DESIGN AND ANALYSIS OF ALGORITHMS
Homework 5

Aman Choudhary
MTech Coursework, CSA 2020
Sr No: 17920

November 14, 2020

1 Matching points on circle

1.1 Problem

We are given a set of n red points and n blue points on a circle. We need to create a matching between the red and the blue points, i.e., each red point needs to be assigned to a unique blue point. An edge in the matching is the line segment connecting a red point and a blue point. A matching is considered valid if no pair of edges in the matching cross each other; otherwise, it is considered invalid. Our task is to compute a valid matching (if it exists) such that length of the longest edge in the matching is minimized.

1.2 Input Set

Our input is a set of $2n$ points. We assume each point is associated with three attributes: x-coordinate, y-coordinate and color. Let us first assign an ordering to these $2n$ points. We pick an arbitrary point on the circle and call it p_1. Then, we traverse along the circumference of the circle in clockwise fashion (direction is again randomly picked) and number the points as $p_2, p_3, ..., p_{2n}$. Thus, our points can be written as,

$$P = \{p_1, p_2, p_3, ..., p_{2n}\}$$

1.3 Key Terms

(i) **Cost of Matching**: We define the cost of any valid matching to be the **length of the longest edge in that matching**. The cost of matching over an empty set is 0, since there are no edges. Also, we assign $+\infty$ to be the cost of an invalid matching.

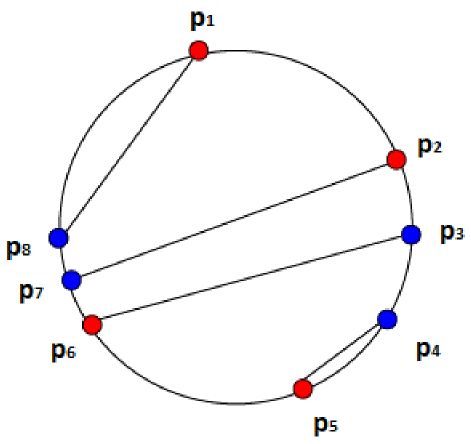
(ii) $S(i, j)$ denotes the **ordered sequence of points** $(p_i, p_{i+1}, ..., p_{j-1}, p_j)$, where $i \leq j$.

(iii) $M(i, j)$ is essentially a **set of edges** which make up the **matching** over $S(i, j)$. $M(i, j) = \phi$, when no valid matching exists for $S(i, j)$.

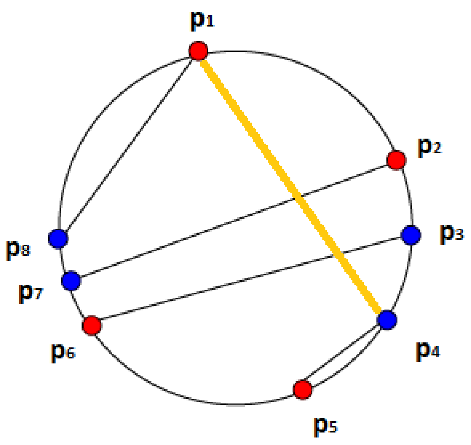
(iv) $C(i, j)$ represents the **optimal (minimum) cost of matching** over all the possible matchings on sequence of points $S(i, j)$. Thus, our final goal is to compute $C(1, 2n)$.

(v) $w(p_i, p_k)$ is defined as **weight** of edge between points p_i and p_k. When points are of different colors, the weight is simply the **Euclidean Distance** between the points. Otherwise, it is defined to be $+\infty$, because that matching will result in an invalid matching.

$$w(p_i, p_k) = \begin{cases} \sqrt{(x_i - x_k)^2 + (y_i - y_k)^2} & \text{, if } p_i.color \neq p_k.color \\ +\infty & \text{, if } p_i.color = p_k.color \end{cases}$$



(a) Ordering the points.



(b) An example breaking of problem (i=1, j=8, k=4). S(1,8) is divided into S(2,3) and S(5,8).

1.4 Optimal Substructure

Let us consider the problem of matching points over $S(i, j)$. We consider the point p_i. Now, if a matching exists over $S(i, j)$, then there must exist a point p_k, such that p_i forms an edge with p_k (obviously, p_i and p_k are of different color). This point p_k could be any point from $S(i + 1, j)$. Hence, we will need to consider all possible choices $i + 1 \leq k \leq$ j. Thus, our solution will be constructed using an **"optimality by exhaustion"** approach: we consider every possible subproblem solution, and then take the optimal one.

