Graph Algorighm - Elementary Graph Algorithm

實作

可以用 Adjacency List 或 Adjacency Matrix 表示。空間複雜度各是 $O(V+E)$ 與 $O(V^2)$。雖然 Adjacenct Matrix 很肥，但是有可以任意存取某個邊的好處，而且邊有權重時不用另外實作邊的資料結構。

Adjacency Matrix

概念上來說利用一個矩陣：

$$\texttt{w}_{\texttt{ij}} = \begin{cases} 1 & \texttt{if } (\texttt{u}_\texttt{i}\,, \texttt{u}_\texttt{j}) \in \texttt{E} \\ 0 & \texttt{otherwise} \end{cases}$$

實作上來說像是：

```
#define N_MAX 20
int W[N_MAX][N_MAX];
```

雖然感覺很肥，很多 Adjacency List 中 $O(V+E)$ 的演算法這邊會變成 $O(V)$，但是如果邊沒有權重的話，可以用 bit field 壓縮空間。而且如果圖很稠密，那麼 Adjacency List 跟 Adjacency Matrix 的複雜度相去不遠，但實作上相較之下較便利。

Adjacency List

概念上來說是一個 List 的 List：

$$\texttt{(u, v)} \in \texttt{E} \iff \texttt{v} \in \texttt{Adj[u]}$$

實作上可以用 vector 陣列，或是 vector<<vector>>：

```
/* include stuffs */

# define N_MAX 1000
vector <int> W[N_MAX];
vector <vector<int>> W_2;
```

存圖用 push_back()：

```
int main()
{
    int i, j;
    while(scanf("%d%d%d", &i, &j) != EOF) {
        W[i].push_back(j);
    }
    return 0;
}
```

若邊帶權重或其他資訊，可以手刻個資料結構，或是開一個陣列存：

```
struct edge{
    int i;
    int j;
    int w;
    edge(int _i, int _j, int _w):i(_i),j(_j),w(_w){}
};
vector <edge> W[N_MAX];
```

Trasversal

DFS

在 CLRS 提供的 pseudo code 長這樣:

```
DFS_VISIT(G, u)
time = time + 1
u.d = time //紀錄進入 u 的時間
u.color = GRAY
for all v in G.adj[u]
    if v.color == WHITE
        v.pi = u //紀錄 v 的父節點是 u
        DFS_VISIT(G, u)
u.color = BLACK
time = time + 1
u.f = time //u pop 出來的時間


DFS(G)
for each vertex u in G.V
    u.color = WHITE
    u.pi = NIL
time = 0
for all vertex u in G.V
    if u.color == WHITE
        DFS_VISIT(u)
```

實際上使用的實作可能像這樣:

```
#define N_MAX 1000

char vis[N_MAX];
vector<int> W[N_MAX];

/* DFS_VISIT() function */
void dfs(int n, vector<int> *G)
{
    vis[n] = 1;
    /* GRAY area for u after this line */
    for (auto &i : G[n]) {
        if (!vis[i]) {
            /* WHITE area for i*/
            dfs(i, G);
        }
    }
    /* BLACK area for u after this line */
}

int main()
{
    /* skip input */
    /* The DFS() */
    fill(vis, vis + NODE_NUM);
    for(int i = 0; i < N_MAX; i++) {
        if (!vis[i])
            dfs(i, &W);
    }

}
```

在 CLRS 中的 DFS 比較高級一點，會把進入時間跟結束時間一併紀錄。可以多新增`int d[N_MAX]`, `int pi[N_MAX]` , `int f[N_MAX]` 等等分別紀錄第 i 個點的結束時間，或是直接寫在 `vis[i]` 裡。

Thm (Parehthesis Theorem)

在對任一圖 $G=(V,E)$ 進行 DFS 時，任兩點 v_i, v_j 的 d 值與 f 值，僅有下面 3 種狀況：

1. `[vi.d, vi.f]` $\cap$ `[vj.d, vj.f]` $= \emptyset$：這時兩點不互為父節點或子節點。
2. `[vi.d, vi.f]` $\subset$ `[vj.d, vj.f]`：這時 v_i 是 v_j 的子節點。
3. `[vj.d, vj.f]` $\subset$ `[vi.d, vi.f]`：這時 v_j 是 v_i 的子節點。

