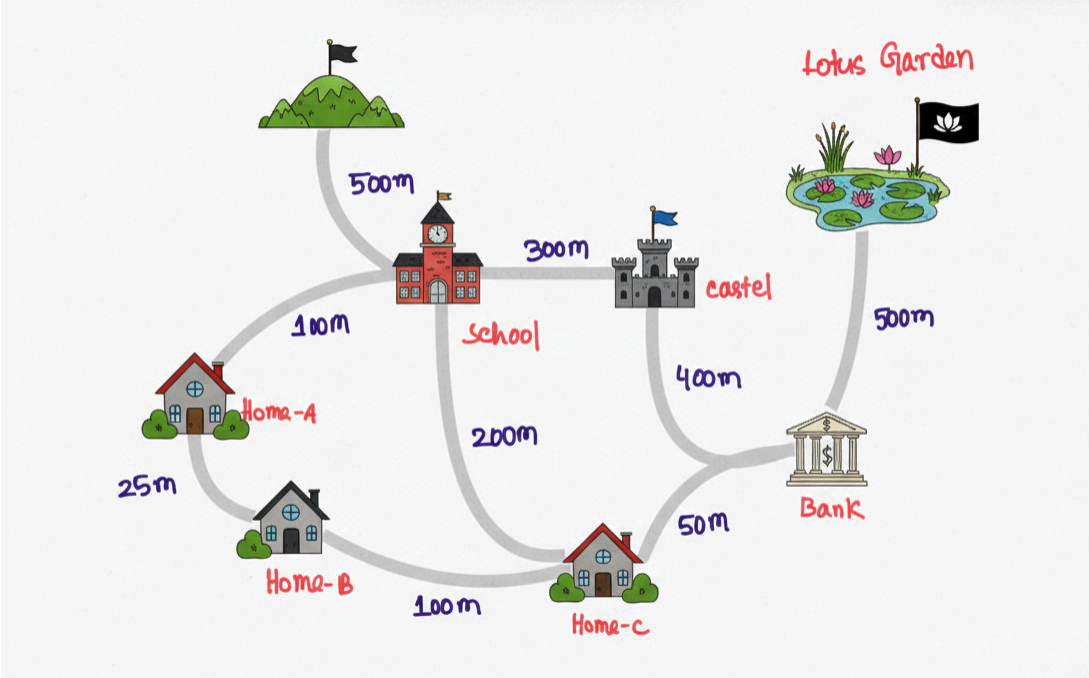
Round 3:

Noya, the sister of **Minsat**, is very shy. She goes to the city once a month with her cow-cart. The cart can carry only limited weight. So she wants to take the most valuable items possible. Since **Noya** is so shy her best friend **Yojnos** raised her hand and told the problem in front of the crowd. The judges liked the idea and asked the contestant to solve it to help **Noya** with this issue.

```
Using default values:
Cart capacity: 50kg
Available items:
1. Weight: 10kg, Value: 60 coins, Ratio: 6 coins/kg
2. Weight: 20kg, Value: 100 coins, Ratio: 5 coins/kg
3. Weight: 30kg, Value: 120 coins, Ratio: 4 coins/kg
4. Weight: 15kg, Value: 70 coins, Ratio: 4.66667 coins/kg
5. Weight: 25kg, Value: 90 coins, Ratio: 3.6 coins/kg
```

Round 4:

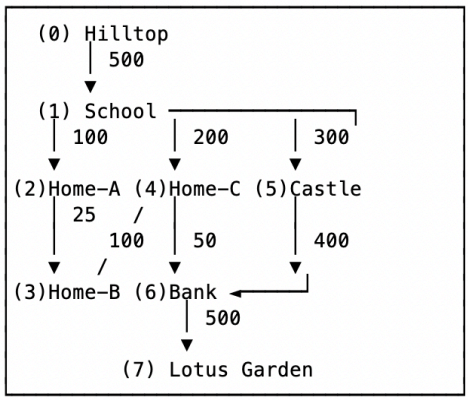
Everyone became so excited seeing the contestant solving all the problems, so a group of concerned villagers asked the judges to set a problem related to the village path finding map. The judges provided a map to the contestants and asked them to find the shortest path from **Hilltop** to **Lotus Garden**.



