AlgoSeacow

Graph II

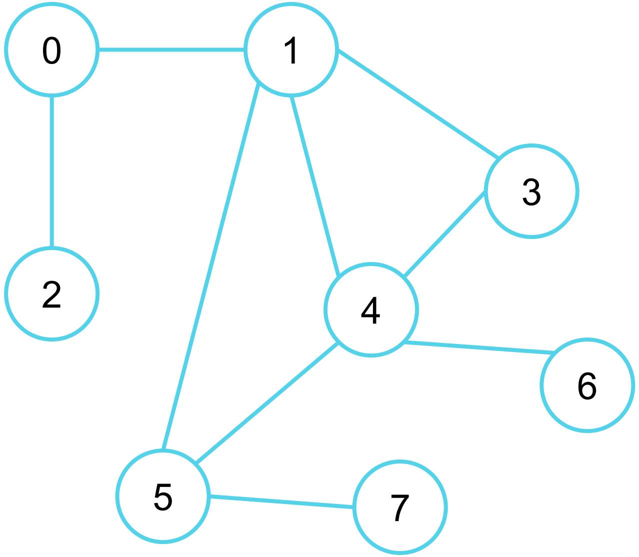
關於這堂課

- 先備知識
  - Graph I
  - recursion
- 最短路徑
  - Dijsktra
  - Bellman-Ford
  - Floyd-Warshall
- Minimum Spanning Tree
  - Kruskal
  - Prim

最短路徑問題

不帶權 Shortest Path

- 邊權都為 1
  - 做一次 bfs, 每個點的 level 值就是最短路



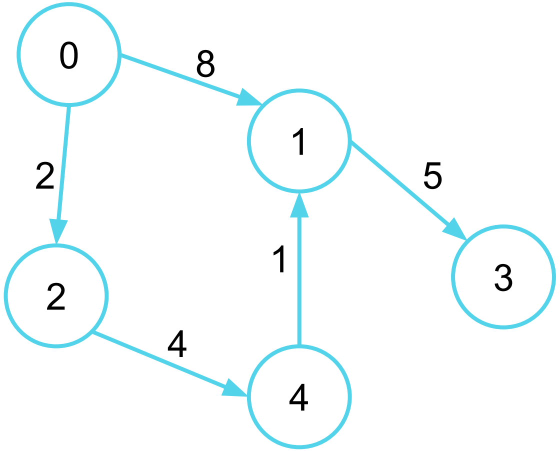
	0	1	2	3	4	5	6	7
dis	0	1	1	2	2	2	3	3

不帶權 Shortest Path - Code

```cpp
int dis[MAXN];

void bfs(int s) {
    queue<int> q;
    for (int i = 0; i < n; i++) dis[i] = INF;
    dis[s] = 0;
    q.push(s);
    while (q.size()) {
        int u = q.front();
        q.pop();
        for (int i = 0; i < G[u].size(); i++) {
            int v = G[u][i];
            if (dis[v] == INF) {
                dis[v] = dis[u] + 1;
                q.push(v);
            }
        }
    }
}
```

