Homework 5

Apply greedy algorithm to the problem of the minimum length triangulation of a convex polygon. How far can greedy be away from optimum? Show the worst case for a pentagon.

**Algorithm:**

Step 1: Add all the vertices of Polygon (p) to set the Vertices(v) which forms a set.

Step 2: Then add all the diagonals which is closer into set (s) which is the heap.

For example, the diagonal has an vertices vd and vd+2

Step 3: Now find the minimum diagonal in set (s) and add it to the new set (J) of smallest length triangulation of (P).

Step 4: Elimination begins by eliminating the vertex vd+1.

Step 5: From the above step remove the diagonal which is related to the eliminated vertex of set V.

Step 6: In this step, change the heap (S) to add not in the neighbor diagonals for the

vertices vd and vd+2.

such as, (d,d+3, (d+2, d-1))

Step 7: Now have to check whether heap E is empty or not. If it is not empty, then have to repeat the step 3 otherwise return the J value which is the minimum length of triangulation.

**Optimum solution for greedy:**

The greedy algorithm will find the optimum solution based on the criteria or the condition given.

Hence for some condition it will find the optimum solution whereas for some case it is difficult

to find out.

The optimization can be defined if the mathematical form is **well behaved** and the greedy

algorithm improves the objective of each iteration function and find the optimal solution.

If the mathematical form is **not well behaved,** then it has to know the **local** optimum.

**Worst Case:**

The worst case for the pentagon is **O(*n2*)**.

**Function to find the minimum cost for convex polygon triangulation:**

// function to find minimum cost for convex polygon

triangulation.

double minPolygon (Point pnt[], int n)

{

// To form a triangle there should be at least 3 points

if (n < 3)

//returns the value

return 0;

// Table is created to store the cost of [i] and [j]

double tab[n][n];

// loop proceeds until "n" times

for (int g = 0; g < n; g++)

{

// checks for the value of i and j

for (int i = 0, j = g; j < n; i++, j++)

{

// checks if j < i+2

if (j < i + 2)

// stores the data

tab[i][j] = 0.0;

// otherwise

else

{

// stores the maximum value in table

tab[i][j] = MAXIMUM;

// checks for the condition

for (int k = i + 1; k < j; k++)

{

/\*the "val" is declared in double and

the required data are stored into the

variable\*/

double val = tab[i][k] + tab[k][j] +

cost (pnt, i, j, k);

/\*checks whether table is greater than

"val"\*/

if (tab[i][j] > val)

//store the "val" in table

tab[i][j] = val;

}

}

}

}

//returns the value

return table [0][n-1];