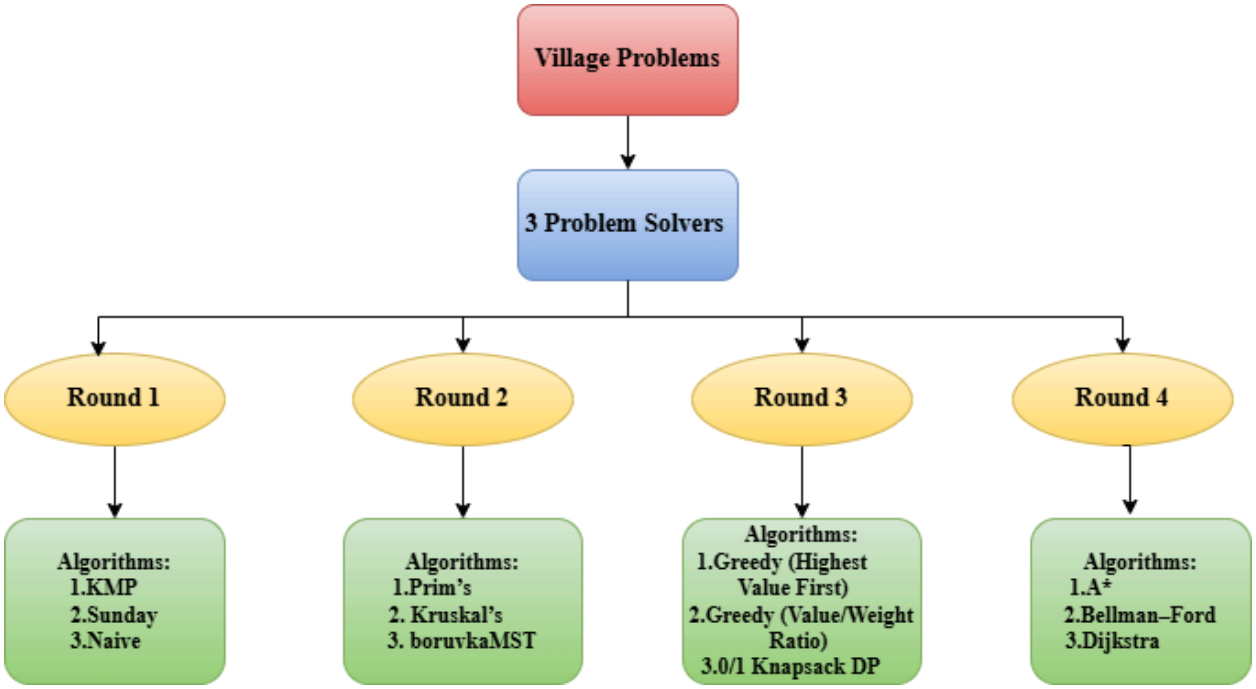
VILLAGE LOCATIONS:

Index	Location
0	Hilltop
1	School
2	Home-A
3	Home-B
4	Home-C
5	Castle
6	Bank
7	Lotus Garden

🗺️ VILLAGE PATH MAP 🗺️



Proposed Method:



Implementation:

Round 1 Result

Maij	Lumihs	Puzzle Man

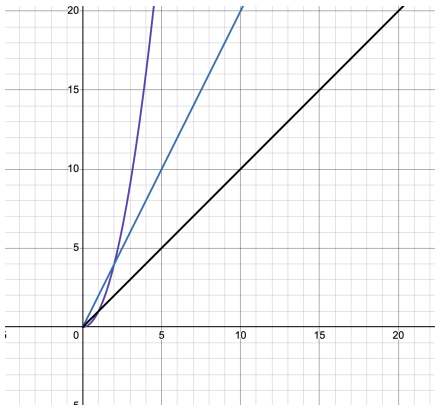
FUNCTION NaiveSearch(Text T, Pattern P)
  n = length of T
  m = length of P
 	
