圖論進階

Yihda Yol

2016 年 11 月 7 日

1 歐拉路徑

相信大家都聽過所謂的「一筆畫問題」，轉換成圖論版本就是：找到一條行跡（不重複經過邊的路徑）使其經過一張圖上所有的邊。這就是所謂的「歐拉路徑」（Eulerian path），而如果要求起終點相同，就稱為「歐拉迴路」（Eulerian circuit）。

假設原圖連通，歐拉路徑存在的條件十分直觀：

1. 無向圖：如果恰有兩點的度數為奇數，則存在歐拉路徑，此二點分別為起終點；如果全部的點度數都是偶數，則存在歐拉迴路。

2. 有向圖：如果恰有一點的出度等於入度 +1、另有一點的入度等於出度 +1，其餘皆入度等於出度，則存在歐拉路徑，此二點分別為起終點；如果全部的點入度等於出度，則存在歐拉迴路。

至於如何找到歐拉迴路呢？可以證明，若一張圖有歐拉迴路，那麼用從任意一點 v 開始走未走過的邊，那麼當沒邊可走時，因為所有點度數都是偶數的關係，一定仍位在點 v。於是可以從任意一點出發開始 DFS（注意這裡的 DFS 和一般的 DFS 不一樣，是判邊有沒有被走過），而所有邊的離開順序就是一條歐拉迴路（有向圖要倒過來）。這是因為把很多個迴路串在一起仍然是一個迴路，而所有的邊一定可以被遍歷完。歐拉路徑的想法也相仿，實際上做法一模一樣。

1.1 漢米爾頓路徑

雖然問題的本質很像，但是找到一條 Hamiltonian path / cycle 是個 NP-complete 問題，目前沒有多項式時間解法。$O(n^2 2^n)$ 的 DP 解相信大家都會了，這裡也就不再重提。據說這個問題目前最好的解是 $O(1.657^n)$，而最小權漢米爾頓路徑（旅行推銷員問題）當前複雜度最好的解依然是 DP，甚至連是否存在底數比 2 還小的指數級演算法都尚未知。

1.2 樹上的歐拉路徑（Euler Tour）

在一棵樹上，如果把每一條樹邊想像成兩條邊一去一回，那麼在這棵樹上就有歐拉迴路，可以簡單地用普通 DFS 做完。如果把這個歐拉迴路的經過的東西依序列出來，可以獲得一些值得利用的性質。

LCA & RMQ

這是 Euler tour 最經典的一個應用。給定一個 N 點的有根樹，若將點以 DFS 順序編號，並把 Euler tour 經過的所有點編號依序列出成為一個序列 E，可以知道 E 的長度為 $2N-1$（因經過 $2N-2$ 條邊）。可以發現，如果點 v 在 E 中出現的位置是 P_v（任意一次都可以，不妨假設為第一次或最後一次），兩個點 a,b 的 LCA 就會是 E 中 $[P_a,P_b]$ 範圍內的最小值，於是 LCA 問題就變成了 RMQ 問題，可以用 sparse table 等資料結構解決掉。之前提過 RMQ 有 $\langle O(N),O(1)\rangle$ 解，因為 LCA 轉換成 RMQ 的時間是 $O(N)$，故這也是 LCA 問題的複雜度。

事實上，對於 RMQ 問題，如果建立一棵對原序列符合堆性質的笛卡爾樹，可以看出這棵樹上的 LCA 即對應原序列的 RMQ，而這個轉換的複雜度也是 $O(N)$，可以用維護 stack 元素遞增的方式建構。因此 RMQ 也可以用 LCA 的演算法處理（如 Tarjan's LCA algorithm）。

距離

相信大家都會用樹高度搭配 LCA 計算樹上兩點的距離，其實它也可以用 Euler tour 處理。序列改成維護 Euler tour 上每個經過的邊權，如果是往上走的話加一個負號。則一點 a 到它子孫 b 的距離就是序列中代表這兩點之間的所有數加總，搭配 LCA 即可算出樹上任兩點的距離。如果將序列以資料結構維護，則還可以支援改變邊權等操作。

