



Graph Algorithms 1

Lecture by Ccucumber12
Credit by double, yp155136, baluteshih
2023/05/27

Sprout



Q & A

- 影片看了嗎？
- 有任何問題嗎？

Sprout



課程內容

- Topological Sort
- Minimum Spanning Tree
- Shortest Path

Sprout



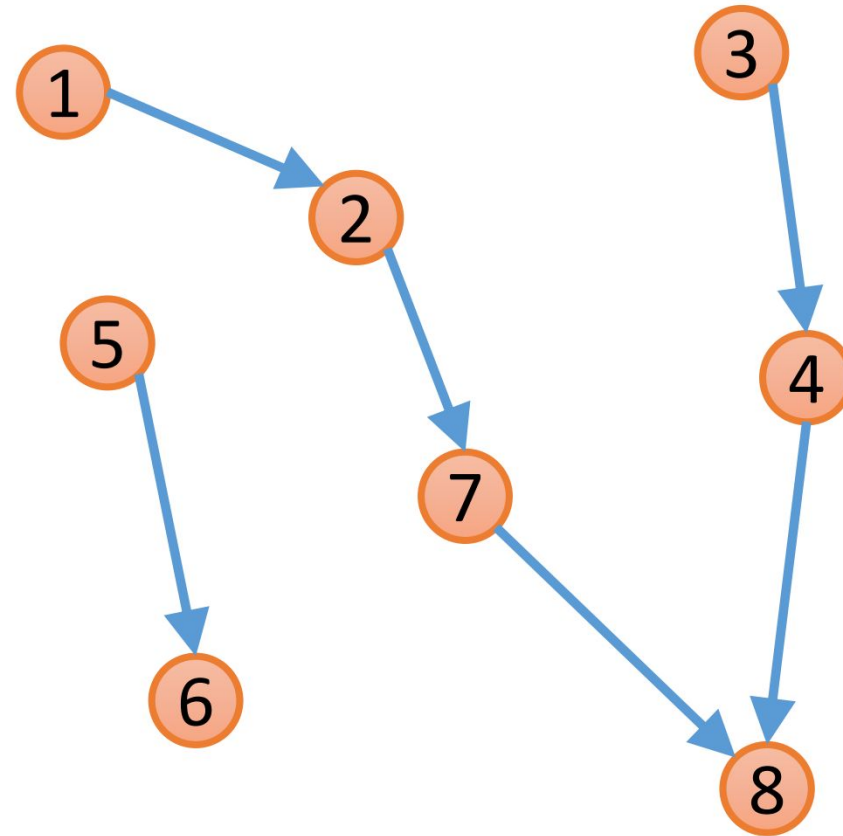
Topological Sort

Sprout



Topological Sort

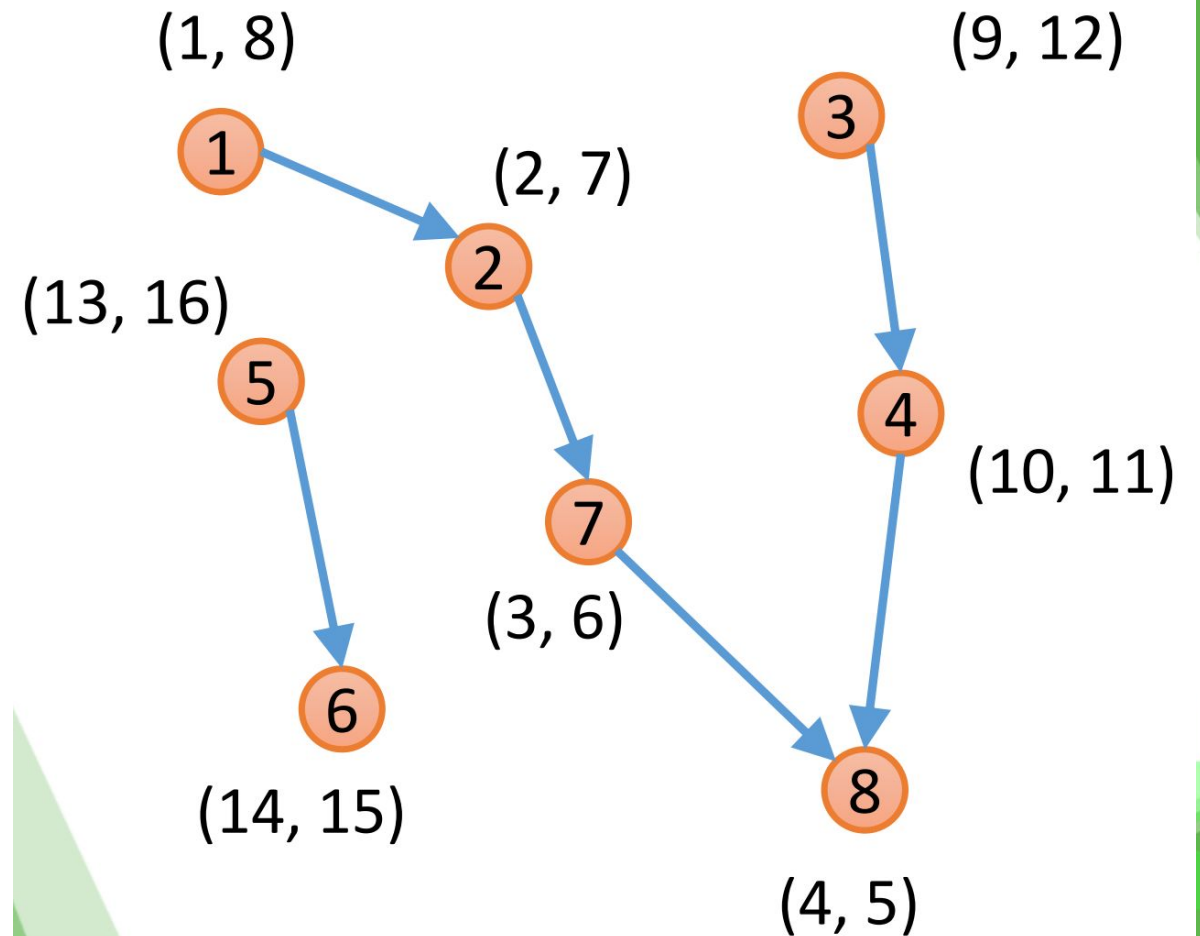
- 方法一: BFS 變形
 - Kahn's Algorithm
 - 一直拔掉入度 0 的點





Topological Sort

- 方法二:DFS 時間戳記
 - 越早離開順序越後面
 - 在 graph2 會用到喔:)





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



Minimum Spanning Tree

Disjoint Set Union

Sprout



Disjoint Set 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

