Data Sctructure

謝嘉穎

TODO TODO 2025

講師介紹

- 謝嘉穎

講師介紹

- 謝嘉穎
- BLAME (blameazu)

講師介紹

- 謝嘉穎
- BLAME (blameazu)
- 因為想學資料結構，所以來當資料結構講師

前言

今天這堂課是教資料結構，雖然課堂上只會講概念，並不會有時間給大家實作。

前言

今天這堂課是教資料結構，雖然課堂上只會講概念，並不會有時間給大家實作。

但希望大家可以先把概念記下來，閒暇時間再自行練習實作。

前言

今天這堂課是教資料結構，雖然課堂上只會講概念，並不會有時間給大家實作。

但希望大家可以先把概念記下來，閒暇時間再自行練習實作。

然後講師應該塞了很多東西進來，~~聽到後面聽不懂應該是正常的~~

前言

今天這堂課是教資料結構，雖然課堂上只會講概念，並不會有時間給大家實作。

但希望大家可以先把概念記下來，閒暇時間再自行練習實作。

然後講師應該塞了很多東西進來，~~聽到後面聽不懂應該是正常的~~

假如大家覺得太簡單/難，可以自己去練題目

前言

今天這堂課是教資料結構，雖然課堂上只會講概念，並不會有時間給大家實作。

但希望大家可以先把概念記下來，閒暇時間再自行練習實作。

然後講師應該塞了很多東西進來，~~聽到後面聽不懂應該是正常的~~

假如大家覺得太簡單/難，可以自己去練題目

但盡量跟上課內容是更好的! omob

上課大綱

1 基礎資料結構

2 進階資料結構

3 超進階資料結構

基礎資料結構

基礎資料結構 – 大綱

1 基礎資料結構

- 資料結構介紹
- Struct & Class
- 並查集 (Disjoint Set Union)
- 在線與離線 (Online or Offline)
- 根號想法 (Sqrt Decomposition)
- 線段樹 (Segment Tree)
- 樹狀樹組 (Binary Index Tree)
- 稀疏表
- 珂朵莉樹

基礎資料結構 – 資料結構介紹

1 基礎資料結構

- 資料結構介紹
- Struct & Class
- 並查集 (Disjoint Set Union)
- 在線與離線 (Online or Offline)
- 根號想法 (Sqrt Decomposition)
- 線段樹 (Segment Tree)
- 樹狀樹組 (Binary Index Tree)
- 稀疏表
- 珂朵莉樹

資料結構介紹 – 前言

什麼是資料結構?

資料結構介紹 – 前言

什麼是資料結構?

在電腦科學中，資料結構（英語：data structure）是電腦中儲存、組織資料的方式。by wiki

資料結構介紹 – 前言

什麼是資料結構?

在電腦科學中，資料結構（英語：data structure）是電腦中儲存、組織資料的方式。by wiki

能將資料結構當成一個工具盒

資料結構介紹 – 前言

什麼是資料結構?

在電腦科學中，資料結構（英語：data structure）是電腦中儲存、組織資料的方式。by wiki

能將資料結構當成一個工具盒

在特殊的情況下找到對應的工具盒是非常重要的

資料結構介紹 – Example

例題

有一個隊伍

Q 筆操作，共有 2 種操作

1. 每次從隊伍後端加入編號為 A_i 的人
2. 每次從隊伍前端拔除一個人，並且需要輸出最前端的人是誰

資料結構介紹 – Example

例題

有一個隊伍

Q 筆操作，共有 2 種操作

1. 每次從隊伍後端加入編號為 A_i 的人
2. 每次從隊伍前端拔除一個人，並且需要輸出最前端的人是誰

假如你有認真上 STL 課的話

資料結構介紹 – Example

例題

有一個隊伍
Q 筆操作，共有 2 種操作

1. 每次從隊伍後端加入編號為 A_i 的人
2. 每次從隊伍前端拔除一個人，並且需要輸出最前端的人是誰

假如你有認真上 STL 課的話 queue!

資料結構介紹 – Example

例題

有一個隊伍

Q 筆操作，共有 2 種操作

1. 每次從隊伍後端加入編號為 A_i 的人
2. 每次從隊伍前端拔除一個人，並且需要輸出最前端的人是誰

假如你有認真上 STL 課的話 queue!

```
1 queue<int> qq;
2 qq.push(a); // add
3 qq.pop(); // remove
```

資料結構介紹 – Example

例題

有一個隊伍

Q 筆操作，共有 2 種操作

1. 每次從隊伍後端加入編號為 A_i 的人
2. 每次從隊伍前端拔除一個人，並且需要輸出最前端的人是誰

假如你有認真上 STL 課的話 queue!

```
1 queue<int> qq;
2 qq.push(a); // add
3 qq.pop(); // remove
```

而 push 及 pop 就是我們與這個工具盒的互動

資料結構介紹 – 總結

也就是說一個能儲存資料、整理資料、拿取資料的工具包

我們其實就可以簡單的稱為 **資料結構**

資料結構介紹 – 總結

也就是說一個能儲存資料、整理資料、拿取資料的工具包

我們其實就可以簡單的稱為 **資料結構**

而資料結構之間有些並無好壞之分

資料結構介紹 – 總結

也就是說一個能儲存資料、整理資料、拿取資料的工具包

我們其實就可以簡單的稱為 **資料結構**

而資料結構之間有些並無好壞之分

像是要用到 First In First Out 的題目，你用 queue 解

Last in First Out 的題目，你用 stack 解

資料結構介紹 – 總結

也就是說一個能儲存資料、整理資料、拿取資料的工具包

我們其實就可以簡單的稱為 **資料結構**

而資料結構之間有些並無好壞之分

像是要用到 First In First Out 的題目，你用 queue 解

Last in First Out 的題目，你用 stack 解

針對這些性質上的差異，我們選取一個 " **適當**" 的資料結構

基礎資料結構 – Struct & Class

1 基礎資料結構

- 資料結構介紹
- Struct & Class
- 並查集 (Disjoint Set Union)
- 在線與離線 (Online or Offline)
- 根號想法 (Sqrt Decomposition)
- 線段樹 (Segment Tree)
- 樹狀樹組 (Binary Index Tree)
- 稀疏表
- 珂朵莉樹

Struct & Class – Struct 介紹

1 基礎資料結構

- Struct & Class
  - Struct 介紹
  - Struct 語法
  - Class 介紹

Struct & Class – Struct 介紹

Struct 是 C++ 中的一個語法

中文應該叫做結構 (Structure)

Struct & Class – Struct 介紹

Struct 是 C++ 中的一個語法

中文應該叫做結構 (Structure)

是一種把一大堆變數、資料丟在一起

使得方便管理他們的地方

Struct & Class – Struct 介紹

也可以想像 Struct 是一個紙皮箱

Struct & Class – Struct 介紹

也可以想像 Struct 是一個紙皮箱

你可以把很多個物品丟進去裡面，最後用膠帶封起來

Struct & Class – Struct 介紹

也可以想像 Struct 是一個紙皮箱

你可以把很多個物品丟進去裡面，最後用膠帶封起來

而你又可以想像，那個紙皮箱上有兩個小孔，分別是 Input 跟 Output

Struct & Class – Struct 介紹

也可以想像 Struct 是一個紙皮箱

你可以把很多個物品丟進去裡面，最後用膠帶封起來

而你又可以想像，那個紙皮箱上有兩個小孔，分別是 Input 跟 Output

所以你唯一溝通整個紙箱子的通道就只有那兩個孔，可以丟東西進去 or 取東西出來

Struct & Class – Struct 介紹

也可以想像 Struct 是一個紙皮箱

你可以把很多個物品丟進去裡面，最後用膠帶封起來

而你又可以想像，那個紙皮箱上有兩個小孔，分別是 Input 跟 Output

所以你唯一溝通整個紙箱子的通道就只有那兩個孔，可以丟東西進去 or 取東西出來

~~當然不排除有把箱子拆開把東西直接拿出來的功能~~

Struct & Class – Struct 介紹

蛤? 跟把變數宣告在外面有什麼差別?

Struct & Class – Struct 介紹

蛤? 跟把變數宣告在外面有什麼差別?

你可以想像箱子裡有一個自動運轉的機器

Struct & Class – Struct 介紹

蛤? 跟把變數宣告在外面有什麼差別?

你可以想像箱子裡有一個自動運轉的機器

當你把東西 Ex. 統一布丁、奶茶丟進去，機器會自動將物品按照某種特定的方式整理好

Struct & Class – Struct 介紹

蛤? 跟把變數宣告在外面有什麼差別?