觀察：要證上面的命題，只要證完下面這兩個：

$$v_i.d < v_j.d \begin{cases} v_i.f > v_j.d \Rightarrow v_i.f > v_j.f & \text{(claim 1)} \\ v_i.f < v_j.d & \text{(顯然兩區間沒交集)} \end{cases}$$

$$v_j.d < v_i.d \begin{cases} v_j.f > v_i.d \Rightarrow v_j.f > v_i.f & \text{(claim 2)} \\ v_j.f < v_i.d & \text{(顯然兩區間沒交集)} \end{cases}$$

就可以證明。而第二個狀況顯然只是把第一個狀況 i, j 對調。所以只要證 claim 1，claim 2 就自動對。

1. $v_i.d < v_j.d$，但 $v_i.f > v_j.d$，表示「v_j 是 v_i 為 `GRAY` 時發現的」，即 `DFS(G, vj)` 是在 `DFS(G, vj)` 內部被呼叫，因此 v_j 是 v_i 的子節點。
2. 而 $v_i.f$ 必定在 `DFS_VISIT(G, vj)` 回傳後才會更新，所以 $v_i.f < v_j.f$。因此得證 claim 1。

另外，在兩區間沒有交集時，表示任一點都不是另一點為 `GRAY` 時發現的，因此不可能互為父子節點。

Corollary (Nesting of descendants' intervals)

對一張圖 G 進行 DFS 時：

$$\texttt{DFS}\ \text{過程中}, v\ \text{是}\ u\ \text{的子節點} \iff u.d < v.d < v.f < u.f$$

由 Parenthesis Theorem 顯然成立。

Thm (White-Path Theorem)

對一張圖 $G = (V, E)$ 進行 DFS 時：

DFS 過程中，v 是 u 的子節點 $\iff$ 在發現 u 時，$\exists u \overset{p}{\leadsto} v. \forall v' \in p.$ `v'.color`

($\Rightarrow$)：使用歸納法

假定 $u = v$，命題顯然成立

假定 $u \neq v$，則由 Nesting of descendants' intervals 知：對於任意子節點 v'，均有 $u.d < v'.d$。所以：

1. 在 `time==u.d` 時，任意子節點 `v'` 都是 `WHITE` 。
2. 而 v 是 u 在進行 DFS 時的子節點，表示存在 DFS 構造出的，一條往 v 的路徑，而且這些路徑上的任一點都是 u 的子節點。
3. 由 2. 知這條路徑上所有的點，必定都是 `WHITE`

($\Leftarrow$)：

反證：假定 DFS 時，$u \overset{p}{\leadsto} v'$ 是一條全白路徑，但路徑中某存在一些不是 u 的子節點的點。令這些點中離 u 最近的為 v。

1. 假定 w 是白路上，v 的前一個點，或是 $w = u$。因為白路上 v 以前的點，都是 u 的子節點，故由「Nesting of descendants' intervals」：

$$u.d \leq w.d < w.f \leq u.f \Rightarrow w.d < u.f \qquad (1)$$

2. 另一方面，在 u 為 `GRAY` 時，v 仍為 `WHITE`，故 v 必定在 u 之後被發現。由「Nesting of descendants' intervals」，若 u 不為 v 的子節點，則：

$$u.d < u.f < v.d < v.f \Rightarrow u.d < v.f \qquad (2)$$

(等一下會用到它)

3. 將 w 由 `WHITE` 轉為 `GRAY` 時，討論 v 的顏色：

   如果 v 不為 `WHITE`，則 $v.f < w.d$。加上 (1) (2) 知道：

   $$u.d < v.f < w.d < u.f$$

   由 「Nesting of descendants' intervals」 知道 $v.d$ 唯一可能狀況是 $u.d < v.d < v.f \leq u.f$，但這表示 v 是 u 的子節點，因此矛盾。

4. 如果 v 仍為 `WHITE`， 又分兩種狀況：

   如果 v 在搜索 w 的白子節點過程中，被標為 `WHITE` 以外的顏色：那 v 就是 w 的子節點。矛盾。

   如果 v 在搜索 w 的白子節點過程中，沒有被標為 `WHITE` 以外的顏色：v 仍然為 `WHITE`，那 v 就會是 DFS 時下一個被標成 `GRAY` 的點，依然是 w 的子節點，也是 u 的子節點，因此也矛盾。