帶權 Shortest Path

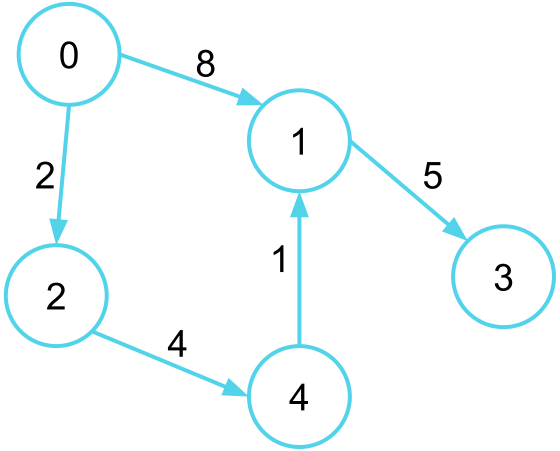
- 邊有權重，權重總和最小化



Dijkstra's 最短路徑演算法

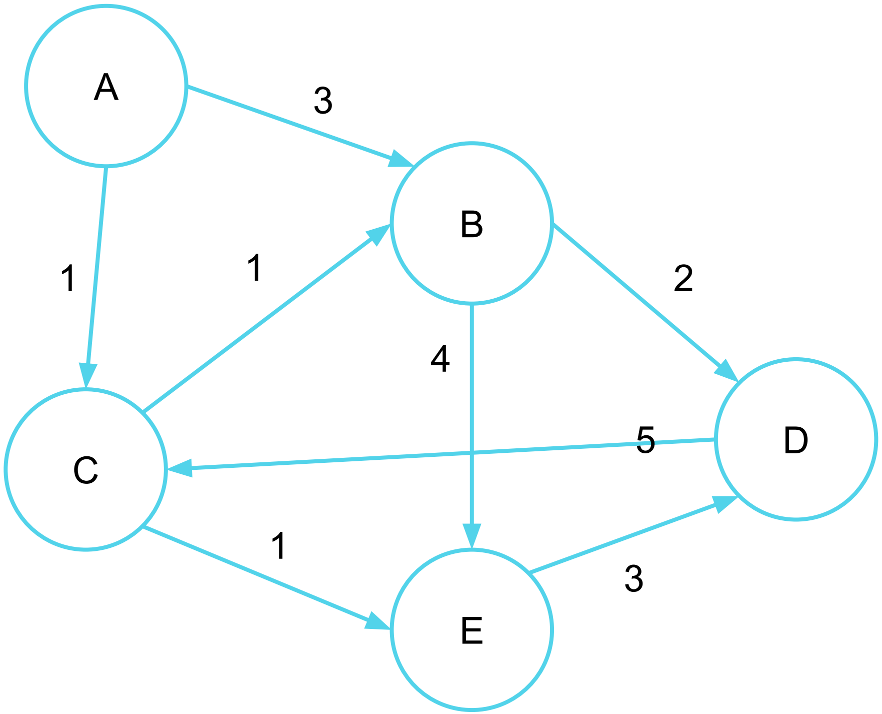
沒有負邊 Shortest Path - Dijkstra

- 給定一張正邊權圖和起點, 求出從起點開始到每個點的最短距離
  - 單源點最短路 (Single Source Shortest Path)
- 作法
  - 從未 visited 的點中挑選最小的值, 將其變為 visited 並更新其鄰居的值
  - 若某點變成 visited, 該點的距離即為最短距離



0	1	2	3	4
0	7	2	12	6

沒有負邊 Shortest Path - Dijkstra 過程



沒有負邊 Shortest Path - Dijkstra $O(n^2)$

```
void Dijkstra(int source) {
    fill(dis, dis + n, INF);
    fill(vis, vis + n, false);
    dis[source] = 0;
    for (int i = 0; i < n; i++) {
        int u = find_target();
        vis[u] = true;
        for (int j = 0; j < (int)G[u].size(); j++) {
            long long w = G[u][j].first;
            int v = G[u][j].second;
            if (vis[v]) continue;
            dis[v] = min(dis[v], dis[u] + w);
        }
    }
}
```

```
int find_target() {
    long long minV = INF;
    int idx = -1;
    for (int i = 0; i < n; i++) {
        if (vis[i]) continue;
        if (dis[i] < minV) {
            minV = dis[i];
            idx = i;
        }
    }
    return idx;
}
```

Dijkstra - 時間複雜度

- 做 n 次 find_target
  - 每次 find_target 時間複雜度 O(n)
  - 總時間複雜度 O(n^2)

- 每條邊最多被走過一次
  - 總時間複雜度 O(m)

- Dijkstra 總時間複雜度 O(n^2 + m)

Dijkstra 優化

- find_target 為從某群點集中找到最小值
  - 適合的資料結構: heap (std::priority_queue)
  - 將 dis 放入 heap 管理, 每次可以快速的求出 target