樹鍊剖分

這東西出現在這裡有些突兀，但是勉強可以和 Euler tour 扯上一點關係，就只好放在這裡了。

之前的課有提到「重心剖分」這種將樹分解成很多份的方法。有另一種常見的有根樹分解方法稱為「樹鍊剖分」（heavy-light decomposition，又稱「重鍊剖分」）。方法是預處理以每個點為根的子樹大小，然後對每個點都連向最大的那棵子樹，如此一棵樹便被分成很多條鍊。

樹鍊剖分有個很重要的性質：任何一條樹上的簡單路徑最多經過 $2\log N$ 條鍊。原因是從上往下走，如果遭遇新鍊，代表該子樹大小不到前一點子樹大小的一半（否則會在同一條鍊上），於是最多經過 $\log N$ 條鍊，而所有路徑都可以拆成兩條上往下的鍊。因此，每一條鍊都可以視為序列，可以用各種資料結構維護，如此可以在樹上做很多神奇的事。

相較於重心剖分通常用來處理子樹，樹鍊剖分比較常用來處理路徑相關問題。

1.3 習題

1. (TIOJ 1084) 找到一個圖上經過點字典序最小的歐拉路徑。

2. (TIOJ 1692) 在一張 N 點 M 邊的無向圖上找到 P 個路徑，使得 P 最小且這些路徑合起來恰好經過每條邊一次。$N \le 1000, M \le 5 \times 10^4$。

3. (CF 528C) 對於一張 N 點 M 邊的無向圖，求一種加上最少數量的邊再將所有邊定向的方法，使得對於每個點，可以將它連接的邊兩兩分組，使得每一組兩個邊的方向相同。$N \le 10^5, M \le 2 \times 10^5$。

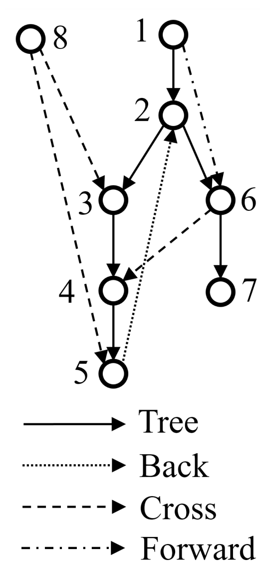
4. (No judge) 一棵樹，每條邊都有權重。要求支援兩種操作：改變一條簡單路徑上所有邊的權重、查詢兩點間的距離。複雜度 $\langle O(N), O(\log^2 N)\rangle$。

5. (No judge) 同上，改為只修改單個邊的權重。複雜度 $\langle O(N), O(\log N)\rangle$。

2 各種連通分量

2.1 DFS Tree

在進入主題前，先複習一下 DFS。上一次圖論課有提到，DFS 的過程可以看成一棵樹。給定一張簡單圖和一種 DFS 方法，根據圖上的邊在 DFS 過程中扮演的角色，可以將所有的邊分為兩類（無向圖）或四類（有向圖）：



1. 樹邊（Tree edge）：在 DFS tree 上的邊。

2. 回邊（Back edge）：從 DFS tree 的後代連向祖先的邊。

3. 交錯邊（Cross edge）：連接 DFS tree 上兩個非祖孫關係的點的邊。

4. 前向邊（Forward edge）：不在 DFS tree 上，但是從祖先連向後代的邊。

不難發現交錯邊和前向邊只會出現在有向圖中出現。

2.2 橋、邊雙連通分量

一張無向圖上，把某些邊移除會導致連通塊數量變多，這種邊稱為「橋」(bridge)。如果把所有的橋移除，那每一個連通塊在原圖上就稱為「邊雙連通分量」(2-edge-connected component）或「橋連通分量」(bridge-connected component，簡稱 BCC)。之所以稱為雙連通，是因為要讓一個邊雙連通圖不連通，至少需要移除兩個邊。