  FOR i FROM 0 TO n - m
  j = 0
  WHILE j < m AND T[i + j] == P[j]
  j = j + 1
 	
  IF j == m
  OUTPUT "Pattern found at index " + i
  END FOR
END FUNCTION
	FUNCTION KMP_Search(Text T, Pattern P)
	  n = length of T
	  m = length of P
	  LPS = new array of size m
	
	  ComputeLPS(P, LPS)
	
	  i = 0 // index for T
	  j = 0 // index for P
	
	  WHILE i < n
	  IF P[j] == T[i]
	  i = i + 1
	  j = j + 1
	
	  IF j == m
	  OUTPUT "Pattern found at index " + (i - j)
	  j = LPS[j - 1]
	  ELSE IF i < n AND P[j] != T[i]
	  IF j == 0
	  i = i + 1
	  ELSE
	  j = LPS[j - 1]
	  END IF
	  END IF
	  END WHILE
	END FUNCTION
		FUNCTION SundaySearch(Text T, Pattern P)
		  n = length of T
		  m = length of P
		  ShiftTable = BuildShiftTable(P)
		  i = 0
		  WHILE i <= n - m
		  j = 0
		  WHILE j < m AND T[i + j] == P[j]
		  j = j + 1
		  IF j == m
		  OUTPUT "Pattern found at index " + i
		  END IF
		  IF i + m < n
		  nextChar = T[i + m]
		  shift = ShiftTable.get(nextChar, m + 1)
		  i = i + shift
		  ELSE
		  // Reached end of text
		  BREAK
		  END IF
		  END WHILE
		END FUNCTION

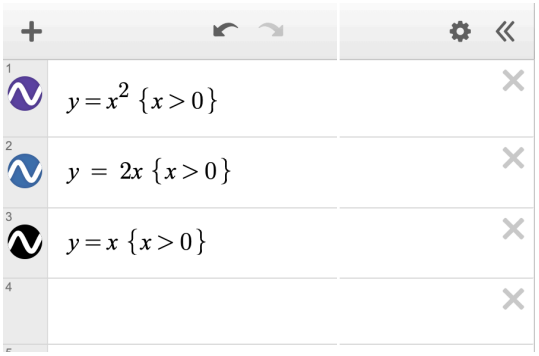
Brute-force, checks every possible alignment and shifts by one after a mismatch.	Uses a precomputed LPS (Longest Prefix Suffix) array to determine the optimal shift, avoiding redundant comparisons.	A variant of the Boyer-Moore algorithm that uses the character immediately after the current text window to determine the shift distance.

Round 1 Analysis:

Algorithm	Time Complexity (Worst)	Preprocessing	Performance	Ease of Implementation	**Remarks**
Naive Pattern Matching	O(n·m)	None	Poor for large text	Very Easy	**5/10**
KMP Algorithm	O(n+m)	Requires O(m) time and space to build the LPS table.	Linear time in the worst case.	Moderate	**7/10**
Sunday Algorithm	O(n/m)	Requires O(m) time and space to build a bad-character shift table.	O(n / m) on average, generally faster in practice. The worst case can be higher.	Easy	**9/10**

Complexity Graph:





Winner: The Puzzle Man for solving string problem using Sunday Algorithm.

Round 2 Result

Maij	Lumihs	Puzzle Man

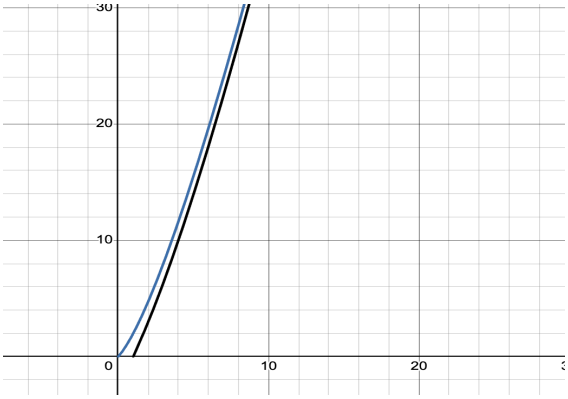
function Prim(Graph G, StartVertex s):
  MST_Edges = Empty set
  Visited = Empty set
  PriorityQueue Q
 	
