Graph Algorithms

Lecture by Ccucumber12

Credit by double, yp155136, baluteshih, HNO2

2024/05/25

Sprout

Q & A

- 影片看了嗎？
- 有任何問題嗎？

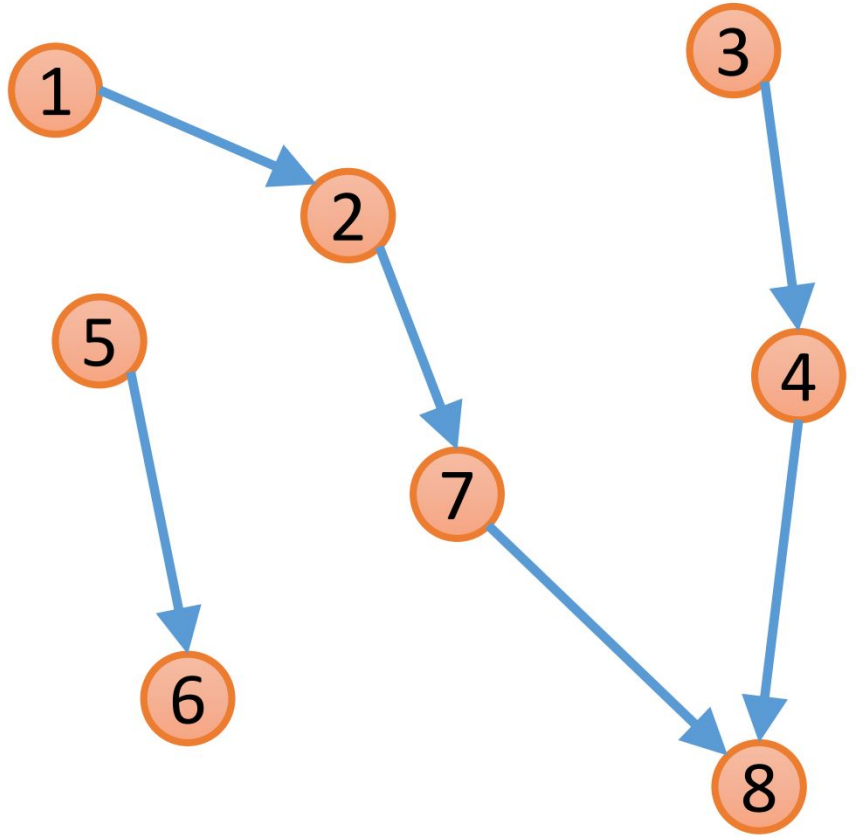
課程內容

- Topological Sort
- Minimum Spanning Tree
- Shortest Path

Topological Sort

Topological Sort

- 方法一:BFS 變形
  - Kahn’s Algorithm
  - 一直拔掉入度 0 的點



Topological Sort

- 想想看：如何輸出字典序最小的 topological sort

方法一：BFS 變形版本

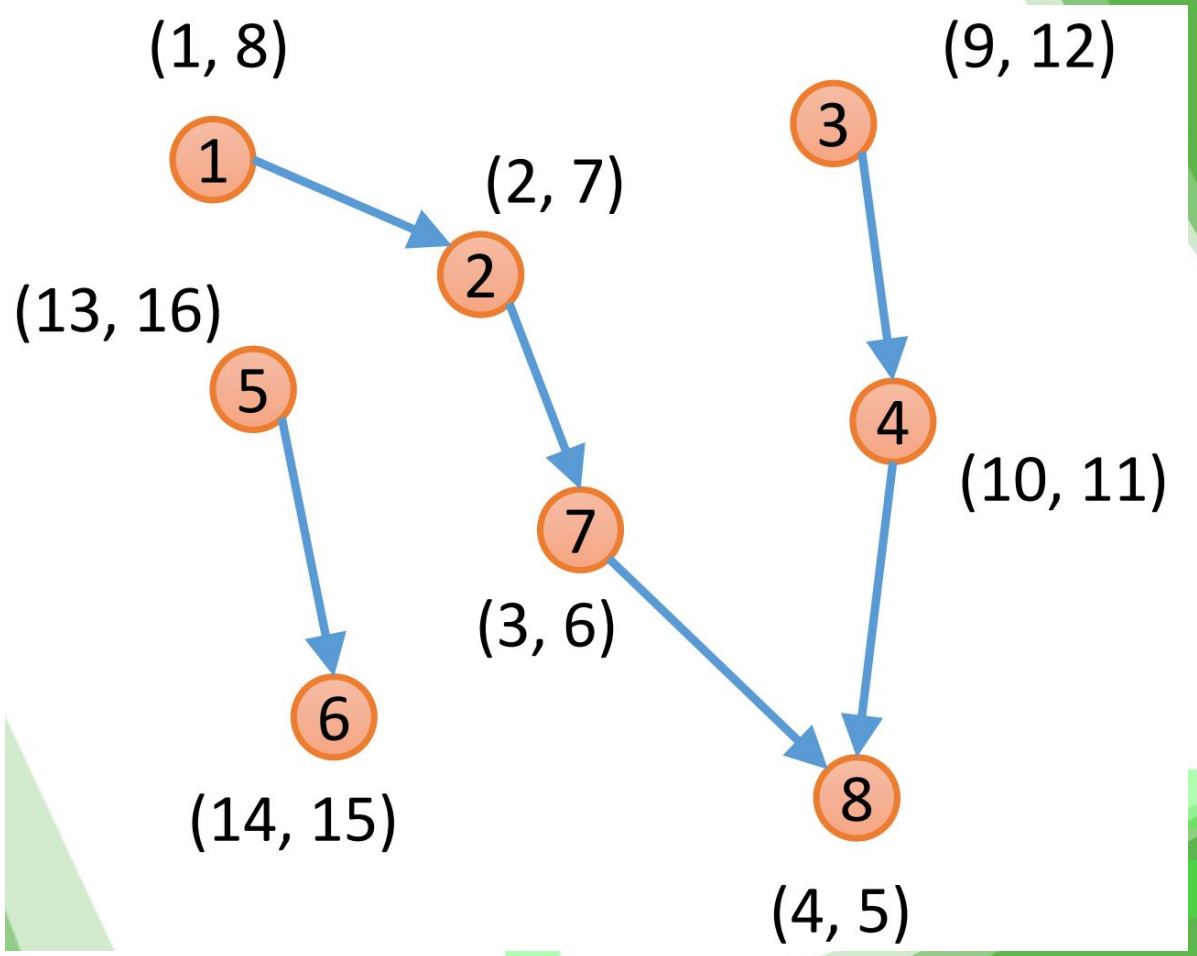
1. 初始將所有入度為 0 的點都推入 queue
2. 從 queue 中取出元素 p
3. 將 p 的出度都移除掉，並維護各個點的入度值
4. 如果某點在上步驟執行後入度變為 0，則將該點推入 queue
5. 若 queue 不為空，回到步驟 2
6. 算法結束時，取出的順序即為一組解

Topological Sort

- 想想看：如何輸出字典序最小的 topological sort
- 不一定要是 queue
- 可以是 set, priority_queue, ...

