Homework 1

Haoran Liang
hl74
Sep 11, 2019

1. Describe two trade-offs between the Bug 1 and Bug 2 algorithms.

a) **Time:** Time is one trade-off between Bug 1 and Bug 2 algorithm. For Bug 1, the robot will travel along the perimeter of the obstacle and record the minimum distance on the obstacle's perimeter from the target point. Thus, robot needs to circle around the obstacle to know the shortest path. The more obstacles, the more distance a robot needs to travel. Different from the Bug 1 algorithm, Bug 2 algorithm will go along the line-m between the start point and target point. The robot will leave the obstacle once it back to the line-m and able to go straight to the target point. Therefore, under most cases, Bug 2 will use less time to reach the target point than Bug 1 algorithm.

b) **Stability:** We know that Bug 1 will circle around an obstacle and find the shortest path from the obstacle towards target point. Thus Bug 1 is an exhaustive search algorithm which will look all choices and find the optimal one. This algorithm may sacrifice time but will have higher stability. We know that Bug 2 will travel along the intersection line-m towards target point. If the obstacle has multiple intersection points with line-m, then Bug 2 will create several inner loops when finding the shortest path. Therefore, the time spent will increase exponentially. However for Bug 1 algorithm, it will always circle around the obstacle no matter obstacle's shape. Thus, Bug 1 has better stability and safer than Bug 2 algorithm.

2. Visibility Graph

a) The upper-bound time complexity should be $O(n^3)$. Suppose we have n vertices, including start point and goal point. Thus, we could compose all vertices pairs such as $<v_1, v_2>$ using $O(n^2)$ times. Then for each distinct vertices pair, we need to check if it is intersect with all obstacle boundary edges. For n vertices, there should be also n edges, and the intersection check function costs constant time. Therefore, the total time complexity should be $O(n^3)$.

Pseudocode:

```
Input: a set of vertices n, and a set of n obstacle boundary edges
for v1 in n vertices do
    for v2 in n vertices do
        if v1 != v2 then we have pair<v1, v2>
        for e in n boundary edges do
            if <v1, v2> is intersects with boundary edge e
                return
            else
                add <v1, v2> as a new edge
```

b) Yes, we can use **Dijkstra algorithm** to find the shortest path from start to goal.
For each vertices we will look for its shortest path to all other connect edges, and the upper bound of time complexity should be $O(n^2)$

Pseudocode:

```
Input: a set of n vertices
Create map[][] as a record distance between two vertices
Create dist[] as shortes distance
Create visted[] as visted vertices
for i from 0 to n vertices do
    dist[i] = map[0][i]
visted[0] = true

for i from 1 to n vertices do
    Create min to MAX_VALUE
    for j from 0 to n vertices do
        if !visted[i] && dist[j] < min
            set min = dist[j]
    visited[j] = true

    for k from 0 to n vertices do
        if !visited[k] && dist[k] > dist[j] + map[j][k]
            dist[k] = dist[j] + map[j][k]
```

3. Algebraic primitive

We have single algebraic primitive $H = \{ (x, y) \mid x^2 + y^2 <= 1 \}$. Thus, we know that this primitive is a circle where the center is the origin. We assume there is a point *A* inside of the circle, where *A'* is the point after rotation.

Suppose: point $A = (x , y)$ and angle between x-axis is *a*, and the distance from A to origin is *d*

point $A' = (x', y')$ and angle of rotation is *b*, and the distance from A' to origin is also *d*

Thus we have:

$x' = d * \cos(a + b)$ and $y' = d * \sin(a + b)$

$d = x / \cos(a) = y / \sin(a)$ so that $x = d * \cos(a)$, $y = d * \sin(a)$

By trigonometric function we know that:

$\cos(a + b) = \cos(a) * \cos(b) - \sin(a) * \sin(b)$

$\sin(a + b) = \sin(a) * \cos(b) + \cos(a) * \sin(b)$

So that:

$x' = d * (\cos(a) * \cos(b) - \sin(a) * \sin(b)) = x * \cos(b) - y * \sin(b)$

$y' = d * (\sin(a) * \cos(b) + \cos(a) * \sin(a)) = y * \cos(b) + x * \sin(b)$

Now we can test if $x'^2 + y'^2 <= 1$:

$$
\begin{aligned}
x'^2 + y'^2 &= x^2\cos(b) - 2*\cos(b)*\sin(b) + y^2\sin(b) + y^2\cos(b) + 2*\cos(b)*\sin(b) + x^2\sin(b) \\
&= x^2\sin(b) + x^2\cos(b) + y^2\sin(b) + y^2\cos(b) \\
&= x^2 * (\sin^2(b) + \cos^2(b)) + y^2 * (\sin^2(b) + \cos^2(b)) \\
&= x^2 * 1 + y^2 * 1 = x^2 + y^2
\end{aligned}
$$

Thus, we proved that point $A' = (x', y')$ is inside of the circle after rotation.

4. Two line intersection

Pseudocode:

```
void computeIntersectionPoint(pair<A1, B1>, pair<A2, B2>):
    // We first check if two line segments are parallel
    if computeCrossProduct((A1.x - B1.x), (A2.y - B2.y), (A1.y - B1.y), (A1.x - B2.x)) == 0
        // We need to check if two line is collinear
        // Collinear means two lines may lie on the same line
        // We chose one point from each segment and calculate cross product with one segment line
        if computeCrossProduct((A1.x - B1.x), (A1.y, B1.y), (B1.x - A2.x), (B1.y - A2.y)) == 0
            // CrossProduct is 0 means two lines are collinear
            if(compareTwoNode(A1, A2) <= 0 && compareTwoNode(B1, B2) >= 0)
                // We have A1 ---- A2 ---- B1 ---- B2
                return A2
            else if(compareTwoNode(A1, A2) >= 0 && compareTwoNode(B1, B2) <= 0)
                // We have A2 ---- A1 ---- B2 ---- B1
                return A1
            else
                return 'no intersection'
        else
            // Since two lines are not collinear, then there is not intersection
            return 'no intersection'
    else
        // Two lines are not parallel, then they may have intersection
        if compareTwoNode(A1, A2) == 0 || compareTwoNode(A1, B2) == 0
            // Two lines intersect at A1
            return A1
        else if compareTwoNode(B1, A2) == 0 || compareTwoNode(B1, B2) == 0
            // Two lines intersect at B1
            return B1
        else
            // Two lines intersect on a point which is not the four endpoints
            // We know that for point (x1,y1), (x2,y2)
            // The line equation is (x - x1) / (x2 - x1) = (y - y1) / (y2 - y1)
            // Then x = t(x2 - x1) + x1 and y = s(y2 - y1) + y1
            // Thus the intersect point is (t(B1.x - A1.x) + A1.x , s(B1.y - A1.y) + A1.y)
            return (t(B1.x - A1.x) + A1.x, s(B1.y - A1.y) + A1.y)
```

```
// Compare two node
int compareTwoNode(node a, node b):
    if a.x < b.x || a.x == b.x && a.y < b.y
        return -1
    else if a.x > b.x || a.x == b.x && a.y > b.y
        return 1
    else if a.x == b.x && a.y == b.y
        // Two points are the same
        return 0


// CrossProduct = x1 * y2 - y1 * x2
// If crossProduct is 0, then two lines are parallel
void computeCrossProduct(point x1, point y1, point x2, point y2):
    return x1 * y2 - y1 * x2
```