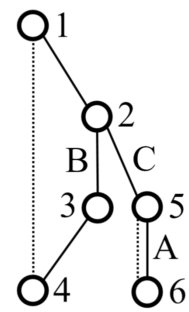
要在一張圖上面找到所有的橋，最直接的想法就是枚舉每一個邊，然後每次都 DFS 看看在沒有這個邊的情況下會不會導致連通塊數量變多，複雜度是 $O((V+E)E)$。

Tarjan's BCC Algorithm

上述的複雜度顯然不太能接受，因為在過程中做了太多重複的事情。但是仔細想想，每次移除一條邊後遍歷，實際上就是要求 DFS 不可以經過那條邊。如果在 DFS 過程中能考慮到每一條邊，就可以用一次的遍歷完成橋的判斷。

這時可以好好利用 DFS tree。在暴力做法中，是枚舉「要移除哪一條邊」，但可以發現在某次的 DFS tree 上，若移除的是回邊，那麼圖的連通性不會有任何改變，因此可能作為橋的只有樹邊。

如何判斷一個樹邊是不是橋呢？由於樹的每個邊都是橋，因此關鍵就在回邊上。對於一條樹邊 $e=(a,b)$，其中 a 是 b 的父親，如果不能從 b「不經由 e」走到 a，那麼將 e 移除即導致 a,b 不連通，也就是說 e 是個橋。而要不經由 e 走到祖先，就必須有一條回邊連接 b 或它的子孫和 a 或它的祖先。



圖中 A、B 不是橋，但 C 是，因為沒有邊從 5、6 連到 1、2。

於是我們可以在 DFS 過程中對每個點 v 維護「從 v 或它的子孫最多往上走一次回邊（也可以不走），最高可到達的點」，通常稱為 low 函數。low 函數的維護也很簡單，只要用每個它連向的邊（父親除外）更新它的 low 函數就好了。如此，對於每條樹邊 $e=(a,b)$（a 是 b 的父親），它是橋的條件就是 $\text{low}(b)=b$。如此複雜度就是一次 DFS 的複雜度。實作上 low 可以記錄 DFS 的進入順序或遞迴深度，更新 low 函數就取 min 即可。

邊雙連通分量有幾個性質。一個兩點以上的邊雙連通圖，任兩點之間都存在兩種不同的簡單路徑（不經過相同的邊）。另外，如果把每個邊雙連通分量變成一個點，畫成一張新圖，這張圖會是樹（或森林）。至於邊雙連通分量的求法也很簡單，一是 DFS 的時候順便把點推進 stack，發現橋 (a,b) 的時候就把 stack 裡面的東西拿出來，一直拿到 b 為止，也就是維護「當前已經遍歷過但還沒形成雙連通分量的點」；另一種方法是找出所有的橋後再重新 DFS 一次。以下附上程式碼。

Algorithm 1: Tarjan's BCC Algorithm in C++

```
vector<int> ed[N];
stack<int> st;
int id[N], low[N], bcc[N], t = 0;
void dfs(int x, int p) {
    id[x] = low[x] = ++t;
    st.push(x);
    for (int i : ed[x]) {
        if (!id[i]) dfs(i, x);
        if (i != p) low[x] = min(low[x], low[i]);
    }
```

```
    if (low[x] == id[x]) {
        int temp;
        do {
            bcc[temp = st.top()] = x;
            st.pop();
        } while (temp != x);
    }
}
```

2.3 割點、點雙連通分量

既然有邊的「橋」，當然也會有點的版本。若一張無向圖移除某點會使連通塊變多，該點就稱為「割點」（cut vertex）或「關節點」（articulation point）。點雙連通分量（2-vertex-connected component），或直接稱為「雙連通分量」（biconnected component，或稱 block，但是注意 BCC 指的通常是邊雙連通），和邊的定義方式也相同。