Sprout



Disjoint Set 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

Sprout



Minimum Spanning Tree

Properties

Sprout



Minimum Spanning Tree

- 性質一：一定是生成「樹」？

Sprout



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便更好

Sprout



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？

Sprout



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

Sprout



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？

Sprout



Minimum Spanning Tree

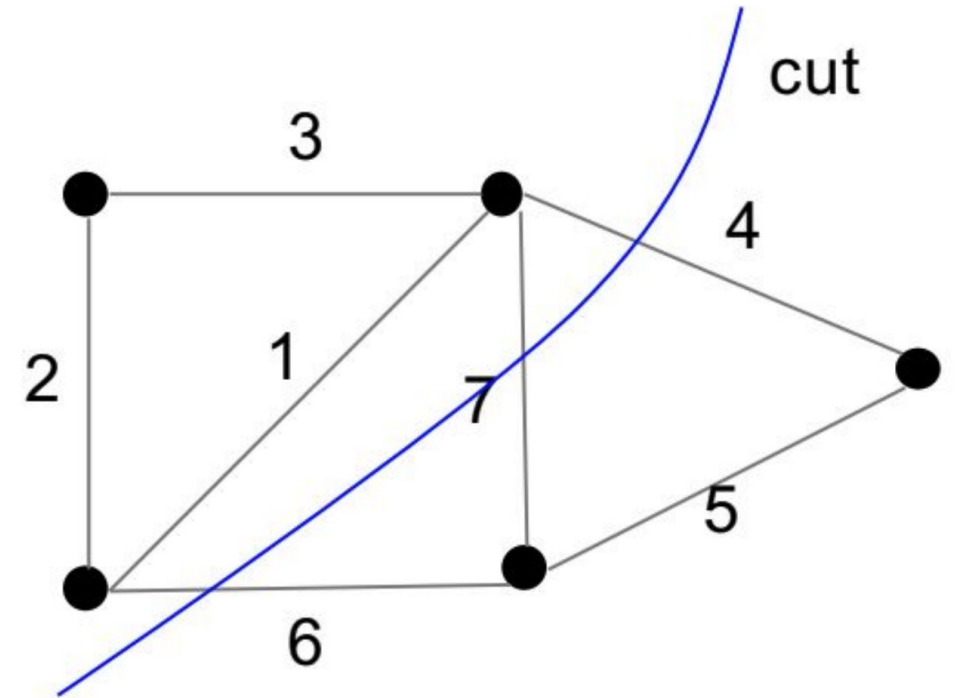
- 性質二: Cycle Property
 - 環上最大邊必然不屬於 MST
- 如果有多條最大邊呢？
 - 環上**唯一**最大邊不屬於 MST
 - 環上最大邊**可以**不屬於 MST
- MST 是否唯一？
 - 如果所有邊權都不一樣那就一定唯一！
 - 但唯一不代表邊權一定都不同