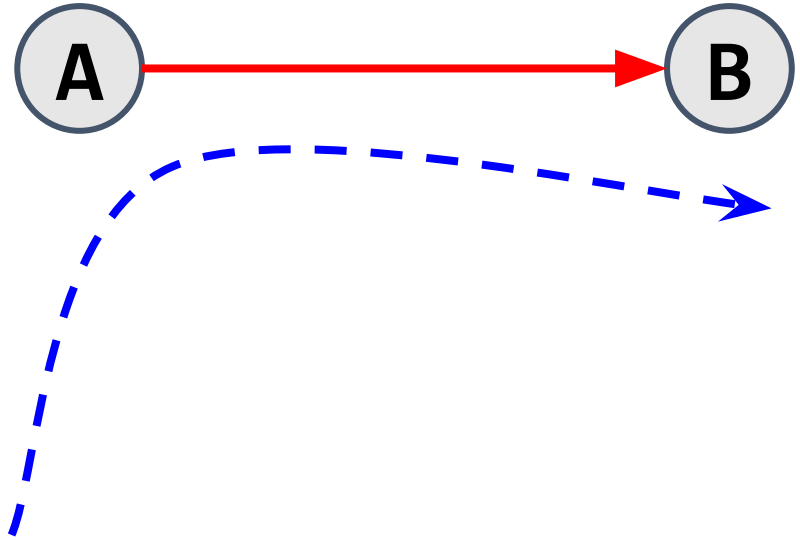
Topological Sort

- 方法二:DFS 時間戳記
  - 越早離開順序越後面



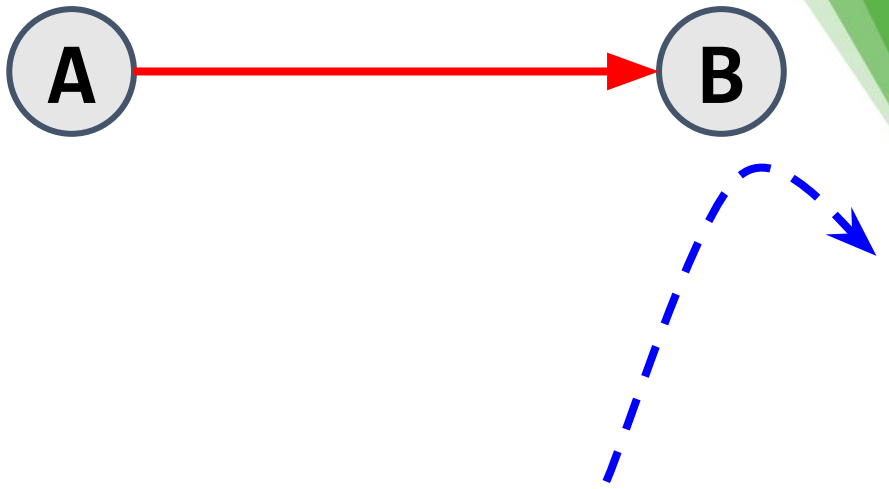
Topological Sort

- 方法二:DFS 時間戳記 - 感性理解
- 如果從 A 進去
  - 會走到 B
  - 然後 B 先離開



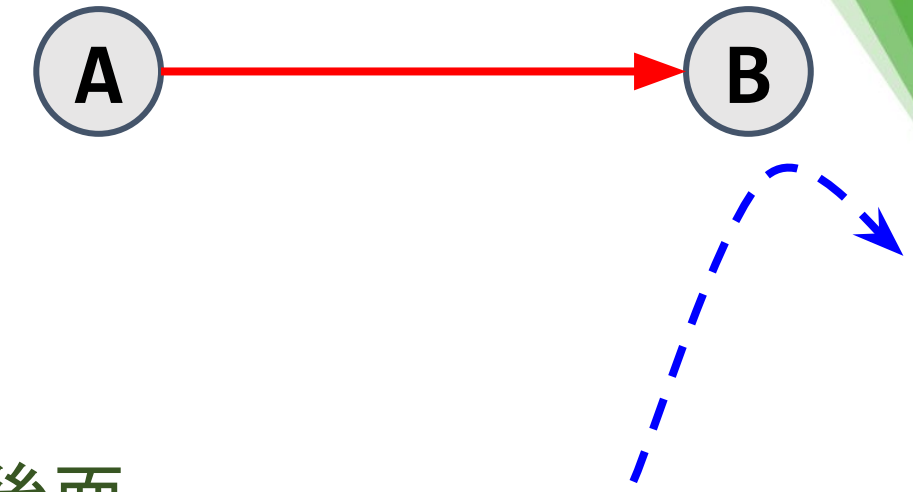
Topological Sort

- 方法二:DFS 時間戳記 - 感性理解
- 如果從 A 進去
  - 會走到 B
  - 然後 B 先離開
- 如果從 B 進去
  - 不會走到 A
  - 然後自己離開



Topological Sort

- 方法二:DFS 時間戳記 - 感性理解
- 如果從 A 進去
  - 會走到 B
  - 然後 B 先離開
- 如果從 B 進去
  - 不會走到 A
  - 然後自己離開
- B 永遠都會先離開 -> 先離開的排後面



Topological Sort

- 在 Excel / Google Spreadsheet 上的應用

比賽編號	L5		輪次	敗部
比賽時間	2024-05-20 21:30		裁判	thomaswang
	ID	排名	總分	扣分
玩家 1	chengbilly92	2	190	
玩家 2	victor	3	164	
玩家 3	ZCKevin	4	151	
玩家 4	Ccucumber12	1	206	

	chengbilly92	victor	ZCKevin	Ccucumber12
禁圖	N64 彩虹之路	GBA 絲帶之路	3DS 新庫巴市	GC 寶寶公園
控制器	Joy-Con (L)	Joy-Con (R)	Joy-Con (R)	Joy-Con (L)
智能駕駛	智能駕駛: OFF	智能駕駛: OFF	智能駕駛: OFF	智能駕駛: OFF
陀螺儀	陀螺儀: ON	陀螺儀: OFF	陀螺儀: OFF	陀螺儀: OFF
自動加速	自動加速: OFF	自動加速: ON	自動加速: ON	自動加速: OFF
角色	耀西	耀西	耀西	碧姬比姬公主
車輛	小熊輕型車	飛行鬱鬱	飛行鬱鬱	豆豆摩托車
輪胎	天空滾輪	天空滾輪	滾輪輪胎	滾輪輪胎
滑翔翼	紙飛機	紙飛機	雲形氣球	紙飛機

賽道狀況

蘑菇盃	瑪利歐賽車競技場	水上樂園	甜點峽谷	咚咚遺跡
鮮花盃	瑪利歐賽道	奇諾比奧海港	扭曲洋樓	嘿呵礦山
星星盃	陽光機場	海豚海角	電子幻界	瓦利歐雪山
特別盃	空中花園	骨骨沙漠	庫巴城堡	彩虹之路
圓蛋盃	GC 耀西賽道	越野摩托車	龍盤之路	寂靜城市
動物盃	GC 寶寶公園	GBA 起司樂園	自然之路	動物森友會
龜殼盃	Wii 牛牛農場	GBA 瑪利歐賽道	DS 泡泡魚海灘	N64 奇諾比奧公路
香蕉盃	GC 乾旱沙漠	SFC 甜圈平原3	N64 碧姬公主賽道	3DS DK叢林

賽道數量	選圖玩家	賽道
16		
1	chengbilly92	滑滑扭扭樂
2	victor	越野摩托車
3	ZCKevin	瑪利歐賽道
4	Ccucumber12	N64 碧姬公主...
5	chengbilly92	SFC 甜圈平原3
6	victor	海豚海角
7	ZCKevin	3DS DK叢林
8	Ccucumber12	海拉魯賽道
9	chengbilly92	GC 冰凍樂園
10	victor	N64 奇諾比奧...
11	ZCKevin	GBA 起司樂園
12	Ccucumber12	鈴鈴地鐵
13	chengbilly92	彩虹之路
14	victor	DS 滴答時鐘
15	ZCKevin	GC 耀西賽道
16	Ccucumber12	碧海

	chengbilly92	victor	ZCKevin	Ccucumber12
	排名	排名	排名	排名

盃賽	賽道	禁圖 次數	禁圖 比例	選圖 次數	選圖 比例	禁選率 次數	禁選率 比例	平均得分 全體	平均得分 自選	平均得分 非自選
蘑菇盃	瑪利歐賽車競技場	0	0.00%	3	15.00%	3	15.00%	10.64	9.67	11.00
	水上樂園	0	0.00%	2	10.00%	2	10.00%	11.25	9.50	11.83
	甜點峽谷	0	0.00%	1	5.00%	1	5.00%	10.25	12.00	9.67
	咚咚遺跡	0	0.00%	8	40.00%	8	40.00%	10.38	10.50	10.33
鮮花盃	瑪利歐賽道	0	0.00%	8	40.00%	8	40.00%	10.81	10.38	10.96
	奇諾比奧海港	0	0.00%	6	30.00%	6	30.00%	10.38	10.50	10.33
	扭曲洋樓	0	0.00%	1	5.00%	1	5.00%	10.00	10.00	10.00
	嘿呵礦山	2	10.00%	5	25.00%	7	35.00%	10.25	12.80	9.40
星星盃	陽光機場	0	0.00%	13	65.00%	13	65.00%	10.92	10.85	10.95
	海豚海角	1	5.00%	5	25.00%	6	30.00%	9.68	11.75	9.13
	電子幻界	2	10.00%	7	35.00%	9	45.00%	10.07	11.86	9.45
	瓦利歐雪山	2	10.00%	3	15.00%	5	25.00%	10.50	13.33	9.56
特別盃	空中花園	2	10.00%	6	30.00%	8	40.00%	10.92	11.83	10.61
	骨骨沙漠	2	10.00%	4	20.00%	6	30.00%	10.56	9.75	10.83
	庫巴城堡	0	0.00%	4	20.00%	4	20.00%	9.25	6.50	10.17
	彩虹之路	4	20.00%	11	55.00%	15	75.00%	10.14	11.18	9.79
圓蛋盃	GC 耀西賽道	0	0.00%	7	35.00%	7	35.00%	11.04	12.00	10.71
	越野摩托車	3	15.00%	13	65.00%	16	80.00%	11.18	11.31	11.13
	龍盤之路	0	0.00%	10	50.00%	10	50.00%	10.40	12.50	9.70
	寂靜城市	3	15.00%	7	35.00%	10	50.00%	10.37	12.29	9.70
動物盃	GC 寶寶公園	12	60.00%	6	30.00%	18	90.00%	10.33	8.83	10.83
	GBA 起司樂園	0	0.00%	5	25.00%	5	25.00%	10.95	10.80	11.00
	自然之路	1	5.00%	10	50.00%	11	55.00%	10.74	11.70	10.41
	動物森友會	0	0.00%	1	5.00%	1	5.00%	10.50	15.00	9.00