Def (DFS Forest)

DFS 搜索結束後，定義 G_π ：

$$\begin{cases} G_\pi = (V, E_\pi) \\ \\ E_\pi = \{(v.\pi, v) : v \in V, \text{and } v.\pi \neq \text{NIL}\} \end{cases}$$

則 G_π 是一個 Forest，稱作 DFS Forest。

因為除了那些起始點以外，所有點恰好有一個父節點，而且該節點不是會自己，因為如果要自己發現自己，自己要是 `GRAY` ，但又要是 `WHITE`，顯然不可能。因此對於瞞一個起始點長出的子樹，都有 $|E| = |V| - 1$。

Def (Classification of Edges)

1. Tree Edge :

$u, v \in V$。若 v 是在 DFS 時，探索 $e = (u, v) \in E$ 時被發現，則稱 e 為 tree edge。

2. Back Edge：

   $u, v \in V$。若在 DFS 時，u 是 v 的父節點，且 $e = (v, u) \in E$，則稱 e 為 back edge。

3. Forward Edge:

   $u, v \in V$。若在 DFS 時，u 是 v 的父節點，且 $e = (u, v) \in E$，但 $e \notin E_\pi$，則稱 e 為 forward edge。

4. Cross Edge:

   不是 Tree Edge, 不是 Back Edge, 不是 Forward Edge 的邊。比如說連接兩顆 DFS Tree 的邊。

Thm (Edge Classification)

在 DFS 第一次探索 (u, v) 時：

1. 若 **`v.color == WHITE`**，則 (u, v) 是 Tree Edge
2. 若 **`v.color == GRAY`**，則 (u, v) 是 Back Edge
3. 若 **`v.color == BLACK`**，則 (u, v) 是 Forward Edge 或 Cross Edge。

---

1. 根據 DFS，如果第一次探索 (u, v) 時，v 為 `WHITE`，則下一步就是把它變成 `GRAY`，而這時 u 也是 `GRAY`，所以 v 是 u 的子節點，因此 (u, v) 為 Tree Edge。

2. 若發現 v 為 `GRAY`，表示 v 在 u 之前被發現，而且仍未探索完所有子節點。因此 u 是 v 在探索子節點時發現的子節點之一，所以 (u, v) 是個 Back Edge

3. 有兩種可能：$u.d < v.d$ 或 $u.d > v.d$。

   1. 若為前者，則由 Nesting 知表示 v 是 u 的子節點，但因為第一次探索 (u, v) 時，v 已經非 `WHITE`，故 v 並不是透過 (u, v) 發現，

因此 $v.\pi \neq u$，故不為 Tree edge。
2. 若為後者，則由 Nesting 知 u, v 不互為父、子節點。因此不可能滿足前 3 種邊。

Thm (Edge of Undirected Graph)

假定 $G = (V, E)$ 是個無向圖，則在 DFS 進行結束後，只可能是 Tree Edge 或 Back Edge。

對於任意一個邊 $(u, v) \equiv (v, u)$，兩個點中一定有一個點比另外一個點先被發現。WLOG 假定 $u.d < v.d$。則 v 是在 u 正為灰色時，被由白圖灰的。

假定這第一次走這條邊時，是從 u 出發走到 v，這時 v 一定是白，不然這條邊在 v 所完之前，u 都不會動它。也就是只能由 v 從 (v, u) 方向探索，矛盾。所以 (u, v) 是 tree edge。

假定第一次走這條邊時，是從 v 出發走到 u，這時 u 已經為 `GRAY`，因此由定義知 $(v, u) = (u, v)$ 是 back edge。

BFS

CLRS 給的：

```
BFS(G, V)
    for all vertex u in G.V - {s}
        u.color = WHITE
        u.d = INF
        u.pi = NIL
    s.color = GRAY
    s.d = 0
    s.pi = NIL
    Q = Ø
    ENQUEUE(Q, s)
    while Q != Ø
        u = DEQUEUE(Q);
```

```
        for all v in G.adj[u]
            if v.color = WHITE
                v.color = GRAY
                v.d = u.d + 1
                v.pi = u
                ENQUEUE(Q, v)
        u.color = BLACK
RETURN
```

```
void bfs(int start)
    fill(vis, vis + NODE_NUM, 0);
    queue<int> q;
    q.push(start);
    vis[start] = 1; /* GRAY starting point */
    while(!q.empty()) {
        int cur = q.front();
        q.pop();
        for (&i in W[cur])
            if (!vis[i]) {
                vis[i] = 1;
                /* GRAY AREA */
                q.push(i);
            }
        /* BLACK AREA */
    }
}
```