- 問題
  - 更新鄰居會修改到 dis 值, 放入 priority_queue 管理的值沒辦法輕易被修改

Dijkstra 優化 - Code

```
void Dijkstra(int source) {
    fill(dis, dis + n, INF);

    priority_queue<pii, vector<pii>, greater<pii>> pq;
    pq.push({0, source});

    while (pq.size()) {
        long long d = pq.top().first;
        int u = pq.top().second;
        pq.pop();

        if (dis[u] != INF) continue;
        dis[u] = d;

        for (int i = 0; i < (int)G[u].size(); i++) {
            long long w = G[u][i].first;
            int v = G[u][i].second;
            pq.push({d + w, v});  // O(log n)
        }
```

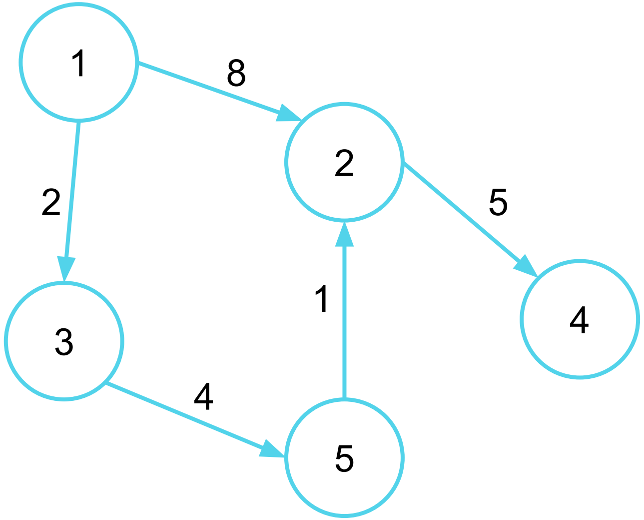
Dijkstra 優化 - 時間複雜度

- 對於每條邊都會將丟一個狀態進去 pq
    - pq 的 push 時間複雜度 O(log n)

- Dijkstra 總時間複雜度 O(n + m log n)

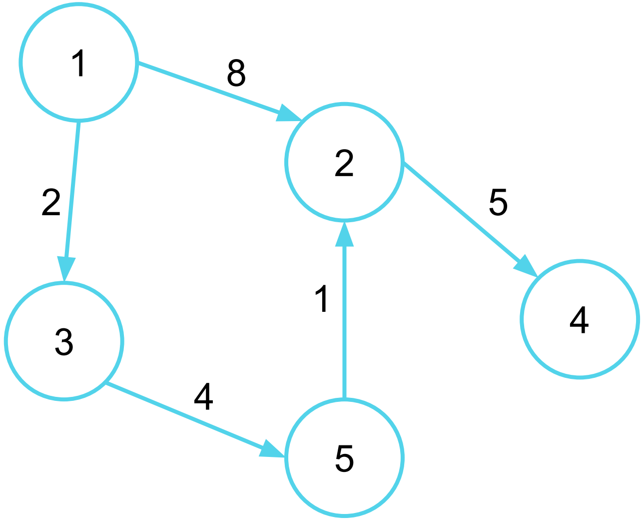
CSES - Shortest Routes I

- 題目敘述
  - 輸入一個帶權有向圖，有 n 個節點和 m 條邊，起點位於編號 1
  - 計算從起點到所有點的最短距離
- 測資範圍
  - $2 \le n \le 100000$
  - $1 \le m \le 200000$
  - $1 \le a, b \le n$
  - $1 \le c \le 10^9$



