Lecture 1
進階圖論 | Advanced Graph Theory

1.1 圖的種類

1.1.1 樹

一個沒有環的連通圖就是一棵樹。相信大家對樹的性質都已經很熟悉，因此這邊介紹一些進階的技巧。

樹的子樹區間

樹分治

就是在樹上使用分治法，對於一棵樹，每次選擇 v 一個點拔掉，將原來的樹分成很多個子樹，並對這些子樹遞迴求解。最後再將這些子樹合併回去。複雜度爲

$$T(n) = \sum T(m_i) + f(n) \tag{1.1}$$

其中 m_i 表示各個子樹的大小，$\sum m_i = n - 1$，而 $f(n)$ 是合併所需的時間。而這個複雜度跟 $\max m_i$ 非常相關，
如果 $\max m_i \leq n/2$ ，那通常就有不錯的複雜度，比如 $f(n) = n$ 則 $T(n) = n \log n$。因此 v 的選擇就很重要了，要滿足拔掉 v 後剩下的子樹都不會太大。事實上有以下定理。

定理 所有樹都有**重心**。一個點是重心表示拔掉他之後所有的子樹的大小都不超過原來的樹大小的一半。

Proof. 定義 $c(v)$ 爲 v 到所有其他點的距離的合。則使 $c(u)$ 最小的那個點一定是重心。否則

假設本來 (u, v) 相鄰，且拔掉 u 後 v 所屬的子樹的大小大於一半，比較 $c(u), c(v)$ 可以發現有超過一半的點到 u 的距離比到 v 近 1，而不超過一半的點距離遠了 1，
因此 $c(v) < c(u)$ ⇒⇐。 □

例題 1-1 Tree — POJ 1741

給一棵樹，樹的邊有權重表示距離，求距離不超過 k 的點對數。($\mathcal{O}\left(n \log^{\mathcal{O}(1)} n\right)$)

用樹分治的想法，我們就只需要考慮如何在線性時間合併子樹即可。對於每個子樹我們可以先做一次 BFS 找出每個點到當前重心 v 的距離，而合併兩個子樹 T_1, T_2 其實就是在問對於所有 $u_1 \in T_1$，有多少 $u_2 \in T_2$ 滿足 $d(u_2, v) \leq k - d(u_1, v)$。這個可用一些如線段樹等的資料結構在 $\mathcal{O}(n \log n)$ 時間完成 (事實上花點巧思，可以用雙指針在線性時間內做到)。總時間複雜度就是 $\mathcal{O}\left(n \log^2 n\right)$。

啓發式合併

樹分治因爲對最大子樹的大小有限制，所以必需額外進行找重心的動作，有點麻煩。不過再某些情況下可以用啓發式合併的方法簡化。

更詳細的說，如果合併兩個子樹 T_1, T_2 的複雜度「幾乎」只和一者的大小有關，如 $\mathcal{O}(T_2), \mathcal{O}(T_2 \log T_1)$ 等等，那麼可以證明每次合併子樹時，找最大的子樹 T_1，把其他的子樹合併進來，就會有不錯的複雜度。

定理 假設合併兩個子樹 T_1, T_2 的複雜度爲 $\mathcal{O}(T_2 f(n))$，如果用上述的方法，則整體的複雜度是 $\mathcal{O}(n \log(n) f(n))$。

Proof. 合併兩個子樹 T_1, T_2 的複雜度爲 $\mathcal{O}(T_2 f(n))$，可以想作把 T_2 裡的點都丟到 T_1 去，且丟一個點的平均複雜度是 $f(n)$。現在一個點被丟到的新的子樹，表示他原來所屬的子樹大小比當前最大的子樹還小，因此合併後他所屬的新子樹會至少是原來的一半，因此每個點最多被丟到新的子樹 $\mathcal{O}(\log n)$ 次，因此總複雜度是 $n \log(n) f(n)$。 □