Sprout



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？

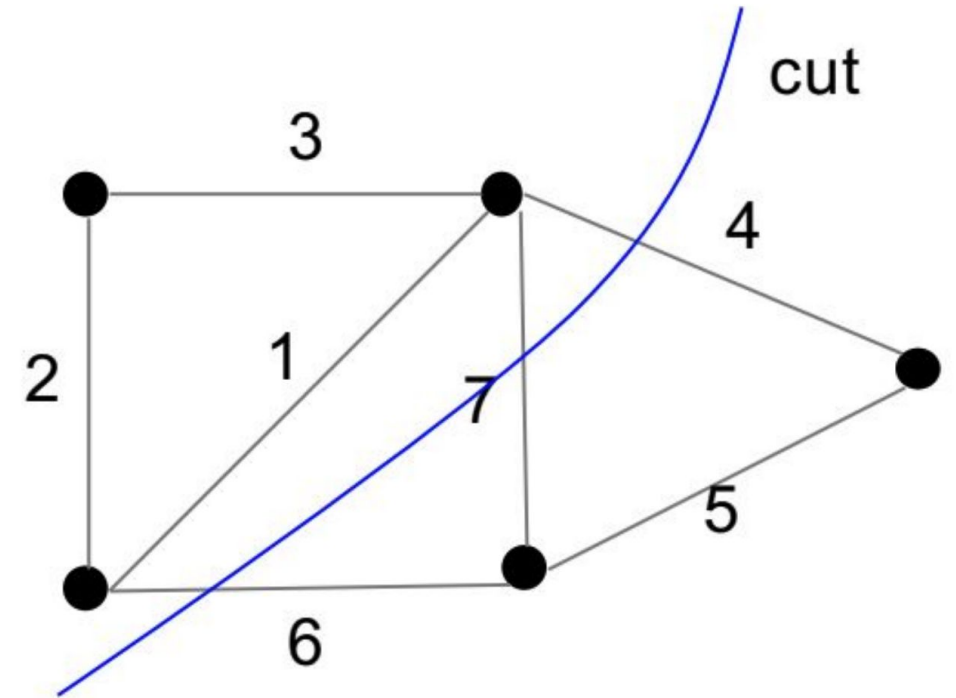


Sprout



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



Sprout



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

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
- 連通？

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
- 連通？
 - Disjoint Set Union !