你可以想像箱子裡有一個自動運轉的機器

當你把東西 Ex. 統一布丁、奶茶丟進去，機器會自動將物品按照某種特定的方式整理好

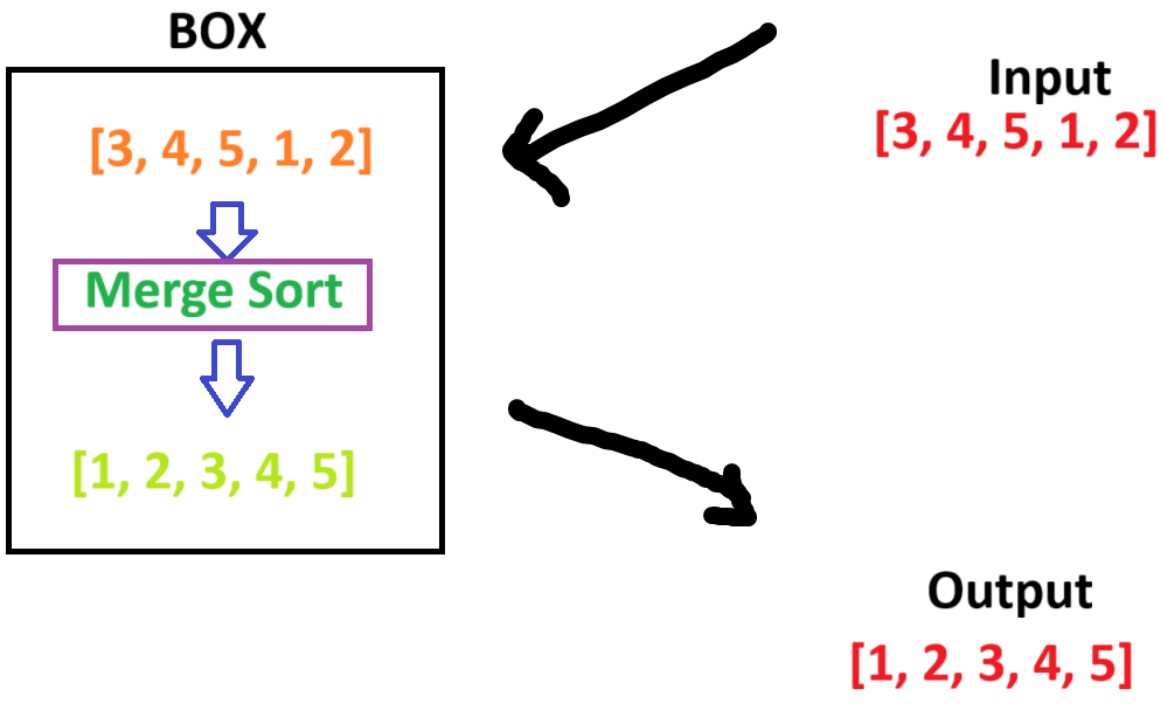
所以當你在箱子外面要把東西取出來的時候，會很方便 (畢竟都幫你整理好了)

Struct & Class – Struct 介紹

舉例來講，有一個 Struct 他在內部會將資料做 Merge Sort 的整理。

Struct & Class – Struct 介紹

舉例來講，有一個 Struct 他在內部會將資料做 Merge Sort 的整理。



Struct & Class – Struct 語法

1 基礎資料結構

- Struct & Class
  - Struct 介紹
  - Struct 語法
  - Class 介紹

Struct & Class – Struct 語法

```
1  struct name {
2    // 變數
3    int b = 0;
4    name(/*可以放參數在這裡面*/) {
5      // 初始化東西可以放這裡
6    }
7    // 底下可以寫 function
8    void f() {
9      cout << "Hello_World!\n";
10   }
11 };
```

Struct & Class – Struct 語法

```
1    name a(/*可以放參數在這裡面*/);
2    // a 是變數名稱
3    a.f(/*參數*/); // 呼叫 f function
4    a.b += 2; // 可以進去修改參數 or 拿出來
```

Struct & Class – Class 介紹

1 基礎資料結構

- Struct & Class
  - Struct 介紹
  - Struct 語法
  - Class 介紹

Struct & Class – Class 介紹

Class

Struct & Class – Class 介紹

Class

我也不會，沒用到過。

基礎資料結構 – 並查集 (Disjoint Set Union)

1 基礎資料結構

- 資料結構介紹
- Struct & Class
- 並查集 (Disjoint Set Union)
- 在線與離線 (Online or Offline)
- 根號想法 (Sqrt Decomposition)
- 線段樹 (Segment Tree)
- 樹狀樹組 (Binary Index Tree)
- 稀疏表
- 珂朵莉樹

並查集 (Disjoint Set Union) – 大綱

1 基礎資料結構

- 並查集 (Disjoint Set Union)
  - 一些基礎圖論的東西
  - DSU 介紹
  - DSU 實作
  - 優化
  - 複雜度分析
  - 題目

並查集 (Disjoint Set Union) – 一些基礎圖論的東西

1 基礎資料結構

- 並查集 (Disjoint Set Union)
  - 一些基礎圖論的東西
  - DSU 介紹
  - DSU 實作
  - 優化
  - 複雜度分析
  - 題目

並查集 (Disjoint Set Union) – 一些基礎圖論的東西

什麼是圖論 (Graph)

並查集 (Disjoint Set Union) – 一些基礎圖論的東西

什麼是圖論 (Graph)

圖論在 Computer Science 中是一門重要的主題

並查集 (Disjoint Set Union) – 一些基礎圖論的東西

什麼是圖論 (Graph)

圖論在 Computer Science 中是一門重要的主題

圖論所主要研究的主題是圖

並查集 (Disjoint Set Union) – 一些基礎圖論的東西

什麼是圖論 (Graph)

圖論在 Computer Science 中是一門重要的主題

圖論所主要研究的主題是圖

圖是由若干個節點，以及連接相鄰兩點的邊所構成的圖形。

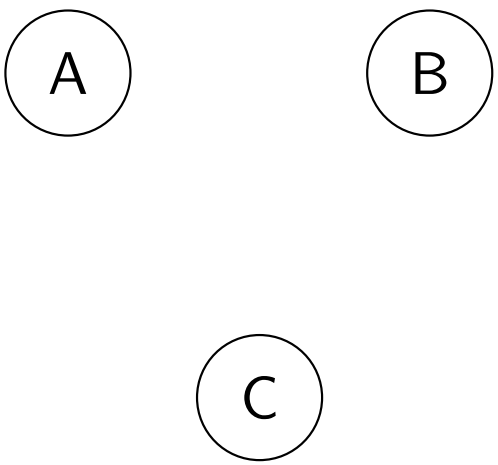
並查集 (Disjoint Set Union) – 一些基礎圖論的東西

節點 (Nodes, Vertexs)

並查集 (Disjoint Set Union) – 一些基礎圖論的東西

節點 (Nodes, Vertexs)

這張圖就是有三個節點，A、B、C



並查集 (Disjoint Set Union) – 一些基礎圖論的東西

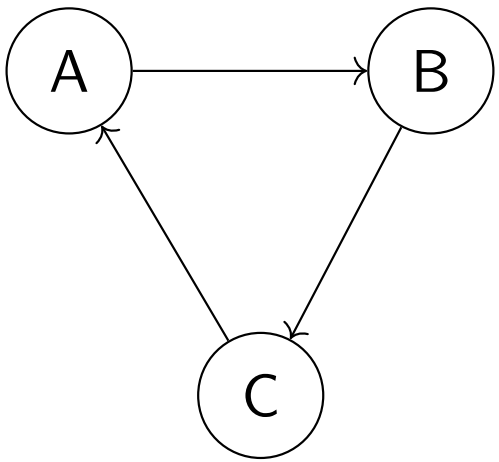
邊 (Edges)

並查集 (Disjoint Set Union) – 一些基礎圖論的東西

邊 (Edges)

這張圖就是有三個節點，A、B、C

三條邊 $A \to B$、$B \to C$、$C \to A$

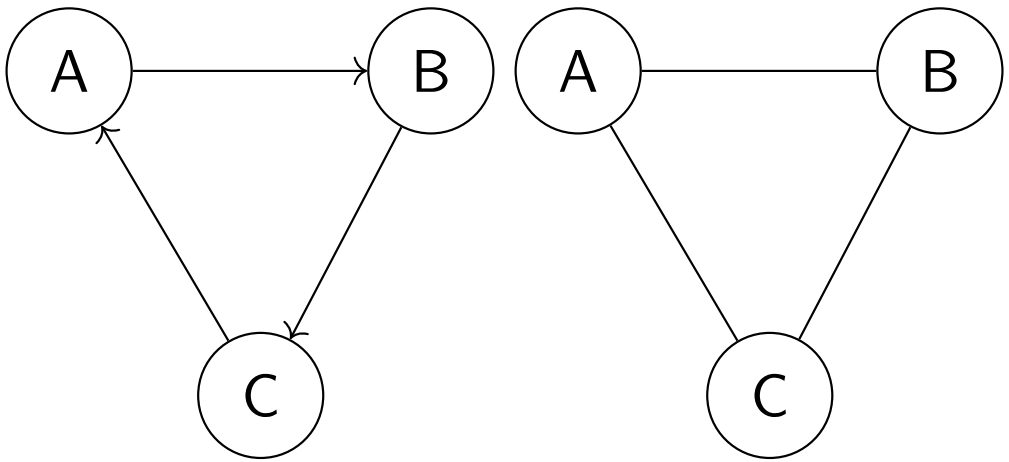


並查集 (Disjoint Set Union) – 一些基礎圖論的東西

邊 (Edges)