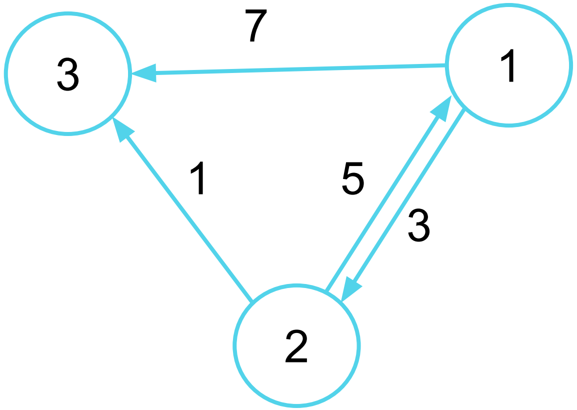
CSES - Flight Discount

- 題目敘述
  - 輸入一個帶權有向圖，有 n 個節點和 m 條邊
  - 可以將一條邊的邊權減半，問從1 到 n 的最短路徑長度
- 測資範圍
  - $2 \le n \le 100000$
  - $1 \le m \le 200000$
  - $1 \le a, b \le n$
  - $1 \le c \le 10^9$



變化題：可以使用兩張折價券的最段路徑

- 與前一題相同，但是可以將兩條邊邊權減半



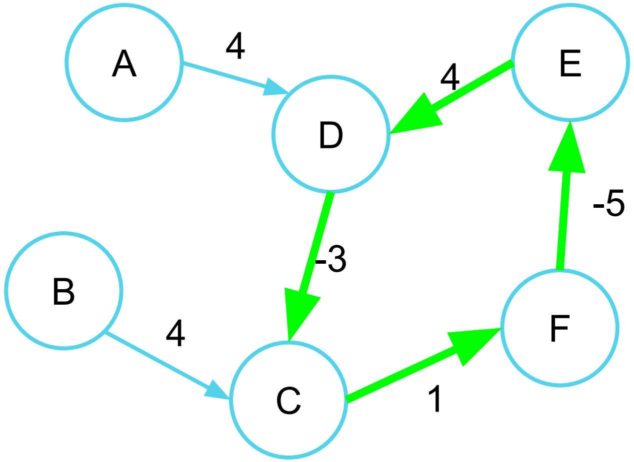
題單 - Dijkstra

- O(m log n)
  - kattis - shortestpath1 : O(m log n)
  - uva 10986 : O(m log n)
  - codeforces 20C. Dijkstra? : O(m log n) Dijkstra && 輸出路徑
  - zerojudge d793 : O(m log n)
  - CSES - Shortest Routes I (模板題)
- O(n^2)
  - kattis - crosscountry : O(n^2)
- 應用題
  - CSES - Flight Discount
  - CSES - Flight Routes

Bellman-Ford's 最短路徑演算法

帶負邊圖 - 負環

- 環上的路徑總和為負數
  - 不存在最短路,
    可以一直繞圈圈會更短



Shortest Path - Bellman Ford

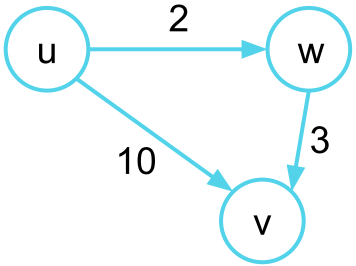
- 給定一張無負環圖和起點，求出從起點開始到每個點的最短距離
    - 單源點最短路 (Single Source Shortest Path)