Minimum Spanning Tree

Disjoint Set Union

Disjoint Set Union

- 操作
  - 詢問元素隸屬的集合
  - 合併兩個集合
- boss[i] 代表 i 上面的老大，依照最頂的那個分類
- 優化技巧
  - 啟發式合併
  - 路徑壓縮

DSU 應用

- 給你一張 N 個點 M 條邊的圖，那 M 條邊會被依序加進圖中

  請在每一條邊被加入的時候，判斷這張圖是不是二分圖

DSU 應用

- 有 N 個人在猜拳，他們每次出的拳都是固定的

  依序會跟你回報：某個人打贏了某個人，或者是某兩個人平手

  請在收到每個回報後，判斷當前的回報們有沒有矛盾

Minimum Spanning Tree

Properties

Minimum Spanning Tree

- 性質一:一定是生成「樹」?

Minimum Spanning Tree

- 性質一:一定是生成「樹」?
- 否則就有環
- 至少可以多拔掉一條邊變更好

Minimum Spanning Tree

- 性質二:Cycle Property
  - 環上最大邊必然不屬於 MST
- 如果有多條最大邊呢？

Minimum Spanning Tree

- 性質二:Cycle Property
  - 環上最大邊必然不屬於 MST
- 如果有多條最大邊呢？
  - 環上**唯一**最大邊不屬於 MST
  - 環上最大邊**可以**不屬於 MST

Minimum Spanning Tree

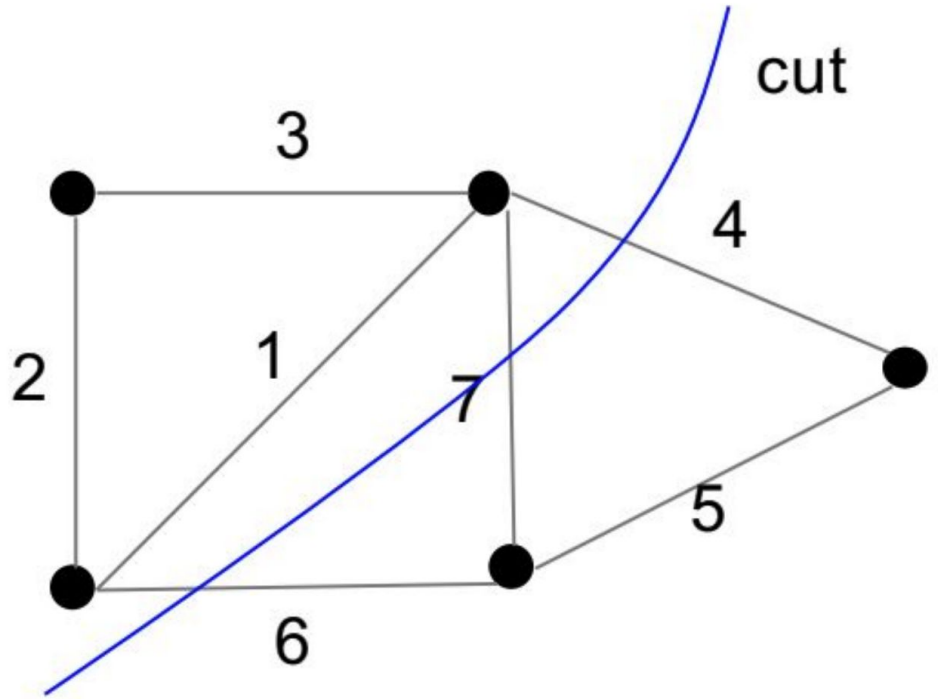
- 性質二:Cycle Property
  - 環上最大邊必然不屬於 MST
- 如果有多條最大邊呢？
  - 環上**唯一**最大邊不屬於 MST
  - 環上最大邊**可以**不屬於 MST
- MST 是否唯一？

Minimum Spanning Tree

- 性質二:Cycle Property
  - 環上最大邊必然不屬於 MST
- 如果有多條最大邊呢？
  - 環上**唯一**最大邊不屬於 MST
  - 環上最大邊**可以**不屬於 MST
- MST 是否唯一？
  - 如果所有邊權都不一樣那就一定唯一！
  - 但唯一不代表邊權一定都不同