1.1.2 平面圖

平面圖就是可以畫在平面上的圖，使得任兩個邊只會在點上相交而已。

而平面圖有一個很重要的定理。

定理 (歐拉定理): 對於一個**連通**的平面圖，有

$$E = V + F - 2 \tag{1.2}$$

其中 E, V, F 分別代表圖的邊數、點數還有面數。一個**面**定義爲被邊所切出的一個區域(包含最外面的無限區域)。

歐拉公式給出了平面圖邊和點的關係式。而從公式中也可以看出平面圖邊的個數不會太多! 事實上對於一個簡單的平面圖，有以下定理:

定理： 對於一個**簡單連通**的平面圖，如果 $|V| \geq 3$ 則有

$$|E| \leq 3|V| - 6 \tag{1.3}$$

Proof. 由 (1.2)，$|E| = |V| + |F| - 2$。但簡單圖沒有自環及重邊，一個面一定至少有三條邊，$|F| \geq 3|E|$。代入 (1.2) 得 $|E| \leq 3|V| - 6$。 □

邊很少的這個條件往往是解題的關鍵!

例題 **1-2** Defense Your Country — NTUJ 2126

給你一個平面圖，請找一個最大團，也就是最大的一個完全子圖。($|V| \leq 2 \cdot 10^5$)

一看這個題目，不得了，連 $|E|$ 都沒有給，但不要忘了由 (1.3) 邊的數量不會超過 3 倍的 $|V|$。

而最大團是個 NPC 問題，如果我們不善用平面圖的性質，肯定解不出來。注意到**平面圖的子圖仍是平面圖**，而完全圖的邊的數量 $\mathcal{O}(E) = \mathcal{O}(V^2)$，平面圖的是 $\mathcal{O}(E) = \mathcal{O}(V)$。肯定**有個上界 m，K_m 絕對不會出現在平面圖上**。事實上算一下會發現對於 K_5，邊有 10 條，點有 5 個，而 $10 > 3 \cdot 5 - 6 = 9$，所以根據 (1.3)，K_5 肯定不是平面圖! 因此平面圖的最大團一定不超過 4。

不過這題還沒有結束，雖然我們只要檢查 4 個點以下的完全圖，但還是要有一個有效率的方法，否則 $\mathcal{O}(V^4)$ 枚舉肯定要超時。再注意到 (1.3) 其實告訴了

$$\sum \deg(v) = 2E \leq 6V - 12.$$

因此 $\deg(v)$ 的平均值小於 6 ，也就是一定有一個點的度數小於 5，所以我們可以先找一個 v_1 使得 $\deg(v_1) \leq 5$ ，然後 2^6 枚舉他和他的鄰居的最大團就可以了。枚舉完了這個點，將這個點從圖上移除，新的圖仍是一個平面圖，因此我們又可以再找一個 v_2 使得 $\deg(v_2) \leq 5$。一路做下去我們就得到了一個 $\mathcal{O}(V)$ 的做法!

這裡再總結一下平面圖的一些結論。

定理： 定義一個圖的 degeneracy 爲 $\min \deg(v)$，則平面圖的 degeneracy ≤ 5。

定理： 一個圖是平面圖的條件若且唯若 $K_5, K_{3,3}$ 不是他的 minor。G 是 H 的 minor 表示 G 可以由

- 刪掉一條邊
- 刪掉一個點
- 收縮一條邊，也就是把兩個點合併。

得到。

1.1.3 競賽圖

一個競賽圖是一個有向圖，且 $(u,v),(v,u)$ 恰好有一個在 E 中。可以想成是「有向完全圖」。競賽圖也有許多有趣的性質，如以下這個例題。

例題 1-3 競賽圖的 Hamilton Path 經典問題

給你一個競賽圖，求他的 Hamilton Path。

如果你把競賽圖想成是「兩兩對局結果」，也就是如果 $u \to v$ 就表示 u 贏 v，那一個 Hamilton path 其實就是一個「幾乎」[1] 正確的強度順序，因此我們不如往排序的方向想。

而衆多排序法，大家第一個想到的應該是最常用的快速排序法。我們可以模仿快速排序法的方法，先隨便挑一個點 v 出來，之後把所有他「贏過」的點 u 挑出來，也就是 $v \to u \in E$ 的點。假說這個集合是 A，那 $\bar{A} \setminus v$ 裡的點 w 由競賽圖的性質可以知道 $w \to v \in E$，因此我們只要遞迴下去做 $A, \bar{A}$，最後用 v 這個點把他們兩個的 Hamilton path 串起來就可以了!