這張圖就是有三個節點，A、B、C

三條邊 $A \to B$、$B \to C$、$C \to A$



左邊跟右邊這張圖是有向邊與無向邊的差異

並查集 (Disjoint Set Union) – 一些基礎圖論的東西

連通塊 (Connected Component)

並查集 (Disjoint Set Union) – 一些基礎圖論的東西

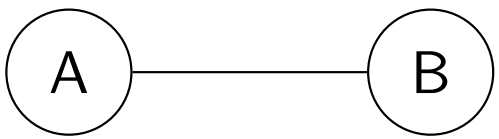
連通塊 (Connected Component)

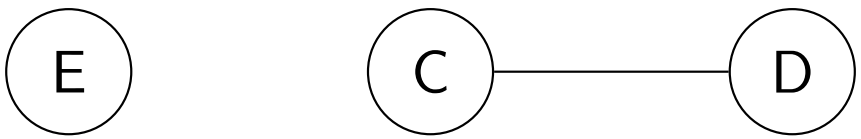
當二節點 A、B，且 A 節點與 B 節點可以透過邊互相到達 (不管邊的方向)，則 A、B 視為在同一個連通塊當中

並查集 (Disjoint Set Union) – 一些基礎圖論的東西

連通塊 (Connected Component)

當二節點 A、B，且 A 節點與 B 節點可以透過邊互相到達 (不管邊的方向)，則 A、B 視為在同一個連通塊當中





$\{A, B\}$ 為一連通塊，$\{C, D\}$ 為一連通塊，$\{E\}$ 為一連通塊

並查集 (Disjoint Set Union) – DSU 介紹

1 基礎資料結構

- 並查集 (Disjoint Set Union)
  - 一些基礎圖論的東西
  - DSU 介紹
  - DSU 實作
  - 優化
  - 複雜度分析
  - 題目

並査集 (Disjoint Set Union) – DSU 介紹

Disjoint Set Union (並査集)

並查集 (Disjoint Set Union) – DSU 介紹

Disjoint Set Union (並查集) ，通常我們都簡稱 DSU

並查集 (Disjoint Set Union) – DSU 介紹

Disjoint Set Union (並查集) ，通常我們都簡稱 DSU

是一種用來解決圖論上節點連通狀塊的資料結構

並查集 (Disjoint Set Union) – DSU 介紹

Disjoint Set Union (並查集) ，通常我們都簡稱 DSU

是一種用來解決圖論上節點連通狀塊的資料結構

雖然通常都是拿來合併集合之類的，但大部分都可以抽象化為圖論去思考

並查集 (Disjoint Set Union) – DSU 介紹

Disjoint Set Union (並查集) ，通常我們都簡稱 DSU

是一種用來解決圖論上節點連通狀塊的資料結構

雖然通常都是拿來合併集合之類的，但大部分都可以抽象化為圖論去思考

DSU 支援的操作

並查集 (Disjoint Set Union) – DSU 介紹

Disjoint Set Union (並查集) ，通常我們都簡稱 DSU

是一種用來解決圖論上節點連通狀塊的資料結構

雖然通常都是拿來合併集合之類的，但大部分都可以抽象化為圖論去思考

DSU 支援的操作

- 將兩個節點相連

並查集 (Disjoint Set Union) – DSU 介紹

Disjoint Set Union (並查集) ，通常我們都簡稱 DSU

是一種用來解決圖論上節點連通狀塊的資料結構

雖然通常都是拿來合併集合之類的，但大部分都可以抽象化為圖論去思考

DSU 支援的操作

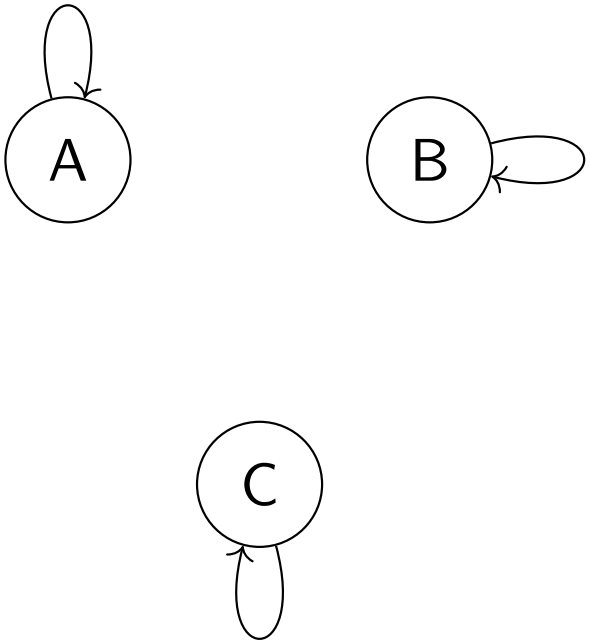
- 將兩個節點相連
- 詢問兩點的連通關係

並查集 (Disjoint Set Union) – DSU 介紹

我們先定義在 DSU 中，所有的節點都有他們的唯一出邊，一開始所有節點的出邊都是指向自己

並查集 (Disjoint Set Union) – DSU 介紹

我們先定義在 DSU 中，所有的節點都有他們的唯一出邊，一開始所有節點的出邊都是指向自己



並查集 (Disjoint Set Union) – DSU 介紹

假設我們今天的圖的每個連通塊都有一個性質

並查集 (Disjoint Set Union) – DSU 介紹

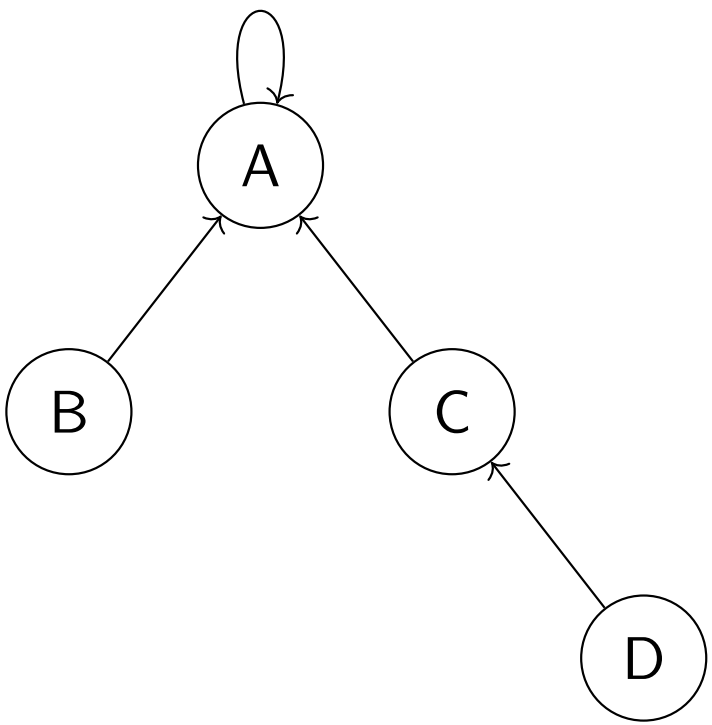
假設我們今天的圖的每個連通塊都有一個性質

每個連通塊皆具有一個唯一的結束點，也就是說不管從哪個節點開始順著邊走，我們最後都將會走到同一節點上，且那個節點的邊指向自己

並查集 (Disjoint Set Union) – DSU 介紹

假設我們今天的圖的每個連通塊都有一個性質

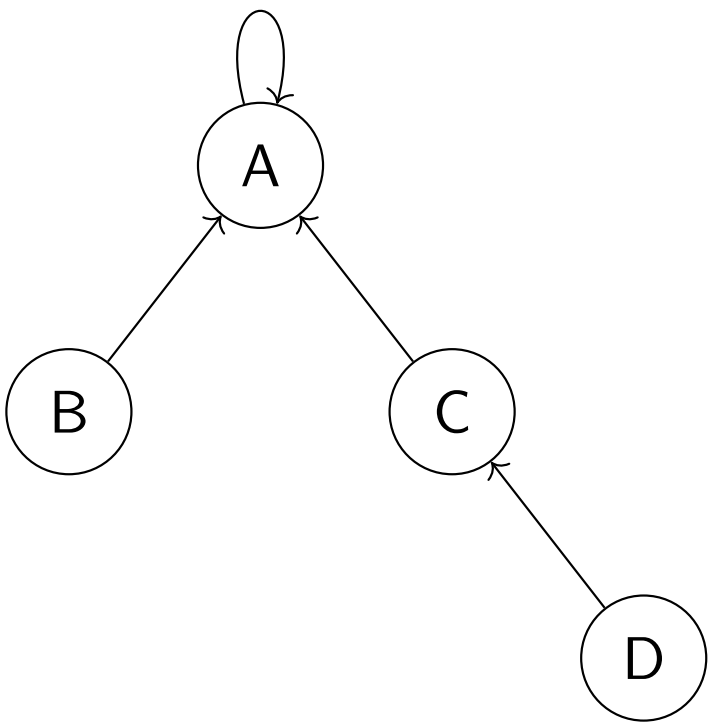
每個連通塊皆具有一個唯一的結束點，也就是說不管從哪個節點開始順著邊走，我們最後都將會走到同一節點上，且那個節點的邊指向自己