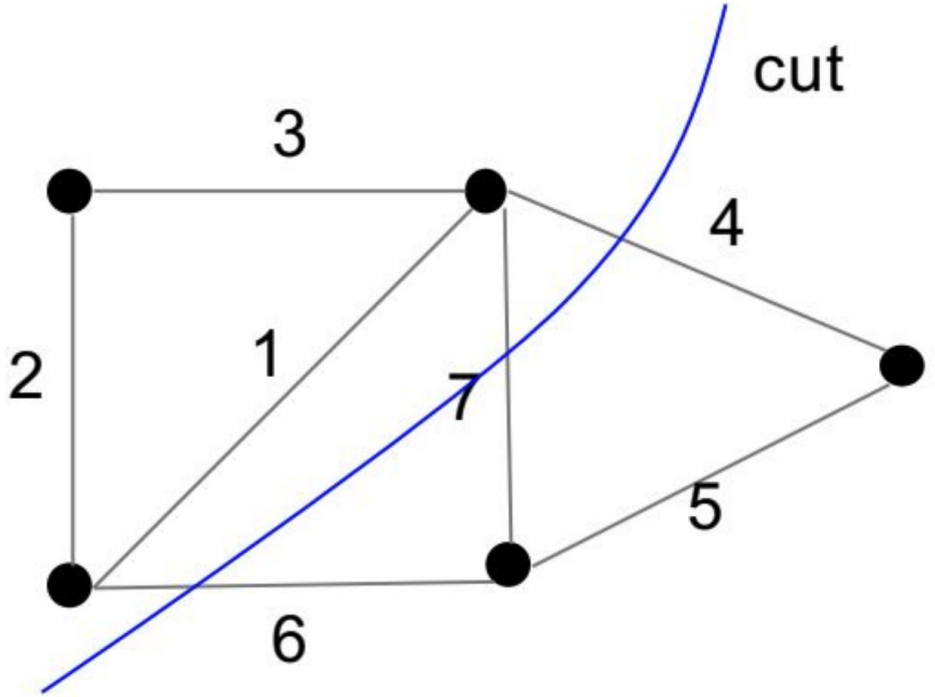
Minimum Spanning Tree

- 性質三:Cut Property
  - cut 上最小邊必然屬於 MST
- 如果有多條最小邊呢？



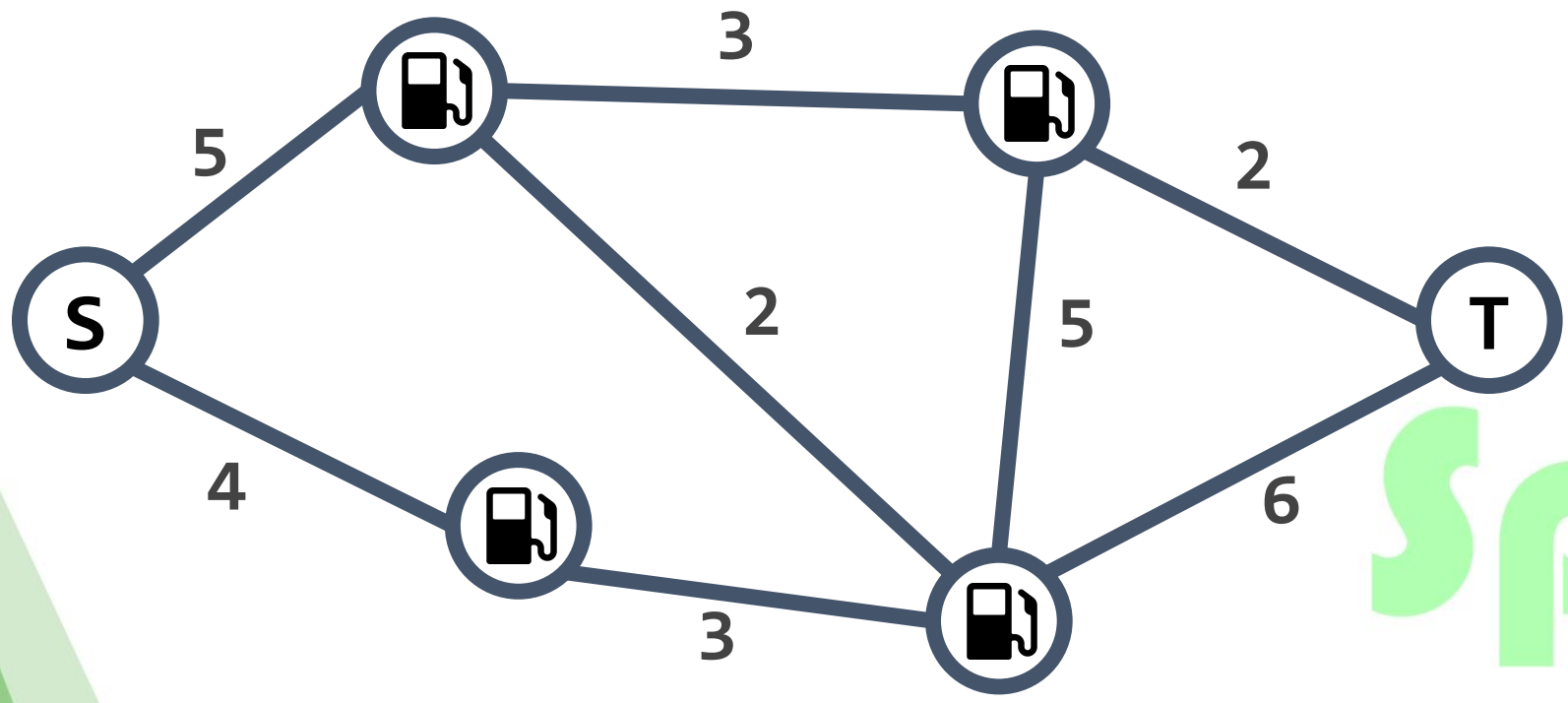
Minimum Spanning Tree

- 性質三:Cut Property
  - cut 上最小邊必然屬於 MST
- 如果有多條最小邊呢？
  - cut 上**唯一**最小邊屬於 MST
  - cut 上最小邊**可以**屬於 MST



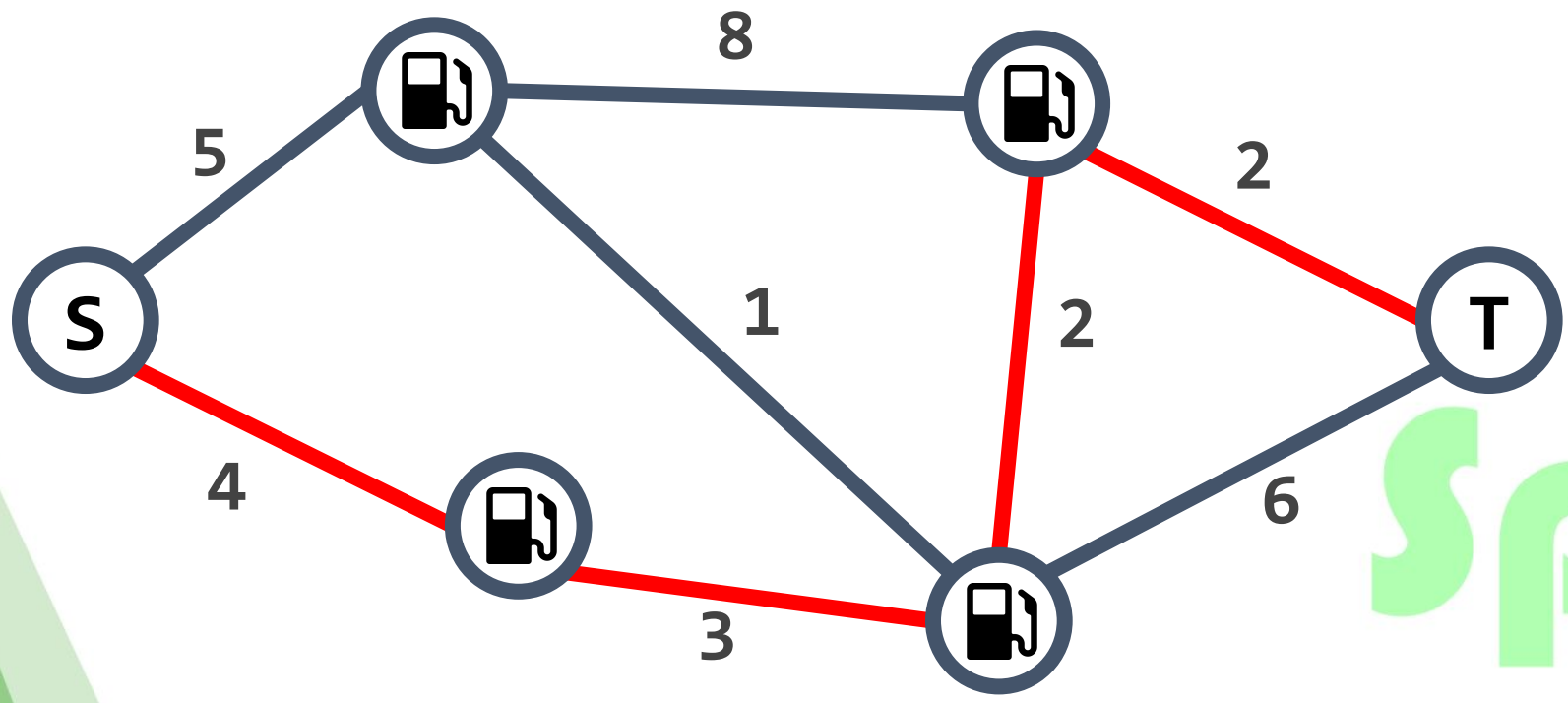
Minimum Spanning Tree

- 給定一張帶權無向圖和兩個點 S、T
- 最小化從 S 走到 T 經過的最大邊



Minimum Spanning Tree

- 給定一張帶權無向圖和兩個點 S、T
- 最小化從 S 走到 T 經過的最大邊
- Sol:先求 MST 然後在上面找 S 到 T 的最大邊
  - cut property！



Minimum Spanning Tree

Kruskal

Sprout

Kruskal

- 從最小邊開始依序檢查
- 如果兩端不連通
  - cut 上最小邊
  - 要選
- 如果兩端連通
  - 環上最大邊
  - 不選

Kruskal

- 從最小邊開始依序檢查
- 如果兩端不連通
  - cut 上最小邊
  - 要選
- 如果兩端連通
  - 環上最大邊
  - 不選
- 連通？

Kruskal

- 從最小邊開始依序檢查
- 如果兩端不連通
  - cut 上最小邊
  - 要選
- 如果兩端連通
  - 環上最大邊
  - 不選
- 連通？
  - Disjoint Set Union！

Kruskal

```
sort edges by weight
for e in edges:
    if dsu.find(e.u) != dsu.find(e.v):
        dsu.unite(e.u, e.v)
        e is an edge in MST
```

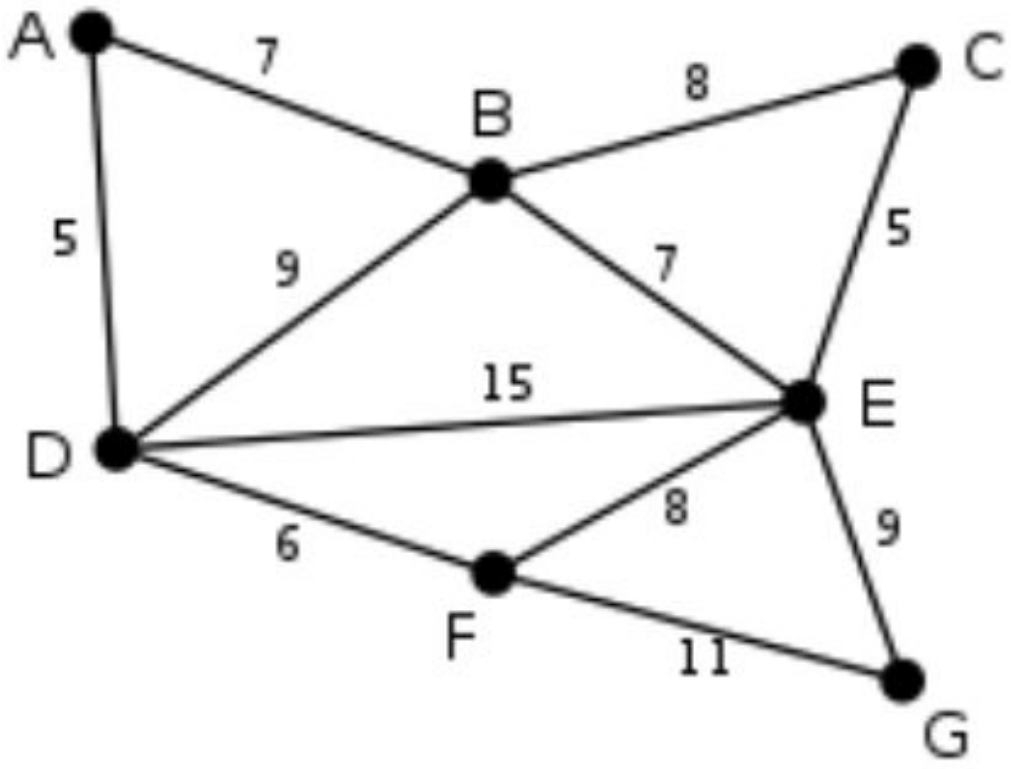
- 一些實作技巧
  - edge 可以開一個 struct 存
    - 或是使用 pair<int, pair<int, int>>
  - disjoint set 可以開一個 struct 維護(個人習慣)