  Add s to Visited
  For each edge (s, v) adjacent to s:
  Add (s, v) to Q
 	
  While Visited has fewer vertices than G.V:
  min_edge = Q.extract_min()
  u, v = min_edge.vertices
 	
  If v is not in Visited:
  Add v to Visited
  Add min_edge to MST_Edges
  For each edge (v, w) adjacent to v:
  If w is not in Visited:
  Add (v, w) to Q
  Else If u is not in Visited:
  Add u to Visited
  Add min_edge to MST_Edges
  For each edge (u, w) adjacent to u:
  If w is not in Visited:
  Add (u, w) to Q
 	
  Return MST_Edges
	function Kruskal(Graph G):
	  MST_Edges = Empty set
	  Sort all edges in G.E by weight in non-decreasing order
	
	  For each vertex v in G.V:
	  MAKE-SET(v)
	
	  For each edge (u, v) in sorted G.E:
	  If FIND-SET(u) is not equal to FIND-SET(v):
	  Add (u, v) to MST_Edges
	  UNION(u, v)
	
	  Return MST_Edges
		function Boruvka(Graph G):
		  MST_Edges = Empty set
		  Components = Set of all vertices (each in its own component)
		
		  While the number of components is greater than 1:
		  cheapest_edges = Map where key is component ID, value is the cheapest edge
		
		  For each edge (u, v) in G.E:
		  comp_u = FIND-SET(u)
		  comp_v = FIND-SET(v)
		
		  If comp_u is not equal to comp_v:
		  If (comp_u is not in cheapest_edges) or (weight(u, v) < weight(cheapest_edges[comp_u])):
		
		cheapest_edges[comp_u] = (u, v)
		  If (comp_v is not in cheapest_edges) or (weight(u, v) < weight(cheapest_edges[comp_v])):
		
		cheapest_edges[comp_v] = (u, v)
		
		  For each edge (u, v) in

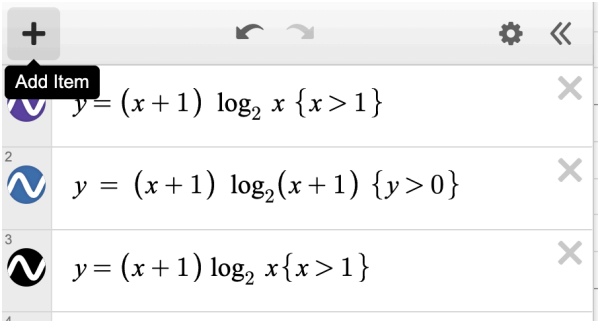
		values of cheapest_edges: If FIND-SET(u) is not equal to FIND-SET(v): Add (u, v) to MST_Edges UNION(u, v) Return MST_Edges
Vertex-based: Grows a single tree from a starting vertex by adding the cheapest edge connecting to an unvisited vertex.	**Edge-based**: Builds a forest (collection of trees) by adding the lowest-weight edges that do not form a cycle, eventually merging into a single tree.	**Component-based (Parallelizable)**: Finds the minimum weight edge for each connected component simultaneously and adds them to the forest, merging components in rounds.

Round 2 Analysis:

Algorithm	Time Complexity	Best Graph Type	Memory Usage	Implementation Difficulty	Remarks
Borůvka's Algorithm	O(E log V)	Any	High	Hard	**8/10**
Kruskal's Algorithm	O(E log E)	Sparse Graph	Medium	Moderate	**8/10**
Prim's Algorithm	O(E log V)	Dense Graph	Low	Easy	**9/10**

Complexity Graph:





Winner: The Puzzle Man - As the Village is dense with electric poles.