並查集 (Disjoint Set Union) – DSU 介紹

假設我們今天的圖的每個連通塊都有一個性質

每個連通塊皆具有一個唯一的結束點，也就是說不管從哪個節點開始順著邊走，我們最後都將會走到同一節點上，且那個節點的邊指向自己



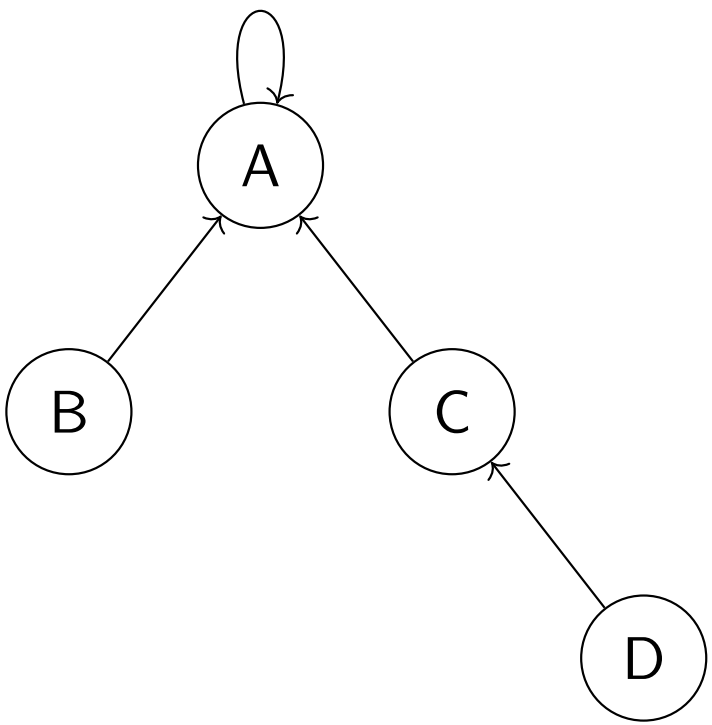
每個點最終都會匯集到點 A

- A

並查集 (Disjoint Set Union) – DSU 介紹

假設我們今天的圖的每個連通塊都有一個性質

每個連通塊皆具有一個唯一的結束點，也就是說不管從哪個節點開始順著邊走，我們最後都將會走到同一節點上，且那個節點的邊指向自己



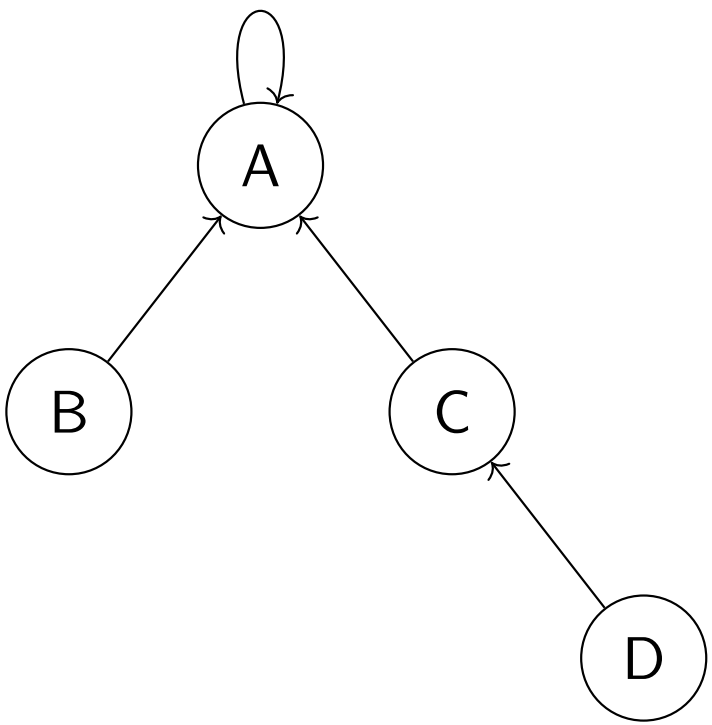
每個點最終都會匯集到點 A

- A
- $B \to A$

並查集 (Disjoint Set Union) – DSU 介紹

假設我們今天的圖的每個連通塊都有一個性質

每個連通塊皆具有一個唯一的結束點，也就是說不管從哪個節點開始順著邊走，我們最後都將會走到同一節點上，且那個節點的邊指向自己



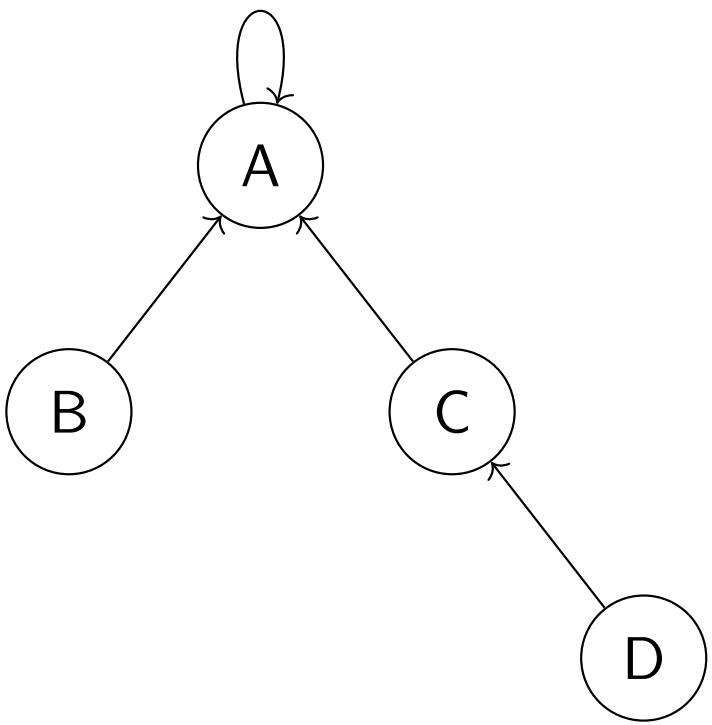
每個點最終都會匯集到點 A

- A
- $B \to A$
- $C \to A$

並查集 (Disjoint Set Union) – DSU 介紹

假設我們今天的圖的每個連通塊都有一個性質

每個連通塊皆具有一個唯一的結束點，也就是說不管從哪個節點開始順著邊走，我們最後都將會走到同一節點上，且那個節點的邊指向自己



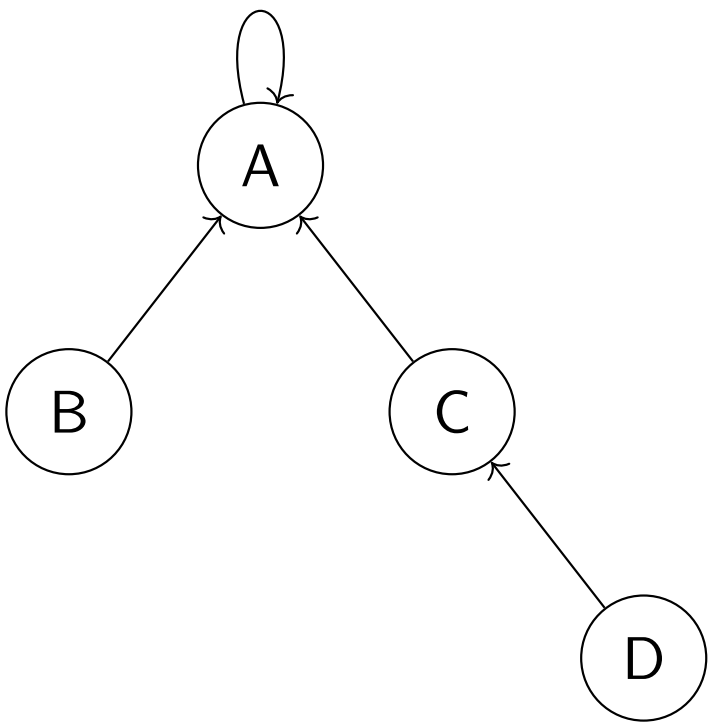
每個點最終都會匯集到點 A

- A
- $B \to A$
- $C \to A$
- $D \to C \to A$

並查集 (Disjoint Set Union) – DSU 介紹

假設我們今天的圖的每個連通塊都有一個性質

每個連通塊皆具有一個唯一的結束點，也就是說不管從哪個節點開始順著邊走，我們最後都將會走到同一節點上，且那個節點的邊指向自己



每個點最終都會匯集到點 A

- A
- $B \to A$
- $C \to A$
- $D \to C \to A$

那麼當我們詢問兩點是否在同一個連通塊，就等價於問他們的結束點是否相同

並查集 (Disjoint Set Union) – DSU 介紹

那麼我們在連接點 A、點 B 時，我們就必須要確保這個性質一直都會存在!