Sprout



Kruskal

sort the edges by their weight
for e in edges:

```
    if dsu.find(e.u) != dsu.find(e.v):  
        dsu.union(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)

Sprout



Minimum Spanning Tree

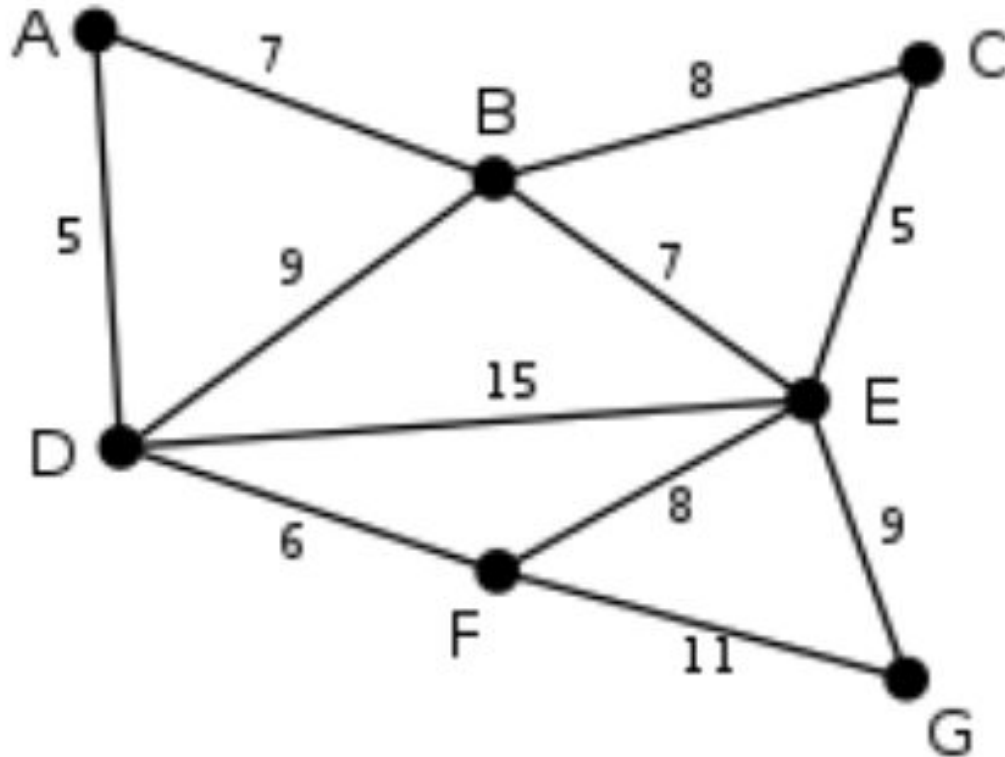
Prim's Algorithm

Sprout



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



Sprout



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
```

rount



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
```

rount



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 $(dis[i], i)$
- 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
```