Minimum Spanning Tree

Prim’s Algorithm

Prim's Algorithm

- 從任意一個點開始，每次從cost最小的點找可以延伸的最小邊



Prim's Algorithm

- O(E + V^2)

```
vector<int> G[n]; // 圖的相鄰串列
bool in_mst[n] = {false};
int dis[n] = {INT_MAX};
int par[n] = {0}

dis[0] = 0;

repeat n-1 times:
    find j such that in_mst[j] == false and dis[j] is minimal
    add (par[j], j) into MST
    in_mst[j] = true
    for E in G[j]:
        dis[E.to] = min(dis[E.to], E.weight)
        if dis[E.to] == E.weight:
            par[E.to] = j
```

Prim's Algorithm

- O((E + V) log E) ?
- 想想看哪個地方可以用資料結構優化

```
vector<int> G[n]; // 圖的相鄰串列
bool in_mst[n] = {false};
int dis[n] = {INT_MAX};
int par[n] = {0}

dis[0] = 0;

repeat n-1 times:
    find j such that in_mst[j] == false and dis[j] is minimal
    add (par[j], j) into MST
    in_mst[j] = true
    for E in G[j]:
        dis[E.to] = min(dis[E.to], E.weight)
        if dis[E.to] == E.weight:
            par[E.to] = j
```

Prim's Algorithm

- O((E + V) log E) ?
- 想想看哪個地方可以用資料結構優化

```
vector<int> G[n]; // 圖的相鄰串列
bool in_mst[n] = {false};
int dis[n] = {INT_MAX};
int par[n] = {0}

dis[0] = 0;

repeat n-1 times:
    find j such that in_mst[j] == false and dis[j] is minimal
    add (par[j], j) into MST
    in_mst[j] = true
    for E in G[j]:
        dis[E.to] = min(dis[E.to], E.weight)
        if dis[E.to] == E.weight:
            par[E.to] = j
```

Prim's Algorithm

- O((E + V) log E) ?
- 想想看哪個地方可以用資料結構優化
- 開一個 min heap 存 (dis[j], j)
- 選 j 的時候, 就看 min heap 的 top
- 如果發現不符合的, 就 pop 到符合為止

```
    find j such that in_mst[j] == false and dis[j] is minimal
```

Prim's Algorithm

- O((E + V) log E) ?
- 想想看哪個地方可以用資料結構優化
- 更新:dis 有被更動過的時候, 就把新的 (dis[E.to], E.to) 丟到 min heap 裡面！

```
    find j such that in_mst[j] == false and dis[j] is minimal
```

```
        dis[E.to] = min(dis[E.to], E.weight)
```

MST 應用

- 有 N 個國家想要石油。這 N 個國家有 M 條路連結。
  對於第 i 個國家可以花 c_i 塊開採石油。
  對於第 i 條邊，可以花 w_i 塊讓 u_i 國跟 v_i 國連通。一個國家開採石油後，可以無限制的分享給連通的其他國家。
- 最少要花多少花費，才能讓每個國家都有石油

MST 應用

- 旅行推銷員問題(Traveling Salesman Problem, TSP)
- 給定一張圖，最小化從 s 出發，經過其他所有點回到 s 的路徑長總和
- NP-Hard :(
- 使用 MST 做估計
  - 最多不超過最佳解兩倍
  - 2-approximation

Shortest Path

Floyd-Warshall

Floyd-Warshall

- 為什麼是對的？
- 為什麼說這個算法是某種 DP 呢？

Floyd-Warshall

- 原本的狀態:

```
d[i][j]:從點 i 走到點 j 的最短距離
for k = 1..n
   for i = 1..n
      for j = 1..n
         d[i][j] = min(d[i][j], d[i][k]+d[k][j]);
```

Floyd-Warshall

- 改變一下:

  d[k][i][j]: i 到 j 的最短距離, 且只能額外經過前 k 個點

```
for k = 1..n
   for i = 1..n
      for j = 1..n
         d[k][i][j] =
         min(d[k-1][i][j], d[k-1][i][k]+d[k-1][k][j]);
```

Floyd-Warshall

```
for k = 1..n
   for i = 1..n
      for j = 1..n
         d[i][j] = min(d[i][j],d[i][k]+d[k][j]);
```

- 如果有聽懂剛剛的推導，應該就不會再打錯順序了吧
- 蛤可是好擔心哪次還是會不小心打錯喔。

Incorrect implementations of the Floyd–Warshall algorithm give correct solutions after three repeats

Ikumi Hide

The University of Tokyo
ihide@es.a.u-tokyo.ac.jp

Soh Kumabe

The University of Tokyo
sohkuma0213@gmail.com

Takanori Maehara

RIKEN Center for Advanced Intelligence Project
takanori.maehara@riken.jp

Abstract

The Floyd–Warshall algorithm is a well-known algorithm for the all-pairs shortest path problem that is simply implemented by triply nested loops. In this study, we show that the incorrect implementations of the Floyd–Warshall algorithm that misorder the triply nested loops give correct solutions if these are repeated three times.

Keywords: graph algorithm; algorithm implementation; common mistake

1. Introduction

The *Floyd–Warshall algorithm* is a well-known algorithm for the all-pairs

Floyd-Warshall

- 這篇2019的新論文告訴了我們，如果你把迴圈的順序打成由外到內是ijk的話，重複跑三遍還是會得到正確的結果喔！
- 哇！不用再怕打錯了呢！

Theorem 1. *If we repeat IJK algorithm three times, it solves the all-pairs shortest path problem. Conversely, there exists an instance that needs three repeats to obtain a correct solution.*

Theorem 2. *If we repeat IKJ algorithm two times, it solves the all-pairs shortest path problem. Conversely, there exists an instance that needs two repeats to obtain a correct solution.*

Shortest Path

Dijkstra

Dijkstra

- 起點 s，還沒拜訪的點集合 U（一開始 U = V）
- 目前最佳距離 d[]（一開始 d[] = INF）
- 目前節點 p（一開始 p = s, d[s] = 0）

Dijkstra

- 起點 s，還沒拜訪的點集合 U（一開始 U = V）
- 目前最佳距離 d[]（一開始 d[] = INF）
- 目前節點 p（一開始 p = s, d[s] = 0）
- 重複直到 U 是空的:
  - 對於 p 的所有相鄰點 i，更新 d[i] = min(d[i], d[i]+w(p,i))
  - 把 p 移出 U
  - 令 p 為 U 中有最近的人（找 d[] 中最小的人）

Dijkstra

- 起點 s，還沒拜訪的點集合 U（一開始 U = V）
- 目前最佳距離 d[]（一開始 d[] = INF）
- 目前節點 p（一開始 p = s, d[s] = 0）
- 重複直到 U 是空的:
  - 對於 p 的所有相鄰點 i，更新 d[i] = min(d[i], d[i]+w(p,i))
  - 把 p 移出 U
  - 令 p 為 U 中有最近的人（找 d[] 中最小的人）
- 答案:d[] 陣列