並查集 (Disjoint Set Union) – DSU 介紹

那麼我們在連接點 A、點 B 時，我們就必須要確保這個性質一直都會存在!

那麼我們可以發現，連通塊 A 與連通塊 B 皆具有自己的一個結束點

並查集 (Disjoint Set Union) – DSU 介紹

那麼我們在連接點 A、點 B 時，我們就必須要確保這個性質一直都會存在!

那麼我們可以發現，連通塊 A 與連通塊 B 皆具有自己的一個結束點

但當我們將他們合併成一個大連通塊 C，C 只能具有一個結束點而已

並查集 (Disjoint Set Union) – DSU 介紹

那麼我們在連接點 A、點 B 時，我們就必須要確保這個性質一直都會存在!

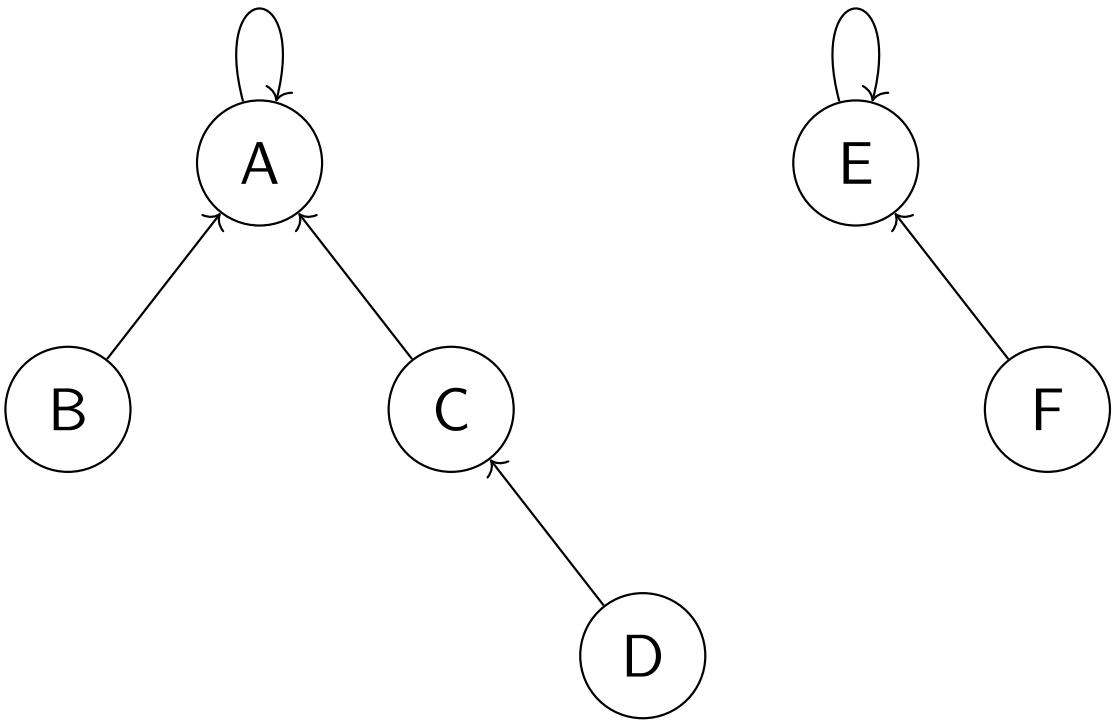
那麼我們可以發現，連通塊 A 與連通塊 B 皆具有自己的一個結束點

但當我們將他們合併成一個大連通塊 C，C 只能具有一個結束點而已

所以我們就將 A 的結束點連接到 B 的結束點即可!

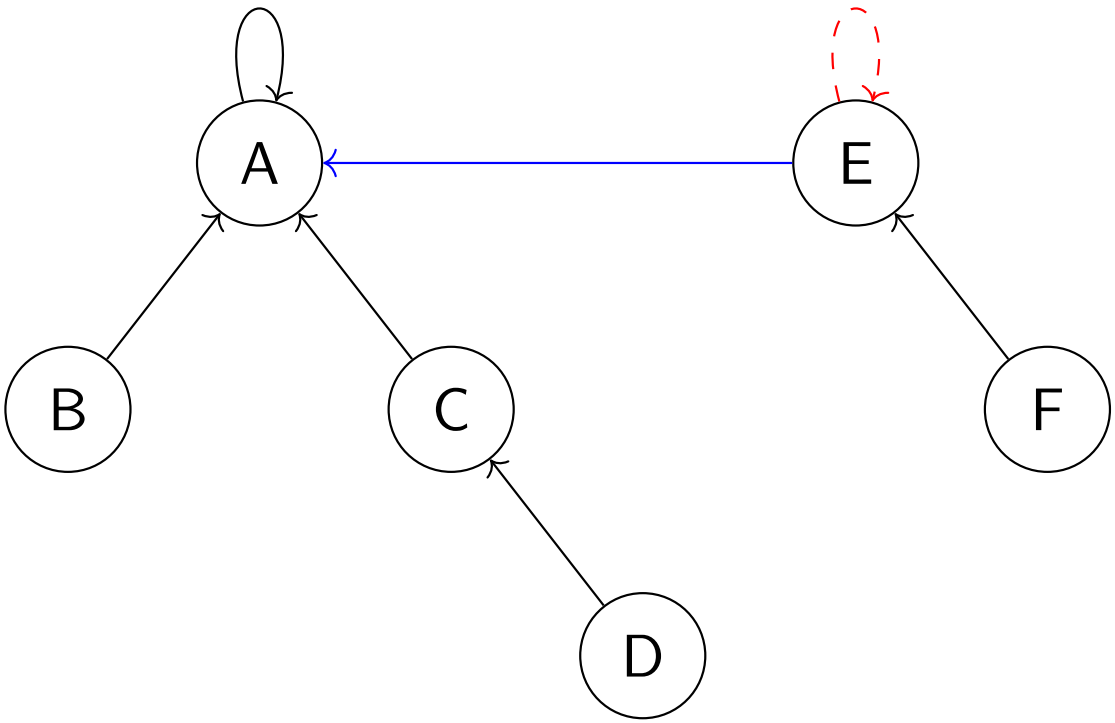
並查集 (Disjoint Set Union) – DSU 介紹

合併連通塊 A 與連通塊 B



並查集 (Disjoint Set Union) – DSU 介紹

合併連通塊 A 與連通塊 B



並查集 (Disjoint Set Union) – DSU 實作

1 基礎資料結構

- 並查集 (Disjoint Set Union)
  - 一些基礎圖論的東西
  - DSU 介紹
  - DSU 實作
  - 優化
  - 複雜度分析
  - 題目

並查集 (Disjoint Set Union) – DSU 實作

初始化圖

```
1 // n: nodes number
2 vector<int> nxt(n+1); // 1-based
3 for(int i = 1; i <= n; i++) {
4   nxt[i] = i; // 自己連到自己
5 }
```

並查集 (Disjoint Set Union) – DSU 實作

詢問 x 點所在的連通塊的結束點

我們通常都用遞迴實作

```
1 int end_point(int x) {
2   if(nxt[x] == x) return x; // 自己是結束點
3   // 當前的結束點是下一個點的結束點
4   return end_point(nxt[x]);
5 }
```

並查集 (Disjoint Set Union) – DSU 實作

連接點 A, B 各自所屬的連通塊

```
1  void merge(int A, int B) {
2    int A_end_point = end_point(A);
3    int B_end_point = end_point(B);
4    // 相同連通塊不需要合併
5    if(A_end_point == B_end_point) return;
6    // 將 A 的結束點連到 B的結束點
7    nxt[A_end_point] = B_end_point;
8  }
```

並查集 (Disjoint Set Union) – DSU 實作

詢問點 A, B 是否在同一個連通塊當中

```
1 bool same(int A, int B) {
2   int A_end_point = end_point(A);
3   int B_end_point = end_point(B);
4   return A_end_point == B_end_point;
5 }
```