Round 3 Result

Maij	Lumihs	Puzzle Man

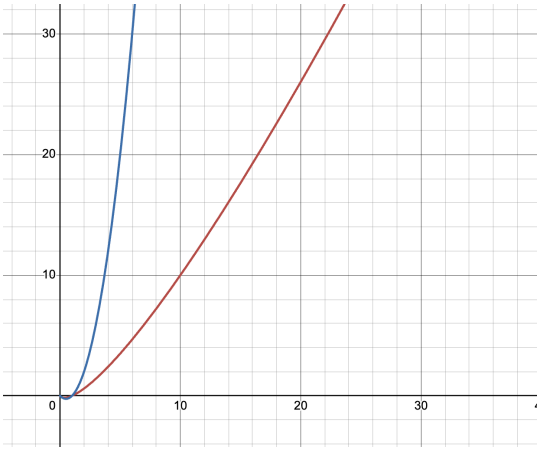
Algorithm FractionalKnapsack(Items[], Capacity W):
 	
for each Item i in Items:
i.ratio = i.value / i.weight
 	
Sort Items by i.ratio in descending order
 	
totalValue = 0
currentWeight = 0
 	
for each Item i in sorted Items:
if currentWeight + i.weight <= W:
currentWeight = currentWeight + i.weight
totalValue = totalValue + i.value
else:
remainingCapacity = W - currentWeight
fraction = remainingCapacity / i.weight
totalValue = totalValue + (fraction * i.value)
currentWeight = currentWeight + (fraction * i.weight)
break
 	
return totalValue
	Algorithm Greedy01Knapsack(Items[], Capacity W):
	
	for each Item i in Items:
	i.ratio = i.value / i.weight
	
	Sort Items by i.ratio in descending order
	
	totalValue = 0
	currentWeight = 0
	
	for each Item i in sorted Items:
	if currentWeight + i.weight <= W:
	currentWeight = currentWeight + i.weight
	totalValue = totalValue + i.value
	
	return totalValue
		Algorithm DP01Knapsack(Items[], Capacity W, Number of items n):
		dp[n+1][W+1]
		
		for i from 0 to n:
		for w from 0 to W:
		dp[i][w] = 0
		
		for i from 1 to n:
		for w from 1 to W:
		if Items[i-1].weight <= w:
		value_if_included = Items[i-1].value + dp[i-1][w - Items[i-1].weight]
		value_if_excluded = dp[i-1][w]
		dp[i][w] = max(value_if_included, value_if_excluded)
		else:
		dp[i][w] = dp[i-1][w]
		
		return dp[n][W]

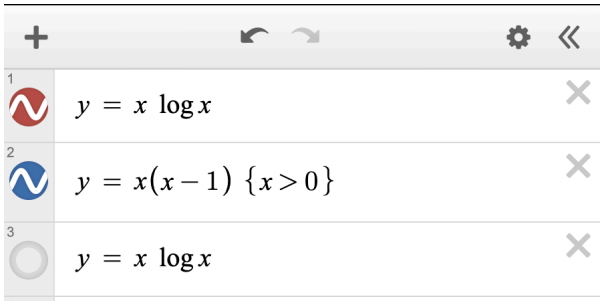
Sorts by value/weight ratio and picks the best items first.	Makes a locally best choice (e.g., max value/weight) which might fail globally.	Solves subproblems and stores results to make globally optimal choices (bottom-up approach).

Round 3 Analysis:

Algorithm	Time Complexity	Optimal Solution	Speed	Space Usage	Remarks
Greedy (Highest Value First)	O(n log n)	No	Very Fast	Low	**6/10**
Greedy (Value/Weight Ratio)	O(n log n)	No	Very Fast	Low	**7/10**
0/1 Knapsack (DP)	O(nW)	Yes	Slower	High	**9/10**

Complexity Graph:





Winner: The Puzzle man, He perfectly utilized capacity using DP.

