You are given a convex polygon $P = \{v_1, v_2, \ldots, v_n\}$, where v_i, $1 \leq i \leq n$, are the vertices that form the polygon. In a convex polygon, each internal angle is less than or equal to 180 degrees, and each line segment between vertices remain inside or on the boundary of the polygon.

A triangulation of a convex polygon is a set of non-intersecting diagonals that partitions the polygon into triangles. We define that the weight of a triangulation is the sum of the lengths of the perimeters of all component triangles. The minimum triangulation problem is to find triangulation of the given polygon that has the minimum weight.

Denote the cost of a minimum triangulation of the polygon formed by $\{v_i, v_{i+1}, \ldots, v_j\}$ as $c(i, j)$. Then the minimum triangulation has a weight of $c(1, n)$.

(a) Fill the blanks below to compute the recurrences which can be used for a dynamic programming algorithm to solve this problem.

$$c(i,j) = \begin{cases} 0 & \text{, \_\_\_(a)\_\_\_} \\ \min_{i<k<j}\{\text{\_\_(b)\_\_}\} & \text{, otherwise} \end{cases}$$

(b) Using the recurrence shown in the previous subproblem to develop a dynamic programming algorithm to solve the problem as efficient as possible. What would be its time complexity in terms of n, the number of vertices in the given convex polygon P? 【102 年台大資工所】

2022. 4.4
11:58

Ans.

(a)

$$c(i,j) = \begin{cases} 0, & \text{if } j \le i+2 \\ \min_{i<k<j} \{ c(i,k) + c(k,j) + dist(i,k) + dist(k,j) + dist(i,j) \}, & \text{otherwise} \end{cases}$$

(b)

```
double TDP(Point P[], int n) {
    if(n<3) return 0;
    int c[n][n];
    for(int gap=0; gap<n; gap++) {
        for(int i=0, j=gap; j<n; i++, j++) {
            if(j<i+2) c[i][j]=0.0;
            else {
                c[i][j] = INT_MAX;
                for(int k=i+1; k<j; k++) {
                    int temp = c[i][k]+c[k][j]
                        + cost(P, i, j, k);
                    if(c[i][j]>temp)
                        c[i][j] = temp;
                } // for 的
            } // else 的
        } // for 的
    } // for 的
    return c[0][n-1];
}
```