Sprout



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 !
- $O((E + V) \log E)$

```
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

Sprout



Shortest Path

Floyd-Warshall

Sprout



Floyd-Warshall

- 為什麼是對的？
- 為什麼說這個算法是某種 DP 呢？

Sprout



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 $j = 1..n$

$d[i][j] = \min(d[i][j], d[i][k] + d[k][j]);$

Sprout



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 $j = 1..n$

$d[k][i][j] =$

$\min(d[k-1][i][j], d[k-1][i][k] + d[k-1][k][j]);$

Sprout



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。

Sprout



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

route



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

Sprout



Dijkstra

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

Sprout



Dijkstra

- 起點 s , 還沒拜訪的點集合 U (一開始 $U = V$)
- 目前最佳距離 $d[i]$ (一開始 $d[i] = \text{INF}$)
- 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 q , 更新 $d[q] = \min(d[q], d[p] + w(p, q))$
 - 把 p 移出 U
 - 令 p 為 U 中有最近的人 (最小 $d[i]$ 的 i)

Sprout



Dijkstra

- 起點 s , 還沒拜訪的點集合 U (一開始 $U = V$)
- 目前最佳距離 $d[i]$ (一開始 $d[i] = \text{INF}$)
- 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 q , 更新 $d[q] = \min(d[q], d[p] + w(p, q))$
 - 把 p 移出 U
 - 令 p 為 U 中有最近的人 (最小 $d[i]$ 的 i)
- 答案: $d[i]$ 陣列

Sprout



Dijkstra

為什麼他是對的？

Sprout



Dijkstra

- **貪心**地選擇目前最近的點
 - $d[i]$ 被更新時, i 是 U 裡面被選到的最近點 p

Sprout



Dijkstra

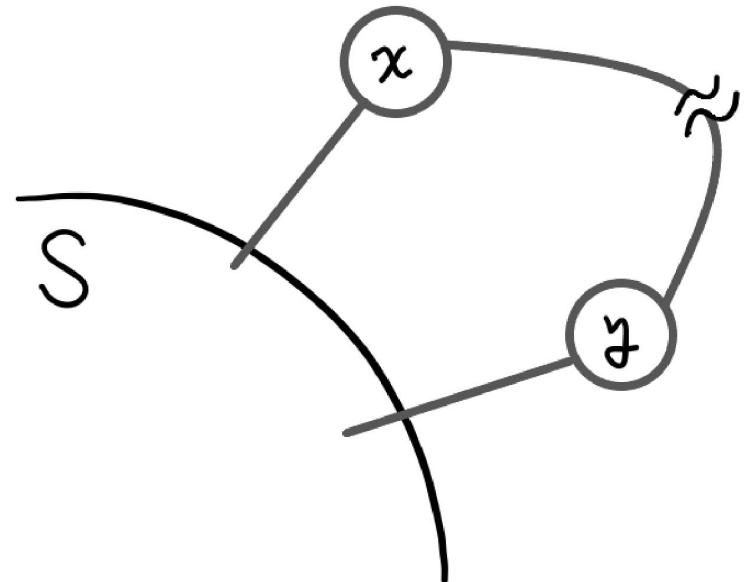
- **貪心**地選擇目前最近的點
 - $d[i]$ 被更新時, i 是 U 裡面被選到的最近點 p
- 為什麼可以貪心？

Sprout



Dijkstra

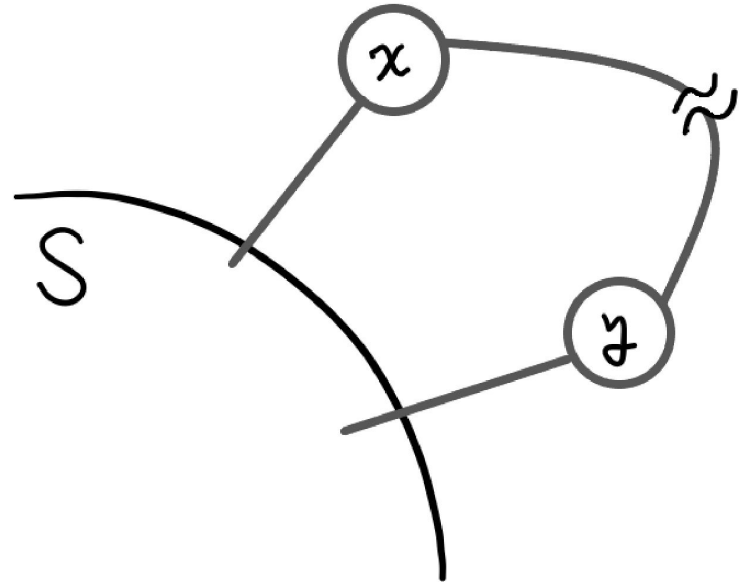
- **貪心**地選擇目前最近的點
 - $d[i]$ 被更新時, i 是 U 裡面被選到的最近點 p
- 為什麼可以貪心？
 - 假設 x 是最近點: $d[x] < d[y]$
 - 如果不選擇走 x 而是走 y
 - 之後繞回來 x 只會更遠
 - 不如一開始就走 x !





Dijkstra

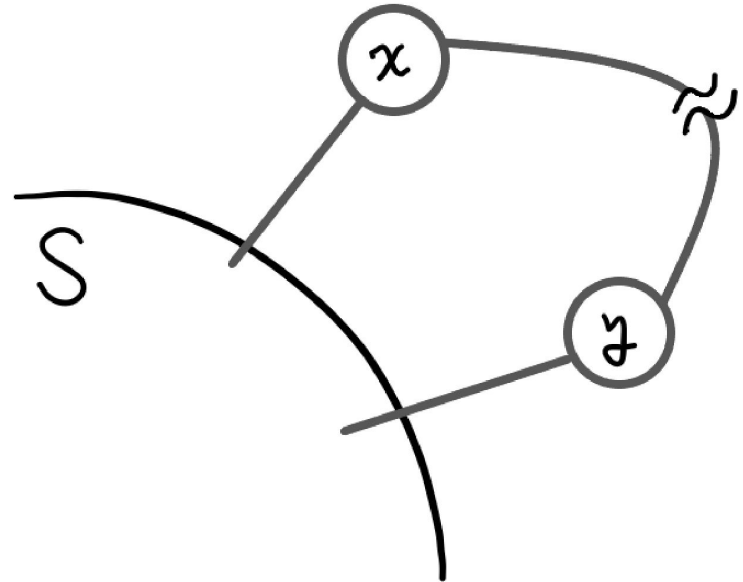
- **貪心**地選擇目前最近的點
 - $d[i]$ 被更新時, i 是 U 裡面被選到的最近點 p
- 為什麼可以貪心？
 - 假設 x 是最近點: $d[x] < d[y]$
 - 如果不選擇走 x 而是走 y
 - **之後繞回來 x 只會更遠 真的嗎？**
 - 不如一開始就走 x !





Dijkstra

- **貪心**地選擇目前最近的點
 - $d[i]$ 被更新時, i 是 U 裡面被選到的最近點 p
- 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 !
- $d[x] < d[y] + w(y, \dots, x)$
 - $w(y, \dots, x) \geq 0$
 - 不能有負邊！





Dijkstra

$O(E+V^2)$
還可以更快嗎？

Sprout



Dijkstra

- 起點 s , 還沒拜訪的點集合 U (一開始 $U = V$)
- 目前最佳距離 $d[i]$ (一開始 $d[i] = \text{INF}$)
- 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 q , 更新 $d[q] = \min(d[q], d[p] + w(p, q))$
 - 把 p 移出 U
 - 令 p 為 U 中有最近的人 (最小 $d[i]$ 的 i)
- 答案: $d[i]$ 陣列

Sprout



Dijkstra

- 起點 s , 還沒拜訪的點集合 U (一開始 $U = V$)
- 目前最佳距離 $d[i]$ (一開始 $d[i] = \text{INF}$)