1.2 有關圖的問題

與圖論有關的問題非常多，幾本上不太可能全部掌握，除了多了解一些結論之外，舉一反三和臨場見招拆招的能力也是很重要的!

1.2.1 計算複雜度理論

「知己知彼、百戰百勝」，要攻克一個題目，我們如果能先知道這個題目有「多難」會是一大幫助。

那要如何區分問題的難度呢？通常我們說一個問題是「容易的」表示我們已經知道有一個多項式時間的演算法可以求出這個問題的答案。比如排序問題，最短路徑問題等等。

但是有一些問題，人們想破頭了都想不出一個多項式時間的演算法可以解出，比如求一個圖的 Hamilton Path 至今都還沒有一個好的演算法解決他，而且也沒有辦法證明不存在一個多項式時間的演算法解決他。

[1] 會說「幾乎」的原因是因爲這個關係不一定有傳遞性，也就是 $(u \to v) \wedge (v \to w) \not\Rightarrow u \to w$。

百般無奈的人們只好先假設我們有一個更加強大的計算體系了，也就是**非確定性圖靈機 (Non-deterministic turing machine)**。

在介紹什麼是非確定性圖靈機之前，我們先定義一個合法的問題是什麼。在這邊我們先只考慮**判定性問題**。什麼是判定性問題呢？其實就是答案只有 Yes, No 兩種的問題，比如說「這個圖上是不是存在一個 Hamilton path 呢?」或是「這個圖是不是一個平面圖?」等等。至於「從 u 到 v 的最短路徑長度是多少?」就不是判定性問題了。但是我們大可以把他轉成「從 u 到 v 的最短路徑長度是否小於 k?」這個判定性問題，然後用比如說二分搜的方法解出原本的問題。

而非確定性圖靈機強大的地方就在於，有別於我們電腦等等的**確定性圖靈機**的計算體系，在進行搜索時有時後我們要面臨抉擇。比如說在找一個圖的 Hamilton path 時，我們不知道起點應該是 $v_1, v_2, \ldots$ 裡的哪一個，於是只好一個個試過一遍。非確定性圖靈機在面臨這種情況時，如果答案是「有解存在」的，他一定會**很幸運的選到一個最好的選擇**。

於是我們把所有用確定性圖靈機可以在多項式時間內解出的題目的集合稱作 P，把所有用確定性圖靈機可以在多項式時間內解出的題目的集合就稱作 NP。[2] 在經過一番推敲後我們發現**一個判定性問題是 NP 問題若且唯若如果答案是 Yes，他的一個解可以在多項式時間被一個內被一個確定性圖靈機驗證**。比如說 Hamilton Path 是 NP 問題，因爲今天如果有人宣稱 P 是一個 Hamiltom Path，那你只要確認 P 通過所有點恰好一次，而且每一條邊都存在於原本的圖中即可。

定義了 NP 後，人們就開始問，NP 裡最難的問題是什麼呢？我們說一個問題 A 比 B 難表示 B 可以化簡成 A，也就是說**如果我們要解 B 問題，我們可以把 B 在多項式時間內作一些轉化，把他變成 A 問題。** 比如我們可以證明找一個 Hamilton Cycle 比找 Hamilton Path 還難，因爲如果要找 Hamilton Path，我們可以再原圖多加一個點 v ，然後把 v 建邊連到所有點。這樣原圖的一個 Hamilton Path 一定可以頭尾用 v 串起來變成 Hamilton Cycle。而在新的圖如果找到了一個 Hamilton Cycle，只要把 Cycle 中的 v 去掉，就是原本 Hamilton Path 的解了。