Round 4 Result

Maij	Lumihs	Puzzle Man

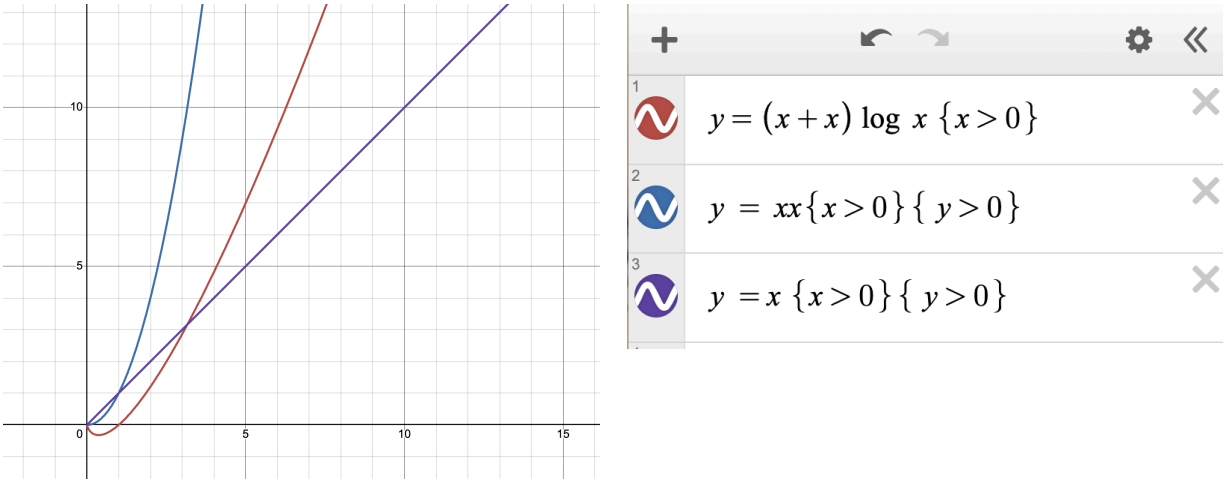
function Dijkstra(Graph, source):
  dist[source] := 0
  for each vertex v in Graph:
  if v is not source:
  dist[v] := INFINITY
  prev[v] := UNDEFINED
  add v to a priority queue Q
 	
  while Q is not empty:
  u := vertex in Q with min dist[u]
  remove u from Q
 	
  for each neighbor v of u:
  alt := dist[u] + length(u, v)
  if alt < dist[v]:
  dist[v] := alt
  prev[v] := u
  // Update v's position in Q if using an efficient priority queue
 	
  return dist[], prev[]
	function BellmanFord(Graph, source):
	  for each vertex v in Graph:
	  dist[v] := INFINITY
	  prev[v] := UNDEFINED
	  dist[source] := 0
	
	  for i from 1 to |V| - 1: // Relax all edges |V| - 1 times
	  for each edge (u, v) with weight w in Graph:
	  if dist[u] + w < dist[v]:
	  dist[v] := dist[u] + w
	  prev[v] := u
	
	  for each edge (u, v) with weight w in Graph: // Check for negative cycles
	  if dist[u] + w < dist[v]:
	  print "Graph contains a negative weight cycle"
	  return FALSE
	
	  return dist[], prev[]
		function A*(start, goal, h):
		
		  create a priority queue openSet ordered by fScore
		
		  gScore[start] := 0
		  fScore[start] := h(start)
		  openSet.push(start)
		
		  while openSet is not empty:
		  current := openSet.pop_min()
		  if current is goal:
		  return reconstruct_path(prev, goal) // Path found
		  for each neighbor neighbor of current:
		  tentative_gScore := gScore[current] + dist(current, neighbor)
		  if tentative_gScore < gScore[neighbor]:
		  Record it!
		  prev[neighbor] := current
		  gScore[neighbor] := tentative_gScore
		  fScore[neighbor] := tentative_gScore + h(neighbor)
		  if neighbor is not in openSet:
		  openSet.push(neighbor)
		
		  return FAILURE // no path found

Greedy	Dynamic Programming	Informed Search (uses heuristic)

Round 4 Analysis:

Algorithm	Time Complexity	Handles Negative Edges	Accuracy	Practical Use	Remarks
Dijkstra's Algorithm	**O(E log V)**	No	**High & reliable**	High	**8/10**
Bellman–Ford	O(VE)	Yes	High	Low	**6/10**
A* Algorithm	Depends on heuristic	No	High (if heuristic good)	Medium	**9/10**

Complexity Graph:



Winner: The puzzle man for finding shortest part using A* algorithm.