割點也可以用 Tarjan's BCC algorithm 求出，一條樹邊 $e = (a, b)$（a 是 b 的父親）若 $\text{low}(b) = a$ 或 b，就代表 a 是個割點。有個特例是 DFS tree 的根，如果有超過一個子樹，那麼它是割點。

需要注意的是割點會被包含在很多點雙連通分量當中，有點像把點邊扮演的角色反轉，所以點雙連通分量一定要跟著找割點一起判斷，和邊雙連通的做法差不多。stack 裡面可以放點或放邊，可以視需要改變實作方法。

點雙連通和邊雙連通有類似的性質。一個三點以上的點雙連通圖，任兩點之間都存在兩種不同的簡單路徑，且對於任三相異點 a, b, c，必定存在依序經過 a, b, c 的簡單路徑。如果將把每個點雙連通分量和割點都變成一個點，畫成一張新圖，則這張圖會是一個割點和點雙連通分量交錯出現的樹。

2.4 強連通分量

之前我們曾經定義過有向圖的連通，就是看它轉換成無向圖後連不連通。但我們發現這樣的連通性似乎有些薄弱，因為如此甚至可能沒辦法用 DFS 之類的一次遍歷完全圖。事實上，有一些更強的連通性，分別稱為「弱連通」、「強連通」。

有向圖中一群點，任取兩個相異點 a, b，若 a 到 b、b 到 a 的路徑至少其一存在，那麼這些點弱連通；若兩者都存在，稱這些點強連通。弱連通比較少討論，在此不特別著墨。

一張有向圖中可以分出很多個強連通分量（strongly connected component，簡稱 SCC）。可以發現，如果把每個強連通分量變成一個點，創造出一個新的圖，那這個圖會是一個 DAG。

求強連通分量的常見方法有兩種：

Tarjan's SCC algorithm

沒錯，剛才出現在 BCC 上的演算法，修改一下依然可以套用在強連通分量上！不難發現，如果一張有向圖的遍歷中只有樹邊和回邊，那麼求 SCC 的方法就和求邊雙連通元件一樣。但是有向圖特有的交錯邊和前向邊就必須要多考慮：不難發現前向邊不影響 SCC（因為可以用樹邊走過去）。唯一有問題的是交錯邊，因為如果交錯邊 f 連向的點，它的子孫有一條回邊連向 f 起終點的 LCA 或其祖先，即會導致 SCC 被改變。

但是仔細觀察「子孫有回邊連向 LCA 或其祖先」這件事發生，必然代表這條交錯邊連向的點尚未形成 SCC（因為形成 SCC 是在離開點時形成的），也就是說它還在 stack 裡面。也就是說，只要在用連向的點更新 low 前加上「這個點必須還沒形成 SCC」這個條件即可。可以發現這樣也會把樹邊的更新篩掉一點，但是並不會影響答案，因為篩掉的點 low 一定比當前點更低。

Korasaju's algorithm

Tarjan's algorithm 固然效率十分高，因為用一次的 DFS 便解決一切。但是它稍微有些難理解，而且 coding 複雜度也稍高。

相對而言，Korasaju's algorithm 是一個非常好理解的演算法。可以發現，如果在圖上用「正確的順序」DFS，那麼有可能用正常的 DFS 便能找出強連通分量。問題是「正確的順序」是什麼呢？有一個簡單的結論：將原圖 E 的所有邊反向得到新圖 E'，在新圖上 DFS 的離開順序倒過來就會是一個「正確的順序」。

為甚麼會這樣呢？我們可以試著找出一個「正確的順序」所要求的性質。同一個 SCC 裡面的點的拜訪順序並不重要（因為永遠都可以用 DFS 拜訪到）；但是對於任意一條連接兩個 SCC 的邊 $e = (a, b)$，b 必須要在 a 以前拜訪（否則會經由 e 從 a 走向 b，而把它們當成同一個 SCC）。剛才有提到將每個 SCC 變成點之後會形成 DAG，由於「將所有邊反向」並不會影響到 SCC 的位置，也就是說在邊反向的圖縮點後的 DAG 的拓樸順序，就會滿足剛才所提到的性質。