並查集 (Disjoint Set Union) – DSU 實作

我們可以先分析一下時間複雜度

並查集 (Disjoint Set Union) – DSU 實作

我們可以先分析一下時間複雜度

- 初始化 $O(N)$

並查集 (Disjoint Set Union) – DSU 實作

我們可以先分析一下時間複雜度

- 初始化 $O(N)$
- 詢問 x 的結束點 $O(\text{鍊長}) = O(N)$

並查集 (Disjoint Set Union) – DSU 實作

我們可以先分析一下時間複雜度

- 初始化 $O(N)$
- 詢問 x 的結束點 $O(\text{鍊長}) = O(N)$
- 合併兩個連通塊 $O(\text{兩次詢問結束點操作}) = O(N)$

並查集 (Disjoint Set Union) – DSU 實作

我們可以先分析一下時間複雜度

- 初始化 $O(N)$
- 詢問 x 的結束點 $O(\text{鍊長}) = O(N)$
- 合併兩個連通塊 $O(\text{兩次詢問結束點操作}) = O(N)$
- 詢問兩點連通關係 $O(\text{兩次詢問結束點操作}) = O(N)$

並查集 (Disjoint Set Union) – DSU 實作

我們可以先分析一下時間複雜度

- 初始化 $O(N)$
- 詢問 x 的結束點 $O(\text{鍊長}) = O(N)$
- 合併兩個連通塊 $O(\text{兩次詢問結束點操作}) = O(N)$
- 詢問兩點連通關係 $O(\text{兩次詢問結束點操作}) = O(N)$

Q 筆操作的話，總體時間複雜度為 $O(QN)$

並查集 (Disjoint Set Union) – DSU 實作

我們可以先分析一下時間複雜度

- 初始化 $O(N)$
- 詢問 x 的結束點 $O(\text{鍊長}) = O(N)$
- 合併兩個連通塊 $O(\text{兩次詢問結束點操作}) = O(N)$
- 詢問兩點連通關係 $O(\text{兩次詢問結束點操作}) = O(N)$

Q 筆操作的話，總體時間複雜度為 $O(QN)$

這樣看起來，時間複雜度好高!

並查集 (Disjoint Set Union) – 優化

1 基礎資料結構

- 並查集 (Disjoint Set Union)
  - 一些基礎圖論的東西
  - DSU 介紹
  - DSU 實作
  - 優化
  - 複雜度分析
  - 題目

並查集 (Disjoint Set Union) – 優化

優化 DSU 的複雜度的方法有兩個

- 路徑壓縮 (Path Compression)
- 啟發式合併 (Union by rank/size)

二者都能將並查集的時間複雜度優化到 $O((Q+N)\log N)$

並查集 (Disjoint Set Union) – 優化

優化 DSU 的複雜度的方法有兩個

- 路徑壓縮 (Path Compression)
- 啟發式合併 (Union by rank/size)

二者都能將並查集的時間複雜度優化到 $O((Q+N)\log N)$

且二者也可以搭配在一起用，時間複雜度為 $O((Q+N)\alpha(N))$

並查集 (Disjoint Set Union) – 優化

優化 DSU 的複雜度的方法有兩個

- 路徑壓縮 (Path Compression)
- 啟發式合併 (Union by rank/size)

二者都能將並查集的時間複雜度優化到 $O((Q+N)\log N)$

且二者也可以搭配在一起用，時間複雜度為 $O((Q+N)\alpha(N))$

$\alpha(N)$ 為反阿克曼函數，待會會提到

並査集 (Disjoint Set Union) – 優化– 路徑壓縮

路徑壓縮 (Path Compression)

並查集 (Disjoint Set Union) – 優化– 路徑壓縮

路徑壓縮 (Path Compression)

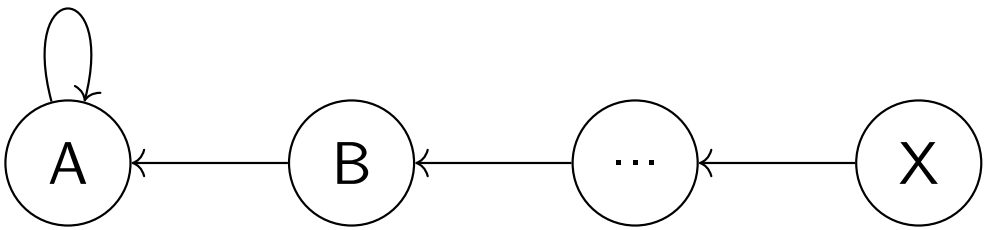
當我們觀察到 DSU 的時間複雜度瓶頸好像就是處於查詢結束點的時候

並查集 (Disjoint Set Union) – 優化– 路徑壓縮

路徑壓縮 (Path Compression)

當我們觀察到 DSU 的時間複雜度瓶頸好像就是處於查詢結束點的時候

因為只要圖是一條鍊，且詢問的點為鍊的最末端，那麼每次都得往上走 N 次才能到達結束點



並查集 (Disjoint Set Union) – 優化– 路徑壓縮

那麼我們就對於整個要走的路徑去做優化好了!

並查集 (Disjoint Set Union) – 優化– 路徑壓縮

那麼我們就對於整個要走的路徑去做優化好了!

當我們今天詢問點 x 的時候，我們會希望它與它的結束點的距離 **越近越好**

並查集 (Disjoint Set Union) – 優化– 路徑壓縮

那麼我們就對於整個要走的路徑去做優化好了!

當我們今天詢問點 x 的時候，我們會希望它與它的結束點的距離 **越近越好**

而我們在鍊上跑的時候，經過的點好像不是那麼的重要?

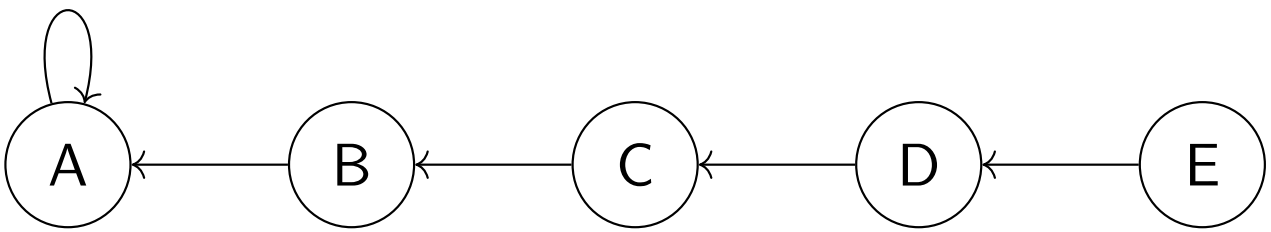
並查集 (Disjoint Set Union) – 優化– 路徑壓縮

那麼我們就對於整個要走的路徑去做優化好了!

當我們今天詢問點 x 的時候，我們會希望它與它的結束點的距離 **越近越好**

而我們在鍊上跑的時候，經過的點好像不是那麼的重要?

所以當我們在詢問完結束點後，是否可以把鍊上經過的所有點的邊，直接連到結束點上?



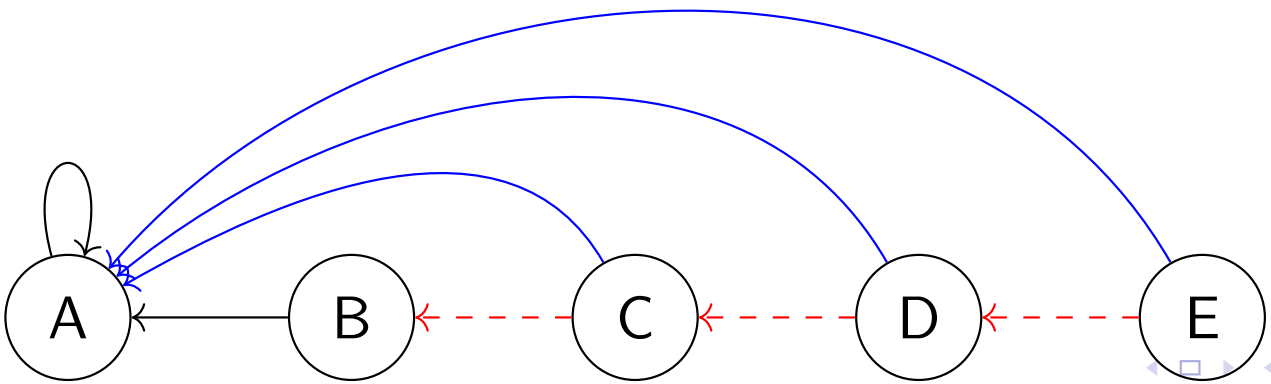
並查集 (Disjoint Set Union) – 優化– 路徑壓縮

那麼我們就對於整個要走的路徑去做優化好了!