類似地，在 CLRS 的程式碼中，多紀錄了進入與離開的 time stamp，以及遍歷時的父節點。在實作上可以跟 DFS 時多開幾個紀錄這些資訊的陣列代替。

另外，CLRS 用白灰黑 3 種狀態表示點在 BFS 中的狀態，但裡面有提到「... In fact, as Exercise 22.2-3 shows, we would get the same result even if we did not distinguish between gray and black vertices. 」，所以在程式的實作中就不區分灰黑。實際上如果需要也可以用 `vis[i]` 的值代替不同狀態。

實際上可以發現頂點的狀態要嘛從頭到尾都是 `WHITE` (也就是不 reachable)，要不然就是依照 `WHITE-GRAY-BLACK` 的順序上色，最後被 `POP` 出來，並不會由黑轉灰，或是由黑轉白，所以實作上不用特地開 `enum` 標註顏色狀態，只要讓知道哪個點有走過就好。

Shortest Path

這邊討論的最短路徑是「通過最少數目的邊，到達另外一點」，是以「邊的數目」決定路徑長度，尚沒有權重的概念。

Def (Shortest-Path Distance)

$G=(V,E)$ 是個圖，Shortest-Path Distance 定義為：

$\delta(u,v) : V \times V \rightarrow \mathbb{N} \cup \{0\}$,

$$\delta(u,v) = \begin{cases} \infty & \text{if u, v not reacha} \\ \min\{|p|-1 : p \text{ is a path from u to v }\} & \text{otherwise} \end{cases}$$

Def (Shortest Path)

$$p \text{ is a shortest past from u to v } \iff \begin{cases} u \overset{p}{\rightsquigarrow} v, \; and \\ |p|-1 = \delta(\text{u}, \text{v}) \end{cases}$$

Lemma (最短路徑性質)

$G=(V,E)$ 是個圖（有向或無向），則：

$$\forall s \in V. \forall (u, v) \in E. \delta(s, v) \leq \delta(s, u) + 1$$

---

1. 假定 s u not reachable，則 $\delta(s, u) = \infty$，命題成立。
2. 假定 s u reachable，則存在一條長度 $\delta(s, u)$ 的最短路徑 $p' = \langle s \ldots u \rangle$，則路徑 $p = p' \cup \{v\}$ 是一條 u, v 間的路徑（未必是最短），長度為 $\delta(s, u) + 1$。因為任何路徑長度都 $\leq$ 最短路徑長度，故：

$$\delta(s, v) \leq \delta(s, u) + 1$$

「=」：如果 u 是 s, v 最短路徑上的前一個點?

Observation (每個點至多被 ENQUEUE 一次)

由程式知：只有 `WHITE` 點會被 `ENQUEUE`，且 `ENQUEUE` 的前一刻會立刻被標記為 `GRAY`。如果有一個點被 `ENQUEUE` 兩次，表示存在「將點標成 `WHITE`」 的步驟，使它再度變為 `WHITE`。但程式中沒有任何動作會將顏色由 `WHITE` 以外的顏色變成 `WHITE`。

Lemma (d 值不短於最短路徑)

當 BFS 在 $G = (V, E)$ 從 $s \in V$ 開始並執行完畢之後：

$$\forall v \in V.\ v.d \geq \delta(s, v)$$

對頂點數目做歸納：

1. 第一次 `ENQUEUE` 時，起始點 s 被塞進 `Q` 裡，並且進行 $s.d = 0 = \delta(s, s)$。而因為此時 s 已經被標為 `GRAY`，故進入迴圈之後，d 值不可能再被更新。而這時：

$$\forall v \in V \setminus \{s\}\ .\ v.d = \infty \geq \delta(s, v)$$

2. 假定在執行過程時，對於任意在 u 被 `DEQUEUE` 至 u 被標為 `BLACK` 前被發現的 `WHITE` 點 v ，並且由歸納法假設：

$$u.d \geq \delta(s, u)$$

而下一步將會將 $v.d$ 指定為 $u.d+1$，但：

$$\begin{aligned} v.d &= u.d + 1 \\ &\geq \delta(s,u) + 1 && (\text{上式}) \\ &\geq \delta(s,v) && (\text{上一個 Lemma}) \end{aligned}$$