- 作法
    - 做 n - 1 次回合更新
    - 每次回合更新
        - 對於每條邊都做一次relax

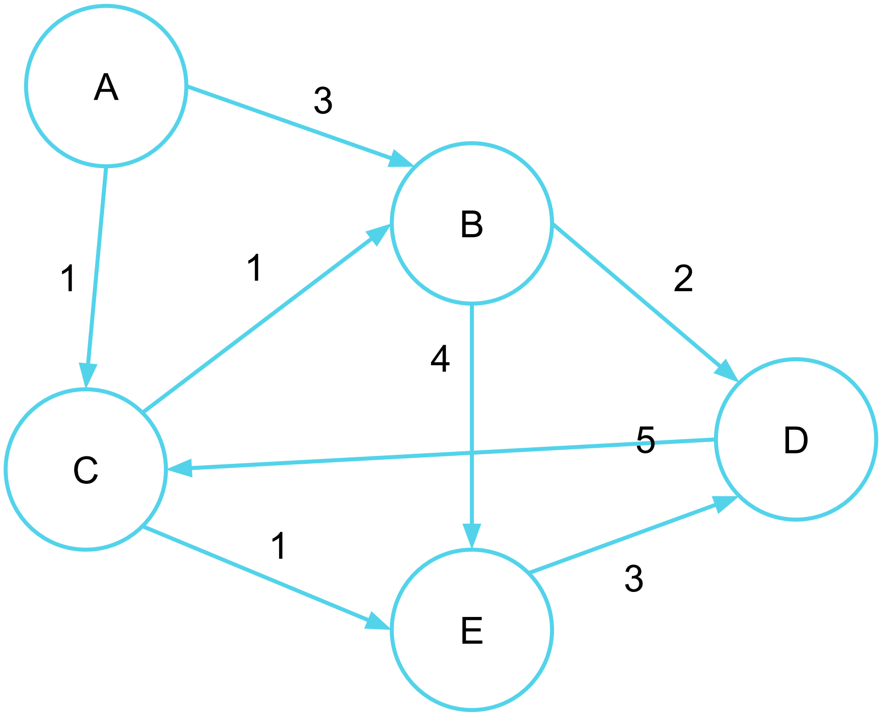
Relax

- 圖上的三個點 u, v, w, 其中
  dis(u, v) 代表點 u 和 點 v 的距離, 則

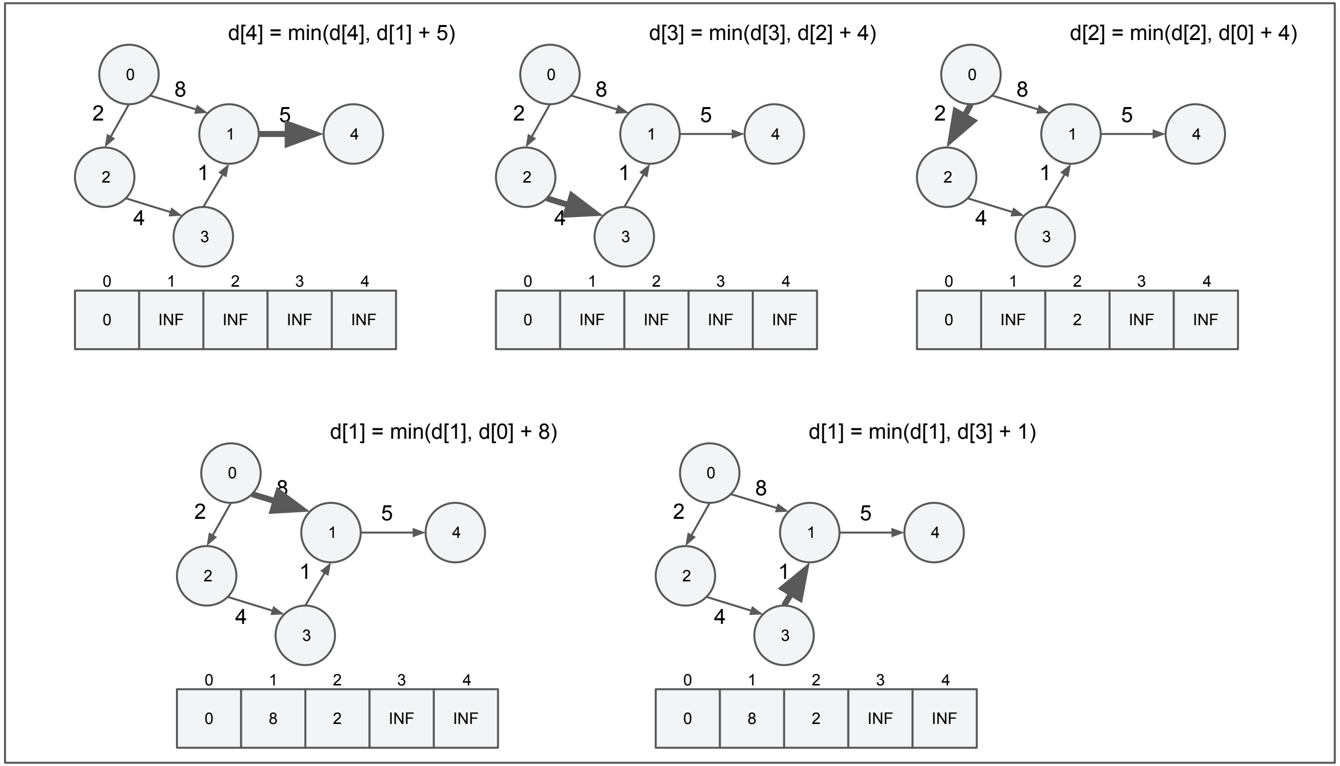
```
dis(u, v) = min(
  dis(u, v),                // 單純從 u 走到 v
  dis(u, w) + dis(w, v)   // 從 u 走到 w 後再走回 v
)
```