Now, let us see what happens when p_i forms an edge with p_k. The edge (p_i, p_k) divides $S(i, j)$ into two non overlapping sequences $S(i + 1, k - 1)$ and $S(k + 1, j)$. Our original problem of matching over $S(i, j)$ is now reduced to matching over two subproblems $S(i + 1, k - 1)$ and $S(k + 1, j)$. We create matching for these two subproblems and take union of these matchings and edge $e(p_i, p_k)$ to generate one possible matching (via edge to p_k) for $M(i, j)$.

1.5 Valid Matching

When we construct a matching over $S(i+1, k-1)$, all the edges occur only between points $p_{i+1}, p_{i+2}, ..., p_{k-2}, p_{k-1}$. Again, while matching over $S(k + 1, j)$, all the edges occur only between points $p_{k+1}, p_{k+2}, ..., p_{j-1}, p_j$. Now, we build the matching over $S(i, j)$ (via edge to p_k) as follows,

$$M(i, j) = M(i + 1, k - 1) \cup e(p_i, p_k) \cup M(k + 1, j)$$

Hence, there exists no edge whose one end point lies in $S(i + 1, k - 1)$ and the other lies in $S(k + 1, j)$. As a result, we introduce no edge which crosses our chosen edge $e(p_i, p_k)$. **Therefore, we conclude that if there exists valid matchings for $S(i + 1, k - 1)$ and $S(k + 1, j)$, then the matching we build for $S(i, j)$ using these two subproblems is also valid.**

1.6 Recurrence relation for $C(i, j)$

While forming an edge from p_i to p_k, we need to consider all possible choices for p_k such that $i + 1 \leq k \leq j$. Then, after computing $M(i + 1, k - 1)$ and $M(k + 1, j)$, we get one possible **candidate matching** for $S(i, j)$. The cost of this candidate matching is,

$$C(i, j)_k = max\{\ C(i + 1, k - 1),\ C(k + 1, j),\ w(p_i, p_k)\ \}$$

Here, subscript k in $C(i, j)_k$ denotes that we are forming an edge from p_i to p_k. The optimal solution will be generated by considering all such possible candidate matchings, and picking the one with minimum cost (optimality by exhaustion). Hence, the optimal cost is given by the following **recurrence relation**,

$$C(i, j) = \min_{i+1\ \leq\ k\ \leq\ j} C(i, j)_k$$

$$C(i, j) = \min_{i+1\ \leq\ k\ \leq\ j} \{\ max\{\ C(i + 1, k - 1),\ C(k + 1, j),\ w(p_i, p_k)\ \}\ \}$$

1.7 Base Cases and Special Case

We define $C(i, j)$ for few base cases:

$$C(i, j) = \begin{cases} 0 & \text{, if } j < i \\ w(p_i, p_k) & \text{, if } j - i + 1 == 2 \\ +\infty & \text{, if } j - i + 1 \text{ is odd} \end{cases}$$

Let n denote the size of subproblem. It is precisely, the number of points in sequence $S(i, j)$.