由歸納法知原題成立。

> *白話文： d 值有限表示兩者有能透過 BFS 到達該點，路徑長為 d 的「發現路徑」。又因為「任意路徑比最短路徑長」，所以這個性質聽起來很合理。 而如果兩點不 reachable，d 值無限，此性質顯然成立。*

Lemma (Q 中 d 遞增，但只有兩種可能值)

在對 $G=(V,E)$ 進行 BFS 的過程中的某個時候，假定 Q 當中的元素依序為：

$$Q = \langle v_1, v_2, \dots v_f \rangle$$

其中 v_1 是 Queue 最前面的元素、v_f 是最後一個元素，則：

$$\begin{cases} v_f.d \leq v_1 + 1, \ and \\ v_i.d \leq v_{i+1}.d & \forall i = 1, 2, .., f-1 \end{cases}$$

證明的思路：所有東西的元素中， d 的上限都是 $v_1.d+1$，但又知道 $v_2.d \geq v_1.d$ ，可以很容易用歸納法證明。

對每一次 `ENQUEUE` 使用歸納法:

- 第 1 次 `ENQUEUE` 時，只有起始點 s 一個元素，顯然成立。
- 在進行某次 `ENQUEUE` 前，假定此時 `Q` 中的元素為:

$$Q = \langle v_1 \dots v_f \rangle$$

且滿足原性質:

$$\begin{cases} v_f.d \leq v_1.d + 1 & (1) \\ \\ \forall i \in \{1, ., f-1\}. v_i.d \leq v_{i+1}.d \Rightarrow v_1.d \leq v_2.d & (2) \end{cases}$$

並且接下來要把 v_1 從 `Q` `DEQUEUE`。令 v_{f+1} 是一個在 v_0 被標記為 `BLACK` 前新發現的節點（如果這樣的節點不存在，性質 (2) 顯然成立; 又因為 $v_2.d + 1 \geq v_1.d + 1$，故性質 (1) 也會成立）由 BFS 步驟知:

$$v_{f+1} = v_1.d + 1$$

由 (2) 可以得到:

$$v_1.d \leq v_2.d \Rightarrow v_{f+1}.d = v_1.d + 1 \leq v_2.d + 1$$

故:

$$v_{f+1}.d \leq v_2.d + 1$$

因此性質 2. 成立。

又因為:

$$v_f.d \leq v_1.d + 1 = v_{f+1}.d$$

加上由歸納法假設已知:

$$v_1.d \leq v_2.d \leq \dots \leq v_f.d$$

因此:

$$v_2.d \leq ... \leq v_f.d \leq v_{f+1}.d$$

故知對於 `DEQUEUE` 之後第一個新找到的節點，在 `ENQUEUE` 之後仍然能使 `Q` 中的元素保持原命題。

而其他在 v_1 被標為 `BLACK` 之前發現的點，d 之大小都跟 $v_{f+1}.d$ 相同，顯然加入 `Q` 之後，`Q` 也可以保持原性質。由此得證。

Corollary(d 值越小，越先 ENQUEUE)

假定 BFS 過程中，v_i 比 v_j 先被 `ENQUEUE`，則:

$$v_i \text{ 比 } v_j \text{ 先 } \texttt{ENQUEUE} \iff v_i.d \leq v_j.d$$

因為每一個點只會被 `ENQUEUE` 一次，`ENQUEUE` 之後 d 立刻被指定值與上色，之後就再也不可能被重複發現。所以一個點只會被指定一次 d 值。

因為 Queue 的性質，先 `ENQUEUE` 者會先 `DEQUEUE` ，但每次 `DEQUEUE` 時，該元素之 d 值將 $\leq$ 所有 Queue 中的元素，包含前一個元素。由此遞推下去，即可證明該命題。

Thm (Correctness of BFS)

$G = (V, E)$ 是一張圖，假定從某一點 s 開始進行 BFS，則:

1. (所有 reachable 的點都可以被發現，且 d 值就是最短路徑長)

$$\begin{aligned} &\forall v \in V, s \leadsto v. \\ &\qquad v.d = \delta(s, v) \qquad\qquad (a) \\ &\qquad v.color = \texttt{BLACK} \qquad (b) \end{aligned}$$

2. 存在由到父節點的最短路徑加一個邊形成的最短路徑:

$$
\begin{aligned}
&\forall v \in V, s \rightsquigarrow v,\ v \neq s.\\
&\quad \exists p' \in \{\text{shortest paths from } s \text{ to } v.\pi\}.\\
&\qquad\qquad p' +\!\!+ v \in \{\text{shortest paths from } s \text{ to } v.\} \qquad (c)
\end{aligned}
$$

對 BFS 時，所有發現 `WHITE` 點的時候進行歸納。

BFS 發現第一個白點時，也就是起始點 s 時，$\delta(s, s) = 0 = s.d$，上述性質顯然成立。

假定 BFS 進行搜索，在 u 被 `DEQUEUE` 時，發現一個 `WHITE` 點 v，且：

1. 所有 `DEQUEUE` 過的的點 v'，都滿足 $\delta(s, v') = v'.d$。
2. 由歸納法假設：

$$u.d = \delta(s, u)$$

3. 利用反證法：假定 $v.d \neq \delta(s, v)$

接著觀察：

1. 由「Lemma(d 值不短於最短路徑)」知 $v.d \geq \delta(s, v)$ 要成立，又反證法假定兩者不等，因此：

$$
\begin{aligned}
v.d &= u.d + 1 && (\text{BFS 規定的})\\
&= \delta(s, u) + 1 && (\text{歸納法假設}: \delta(s, u) = u.d)\\
&> \delta(s, v) && (\text{Lemma}, v.d \neq \delta(s, v))
\end{aligned}
$$

因此：

$$u.d \geq \delta(s, v)$$

2. 假定 $s \overset{p_m}{\rightsquigarrow} v$ 是 s 往 v 的「最短路徑」，並令 v' 是最短路徑中，v 的前一個點。
3. 由於 $p'_m := p_m \setminus \{v\}$ 必須是一條$s \overset{p'_m}{\rightsquigarrow} v'$ 的最短路徑，否則就可以構造出「更短的最短路徑」。由歸納法假設：

$$\delta(s, v') = v'.d$$

比較：

$$
\begin{aligned}
\delta(s,v) &= \delta(s,v')+1 && (p_m, p'_m\text{都是最短路徑}) \\
&= v'.d+1 && (\text{歸納法假設：}v'.d=\delta(s,v')) \\
&\le u.d && (\text{剛剛推論：}u.d \ge \delta(s,v)) \\
&\Rightarrow v'.d \le u.d-1 \\
&\Rightarrow v'.d < u.d
\end{aligned}
$$

但由 $v'.d < u.d$，加上「Lemma(d 值越小，越先 ENQUEUE)」可發現：「 v' 在 u 之前被發現」。又因為 v' 跟 v 相鄰，所以：

1. 如果這時 v 是 `WHITE`，那麼 d 值應該要是 $v'.d+1$，但 $v'.d+1 < u.d+1$ ，因此與「u 是第一個發現 v 的點」矛盾。
2. 如果這時 v 不是 `WHITE`，那不管是 `GRAY` 或是 `BLACK`，都表示 v 點比 v' 早被 `ENQUEUE` ，所以 $v.d \le v'.d < u.d$ ，仍然推到 $v.d \neq u.d+1$，因此也矛盾。

故不可能 $v.d \neq \delta(s,v)$。得證性質 (a)。

由性質 (a) 可證性質 (b) ： 假定 BFS 結束後，有些 reachable 的點沒有被搜尋到，因為 d 值不會被更動，故 $d=\infty > \delta(s,v)$，故矛盾。

性質 (c) 可以觀察到 $v.\pi = u \Rightarrow u.d+1 = v.d$。因此沿著父節點一直往上找，找到 $d=0$ 為止，就可以找到滿足該性質的最短路徑。

參考資料

1. CLRS
2. BFS 的證明

Online Judge

> *OJ 習題*
>
> *UVA : 336,352,383,532,567,571,601,705, 762,10004,10009,10474, 10505,10592,10603, 10946, 11624*
>
> *POJ : 1129,1154,1416,1606,1753,1915,1979,2243*