In this lecture, we covered Rectangular Partitions problem with Dynamic Programming and Divide-and-Conquer.

### 24.1 Rectangular Partitions Question

Connect all the points with same lab together by wires located inside the large rectangle and on the outside of the interior rectangles. The problem is computationally intractable.

![Rectangular Partitions](image1.png)

Figure 1: Rectangular Partitions

The question is to find the minimum of sum of the length of all wires.

A good way is reduce Rectangular Partitions to Channel Routing Problems each of which can be solved independently.

![Channel Routing Problems](image2.png)

Figure 2: Channel Routing Problems
Steps:

- Replace each interior rectangle by its four corner points.
- Partition a rectangle with interior points so that all the points end up on a partitioning line.
- Then delete the segments inside the interior rectangles to obtain the individual channels.

One thing should be paid attention to is: Line segments cannot be acrosed.
The following table is the comparison of approximation methods to the problem:

<table>
<thead>
<tr>
<th>Approximation Algorithm</th>
<th>time complexity</th>
<th>approximation bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamic Programming</td>
<td>(O(n^5))</td>
<td>2 (simple proof)</td>
</tr>
<tr>
<td>Problem transformation</td>
<td>(O(n^2))</td>
<td>3</td>
</tr>
<tr>
<td>Divide-and-Conquer</td>
<td>(O(n \log n))</td>
<td>4</td>
</tr>
</tbody>
</table>

### 24.2 Guillotine Partition

Definitions: assuming there is \(E(I)\): Rectangular partition for \(I = (R, P)\)

- Guillotine cut: A line segment in Rectangular partition that partitions it into two rectangles.
- Guillotine partition: \(E(I)\) is a guillotine partition if either \(E(I)\) is empty or \(E(I)\) has a guillotine cut that partitions \(R\) into \(R_1\) and \(R_2\), and both \(E(I_1)\) (edges from \(E(I)\) in \(R_1\)) and \(E(I_2)\) (edges from \(E(I)\) in \(R_2\)) are guillotine partitions for \(I_1 = (R_1, P_1)\) and \(I_2 = (R_2, P_2)\), respectively.

One thing should be paid attention to is: A guillotine partition is a rectangular partition, but a rectangular partition is not necessarily a guillotine partition.

### 24.3 Chain Matrix Multiplication

24.3.1

If we want to do multiplication on \(A \times B\), \(A\) and \(B\) must be in the form of \(R_1 \times R_2\) and \(R_3 \times R_4\) where \(R_i\) is the number of columns and \(|R_2| = |R_3|\).

24.3.2

Assuming \(m = m_1, m_2, m_3, ..., m_i, ..., m_n\), we define the form in \(m_i\) looks like \(R_{i-1} \times R_i\), for example: \(m_1 = R_0 \times R_1\), \(m_n = R_{n-1} \times R_n\), then we can calculate \(C_{ij}\) by following equation:

\[
A \times B = C_{ij} \text{ where } A = P \times Q \text{ and } B = Q \times R, \text{ then } \]
\[
C_{ij} = P \times R \text{ and the time complexity is } O(PQR) \]
\[
C_{ij} = \sum_{k=1,1 \leq i \leq P, 1 \leq j \leq R}^{q} a_{ik} b_{jk}
\]

24.3.3

As we know in matrix, no matter how we parenthesize the product, the result will be the same:
\[((m_1m_2)m_3)m_4)m_5\) = \((m_1m_2)(m_3m_4)m_5\))

However, the order in which we parenthesize the product will affect the efficiency because \(m_i\) will contributes to different partitions:

in the case of: \(m_i \times ... m_k = R_{i-1} \times R_k\) and \(m_{k+1} \times ... m_j = R_k \times R_{j-1}\), we could find that:

\[C_{ij} = \min\{R_{i-1}R_kR_j + C_{ik} + C_{k+1,j}\}\]

With this equation, we could calculate all the \(C_{ij}\) like:

\begin{table}
\begin{tabular}{cccccccc}
\(C_{11}\) & \(C_{22}\) & \(C_{33}\) & \(C_{44}\) & \(C_{55}\) & ... \\
\(C_{12}\) & \(C_{23}\) & \(C_{34}\) & \(C_{45}\) & ... & ... \\
\(C_{13}\) & \(C_{24}\) & \(C_{35}\) & ... & ... & ... \\
\(\cdots\) & & & & & ... \\
\(C_{15}\) & ... & ... & ... & ... & ... \\
\end{tabular}
\end{table}

\section{24.4 Dynamic Programming method}

\subsection{24.4.1}

Assuming we have following rectangle:

\begin{figure}
\centering
\includegraphics[width=\textwidth]{rectangle.png}
\caption{Figure 6:}
\end{figure}

We would like to find the optimal solution with dynamic programming, then we could think that the minimum edge length guillotine partitioning problem consists of finding a guillotine partition of least total edge length.