- 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 q , 更新 $d[q] = \min(d[q], d[p] + w(p, q))$
 - 把 p 移出 U
 - **令 p 為 U 中有最近的人 (最小 $d[i]$ 的 i)**
- 答案: $d[i]$ 陣列

Sprout



Dijkstra

- 從一個集合裡面找出最小的數字

Sprout



Dijkstra

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

Sprout



Dijkstra

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

Sprout



Dijkstra

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

Sprout



Dijkstra

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

Sprout



Dijkstra

怎麼找出最短路徑本身？

Sprout



最短路徑樹

- 每一個最短路徑一定都是簡單路徑

Sprout



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

Sprout



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！

Sprout



Shortest Path

Bellman-Ford

Sprout



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

- 設 $d[i]$ 是從起點走到點 i 的最短距離

Sprout



Bellman-Ford

- 設 $d[i]$ 是從起點走到點 i 的最短距離
- 根據定義： $d[j] \leq d[i] + w(i, j)$, 否則 $d[j]$ 就不是最短距離

Sprout



Bellman-Ford

- 設 $d[i]$ 是從起點走到點 i 的最短距離
- 根據定義： $d[j] \leq d[i] + w(i, j)$, 否則 $d[j]$ 就不是最短距離
- 小性質：
 - 對於所有的 (i, j) 都滿足 $d[j] \leq d[i] + w(i, j)$
 - 且每個 j 都存在 $d[j] = d[i] + w(i, j)$
 - 那麼 $d[]$ 就是最短距離陣列

Sprout



Bellman-Ford

- 設 $d[i]$ 是從起點走到點 i 的最短距離
- 根據定義： $d[j] \leq d[i] + w(i, j)$, 否則 $d[j]$ 就不是最短距離
- 小性質：
 - 對於所有的 (i, j) 都滿足 $d[j] \leq d[i] + w(i, j)$
 - 且每個 j 都存在 $d[j] = d[i] + w(i, j)$
 - 那麼 $d[]$ 就是最短距離陣列
- 想法：
 - 對於 (i, j) 滿足 $d[j] > d[i] + w(i, j)$
 - 把 $d[j]$ 更新成 $d[i] + w(i, j)$

Sprout



Bellman-Ford

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

Sprout



Bellman-Ford

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

Sprout



Bellman-Ford

算法

- 初始化所有 $\text{dis}(u)$ 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



Bellman-Ford

還記得為什麼是對的嗎？

Sprout



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

Sprout



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

Sprout



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

Sprout



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 > 所有點！

Sprout



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 > 所有點！

Sprout



Bellman-Ford

- 如果圖存在**負環**

Sprout



Bellman-Ford

- 如果圖存在**負環**
- 可以在負環上面瘋狂打轉

Sprout



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

Sprout



Shortest Path

Negative Cycle

Sprout



負環

- 什麼叫做處理負環呢？

Sprout



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？

Sprout



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

Sprout



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
 - ...?

Sprout



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