Shortest Path - Bellman Ford 過程



Shortest Path - Bellman Ford 例子



x 4次 (n-1)

Bellman Ford - Code (使用 edge list)

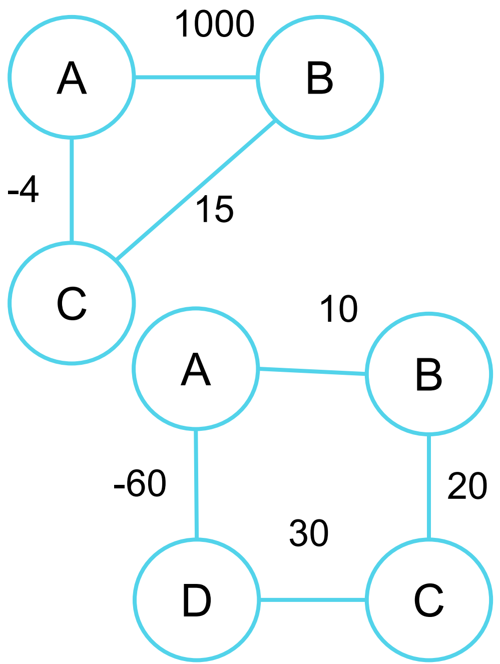
```
struct Edge {
    int u, v;
    long long w;
};

int n, m;
vector<Edge> edges;
long long d[MAXN];

void BellmanFord(int s) {
    fill(d, d + n, INF);
    d[s] = 0;
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < m; j++) {
            Edge e = edges[j];
            d[e.v] = min(d[e.v], d[e.u] + e.w);
        }
    }
}
```

Bellman Ford - 時間複雜度

- 做 n - 1 回合
    - 每回合跑過所有的邊, 時間複雜度 O(m)
    - 總時間複雜度 O(n * m)

UVA 558 Wormholes

- 給一張有向帶權圖, 問有沒有負環
- $1 \le n \le 1000$, $0 \le m \le 2000$
- $-1000 \le t \le 1000$



題單 - Bellman-Ford

- 判斷負環
  - UVa 558 Wormholes
  - POJ 3259 蟲洞
- 有負邊求最短路徑
  - CSES - High Score
  - CSES - Cycle Finding

Floyd-Warshall’s 最短路徑演算法

Floyd Warshall

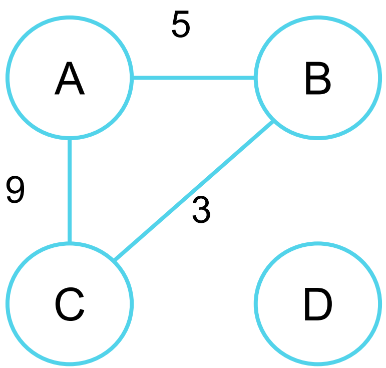
- 給定一張無負環圖和起點, 求任兩點之間的最短距離
  - 全點對最短路徑 (All Pair Shortest Path)