當我們今天詢問點 x 的時候，我們會希望它與它的結束點的距離 **越近越好**

而我們在鍊上跑的時候，經過的點好像不是那麼的重要?

所以當我們在詢問完結束點後，是否可以把鍊上經過的所有點的邊，直接連到結束點上?



並查集 (Disjoint Set Union) – 優化– 路徑壓縮

蛤? 這樣還不是要走整條鏈?

並查集 (Disjoint Set Union) – 優化– 路徑壓縮

蛤? 這樣還不是要走整條鏈?

這個是 $O((N+Q)\log N)$ 算法?

並查集 (Disjoint Set Union) – 優化– 路徑壓縮

蛤? 這樣還不是要走整條鏈?

這個是 $O((N+Q)\log N)$ 算法?唬爛吧?

並查集 (Disjoint Set Union) – 優化– 路徑壓縮

蛤? 這樣還不是要走整條鏈?

這個是 $O((N+Q)\log N)$ 算法?唬爛吧?

別急，你先別急

並查集 (Disjoint Set Union) – 優化– 路徑壓縮

蛤? 這樣還不是要走整條鏈?

這個是 $O((N+Q)\log N)$ 算法?唬爛吧?

別急，你先別急

我們可以先感性理解一下

並查集 (Disjoint Set Union) – 優化– 路徑壓縮

蛤? 這樣還不是要走整條鏈?

這個是 $O((N+Q)\log N)$ 算法?唬爛吧?

別急，你先別急

我們可以先感性理解一下

當我們每次查詢的時候，都會走完一個鍊的長度，但我們相信他，所以他的長鍊的數量應該只有 $\log N$ 個，所以整體複雜度為 $O((N+Q)\log N)$

並查集 (Disjoint Set Union) – 優化– 路徑壓縮

蛤? 這樣還不是要走整條鏈?

這個是 $O((N+Q)\log N)$ 算法?唬爛吧?

別急，你先別急

我們可以先感性理解一下

當我們每次查詢的時候，都會走完一個鍊的長度，但我們相信他，所以他的長鍊的數量應該只有 $\log N$ 個，所以整體複雜度為 $O((N+Q)\log N)$?

並查集 (Disjoint Set Union) – 優化– 路徑壓縮

蛤? 這樣還不是要走整條鏈?

這個是 $O((N+Q)\log N)$ 算法?唬爛吧?

別急，你先別急

我們可以先感性理解一下

當我們每次查詢的時候，都會走完一個鍊的長度，但我們相信他，所以他的長鍊的數量應該只有 $\log N$ 個，所以整體複雜度為 $O((N+Q)\log N)$?

會在之後的章節稍微證明一下正確的複雜度

並查集 (Disjoint Set Union) – 優化– 路徑壓縮

路徑壓縮的實作

```
1  int end_point(int x) {
2    if(nxt[x] == x) return x; // 自己是結束點
3    // 當前的結束點是下一個點的結束點
4    // 直接將下一個點接到查詢後的結束點
5    nxt[x] = end_point(nxt[x]);
6    return nxt[x];
7  }
```

並査集 (Disjoint Set Union) – 優化– 啟發式合併

啟發式合併 (Union By Rank/Size)

並查集 (Disjoint Set Union) – 優化– 啟發式合併

啟發式合併 (Union By Rank/Size)

剛剛聊的路徑壓縮是對於詢問結束點的操作所進行的優化

並查集 (Disjoint Set Union) – 優化– 啟發式合併

啟發式合併 (Union By Rank/Size)

剛剛聊的路徑壓縮是對於詢問結束點的操作所進行的優化

那我們是否可以對合併的時候進行優化呢?

並查集 (Disjoint Set Union) – 優化– 啟發式合併

我們發現，當我們做合併連通塊的時候，我們好像是隨便將一個連通塊的結束點連接到另一個連通塊的結束點上

並查集 (Disjoint Set Union) – 優化– 啟發式合併

我們發現，當我們做合併連通塊的時候，我們好像是隨便將一個連通塊的結束點連接到另一個連通塊的結束點上

我們可以發現，連接過去的連通塊的最大深度會 $+1$

並查集 (Disjoint Set Union) – 優化– 啟發式合併

我們發現，當我們做合併連通塊的時候，我們好像是隨便將一個連通塊的結束點連接到另一個連通塊的結束點上

我們可以發現，連接過去的連通塊的最大深度會 $+1$

且換個角度思考，他新接的那條邊最糟會被走下面那個連通塊的節點個數的次數

並查集 (Disjoint Set Union) – 優化– 啟發式合併

我們發現，當我們做合併連通塊的時候，我們好像是隨便將一個連通塊的結束點連接到另一個連通塊的結束點上

我們可以發現，連接過去的連通塊的最大深度會 $+1$

且換個角度思考，他新接的那條邊最糟會被走下面那個連通塊的節點個數的次數

那麼我們究竟要考慮深度還是節點個數呢?

並查集 (Disjoint Set Union) – 優化– 啟發式合併

不難發現，兩種性質都可以影響到之後的複雜度

並查集 (Disjoint Set Union) – 優化– 啟發式合併

不難發現，兩種性質都可以影響到之後的複雜度

且兩個連通塊的兩種性質大小關係不一定完全小/大於

並查集 (Disjoint Set Union) – 優化– 啟發式合併

不難發現，兩種性質都可以影響到之後的複雜度

且兩個連通塊的兩種性質大小關係不一定完全小/大於

所以我們無論選哪個性質來做，複雜度都將會是 $O(\log N)$

並查集 (Disjoint Set Union) – 優化– 啟發式合併

不難發現，兩種性質都可以影響到之後的複雜度

且兩個連通塊的兩種性質大小關係不一定完全小/大於

所以我們無論選哪個性質來做，複雜度都將會是 $O(\log N)$

複雜度我們以後也會證明

並查集 (Disjoint Set Union) – 優化– 啟發式合併

啟發式合併有兩種做法，而我們這裡使用節點個數總合為啟發式合併的依據

```
1  // sz[x] 代表著 x 連通塊的大小，預設每個都為 1
2  void merge(int A, int B) {
3    int A_end_point = end_point(A);
4    int B_end_point = end_point(B);
5    // 相同連通塊不需要合併
6    if(A_end_point == B_end_point) return;
7    if(sz[A_end_point] > sz[B_end_point]) {
8      // 需要將 A 連到 B
9      // 且確保 A 的大小 < B 的大小
10     swap(A_end_point, B_end_point);
11   }
12   // 將 A 的結束點連到 B的結束點
13   nxt[A_end_point] = B_end_point;
14 }
```

並查集 (Disjoint Set Union) – 複雜度分析

1 基礎資料結構

- 並查集 (Disjoint Set Union)
  - 一些基礎圖論的東西
  - DSU 介紹
  - DSU 實作
  - 優化
  - 複雜度分析
  - 題目

並查集 (Disjoint Set Union) – 複雜度分析– 總結

有時候路徑壓縮以及啟發式合併並不能套用到某些 DSU 的題目上，但二者只要選一者使用，就能達到 $O(\log N)$ 等級的複雜度

並查集 (Disjoint Set Union) – 複雜度分析– 總結

有時候路徑壓縮以及啟發式合併並不能套用到某些 DSU 的題目上，但二者只要選一者使用，就能達到 $O(\log N)$ 等級的複雜度

~~且理論上複雜度不會證也沒關係~~

並查集 (Disjoint Set Union) – 題目

1 基礎資料結構

- 並查集 (Disjoint Set Union)
  - 一些基礎圖論的東西
  - DSU 介紹
  - DSU 實作
  - 優化
  - 複雜度分析
  - 題目

並查集 (Disjoint Set Union) – 題目– 例題 1

例題

TOJ 701 電學大師
N 點，Q 筆詢問
每次有 2 種操作

1. 連通兩點
2. 詢問兩點是否有連通

題目限制

- $1 \leq N \leq 10^5$
- $1 \leq Q \leq 2 \times 10^5$

並查集 (Disjoint Set Union) – 題目– 例題 1

我們可以發現這題是一題 DSU 裸題

並查集 (Disjoint Set Union) – 題目– 例題 1

我們可以發現這題是一題 DSU 裸題

而實作上我們通常都是直接寫一個 struct 去把 DSU 的整體架構包住

並查集 (Disjoint Set Union) – 題目– 例題 1

我們可以發現這題是一題 DSU 裸題

而實作上我們通常都是直接寫一個 struct 去把 DSU 的整體架構包住

這樣我們只要確保 struct 裡面的 function 寫好，那麼就不會汙染到全域環境了~~(也不會佔到變數名稱)~~

並查集 (Disjoint Set Union) – 題目– 例題 1

這個是我常用的模板，將連通塊大小跟下一個點的指向寫在一起 ~~(優美的 code)~~

```
1  struct DSU{
2    vector<int> p;
3    DSU(int n) : p(n+1, -1) {}
4    int fp(int id) {return p[id] < 0 ? id : p[id] =
         fp(p[id]);}
5    bool same(int a, int b) {return fp(a) == fp(b);}
6    void upd(int a, int b) {
7      int ar = fp(a), br = fp(b);
8      if(ar != br) {
9        if(p[ar] > p[br]) swap(ar, br);
10       p[ar] += p[br];
11       p[br] = ar;
12     }
13   }
14 };
```