Sprout



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...

Sprout



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！**

Sprout



負環

- Floyd-Warshall 呢？

Sprout



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 $j = 1..n$

$d[i][j] = \min(d[i][j], d[i][k] + d[k][j]);$

Sprout



負環

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

Sprout



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

Sprout



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: $d[i][i] < 0$!

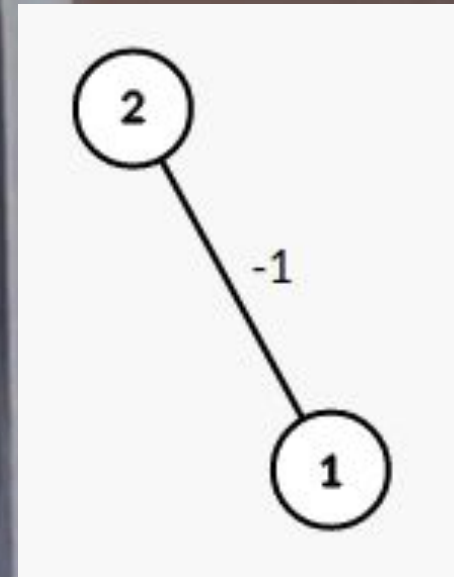
Sprout



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？

Sprout

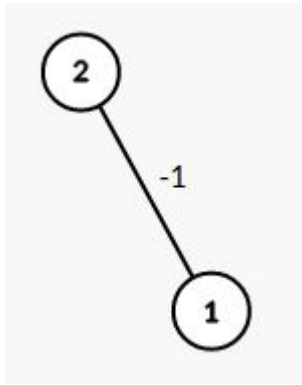


請問,這是負環嗎?



負環

- 無向圖？



- 這...是不是負環啊？

Sprout



負環

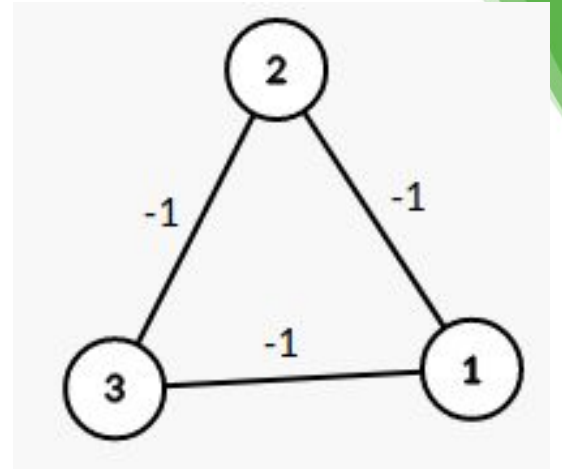
- 無向圖的負邊, Floyd-Warshall, 都會判斷成負環。

Sprout



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



Sprout



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

Sprout



Shortest Path

Applications

Sprout



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？
- $N \leq 2 \cdot 10^5$, $M \leq 5 \cdot 10^5$, $Q \leq 10^5$

Sprout



2017認證考 – 江神與他的小火車

- 如果有用新加進來的邊 $a \rightarrow b$

Sprout



2017認證考 – 江神與他的小火車

- 如果有用新加進來的邊 $a \rightarrow b \dots\dots$
- 路徑可以被拆解成 $1 \rightarrow a + a \rightarrow b + b \rightarrow N$

Sprout



2017認證考 – 江神與他的小火車

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

Sprout



2017認證考 – 江神與他的小火車

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

Sprout



2017認證考 – 江神與他的小火車

- 如果有用新加進來的邊 $a \rightarrow b$
- 路徑可以被拆解成 $1 \rightarrow a + a \rightarrow b + b \rightarrow N$
- 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？

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。
- $N, M \leq 1000$

Sprout



YP們

- 我們有哪些限制？

Sprout



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}

Sprout



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}

Sprout



YP們

- 我們有哪些限制？
- a_i 和 b_i 不超過 d_i
- $p_a + d \geq p_b$

Sprout



YP們

- 我們有哪些限制？
- a_i 和 b_i 不超過 d_i
- $p_a + d \geq p_b$
- 所有人在 θ 一定是合法解

Sprout



YP們

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

Sprout



YP們

- 但 p_1 和 p_N 可以最大離多遠呢？

Sprout



YP們

- 但 p_1 和 p_N 可以最大離多遠呢？
- 可能一
 - $p_1 + d_1 \geq p_N$

Sprout



YP們

- 但 p_1 和 p_N 可以最大離多遠呢？
- 可能一
 - $p_1 + d_1 \geq p_N$
- 可能二
 - $p_1 + d_2 \geq p_2$
 - $p_2 + d_3 \geq p_N$
 - $\rightarrow p_1 + (d_2 + d_3) \geq p_N$

Sprout



YP們

- 但 p_1 和 p_N 可以最大離多遠呢？
- 可能一
 - $p_1 + d_1 \geq p_N$
- 可能二
 - $p_1 + d_2 \geq p_2$
 - $p_2 + d_3 \geq p_N$