這個演算法總共需要兩次 DFS 和多建一張圖，常數上比 Tarjan's SCC algorithm 差，但是由於想法簡單，因此也不失為一個競賽中的好選擇。

2-Satisfiability (2-SAT)

所謂 2-SAT 問題，即是給定一個由 N 個布林變數（只能是 true 或 false）的式子 $E(X_1, X_2, \cdots, X_N)$，並且滿足這個式子是由 P 個部分取 and，每個部分都是 $(X_i \lor X_j)$、$(\neg X_i \lor X_j)$ 或 $(\neg X_i \lor \neg X_j)$ 的其中一種，求滿足 $E =$ true 的解，或判斷其無解。

可以發現它實際上就是要求 P 個部分全部都是 true。如果換個想法，$(X_i \lor X_j) =$ true 就代表「如果 $X_i =$ false 那麼 $X_j =$ true」（以及 i, j 反過來），有 not 的也可以類推。若將所有「若 A 則 B」這樣的關係建成一張有向圖，那原式存在解若且唯若 $X_i =$ false 和 $X_i =$ true 不能在同一個強連通分量中（否則將會導致矛盾）。如果有解的話，依照縮點後

圖的拓樸順序，逆向設定解即可。

2.5 習題

1. (TIOJ 1149) 裸 2-SAT。

2. (CF 487E) 在一張 N 點 M 邊的無向圖上，每個點都有權重。要求支援改變一個點的點權，或查詢從 a 到 b 的所有簡單路徑可以經過的點中（起終點也算），點權最小的點的權重。操作有 Q 次，$N, M, Q \leq 10^5$。

3. (TIOJ 1684) 有 $N \leq 1000$ 個武士和 $M \leq 10^6$ 個武士間的仇恨關係。如果能選出奇數個武士（不能只有一個）讓他們坐在圓桌周圍，使得任兩個座位相鄰的人都不互相憎恨，那麼他們就可以開會。求有多少個人永遠不可能開會。

4. (TIOJ 1683) 一張 N 點 M 邊的有向圖，求包含最多點的弱連通分量有幾個點。$N \leq 10^4, M \leq 10^5$。

5. (TIOJ 1484) 一張強連通圖如果每個邊都只屬於一個環，稱它符合「仙人掌」性質。給一張 N 點 M 邊的有向圖，求它是否符合仙人掌性質。$N, M \leq 10^5$。

6. (No judge) 有一個 $N \times M$ 的區域，而你有一種炸彈可以轟炸一整列或一整欄，但是同一列或欄只能轟炸一次。在這個區域中有 P 個特殊格子，可能要求你要必須炸兩次、至少炸一次、至多炸一次或不能炸。求一種滿足所有特殊格子條件的轟炸方法。複雜度 $O(N + M + P)$。

7. (CF 732F) 一張 N 點 M 邊的無向簡單連通圖，求一種定向的方法，使得從任意點開始走可以到達的點數最小值最大。$N, M \leq 4 \times 10^5$。

3 匹配

所謂匹配，就是在一張無向圖上找到相異的 $2n$ 個點 $A_1, B_1, A_2, B_2, \cdots, A_n, B_n$，使得對於所有 i，都存在連接 A_i, B_i 的邊 E_i，而這個匹配的權重就是 E_i 權重的總和。

3.1 二分圖最大匹配

二分圖最大匹配（maximum cardinality bipartite matching）是所有匹配問題裡面最簡單的一種，就是在一張所有邊的權重都是 1 的二分圖上找到一個最大權重的匹配。