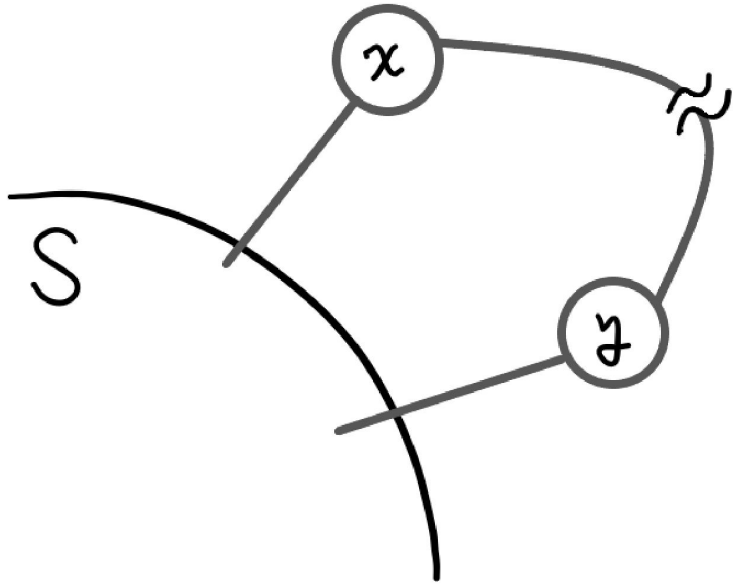
為什麼他是對的？

Dijkstra

- **貪心**地選擇目前最近的點
- 為什麼可以貪心？

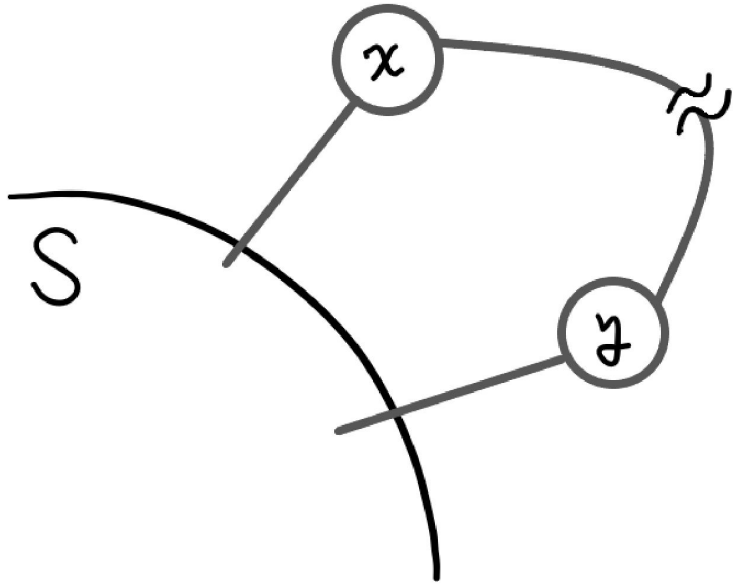
Dijkstra

- **貪心**地選擇目前最近的點
- 為什麼可以貪心？
  - 假設 x 是最近點:d[x] < d[y]
  - 如果不選擇走 x 而是走 y
  - 之後繞回來 x 只會更遠
  - 不如一開始就走 x！



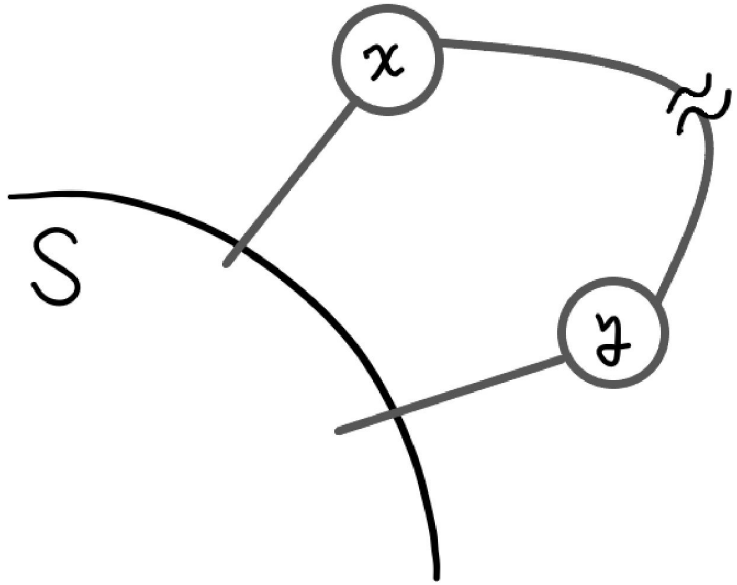
Dijkstra

- **貪心**地選擇目前最近的點
- 為什麼可以貪心？
  - 假設 x 是最近點:d[x] < d[y]
  - 如果不選擇走 x 而是走 y
  - 之後繞回來 x 只會更遠 **真的嗎？**
  - 不如一開始就走 x！



Dijkstra

- **貪心**地選擇目前最近的點
- 為什麼可以貪心？
  - 假設 x 是最近點:d[x] < d[y]
  - 如果不選擇走 x 而是走 y
  - 之後繞回來 x 只會更遠 **真的嗎？**
  - 不如一開始就走 x！
- d[x] < d[y] + w(y, ..., x)
  - w(y, ..., x) >= 0
  - 不能有負邊！



Dijkstra

$O(E+V^2)$

還可以更快嗎？

Dijkstra

- 起點 s，還沒拜訪的點集合 U（一開始 U = V）
- 目前最佳距離 d[]（一開始 d[] = INF）
- 目前節點 p（一開始 p = s, d[s] = 0）
- 重複直到 U 是空的:
  - 對於 p 的所有相鄰點 i，更新 d[i] = min(d[i], d[i]+w(p,i))
  - 把 p 移出 U
  - 令 p 為 U 中有最近的人（找 d[] 中最小的人）
- 答案:d[] 陣列

Dijkstra

- 起點 s，還沒拜訪的點集合 U（一開始 U = V）
- 目前最佳距離 d[]（一開始 d[] = INF）
- 目前節點 p（一開始 p = s, d[s] = 0）
- 重複直到 U 是空的:
  - 對於 p 的所有相鄰點 i，更新 d[i] = min(d[i], d[i]+w(p,i))
  - 把 p 移出 U
  - **令 p 為 U 中有最近的人（找 d[] 中最小的人）**
- 答案:d[] 陣列

Dijkstra

- 從一個集合裡面找出最小的數字

Dijkstra

- 從一個集合裡面找出最小的數字
- heap / priority_queue!

Dijkstra

- 從一個集合裡面找出最小的數字
- heap / priority_queue!
- 用priority_queue找到d[]最小的點，然後更新其他點的d[]

Dijkstra

- 從一個集合裡面找出最小的數字
- heap / priority_queue!
- 用priority_queue找到d[]最小的點，然後更新其他點的d[]
- 原本:(V 輪) * (V 找最小 + 更新相連點) = $O(E+V^2)$

Dijkstra

- 從一個集合裡面找出最小的數字
- heap / priority_queue!
- 用priority_queue找到d[]最小的點，然後更新其他點的d[]
- 原本:(V 輪) * (V 找最小 + 更新相連點) = $O(E+V^2)$
- 新做法:
  - (V 輪) * ((logE 找最小) + (相鄰點 logE 更新進heap))
  - (V輪 * logE 找最小) + (E條邊 * logE 更新進heap)
  - **O((V+E)logE)**

怎麼找出最短路徑本身？

最短路徑樹

- 每一個最短路徑一定都是**簡單路徑**

最短路徑樹

- 每一個最短路徑一定都是簡單路徑
- 把這些路徑疊起來會變成一棵樹
- 我們稱這種樹叫做最短路徑樹

最短路徑樹

- 每一個最短路徑一定都是簡單路徑
- 把這些路徑疊起來會變成一棵樹
- 我們稱這種樹叫做最短路徑樹
- 樹上的節點，父節點是唯一的
- 紀錄轉移的前一項！

Shortest Path

Bellman-Ford

Bellman-Ford

核心概念：放鬆(RELAX)

- 對一條邊 (u, v)
- 如果滿足 $\text{dis}(u) + \text{weight}(u, v) < \text{dis}(v)$
- 就更新 $\text{dis}(v)$

Bellman-Ford

- d[i] 是從起點走到點 i 的最短距離
- w(i, j) 是某條從 i 走到 j 的邊權

Bellman-Ford

- d[i] 是從起點走到點 i 的最短距離
- w(i, j) 是某條從 i 走到 j 的邊權
- 根據定義:d[j] <= d[i]+w(i, j), 否則d[j]不是最短距離

Bellman-Ford

- d[i] 是從起點走到點 i 的最短距離
- w(i, j) 是某條從 i 走到 j 的邊權
- 根據定義:d[j] <= d[i]+w(i, j), 否則d[j]不是最短距離
- 想法:
  - 若對於 (i, j) 有 d[j] > d[i]+w(i, j)
  - 把 d[j] 更新成 d[i]+w(i, j)

Bellman-Ford

- 想法:
  - 若對於 (i, j) 有 d[j] > d[i]+w(i, j)
  - 把 d[j] 更新成 d[i]+w(i, j)
- 怎麼更新？

Bellman-Ford

- 想法:
  - 若對於 (i, j) 有 d[j] > d[i]+w(i, j)
  - 把 d[j] 更新成 d[i]+w(i, j)
- 怎麼更新？
- 隨便亂更新！