也就是說，如果你會解 Hamilton Cycle，那你一定也能解 Hamilton Path。因此 NP 裡「最難」的問題就代表**如果你會解這個問題，那所有 NP 問題都可以在多項式時間化簡為這個問題。**不過「最難」的問題眞的存在嗎？有的！其實以經有許多問題被證明是「最難」的 NP 問題了，如 3-SAT、最大團還有剛剛說的 Hamilton Path 等都是！我們把這些問題叫作 NPC (NP-complete) 問題。

而如果我們有一天找到了一個可以在多項式時間內解出某一個 NPC 問題的演算法，那不得了了，因爲 NPC 已經是 NP 裡最難的問題了，所有 NP 問題都可以在多項式時間內解出！

很遺憾的是這些 NPC 問題我們到目前爲止都找不到一個多項式時間的演算法，也沒有辦法證明這樣的演算法不存在！

[2]所以 NP 並不是 Non-Polynomial 的縮寫，而是 Non-deterministic Polynomial 的縮寫。

這和競賽解題有何關係?

有的! 這往往可以幫助你往正確的方向思考。在解題的過程中我們往往會把題目做一些變換，轉成其他的題目來解。而如果你發現你轉成了一個 NPC 問題，除非測資範圍很小，否則通常表示你一定有什麼題目的性質沒有用到，要從新思考思路! 比如前面說的「平面圖最大團」問題，一般圖的最大團問題是 NPC，因此關鍵一定在平面圖的性質上!

首先我們定義什麼是字串。

定義： 給定一個字元集 Σ，一個字串是一些字元的有序序列，我們寫作

$$A = a_0 a_1 \cdots a_{n-1}$$

其中 $a_i \in \Sigma$。我們把 n 叫作字串的長度。

舉個例子，如果 $\Sigma = \{\texttt{"A"}, \texttt{"C"}, \texttt{"G"}, \texttt{"T"}\}$，$A$ 可能表示你的 DNA 序列，或者如果 $\Sigma = \{\texttt{"0"}, \texttt{"1"}, \cdots \texttt{"9"}\}$，那麼 A 可能代表客戶的電話號碼。這時候就會有許多有趣的問題了，比如:

- 假設另一段 DNA 字串 B 代表一個可能的致癌基因，如何快速的判斷一個人是否帶有這種基因?
- 現在有幾種不同的生物的 DNA 序列，如何判斷這些字串間的相似成度，以了解演化順序?

對於這些問題，數據的大小可以是非常大的，我們勢必會需要一些好的演算法來解決它們! 不過在這之前我們必需先說清楚一些定義。

定義： 給定一個字串 $A = a_0 a_1 \cdots a_{n-1}$

- 一個子字串是其連續的一段 $a_i a_{i+1} a_{i+2} \cdots a_j$ 記作 $A[i, j]$。
- 一個子序列是一個字串 $B = a_{q_1} a_{q_2} a_{q_3} \cdots a_{q_m}$，其中 $0 \le q_1 < q_2 < q_3 < \cdots < q_{m-1} < q_m < n$ 。
- 一個 A 的前綴是 A 的一個子字串 $a_0 a_1 a_2 \cdots a_h$，其中 $0 \le h < n$，記作 $P_A(h)$。
- 一個 A 的後綴是 A 的一個子字串 $a_k a_{k+1} a_{k+2} \cdots a_n$，其中 $0 \le k < n$，我們特別記作 $S_A(k)$。並讓所有後綴的集合稱作 $\sigma(A)$。

舉例來說，如果 $A =$ "abcbbab"，那"bcb" 是他的子字串，"acb" 是他的子序列，而"bbab" 是他的一個後綴。

1.3 字串的儲存

通常最基本的儲存方式就是用一個陣列依序將字串的每一個字元存下來。不過當我們要同時儲存許多字串時，可能就要花點巧思了。而這邊要介紹一個可以同時儲存多個字串的資料結