在一個匹配當中，可以將所有的邊分為「匹配邊」和「未匹配邊」。而如果一條非環的簡單路徑經過的邊是匹配邊和未匹配邊交替出現，則稱為「交錯路徑」。如果交錯路徑兩端的邊都是未匹配邊、兩端的點都是未匹配點，稱為「擴充路徑」。之所以稱為擴充路

徑，原因是如果把它的未匹配邊、匹配邊顛倒，則此匹配的大小便加一。

而有個很重要的定理，稱為 Berge's lemma：一個匹配是最大匹配，若且唯若找不到任何的擴充路徑。此定理還可以延伸得到，如果找不到從一個未匹配點開始的擴充路徑，那麼一定存在最大匹配不包含這個點。

那麼要如何尋找擴充路徑呢？簡單的做法是從未匹配點開始，窮舉所有可能的交錯路徑，如果遇到了一個未匹配點，就表示找到了擴充路徑。但是我們可以好好利用二分圖的性質：如果把離開始點偶數條邊的點稱為偶點、奇數條的稱為奇點，則圖上所有的邊必連接一個奇點和一個偶點，也就是所有的路徑必定奇偶點相間。而如果兩條不同的擴充路徑包含重複的點，也只能選擇其中一條擴充。因此，我們只要簡單地進行 DFS，並忽略一切不是樹邊的邊即可。

最後是複雜度。由於一開始會有 $|V|$ 個未匹配點，而每一次 DFS 不是移除一個點（找不到擴充路徑）或是讓它變成匹配點（找到擴充路徑），因此最多進行 $|V|$ 次 DFS，複雜度 $O(|V||E|)$。另外，如果在進行此演算法前先在圖上隨便遍歷一次，每遇到一條邊兩端點都是未匹配點就將它們匹配起來，雖然只是個常數優化，但是實際上執行速度可以得到蠻大的改進。

順帶一提，二分圖的圖表示法可以進行優化。將圖分為 X 和 Y 兩群點，如果 X_i 和 Y_j 連邊，就記 $M_{i,j}$ 為 1（鄰接矩陣）或在 M_i 加入 j（鄰接串列），可以節省不少空間。以下附上程式碼。

Algorithm 2: Maximum Cardinality Bipartite Matching in C++

```cpp
vector<int> ed[Nx]; //優化過的儲存法
int mx[Nx], my[Ny]; //兩邊的點分別匹配到另一邊的哪個點，初始化為-1
bitset<Nx> vis;
bool dfs(int x) {
    vis[x] = true;
    for (int i : ed[x]) {
        if (!~my[i] || !vis[my[i]] && dfs(my[i])) {
            my[mx[x] = i] = x;
            return true;
        }
    }
    return false;
}
int matching() {
    fill(mx, mx + Nx, -1); fill(my, my + Ny, -1);
    int ans = 0;
    for (int i = 0; i < Nx; i++) {
        vis.reset();
        if (dfs(i)) ans++;
    }
    return ans;
}
```

其實一般圖上也可以用類似的方法解決最大匹配問題。但是一般圖和二分圖最大的不同，就是一般圖存在奇環。由於在奇環上繞一圈會導致奇偶點互換，因此實作上需要把奇環縮成一個點，稱為「縮花」。這個演算法很難寫，一般來講不會出現在競賽中。

另外，如果改為對所有未匹配點用 BFS 尋找最短的擴充路徑，則可以證明複雜度變為 $O(|E|\sqrt{|V|})$。這個演算法和流（flow）有緊密的關係，在此就先略過不提。至於帶權最大匹配的演算法是匈牙利演算法（Hungarian algorithm），基礎算法複雜度是 $O(V^2E)$，用類似 Dijkstra 的技巧可以到 $O(VE + V^2 \log V)$。這兩個東西理論上不會出現在 IOI 裡面（當然可以用它們拿部分分），在此略過不提。