 - $\rightarrow p_1 + (d_2 + d_3) \geq p_N$
- 要取 $\min(d_1, d_2 + d_3)$

Sprout



YP們

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

Sprout



YP們

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

Sprout



YP們

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

Sprout



YP們

- 但 p_1 和 p_N 可以最大離多遠呢？
- 可能一
 - $p_1 + d_1 \geq p_N \rightarrow 1$ 到 N 建一條長度 d_1 的邊
- 可能二
 - $p_1 + d_2 \geq p_2 \rightarrow 1$ 到 2 建一條長度 d_2 的邊
 - $p_2 + d_3 \geq p_N \rightarrow 2$ 到 N 建一條長度 d_3 的邊
 - $\rightarrow p_1 + (d_2 + d_3) \geq p_N$
- 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 $\min$!

Sprout



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}

Sprout



YP們

- p_i 小於 p_{i+1}
- $p_i < p_{i+1}$

Sprout



YP們

- p_i 小於 p_{i+1}
- $p_i < p_{i+1}$
- $p_{i+1} - 1 \geq p_i$

Sprout



YP們

- p_i 小於 p_{i+1}
- $p_i < p_{i+1}$
- $p_{i+1} - 1 \geq p_i$
- p_{i+1} 到 p_i 有一條 -1 的邊

Sprout



YP們

- p_i 小於 p_{i+1}
- $p_i < p_{i+1}$
- $p_{i+1} - 1 \geq p_i$
- 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

Sprout



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}

Sprout



YP們

- a_i 和 b_i 超過 d_i
- $p_b - d > p_a$
- $p_b - (d+1) \geq p_a$
- 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

Sprout



YP們

- a_i 和 b_i 超過 d_i
- $p_b - d > p_a$
- $p_b - (d+1) \geq p_a$
- 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?

Sprout



YP們

- 什麼時候會產生負環

Sprout



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

Sprout



YP們

- 什麼時候會產生負環
- 可能狀況：
 - p_a 到 p_b 不超過 3 $\rightarrow p_a + 3 \geq p_b$
 - p_a 到 p_b 超過 3 $\rightarrow p_b - 4 \geq p_a$

Sprout



YP們

- 什麼時候會產生負環
- 可能狀況：
 - p_a 到 p_b 不超過 3 $\rightarrow p_a + 3 \geq p_b$
 - p_a 到 p_b 超過 3 $\rightarrow p_b - 4 \geq p_a$
- $p_a - 1 \geq p_a$

Sprout



YP們

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

Sprout



YP們

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

Sprout



YP們

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

Sprout



YP們

- 把所有限制改成 $p_a + d \geq p_b$ 的形式
- 對於 a 到 b 建立長度為 d 的邊
- 最短路的结果即是答案
 - $d[N] = \text{inf}$

Sprout



YP們

- 把所有限制改成 $p_a + d \geq p_b$ 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
 - $d[N] = \text{inf} \rightarrow N$ 可以在無限遠

Sprout



YP們

- 把所有限制改成 $p_a + d \geq p_b$ 的形式
- 對於 a 到 b 建立長度為 d 的邊
- 最短路的结果即是答案
 - $d[N] = \text{inf} \rightarrow N$ 可以在無限遠
- 存在負環代表無解

Sprout



YP們

- 把所有限制改成 $p_a + d \geq p_b$ 的形式
- 對於 a 到 b 建立長度為 d 的邊
- 最短路的结果即是答案
 - $d[N] = \text{inf} \rightarrow N$ 可以在無限遠
- 存在負環代表無解
- **差分約束系統**

Sprout



Sli.do 回覆~~~

Sprout