匹配理論的研究

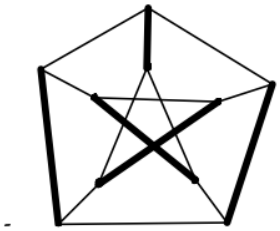
Study of matching theory

指導教授: 謝孫源
專題成員: 曾任謙
開發工具: g++13.3.0
測試環境: Linux Ubuntu 24.04.2

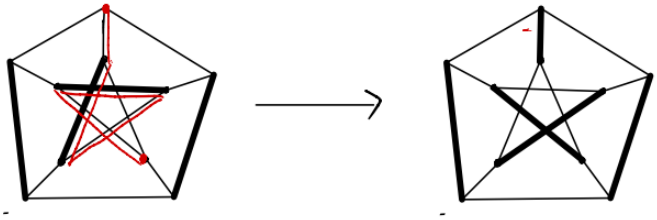
一、簡介：（標題字形16）

本專題聚焦於「一般圖最大匹配問題」的演算法研究與實作，並以 Vazirani 所提出的 $O(n^{0.5}m)$ 時間複雜度演算法為核心內容進行分析與實作。

最大匹配是一個經典的圖論問題，其目標是在一張簡單圖中尋找一個邊集合，使得其中任兩條邊不共享端點，並且整體匹配大小最大。這類問題的主流解法之一，是基於**擴增路徑**（augmenting path）的概念：從一個非最大匹配出發，透過尋找擴增路徑並進行替換，逐步逼近最大匹配。基於上述概念的，最簡單的解法之一，是透過深度搜索(DFS)尋找任何一條 augmenting path，然後進行替換，以此來逐步逼近最大匹配，這個方法的時間複雜度為$O(nm)$。

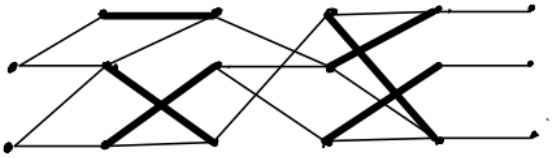


(1.) A Maximum matching on a graph.



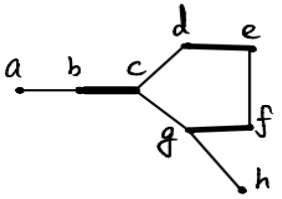
(2.) An augmenting path increases the size of the matching.

在二分圖中，Hopcroft-Karp 演算法透過「**批量尋找最短擴增路徑**」來提升效率，並利用**最短擴增路徑(shortest augmenting path)**長度在增廣過程中的遞增性，成功將複雜度降至$O(n^{0.5}m)$。然而，這種方法難以直接套用到一般圖，原因並非這個概念不適用於一般圖，而是**最短交錯路徑(shortest alternating path)的長度**在一般圖中，就失去了在二分圖中具備的簡單性: **考慮任何一個最短交錯路徑，至路徑上任意點的交錯路徑恰好是一個該點的最短交錯路徑。**



(3.) In Bipartite Graph, on any shortest alternating path p, for any vertex , the path to p is a shortest alternating path of that vertex.

本專題中，我研究並實作了 V. V. Vazirani 提出的$O(n^{0.5}m)$**一般圖最大匹配演算法**。該演算法將 Hopcroft-Karp 的思想推廣到一般圖中，成功處理了一般圖特有的困難結構。其核心概念在於：每個點的最短交錯路徑可以分為偶數長度的 **evenlevel path** 與奇數長度的 **oddlevel path**，並同時可分為 **minlevel path**（較短）與 **maxlevel path**（較長）。minlevel path 可以透過類似 Hopcroft-Karp 的層級建構法獲得，而 maxlevel path 的建構則仰賴稱為 **bridge** 的特殊結構：對於所有具有 maxlevel path 的點，皆存在一個符合條件的 bridge，藉此可以系統性地完成整體層級建構流程。透過這種方式，我們得以完整建立所有點的 evenlevel path 與 oddlevel path，進而得以批量找到最短增廣路徑，最終有效求解一般圖最大匹配問題。



(4.) In General Graph, the path to an vertex on an shortest alternating path, is not necessary to be a shortest alternating path. Consider the shortest alternating path a,b,c,d,e,f,g,h. In the path, a,b,c,d,e,f,g, is not a shortest alternating path of g(a,b,c,g).

系統架構圖:

While (There is an augmenting path):

For each minlevel i:

Give nodes with minlevel i its minlevel.

Find all bridges with tenacity $2i+1$.

For each bridge g with tenacity $2i+1$:

Give all node that its tenacity $2i+1$ and there's a maxlevel path pass the bridge its maxlevel. (Call the process DDFS)

If DDFS find an augmenting path,

Then erase the nodes on it, and those cannot be in other augmenting paths.

If an alternating path is found, then break.