\(OPT(i_l, i_v, j_d, j_u): \) Value of optimal solution from problem in the shaded region.

How to solve it? Assuming we have a instance of the question in following rectangle:

- If we partition the rectangle by a horizontal line through point (5,2), then we could have a guillotine partition(1, 6, 2, 6), the one of the solution will be \(x + OPT(1,6,2,6)\)
If we partition the rectangle by a horizontal line through point (3,3), then we could have two guillotine partitions \((1, 6, 1, 3)\) and \((1, 6, 3, 4)\), the one of the solution will be \(x + \text{OPT}(1,6,1,3) + \text{OPT}(1,6,3,4)\).

Do the steps like above and find all the possible solutions:

Then we can get the minimum of the solutions as our optimal solution:

\[
\text{OPT}(1,6,1,6) = \min \{x + \text{OPT}(1,6,2,6), x + \text{OPT}(1,6,1,3) + \text{OPT}(1,6,3,6), x + \text{OPT}(1,6,1,4) + \text{OPT}(1,6,4,6), x + \text{OPT}(1,6,1,5), y + \text{OPT}(2,6,1,6), y + \text{OPT}(1,3,1,6) + \text{OPT}(3,6,1,6), y + \text{OPT}(1,4,1,6) + \text{OPT}(4,6,1,6), y + \text{OPT}(1,5,1,6)\}
\]

### 24.4.2 Approximation Algorithm

Let \(G(I)\) be an optimal guillotine partition and \(L(G(I))\) be the edge length of \(G(I)\).

Let \(E_{\text{opt}}(I)\) be an optimal rectangular partition and \(L(E_{\text{opt}}(I))\) be the edge length of \(E_{\text{opt}}(I)\)

\[
L(G(I)) \leq 2L(E_{\text{opt}}(I))
\]

### 24.5 Divide-and-Conquer

\(O(n \log n)\)-time divide-and-conquer approximation algorithm.

\(L(I)\) represents the total length the line segments in the solution for problem instance \(I\) generated by our algorithm, and \(\text{opt}(I)\) represents the corresponding one in an optimal solution.

Generates solutions within 4 times the optimal solution value, i.e. for every \(I\),

\[
L(I) \leq 4\text{opt}(I)
\]

### 24.5.1 Problem Instance

Assume \(I = ((x, y), (X, Y), P)\), where \((x, y)\) and \((X, Y)\) define a rectangle \(R(x, y)\) is the lower-left corner, and \((X, Y)\) are the dimensions), and \(P = p_1, p_2, ..., p_n\) is a nonempty set of points inside \(R\).
- mid-cut: a line segment orthogonal to the x-axis that intersects the center of the rectangle (i.e., it includes point \((x + \frac{X}{2}, y + \frac{Y}{2})\)).

- end-cut: a line segment orthogonal to the x-axis that contains either one of the leftmost or the rightmost points in \(P\).

Algorithm of PROCEDURE PARTITION:

**Algorithm 1** PROCEDURE PARTITION((\(x, y\), (\(X\), \(Y\)), \(P\))

1: Relabel the dimensions so that \(X \geq Y\)
2: \(P_1 \leftarrow \{p_k | p_k \in P \text{ and } x_k < x + \frac{X}{2}, \text{ where } p_k = (x_k, y_k)\}\)
3: \(P_2 \leftarrow \{p_k | p_k \in P \text{ and } x_k > x + \frac{X}{2}, \text{ where } p_k = (x_k, y_k)\}\)
4: if \(P_1 \neq \emptyset \) and \(P_2 \neq \emptyset\) then
5: Introduce the line segment orthogonal to the x-axis that partitions \(R\) through its center;
6: \(PARTITION((x, y), (\frac{X}{2}, Y), P_1)\);
7: \(PARTITION((x + \frac{X}{2}, y), (\frac{X}{2}, Y), P_2)\);
8: else
9: Let \(c\) be the coordinate value along the x-axis of a point in \(P\) with smallest \(|c - (x + \frac{X}{2})|\);
10: Introduce the line segment orthogonal to the x-axis that partitions \(R\) through the points with x-coordinate value equal to \(c\);
11: Delete from \(P_1\) and \(P_2\) all the points located along the end-cut;
12: if \(P_1 \neq \emptyset\) then \(PARTITION((x, y), (c - x, Y), P_1)\);
13: if \(P_2 \neq \emptyset\) then \(PARTITION((c, y), (X - (c - x), Y), P_2)\);
14: end if