Design and Analysis of Algorithms

Digital Assignment 1

SLOT : A2 + TA2

COURSE CODE : BCSE204

1: Backtracking — Knights Tour on Chess board

i) Problem Statement

The Knight's Tour problem is a classic problem in the realm of backtracking algorithms. The objective is to find a sequence of moves for a knight on a chessboard such that the knight visits every square exactly once.

ii) Explanation with a sample problem

Let's consider an 8 × 8 chessboard. We start with the knight placed on any square on the board. The goal is to move the knight around the board, covering all squares exactly once, using only the allowed moves of the knight.

Suppose we start with the knight at position (0,0) on the chessboard.

We use the following notation to represent moves :—

→ Up moves : U
→ Down moves : D
→ Left moves : L
→ Right moves : R

One possible solution to the Knight's Tour problem :

1. Start at (0,0)
2. Move to (2,1) (two squares right, one square up) — RRU
3. Move to (0,2) — ULL
4. Move to (1,0) — DUR
5. Move to (3,1) — RRD
6. Move to (1,2) — URU
7. Move to (0,4) — UULL
8. Move to (2,5) — RRDD
9. Move to (4,4) — RDDLL
10. Move to (6,5) — RRUUR
11. Move to (7,7) — RRUURU

And so on,

until all squares are covered.

iii) Pseudocode

```
Procedure KnightTour(board, n, row, col, move_count)
  if move_count equals n*n
    return true // all squares visited

  for each legal move (next_row, next_col)
  from (row, col)
    if next_row and next_col are inside
    the board and board [next_row][next_col]
    is not visited
      mark (next_row, next_col) as visited

      if KnightTour(board, n, next_row,
      next_col, move_count + 1)
        return true // tour completed
                       successfully

      unmark (next_row, next_col)

  return false // no tour possible
                  from this point
```

2. Least Cost Branch and Bound

– Travelling Salesperson Problem

i) Problem Statement

The Traveling Salesperson Problem (TSP) is a well-known problem in a combinatorial optimization.

Given a list of cities and the distances between each pair of cities, the task is to find the shortest possible route that visits each city exactly once and returns to the original city.

ii) Explanation with a sample problem.

Consider a scenario where a salesperson needs to visit four cities: A, B, C and D.

The distances between these cities are as follows :—

→ A to B : 10

→ A to C : 15

→ A to D : 20

→ B to C : 35

→ B to D : 25

→ C to D : 30

The objective is to find the shortest possible route that visits all cities exactly once and returns to the starting city.

One possible solution to this problem is the route :

$$A \rightarrow B \rightarrow D \rightarrow C \rightarrow A$$

with a total distance of :

$$10 + 25 + 30 + 15$$

$$= \underline{\underline{80 \text{ units}}}.$$

iii) Pseudocode

```
Procedure TSP (branch, bound, cost, path)
    if branch is equal to n  // all cities visited
        if cost + distance (path [n], path[1] < min_cost

            update min_cost and optimal_path
        return

    for each city not yet visited
        if cost + lower_bound(path) < min_cost
            // pruning
            update bound with the new lower bound

            TSP (branch + 1, bound, new_cost,
                 new_path)
```

The search space is explored by generating child nodes from the current node, representing potential routes. The priority queue ensures that we explore the most promising routes first, based on their estimated costs. As we traverse the search space, we prune branches that are unlikely to lead an optimal solution, thus improving efficiency.

3. Line Segments and its properties

i) Line segments are portions of lines that have a defined starting and ending point. They are characterized by various properties.

1. Length : The distance between the two endpoints.

2. Direction : Determined by the orientation of the endpoints.

3. Slope : The ratio of vertical change to horizontal change between two points.

4. Intersection : Two line segments intersect if they share a common point. In this case, the intersection point must lie within the range of both segments.

The naive approach to determining the intersection of two line segments involves checking all possible combinations of endpoints and ensuring that they do not lie on the same side of the other segment's line. If any two endpoints of the segments are collinear, further checks are needed to verify that the intersection point falls within the range of both segments.

ii) Pseudocode for Naive Intersection Approach

```
Procedure intersect (segment1, segment2)
  // define endpoints of segments

 (x1, y1), (x2, y2) = segment1.start, segment1.end
 (x3, y3), (x4, y4) = segment2.start, segment2.end

  // calculate slopes
 slope1 = (y2 - y1) / (x2 - x1)
 slope2 = (y4 - y3) / (x4 - x3)

 // check for parallel segments
 if slope1 equals slope2
     return "Segments are parallel, no intersection"

 // calculate intersection point
 intersection x = ((x1 * y2 - y1 * x2) * (x3 - x4)
 - (x1 - x2) * (x3 * y4 - y3 * x4)) /
 ((x1 - x2) * (y3 - y4) - (y1 - y2) * (x3 - x4))

 intersection y = ((x1 * y2 - y1 * x2) * (y3 - y4)
 - (y1 - y2) * (x3 * y4 - y3 * x4)) / ((x1 - x2)
 * (y3 - y4) - (y1 - y2) * (x3 - x4))

 // check if intersection point lies within
 both segments

 if (intersection x >= min(x1, x2) and
 intersection x <= max(x1, x2) and
 intersection y >= min(y1, y2) and intersection y
 <= max(y1, y2)) and
```

(intersection_x >= min(x3, x4) and intersection_x <= max(x3, x4) and intersection_y >= min(y3, y4) and intersection_y <= max(y3, y4))

return "Segments intersect at point (intersection_x, intersection_y)"

else

return "Segments do not intersect"

4. Sweeping Lines — Properties and Applications

Sweeping lines typically refer to curves or lines that extend smoothly and continuously.

Properties :—

1. Continuity : Sweeping lines exhibit smooth and continuous curves without abrupt changes in directions.

2. Variable Curvature : These lines can have varying degrees of curvature along their path, allowing for flexibility in design.

3. Infinite points : Sweeping lines theoretically have an infinite number of points, making them suitable for creating intricate shapes.

4. Elegance : The aesthetic appeal of sweeping lines often lies in their graceful and flowing appearance.

Applications :—

1. Industrial design : Sweeping lines are commonly used in product design, adding a sleek and modern look to objects like cars, furniture, and electronic devices.

2. Architecture : Architects incorporate sweeping lines in building facades and interior designs to create visually striking and dynamic spaces.

3. Graphic design : Sweeping lines are employed for creating dynamic and visually appealing illustrations, logos and patterns.

4. Automotive design : Car designers utilize sweeping lines to enhance aerodynamics, improve aesthetics, and convey a sense of motion in vehicle exteriors.

5. Fashion design : Sweeping lines are often integrated into clothing designs to create elegant and flattering silhouettes.

6. Animation and digital art : Artists use sweeping lines to convey motion and fluidity in animated characters and digital art compositions.

These properties and applications showcase the versatility and aesthetic appeal of sweeping lines in various creative fields.