Input / Output Analysis:

Round-01:

```
================================================
      PUZZLE MAN ALGORITHM CHALLENGE
        Can You Beat The Village Champion?
      Featuring: Maij vs Lumihs vs Puzzle Man
================================================

TEST ROUND 1: STRING SEARCH ALGORITHMS
================================================

Enter register book text (default: maijlumihsdihutzifanaizrafametafihsin):
>
Using default text

Enter name to search (default: ihsin):
>
Using default pattern

================================================
TESTING ALL ALGORITHMS:

1. NAIVE SEARCH ALGORITHM:
   ✓ Found at position: 32
   Complexity: O(n*m) worst case

2. KMP ALGORITHM:
   ✓ Found at position: 32
   Complexity: O(n+m) guaranteed

3. SUNDAY ALGORITHM:
   ✓ Found at position: 32
   Complexity: Average O(n)

================================================
SUMMARY:
Text length: 37 characters
Pattern length: 5 characters
All algorithms found the pattern!
```

Round-02

```
============================================================
TESTING ALGORITHMS:
============================================================

PRIM'S ALGORITHM:
  Minimum cost: 14 coins
  Complexity: O(E log V)

KRUSKAL'S ALGORITHM:
  Minimum cost: 14 coins
  Complexity: O(E log E)

BORUVKA'S ALGORITHM:
  Minimum cost: 14 coins
  Complexity: O(E log V)

------------------------------------------------------------
RESULTS SUMMARY:

✓ ALL ALGORITHMS AGREE!
Minimum wire cost: 14 coins

------------------------------------------------------------
ALGORITHM COMPARISON:

• Prim's: Best for dense graphs
• Kruskal's: Best for sparse graphs
• Boruvka's: Best for parallel processing

------------------------------------------------------------
VILLAGE BUDGET ANALYSIS:

Village budget: 15 coins
Minimum cost: 14 coins
✓ Project is within budget!
Remaining: 1 coins
```

Round-03:

```
==================================================
TESTING ALGORITHMS:

1. GREEDY FRACTIONAL KNAPSACK:
   Maximum value: 250 coins
   Complexity: O(n log n)
   Approach: Sort by value/weight ratio, take fractions
   Items taken (fractional allowed):
     • Full item: 10kg, 60 coins
     • Full item: 20kg, 100 coins
     • Full item: 15kg, 70 coins
     • Fraction: 5/30kg, 20 coins (16.6667%)
   Total: 250 coins

2. GREEDY 0/1 KNAPSACK:
   Maximum value: 230 coins
   Complexity: O(n log n)
   Approach: Sort by value/weight ratio, take whole items only
   Items taken (whole items only):
     • Item: 10kg, 60 coins
     • Item: 20kg, 100 coins
     • Item: 15kg, 70 coins
   Space left: 5kg
   Total: 230 coins

3. DYNAMIC PROGRAMMING (0/1):
   Maximum value: 230 coins
   Complexity: O(n*capacity) = O(5*50) = O(250)
   Approach: Build DP table, guaranteed optimal


--- RESULTS ---
Fractional (O(n log n)): 250 coins
Greedy 0/1 (O(n log n)): 230 coins
DP 0/1 (O(n*W)):         230 coins

✓ Greedy optimal
```