Bellman-Ford

算法

- 初始化所有 $\mathrm{dis}(u)$ 為無窮大（起點設為 0）
- 對圖上的每一條邊放鬆一次
- 重複上面的步驟 $V-1$ 次

為什麼他是對的？

Bellman-Ford

- 如果最短路存在，距離(經過的邊數)不超過 V-1
  - 在最短路徑樹上兩點距離不超過 V-1
  - 否則就有環了

Bellman-Ford

- 如果最短路存在，距離(經過的邊數)不超過 V-1
  - 在最短路徑樹上兩點距離不超過 V-1
  - 否則就有環了
- 第一輪放鬆會完成距離 1 的點

Bellman-Ford

- 如果最短路存在，距離(經過的邊數)不超過 V-1
  - 在最短路徑樹上兩點距離不超過 V-1
  - 否則就有環了
- 第一輪放鬆會完成距離 1 的點
- 第二輪放鬆會完成距離 2 的點

Bellman-Ford

- 如果最短路存在，距離(經過的邊數)不超過 V-1
  - 在最短路徑樹上兩點距離不超過 V-1
  - 否則就有環了
- 第一輪放鬆會完成距離 1 的點
- 第二輪放鬆會完成距離 2 的點
- …
- 第 V-1 輪放鬆會完成距離 V-1 的點 -> 所有點！

Bellman-Ford

- **如果最短路存在**，距離(經過的邊數)不超過 V-1
  - 在最短路徑樹上兩點距離不超過 V-1
  - 否則就有環了
- 第一輪放鬆會完成距離 1 的點
- 第二輪放鬆會完成距離 2 的點
- …
- 第 V-1 輪放鬆會完成距離 V-1 的點 -> 所有點！

Bellman-Ford

- 如果圖存在**負環**
- 可以在負環上面瘋狂打轉

Bellman-Ford

- 如果圖存在**負環**
- 可以在負環上面瘋狂打轉
- 路徑權重和變成無限小
- 最短路不存在

Shortest Path

Negative Cycle

負環

- 什麼叫做處理負環呢？
- 不是說好有負環的話就沒有最短路嗎？

負環

- 什麼叫做處理負環呢？
- 不是說好有負環的話就沒有最短路嗎？
- 有一些技巧可以拿來看有沒有負環
  - 更新的次數太多
  - …?

Bellman-Ford

- 如果最短路存在，距離(經過的邊數)不超過 V-1
  - 在最短路徑樹上兩點距離不超過 V-1
  - 否則就有環了
- 第一輪放鬆會完成距離 1 的點
- 第二輪放鬆會完成距離 2 的點
- …
- 第 V-1 輪放鬆會完成距離 V-1 的點 -> 所有點！

Bellman-Ford

- 如果最短路存在，距離(經過的邊數)不超過 V-1
  - 否則就有環了
  - 最短路徑樹
- 第一輪放鬆會完成距離 1 的點
- 第二輪放鬆會完成距離 2 的點
- …
- 第 V-1 輪放鬆會完成距離 V-1 的點
- **第 V 輪還成功放鬆某個人...**

Bellman-Ford

- 如果最短路存在，距離(經過的邊數)不超過 V-1
  - 否則就有環了
  - 最短路徑樹
- 第一輪放鬆會完成距離 1 的點
- 第二輪放鬆會完成距離 2 的點
- …
- 第 V-1 輪放鬆會完成距離 V-1 的點
- **第 V 輪還成功放鬆某個人...**
  - **負環存在！**

負環

- Floyd-Warshall 呢？

Floyd Warshall

- 原本的狀態:

```
d[i][j]:從點 i 走到點 j 的最短距離
for k = 1..n
   for i = 1..n
      for j = 1..n
         d[i][j] = min(d[i][j], d[i][k]+d[k][j]);
```

負環

- Floyd-Warshall 呢？
- d[i][j]: 從 i 走到 j 的最短距離

負環

- Floyd-Warshall 呢？
- d[i][j]: 從 i 走到 j 的最短距離
- d[i][i]: 從 i 走到 i 的最短距離

負環

- Floyd-Warshall 呢？
- d[i][j]: 從 i 走到 j 的最短距離
- d[i][i]: 從 i 走到 i 的最短距離
  - 若負環存在:d[i][i] < 0！

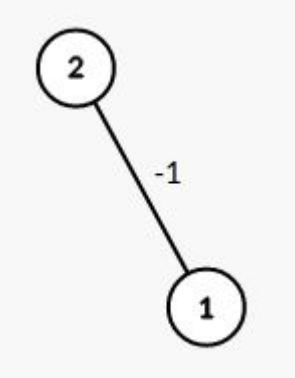
負環

- Floyd-Warshall 呢？
- d[i][j]: 從 i 走到 j 的最短距離
- d[i][i]: 從 i 走到 i 的最短距離
  - 若負環存在:d[i][i] < 0！
- 無向圖？


2
-1
1
請問,這是負環嗎?

負環

- 無向圖？



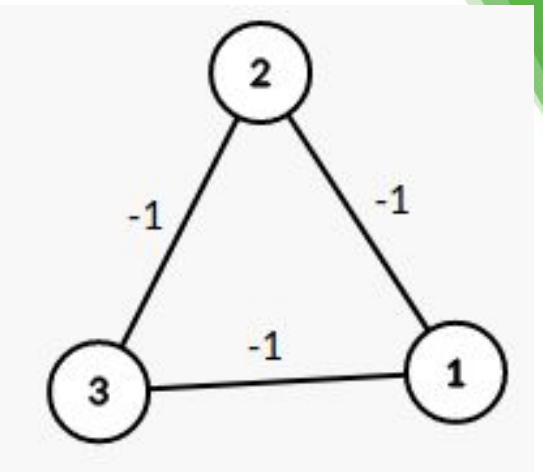
- 這...是不是負環啊？

負環

- 無向圖的負邊，Floyd-Warshall，都會判斷成負環。

負環

- 無向圖的負邊，Floyd-Warshall，都會判斷成負環
- 我們會在感情上(?)不太能接受這個結論。我們希望被判斷出來的負環是**簡單負環**，就是不能使用同一條邊。
- 所以怎麼在無向圖判斷存不存在簡單負環？
- 很難 QwQ
- 2009年時有一篇論文提出了一個 O(N^3) 的做法有興趣的可以去查查看(?)
  - Improved Algorithms for Detecting Negative Cost Cycles in Undirected Graphs



各種比較

	Bellman-Ford	Dijkstra	Floyd-Warshall
類型	單點源	單點源	全點對
限制	任意圖	無負邊	任意圖
時間複雜度	$O(VE)$	$O(V^2 + E)$ $O(E \log E)$	$O(V^3)$
用途	檢查負環	快速單點源	全點對

Shortest Path

Applications

2017認證考 – 江神與他的小火車

- 有一個王國目前有 N 個城市跟 M 條火車單向軌道，每條軌道有其長度。
- 為了即將到來的世界大學運動會，江神打算再多建造一條長度 1 的軌道來提昇從城市 1 到城市 N 的運輸效率。
- 請你回答 Q 次，如果建立一條長度 1 的單向軌道，連結城市 a 到 b，從城市 1 到城市 N 的運輸效率會變成多少？

- N <= 2*10^5, M <= 5*10^5, Q <= 10^5

2017認證考 – 江神與他的小火車

- 如果有用新加進來的邊 a->b .....
- 路徑可以被拆解成 1->a + a->b + b->N

2017認證考 – 江神與他的小火車

- 如果有用新加進來的邊 a->b .....
- 路徑可以被拆解成 1->a + a->b + b->N
- 左側固定都是求 1 出發到某點的最短距離
- 右側固定都是求某點出發到 N 的最短距離

2017認證考 – 江神與他的小火車

- 如果有用新加進來的邊 a->b .....
- 路徑可以被拆解成 1->a + a->b + b->N
- 左側固定都是求 1 出發到某點的最短距離
- 右側固定都是求某點出發到 N 的最短距離
- 左側我們會求，那右側呢？

2020 全國賽 – pB.村莊與小徑

- 小鎮有 N 個村莊 M 條單向路徑。每個路徑有一個危險程度 wi，且保證不會出現環狀路徑。
- 定義村莊 x 的危險評分為從村莊 1 走到 x 的最小危險程度總和。
- 求所有村莊的危險評分

- N <= 1e5, m <= 2e5, |wi| <= 1e6

TIOJ 題目

2020 全國賽 – pB.村莊與小徑

- 小鎮有 N 個村莊 M 條單向路徑。每個路徑有一個危險程度 wi，且保證不會出現環狀路徑。
- 定義村莊 x 的危險評分為從村莊 1 走到 x 的最小危險程度總和。
- 求所有村莊的危險評分

- N <= 1e5, m <= 2e5, **|wi| <= 1e6**

TIOJ 題目

2020 全國賽 – pB.村莊與小徑

- 小鎮有 N 個村莊 M 條單向路徑。每個路徑有一個危險程度 wi，且**保證不會出現環狀路徑**。
- 定義村莊 x 的危險評分為從村莊 1 走到 x 的最小危險程度總和。
- 求所有村莊的危險評分

- N <= 1e5, m <= 2e5, **|wi| <= 1e6**

TIOJ 題目

2020 全國賽 – pB.村莊與小徑

- Topological Sort！
- 一條邊一旦經過了就再也不能回來了
- 照順序用每條邊更新 d[] 陣列

Questions?