- 作法
  - 三層迴圈

```
void FloydWarshall() {
    for (int k = 0; k < n; k++)
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                d[i][j] = min(d[i][j], d[i][k] + d[k][j]);
}
```

CSES - Shortest Routes II

- 題目敘述
  - 給一個無向圖，詢問 q 次某兩點之間的最短路徑
- 測資範圍
  - $2 \le n \le 500, 1 \le m \le n^2$
  - $1 \le a, b \le n, 1 \le c \le 10^9$
  - $1 \le q \le 10^5$



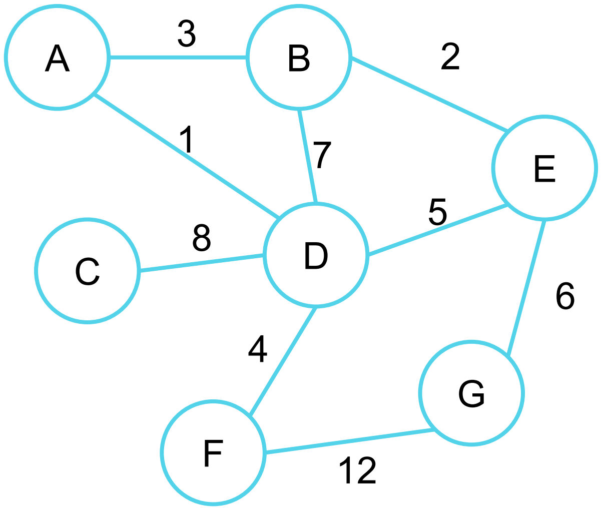
More Practices - Floyd Warshall

- CSES - Shortest Routes II (模板題)
- zerojudge a674 (最小化最大路徑)
- zerojudge d908 (最大邊權合)
- zerojudge d282 (帶權樹重心)
- zerojudge c128 (最大化最小路徑)
- zerojudge a874 (最短路徑)
- uva 247 (強連通分量)
- codeforces 25C. Roads in Berland (任兩點的最短路合)
- leetcode 399. Evaluate Division (DFS)
- Atcoder abc208D (演算法本質意義)
- TIOJ 1212 (有向圖最小環)

最小生成樹
(Minimum Spanning Tree)

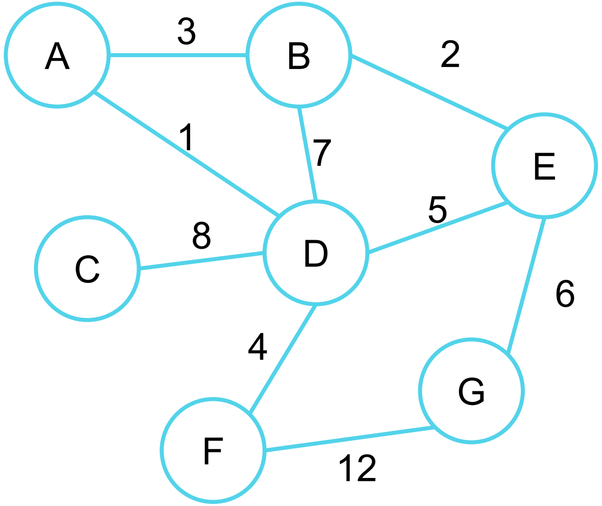
Minimum Spanning Tree

- 定義
  - 輸入一個帶權簡單無向圖
  - 選一些邊讓圖依然是連通的，選出的邊總和愈小愈好



Kruskal's 演算法

- 將 edge 按照邊權由小到大排序
- 依序嘗試加入樹邊
  - 若加入不會造成環
    - 此邊為樹邊
  - 若加入會造成環
    - 此邊不為樹邊
  - 可用 Disjoint Set 維護嘗試加邊後是否有環