並查集 (Disjoint Set Union) – 題目– 例題 1

呼叫

```
// main 裡面
int n, q; cin >> n >> q;
DSU dsu(n);
while(q--) {
  int op, a, b; cin >> op >> a >> b;
  if(op == 0) {
    cout << (dsu.same(a, b) ? "YES\n" : "NO\n");
  } else {
    dsu.upd(a, b);
  }
}
```

並查集 (Disjoint Set Union) – 題目– 例題 2

例題

TOJ 89 可愛的小動物
N 個動物，Q 筆詢問
每次有 4 種操作

1. A 與 B 為朋友
2. A 與 B 為敵人
3. 詢問 A 與 B 為朋友
4. 詢問 A 與 B 為敵人

若 $1, 2$ 操作已與之前的操作衝突，則輸出"angry"
若 $3, 4$ 操作為真，則輸出"yeap"，反之輸出"nope"
題目限制

- $1 \leq N, Q \leq 5 \cdot 10^5$

基礎資料結構 – 在線與離線 (Online or Offline)

1 基礎資料結構

- 資料結構介紹
- Struct & Class
- 並查集 (Disjoint Set Union)
- 在線與離線 (Online or Offline)
- 根號想法 (Sqrt Decomposition)
- 線段樹 (Segment Tree)
- 樹狀樹組 (Binary Index Tree)
- 稀疏表
- 珂朵莉樹

基礎資料結構 – 根號想法 (Sqrt Decomposition)

1 基礎資料結構

- 資料結構介紹
- Struct & Class
- 並查集 (Disjoint Set Union)
- 在線與離線 (Online or Offline)
- 根號想法 (Sqrt Decomposition)
- 線段樹 (Segment Tree)
- 樹狀樹組 (Binary Index Tree)
- 稀疏表
- 珂朵莉樹

基礎資料結構 – 線段樹 (Segment Tree)

1 基礎資料結構

- 資料結構介紹
- Struct & Class
- 並查集 (Disjoint Set Union)
- 在線與離線 (Online or Offline)
- 根號想法 (Sqrt Decomposition)
- 線段樹 (Segment Tree)
- 樹狀樹組 (Binary Index Tree)
- 稀疏表
- 珂朵莉樹

線段樹 (Segment Tree) – 單點修改

1 基礎資料結構

- 線段樹 (Segment Tree)
  - 單點修改
  - 區間修改

線段樹 (Segment Tree) – 區間修改

1 基礎資料結構

- 線段樹 (Segment Tree)
  - 單點修改
  - 區間修改

基礎資料結構 – 樹狀樹組 (Binary Index Tree)

1 基礎資料結構

- 資料結構介紹
- Struct & Class
- 並查集 (Disjoint Set Union)
- 在線與離線 (Online or Offline)
- 根號想法 (Sqrt Decomposition)
- 線段樹 (Segment Tree)
- 樹狀樹組 (Binary Index Tree)
- 稀疏表
- 珂朵莉樹

樹狀樹組 (Binary Index Tree) – 單點修改

1 基礎資料結構

- 樹狀樹組 (Binary Index Tree)
  - 單點修改
  - 區間修改

樹狀樹組 (Binary Index Tree) – 區間修改

1 基礎資料結構

- 樹狀樹組 (Binary Index Tree)
  - 單點修改
  - 區間修改

基礎資料結構 – 稀疏表

1 基礎資料結構

- 資料結構介紹
- Struct & Class
- 並查集 (Disjoint Set Union)
- 在線與離線 (Online or Offline)
- 根號想法 (Sqrt Decomposition)
- 線段樹 (Segment Tree)
- 樹狀樹組 (Binary Index Tree)
- 稀疏表
- 珂朵莉樹

基礎資料結構 – 珂朵莉樹

1 基礎資料結構

- 資料結構介紹
- Struct & Class
- 並查集 (Disjoint Set Union)
- 在線與離線 (Online or Offline)
- 根號想法 (Sqrt Decomposition)
- 線段樹 (Segment Tree)
- 樹狀樹組 (Binary Index Tree)
- 稀疏表
- 珂朵莉樹

進階資料結構

進階資料結構 – 回滾 DSU

2 進階資料結構

- 回滾 DSU
- 帶權 DSU
- 線段樹/樹狀數組上二分搜
- 字典樹 (Trie)
- 動態開點線段樹
- 持久化線段樹
- 樹套樹
- 樹堆 (Treap)

進階資料結構 – 帶權 DSU

2 進階資料結構

- 回滾 DSU
- 帶權 DSU
- 線段樹/樹狀數組上二分搜
- 字典樹 (Trie)
- 動態開點線段樹
- 持久化線段樹
- 樹套樹
- 樹堆 (Treap)

進階資料結構 – 線段樹/樹狀數組上二分搜

2 進階資料結構

- 回滾 DSU
- 帶權 DSU
- 線段樹/樹狀數組上二分搜
- 字典樹 (Trie)
- 動態開點線段樹
- 持久化線段樹
- 樹套樹
- 樹堆 (Treap)

進階資料結構 – 字典樹 (Trie)

2 進階資料結構

- 回滾 DSU
- 帶權 DSU
- 線段樹/樹狀數組上二分搜
- 字典樹 (Trie)
- 動態開點線段樹
- 持久化線段樹
- 樹套樹
- 樹堆 (Treap)

2 進階資料結構

- 回滾 DSU
- 帶權 DSU
- 線段樹/樹狀數組上二分搜
- 字典樹 (Trie)
- 動態開點線段樹
- 持久化線段樹
- 樹套樹
- 樹堆 (Treap)

2 進階資料結構

- 回滾 DSU
- 帶權 DSU
- 線段樹/樹狀數組上二分搜
- 字典樹 (Trie)
- 動態開點線段樹
- 持久化線段樹
- 樹套樹
- 樹堆 (Treap)

進階資料結構 – 樹套樹

2 進階資料結構

- 回滾 DSU
- 帶權 DSU
- 線段樹/樹狀數組上二分搜
- 字典樹 (Trie)
- 動態開點線段樹
- 持久化線段樹
- 樹套樹
- 樹堆 (Treap)

進階資料結構 – 樹堆 (Treap)

2 進階資料結構

- 回滾 DSU
- 帶權 DSU
- 線段樹/樹狀數組上二分搜
- 字典樹 (Trie)
- 動態開點線段樹
- 持久化線段樹
- 樹套樹
- 樹堆 (Treap)

超進階資料結構

超進階資料結構 – 李超線段樹

3 超進階資料結構

- 李超線段樹
- 時間線段樹
- 線段樹優化建圖
- 吉如一線段樹 (Segment Beat)
- 持久化 DSU
- 動態凸包
- 貓樹

3 超進階資料結構

- 李超線段樹
- 時間線段樹
- 線段樹優化建圖
- 吉如一線段樹 (Segment Beat)
- 持久化 DSU
- 動態凸包
- 貓樹

3 超進階資料結構

- 李超線段樹
- 時間線段樹
- 線段樹優化建圖
- 吉如一線段樹 (Segment Beat)
- 持久化 DSU
- 動態凸包
- 貓樹

超進階資料結構 – 吉如一線段樹 (Segment Beat)

3 超進階資料結構

- 李超線段樹
- 時間線段樹
- 線段樹優化建圖
- 吉如一線段樹 (Segment Beat)
- 持久化 DSU
- 動態凸包
- 貓樹

超進階資料結構 – 持久化 DSU

3 超進階資料結構

- 李超線段樹
- 時間線段樹
- 線段樹優化建圖
- 吉如一線段樹 (Segment Beat)
- 持久化 DSU
- 動態凸包
- 貓樹

超進階資料結構 – 動態凸包

3 超進階資料結構

- 李超線段樹
- 時間線段樹
- 線段樹優化建圖
- 吉如一線段樹 (Segment Beat)
- 持久化 DSU
- 動態凸包
- 貓樹

3 超進階資料結構

- 李超線段樹
- 時間線段樹
- 線段樹優化建圖
- 吉如一線段樹 (Segment Beat)
- 持久化 DSU
- 動態凸包
- 貓樹

參考資料

參考資料

Tfcis - 2024 寒訓資料結構簡報 by tobiichi3227

Tfcis T26 資料結構簡報 by tw20000807

Tfcis - 2024 暑訓資料結構簡報 by tw20000807

Graph_Theory wiki

Tarjan, Robert E. ， van Leeuwen, Jan - Worst-case analysis of set union algorithms