Sprout

YP們

- 現在有 N 個 YP 編號為 1 到 N，他們很乖的照著編號順序排在一條整數數線上但不能重疊。
- 不過這些 YP 有 M 條要求，每條要求可能是：
  - 編號 a_i 跟編號 b_i 的 YP 之間不願意相離超過 d_i。
  - 編號 a_i 跟編號 b_i 的 YP 之間不願意相距 d_i 以內。
- 請問是否可以滿足所有要求，使得 YP 們可以依舊照著編號順序排在一條數線上？
- 若可以，請輸出編號 1 和編號 N 的 YP 之間的最大距離，或者是判斷他們是不是可以相距無限遠。

- N,M <= 1000

YP們

- 我們有哪些限制？

YP們

- 我們有哪些限制？
- a_i 和 b_i 不超過 d_i
- a_i 和 b_i 超過 d_i
- p_i 小於 p_{i+1}

YP們

- 我們有哪些限制？
- **a_i 和 b_i 不超過 d_i**
- a_i 和 b_i 超過 d_i
- p_i 小於 p_{i+1}

YP們

- 我們有哪些限制？
- a_i 和 b_i 不超過 d_i
- p_a + d >= p_b

YP們

- 我們有哪些限制？
- a_i 和 b_i 不超過 d_i
- p_a + d >= p_b
- 所有人在 0 一定是合法解

YP們

- 我們有哪些限制？
- a_i 和 b_i 不超過 d_i
- p_a + d >= p_b
- 所有人在 0 一定是合法解
- 但 p_1 和 p_N 可以最大離多遠呢？

YP們

- 但 p_1 和 p_N 可以最大離多遠呢？

YP們

- 但 p_1 和 p_N 可以最大離多遠呢？
- 可能一
  - p_1 + d_1 >= p_N

YP們

- 但 p_1 和 p_N 可以最大離多遠呢？
- 可能一
  - p_1 + d_1 >= p_N
- 可能二
  - p_1 + d_2 >= p_2
  - p_2 + d_3 >= p_N
  - → p_1 + (d_2 + d_3) >= p_N

YP們

- 但 p_1 和 p_N 可以最大離多遠呢？
- 可能一
  - p_1 + d_1 >= p_N
- 可能二
  - p_1 + d_2 >= p_2
  - p_2 + d_3 >= p_N
  - → p_1 + (d_2 + d_3) >= p_N
- 要取 min(d_1, d_2 + d_3)

YP們

- 但 p_1 和 p_N 可以最大離多遠呢？
- 可能一
  - p_1 + d_1 >= p_N
- 可能二
  - p_1 + d_2 >= p_2
  - p_2 + d_3 >= p_N
  - → p_1 + (d_2 + d_3) >= p_N
- 要取 min(d_1, d_2 + d_3)
- 要怎麼找到所有可能並取 min ？

YP們

- 但 p_1 和 p_N 可以最大離多遠呢？
- 可能一
  - p_1 + d_1 >= p_N
- 可能二
  - p_1 + d_2 >= p_2
  - p_2 + d_3 >= p_N
  - → p_1 + (d_2 + d_3) >= p_N
- 要取 min(d_1, d_2 + d_3)
- 要怎麼找到所有可能並取 min ？
  - 圖論！

YP們

- 但 p_1 和 p_N 可以最大離多遠呢？
- 可能一
  - p_1 + d_1 >= p_N → 1 到 N 建一條長度 d_1 的邊
- 可能二
  - p_1 + d_2 >= p_2 → 1 到 2 建一條長度 d_2 的邊
  - p_2 + d_3 >= p_N → 2 到 N 建一條長度 d_3 的邊
  - → p_1 + (d_2 + d_3) >= p_N
- 要取 min(d_1, d_2 + d_3)
- 要怎麼找到所有可能並取 min ？
  - 圖論！

YP們

- 但 p_1 和 p_N 可以最大離多遠呢？
- 可能一
  - p_1 + d_1 >= p_N → 1 到 N 建一條長度 d_1 的邊
- 可能二
  - p_1 + d_2 >= p_2 → 1 到 2 建一條長度 d_2 的邊
  - p_2 + d_3 >= p_N → 2 到 N 建一條長度 d_3 的邊
  - → p_1 + (d_2 + d_3) >= p_N
- 要取 min(d_1, d_2 + d_3)
- 要怎麼找到所有可能並取 min ？
  - 圖論！
  - 到 N 的最短路就是 min！

YP們

- 我們有哪些限制？
- a_i 和 b_i 不超過 d_i
- a_i 和 b_i 超過 d_i
- **p_i 小於 p_{i+1}**

YP們

- p_i 小於 p_{i+1}
- p_i < p_{i+1}

YP們

- p_i 小於 p_{i+1}
- p_i < p_{i+1}
- p_{i+1} - 1 >= p_i

YP們

- p_i 小於 p_{i+1}
- p_i < p_{i+1}
- p_{i+1} - 1 >= p_i
- p_{i+1} 到 p_i 有一條 -1 的邊

YP們

- p_i 小於 p_{i+1}
- p_i < p_{i+1}
- p_{i+1} - 1 >= p_i
- p_{i+1} 到 p_i 有一條 -1 的邊
- 因為要找最短路，所以 p_{i+1} 一定會去更新 p_i

YP們

- 我們有哪些限制？
- a_i 和 b_i 不超過 d_i
- **a_i 和 b_i 超過 d_i**
- p_i 小於 p_{i+1}

YP們

- a_i 和 b_i 超過 d_i
- p_b - d > p_a
- p_b - (d+1) >= p_a
- p_b 到 p_a 有一條 -(d+1) 的邊
- 因為要找最短路，所以 p_b 一定會去更新 p_a

YP們

- a_i 和 b_i 超過 d_i
- p_b - d > p_a
- p_b - (d+1) >= p_a
- p_b 到 p_a 有一條 -(d+1) 的邊
- 因為要找最短路，所以 p_b 一定會去更新 p_a
- 負環？

YP們

- 什麼時候會產生負環

YP們

- 什麼時候會產生負環
- 可能狀況:
  - p_a 到 p_b 不超過 3
  - p_a 到 p_b 超過 3

YP們

- 什麼時候會產生負環
- 可能狀況:
  - p_a 到 p_b 不超過 3 → p_a + 3 >= p_b
  - p_a 到 p_b 超過 3 → p_b - 4 >= p_a

YP們

- 什麼時候會產生負環
- 可能狀況:
  - p_a 到 p_b 不超過 3 → p_a + 3 >= p_b
  - p_a 到 p_b 超過 3 → p_b - 4 >= p_a
- p_a - 1 >= p_a

YP們

- 什麼時候會產生負環
- 可能狀況:
  - p_a 到 p_b 不超過 3 → p_a + 3 >= p_b
  - p_a 到 p_b 超過 3 → p_b - 4 >= p_a
- p_a - 1 >= p_a
- 負環代表自己要比自己小

YP們

- 什麼時候會產生負環
- 可能狀況:
  - p_a 到 p_b 不超過 3 → p_a + 3 >= p_b
  - p_a 到 p_b 超過 3 → p_b - 4 >= p_a
- p_a - 1 >= p_a
- 負環代表自己要比自己小
- 限制形成矛盾 → 無解！

YP們

- 把所有限制改成 p_a + d >= p_b 的形式
- 對於 a 到 b 建立長度為 d 的邊

YP們

- 把所有限制改成 `p_a + d >= p_b` 的形式
- 對於 `a` 到 `b` 建立長度為 `d` 的邊
- 最短路的結果即是答案
  - `d[N] = inf`

YP們

- 把所有限制改成 `p_a + d >= p_b` 的形式
- 對於 `a` 到 `b` 建立長度為 `d` 的邊
- 最短路的結果即是答案
  - `d[N] = inf` → `N` 可以在無限遠

YP們

- 把所有限制改成 `p_a + d >= p_b` 的形式
- 對於 `a` 到 `b` 建立長度為 `d` 的邊
- 最短路的結果即是答案
  - `d[N] = inf` → `N` 可以在無限遠
- 存在負環代表無解

YP們

- 把所有限制改成 `p_a + d >= p_b` 的形式
- 對於 `a` 到 `b` 建立長度為 `d` 的邊
- 最短路的結果即是答案
  - `d[N] = inf` → `N` 可以在無限遠
- 存在負環代表無解
- **差分約束系統**