Kruskal - Code (使用 edge list)

```
bool cmp(Edge a, Edge b) {
    return a.w < b.w;
}

long long Kruskal() {
    sort(edges.begin(), edges.end(), cmp);
    long long ans = 0;
    for (int i = 0; i < m; i++) {
        Edge e = edges[i];
        if (find(e.u) != find(e.v)) {
            merge(e.u, e.v);
            ans += e.w;
        }
    }
    return ans;
}
```

Kruskal - 時間複雜度

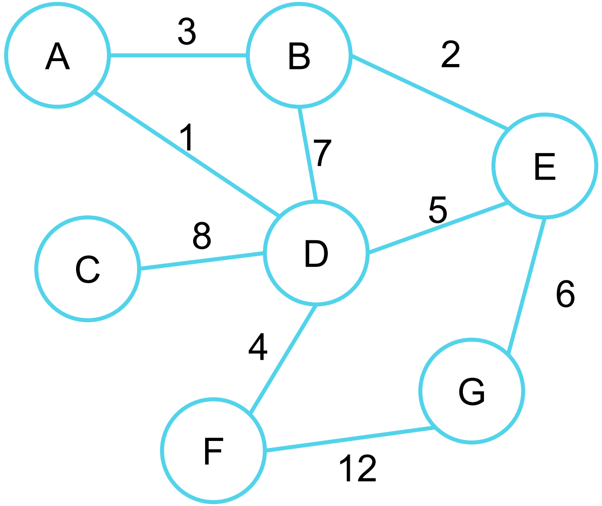
- 排序整個 edge list
  - 時間複雜度 O(m log m)
- merge 最多做 n - 1 次
  - 每次 merge 時間複雜度 O(log n)
  - 總時間複雜度 O(n log n)
- Kruskal 時間複雜度 O(m log m)

Prim's 演算法

- 和 Dijkstra 相似
  - 修改更新的部分
  - 從 priority_queue 拿出的合法狀態就是在新的一條樹邊
    - 注意判掉重複情形

Prim 過程

- 嘗試使用 Prim 找出最小生成樹



Prim - Code

```cpp
void Prim(int source) {
    fill(dis, dis + n, INF);
    priority_queue<pii, vector<pii>, greater<pii>> pq;
    pq.push({0, source});

    while (pq.size()) {
        long long d = pq.top().first;
        int u = pq.top().second;
        pq.pop();

        if (dis[u] != INF) continue;
        dis[u] = d;

        for (int i = 0; i < (int)G[u].size(); i++) {
            long long w = G[u][i].first;
            int v = G[u][i].second;
            pq.push({w, v});  // O(log n)
        }
    }
}
```

```cpp
int main() {
    cin.tie(0);
    cin.sync_with_stdio(0);

    init();
    Prim(0);

    long long ans = 0;
    for (int i = 0; i < n; i++) {
        ans += dis[i];
    }
    cout << ans << '\n';

    return 0;
}
```

Prim - 時間複雜度

- 對於每條邊都會將丟一個狀態進去 pq
  - pq 的 push 時間複雜度 O(log n)

- Prim 總時間複雜度 O(m log m)

ZeroJudge a129: 最小生成樹

- 給一張無向圖(可能不連通), 若不存在生成樹則輸出 `-1`, 否則輸出最小生成樹的邊權總和
- $2 \le n \le 10^5$, $1 \le m \le 2 * 10^5$
- $1 \le a, b \le n$, $|c| \le 2^{31}-1$

題單

- O(m log n) Kruskal MST
  - uva 10034 (MST 邊權總和)
  - uva 10147
  - uva 10397
  - uva 1665
  - NPSC 2010 高中組 初賽 D
  - zerojudge e509
- $O(n^2)$ Prim MST
  - leetcode 1584. Min Cost to Connect All Points $O(n^2)$ 曼哈頓距 離 MST
  - zj d909
  - NPSC 2013 高中組 初賽 B : $O(n^2)$ MST

題單

- 應用問題
    - uva 1395 : 最大邊跟最小邊差最少的spanning tree
    - leetcode 1579. Remove Max Number of Edges to Keep Graph Fully Traversable