3.2 Hall's Marriage Theorem

說到二分圖匹配就不得不提一下這個定理。考慮一個點分為 X, Y 兩部分的二分圖，對於任意 X 的子集 S，將 Y 中與 S 中任意點相鄰的點集記為 $M(S)$。這個定理是說，「對於任意 X 的子集 S 都有 $|S| \leq |M(S)|$」若且唯若「最大匹配是 $|X|$」。

3.3 獨立集與覆蓋

先來做名詞簡介。

1. 最大點獨立集（maximum independent set）：在圖中最大的一個點集，使得在此集合中任兩點都不相鄰，記為 I。

2. 最大邊獨立集（maximum independent edge set）：在圖中最大的一個邊集，使得在此集合中任兩邊都不相鄰。其實就是最大匹配，記為 M。

3. 最小點覆蓋（minimum vertex cover）：在圖中最小的一個點集，使得對於任意一條邊，它的兩端點至少有一個在此集合中，記為 C_v。

4. 最小邊覆蓋（minimum edge cover）：在圖中最小的一個邊集，使得對於任意一個點，都至少有一個相鄰的邊在此集合中，記為 C_e。

可以證明，一張圖 $G = (V, E)$ 中，$|I| + |C_v| = |M| + |C_e| = |V|$。由於對於任何點獨立集 I'，$V - I'$ 都是一個點覆蓋，因此 $|I| + |C_v| = |V|$。

至於對於一個最大匹配，可以發現這個最大匹配蓋住了 $2|M|$ 個點，因此最多只需要再 $|V| - 2|M|$ 條邊就可以覆蓋完所有點，故 $|C_e| \leq |V| - |M|$。然而一個最小邊覆蓋和所有點，會形成一個森林（沒有環，否則就不會是最小），如果把每個連通塊挑一個邊出來，會形成邊獨立集。因為每個連通塊都是一棵樹，滿足點數減邊數等於一，也就是連通塊個數等於 $|V| - |C_e|$，因此 $|M| \geq |V| - |C_e|$。綜合上述兩點，$|M| + |C_e| = |V|$。

在一般圖上，最大點獨立集和最小點覆蓋都是 NP-complete 問題，目前不存在多項式時間解法。然而，對於二分圖，$|M| = |C_v|$（稱為 König's theorem），也就是以上四個問題都存在多項式時間解法。這個證明與構造有些複雜，在此略過。

另外，如果原圖是樹，那麼以上四個問題都可以用 greedy 或樹 DP 解決，做法也不困難，可以自行思考。

3.4 習題

1. (TIOJ 1089)(TIOJ 1253) 一個 $N \times N$ 的方格上，散布著 K 隻怪物。現在你有一種武器，用一次可以打敗一整列或一整欄的怪物。求你需要用多少次這個武器才能把所有怪物打敗。$N \leq 1000, K \leq 2 \times 10^4$。

2. (TIOJ 1069) 某天有 $m \leq 1000$ 個事件會發生，第 i 個事件會在 t_i 時刻與座標上 (x_i, y_i) 的位置發生，每個事件都需要有人到場處理。每個人都可以在前一天先抵達任意位置待命，而一個人從一點到另外一點所耗費的時間是兩點的曼哈頓距離。求至少要在前一天派出幾個人才能處理所有事件。

3. (IOI 2015)(TIOJ 1886) 有 $N \leq 5 \times 10^5$ 個人和 Q 個任務，第 i 個任務需要將這些人分成 M_i 組，人數分別為 $P_1, P_2, \cdots, P_{M_i}$。然而每個人都要求他在的那一組人數必須要在 $[A_j, B_j]$ 之間，求每次的任務是否可以成功分組。$\sum M_i \leq 2 \times 10^5$。
（註：可能需要平方分割和二維線段樹或者 DP 優化之類的東西。）

4. (UVa 10243) 求樹上的最小點覆蓋。

5. (TIOJ 1528) 給一個 N 點 M 邊的無向簡單圖，求它的最大權點獨立集。$N \leq 5 \times 10^4, M \leq N + 7$。