Round-04:

```
VILLAGE LOCATIONS:
┌─────────┬──────────────┐
│ Index   │ Location     │
├─────────┼──────────────┤
│   0     │ Hilltop      │
│   1     │ School       │
│   2     │ Home-A       │
│   3     │ Home-B       │
│   4     │ Home-C       │
│   5     │ Castle       │
│   6     │ Bank         │
│   7     │ Lotus Garden │
└─────────┴──────────────┘


       🗺️ VILLAGE PATH MAP 🗺️
┌──────────────────────────────────┐
│  (0) Hilltop                     │
│      | 500                       │
│      ▼                           │
│  (1) School ──────────────┐      │
│   | 100     | 200     | 300      │
│   ▼         ▼         ▼          │
│(2)Home-A (4)Home-C (5)Castle     │
│   | 25   /  |         |         │
│   |    100  | 50      | 400     │
│   ▼   /     ▼         ▼         │
│(3)Home-B (6)Bank ◄──────┘        │
│             | 500                │
│             ▼                    │
│         (7) Lotus Garden         │
└──────────────────────────────────┘
Choose:
1. Use default (Hilltop → Lotus Garden)
2. Choose custom start and end
Enter choice (1-2): 1

Using: Hilltop (0) → Lotus Garden (7)

================================================
TESTING ALGORITHMS:

1. DIJKSTRA'S ALGORITHM:
2. BELLMAN-FORD ALGORITHM:
3. A* SEARCH ALGORITHM:

--- RESULTS ---
Dijkstra (O((V+E)logV)): 1250 units
Bellman-Ford (O(V*E)):   1250 units
A* Search:               1250 units

✓ All agree: 1250 units
```

Result Analysis:

The evaluation was conducted in four competitive rounds, followed by a final cumulative result. Three contestants—Maij, Lumihs, and Puzzle Man—applied different algorithms to solve core problems. Performance was judged based on efficiency, accuracy, practicality, and implementation quality, with each round scored out of 10.

Round 1: String Search	**Round 2: Minimum Spanning Tree (MST)**
Maij (Naive Algorithm): Simple but slow for large inputs — 5/10 Lumihs (KMP): Efficient and reliable — 7/10 Puzzle Man (Sunday Algorithm): Fast in practice and easy to implement — 9/10	Maij (Prim's): Efficient for dense graphs — 8/10 Lumihs (Kruskal's): Best for sparse graphs — 8/10 Puzzle Man (Borůvka's): Strong theoretical performance — 9/10 All algorithms produced an MST with 13 coins.
Winner: Puzzle Man	Winner: Puzzle Man

Round 3: Knapsack Problem	**Round 4: Path Finding**
Maij (Greedy 0/1): Fast but not optimal — 6/10 Lumihs (Greedy Fractional): Better results but not always optimal — 7/10 Puzzle Man (DP 0/1): Guaranteed optimal solution — 9/10 Optimal value: 230 coins	Maij (Dijkstra): Fast and reliable — 8/10 Lumihs (Bellman–Ford): Handles negatives but slower — 6/10 Puzzle Man (A*): Very fast with good heuristic — 9/10 Shortest distance: 1250 units
Winner: Puzzle Man	Winner: Puzzle Man

Final Results

Contestant	**Total Score**	**Position**
Maij	27 / 40	3rd
Lumihs	28 / 40	2nd
Puzzle Man	36 / 40	1st

DLOG Medalist: The Puzzle Man.

Conclusion:

This project compared different algorithms through four competitive rounds: string search, minimum spanning tree, knapsack optimization, and path finding. The results clearly show that choosing the right algorithm for a problem is more important than choosing the simplest one.

Simple algorithms like Naive pattern matching and Greedy knapsack were easy to implement and fast, but they often gave slower performance or non-optimal results. More advanced algorithms such as KMP, Dynamic Programming, and A* required more understanding, but they produced better and more reliable outcomes.

Across all rounds, Puzzle Man consistently achieved the highest scores by selecting algorithms that balanced speed, accuracy, and practical efficiency. This is why Puzzle Man became the overall winner and DLOG Medalist.

“After a long time analysis Dihut and Zifan came in front of the crowd and announced the winner. It is none other than everyone's favourite "