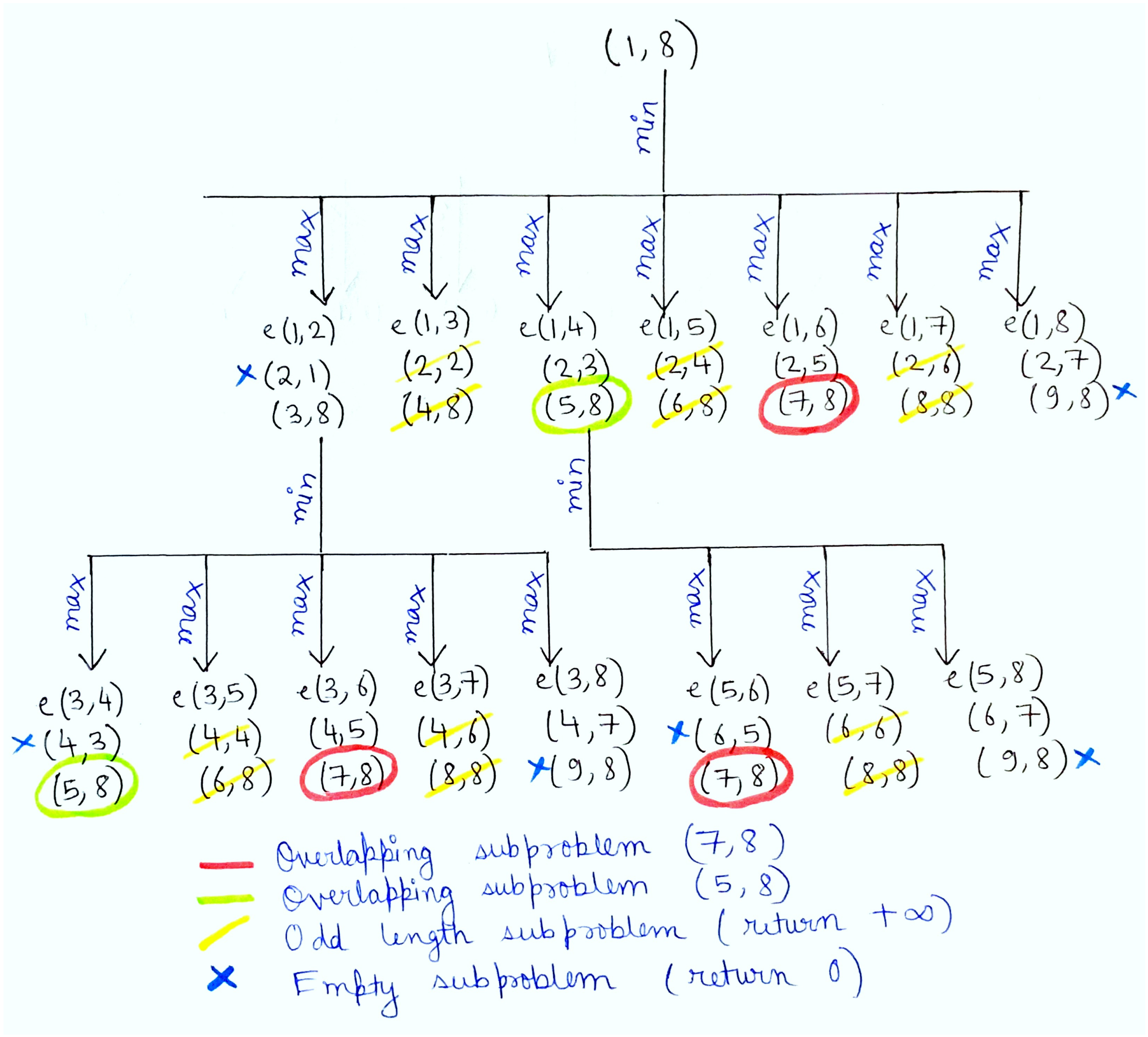
(i) **Base case (n = 0) :** Our subproblems are defined as computing matching over $S(i, j)$ where $i \leq j$. When $i > j$, the sequence is empty, and cost of matching is 0, as we have no points.

(ii) **Base case (n = 2) :** It is a trivial problem, and the cost of matching is defined as the weight of the edge between the two points.

(iii) **Special Case (n is odd) :** When there are odd number of points, no matching is possible, since one point will always be left out. Hence the cost is defined to be $+\infty$.

1.8 Overlapping Subproblems

Here, we show a partially expanded recursive tree of our problem. Only few nodes have been expanded for sake of brevity. We can clearly see there are overlapping subproblems $(5, 8)$ and $(7, 8)$.



1.9 Time Complexity

We will solve the above problem using **dynamic programming** as it has an optimal substructure as well as overlapping subproblems. We use a matrix $Cost[1...2n][1..2n]$ with $2n$ rows and $2n$ columns. The final solution i.e. the minimum cost of matching over $S(1, 2n)$ will be stored in $Cost[1][2n]$. We will only use **upper half** of the matrix because our matchings need to be computed over $S(i, j)$ such that $i \leq j$. The number of entries in the upper half of matrix defines the total number of subproblems. Hence,

$$\textbf{Total number of subproblems} = \frac{n * (n-1))}{2} = O(n^2)$$

The entry $Cost[i][j]$, stores the minimum cost over all possible candidate matchings on $S(i, j)$. It is computed using the recurrence relation defined above. The entry $Cost[i][j]$ denotes a problem of size of $j - i + 1$. In order to compute this entry, we need to check over all possible candidate matchings. As defined in the recurrence relation, a total of $j - i$ cases will be checked, one for each k ($i + 1 \leq k \leq j$), while filling $Cost[i][j]$. Hence, to solve a problem of size $j - i + 1$, we need to examine $j - i$ cases. Since, the time to solve an individual subproblem depends on its size, the time complexity for solving each subproblem is $O(n)$. Hence,

Total Time Complexity = Total number of subproblems * Time to solve a subproblem

$$= O(n^2) \; * \; O(n) \; = \; O(n^3)$$

NOTE: Ideas and different approaches pertaining to solving of problem were discussed with, **Shashank Singh (M.Tech Coursework, CSA 2020).**