
# NCKU Programming Contest Training Course
# Course 5
# 2013/01/22

**Pin-Chieh Huang (free999)**

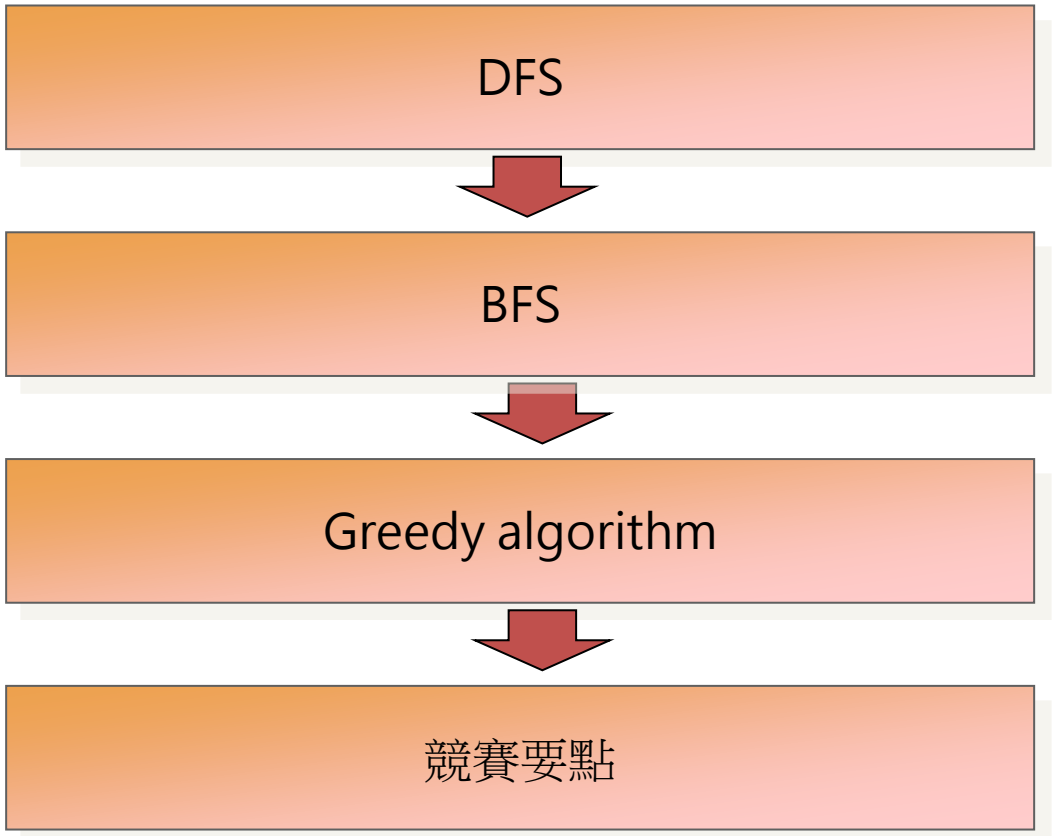
*Pinchieh.huang@gmail.com*

Department of Computer Science and Information Engineering
National Cheng Kung University
Tainan, Taiwan



***made by electron & free999***

# Outline

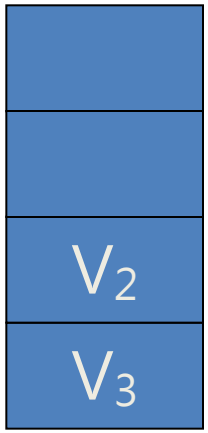


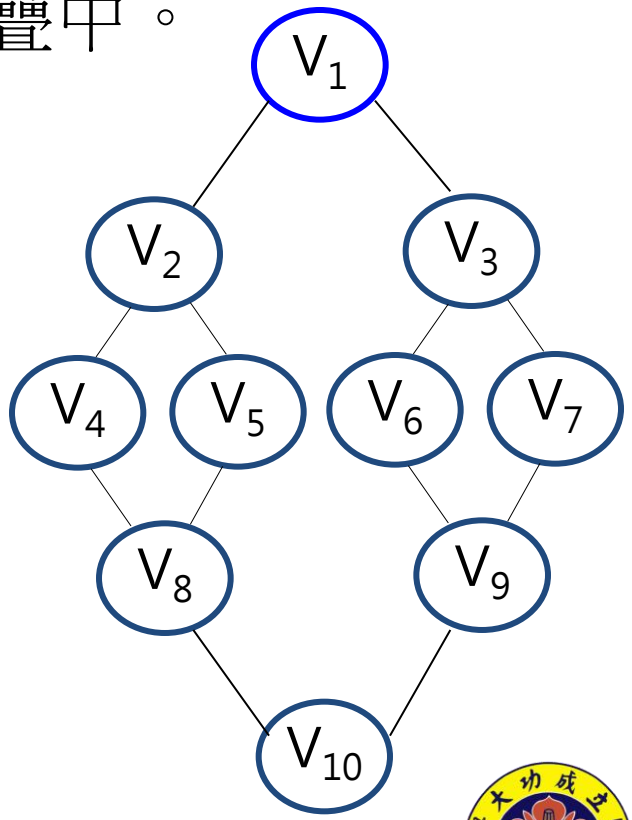



made by electron & free999

# DFS

- 先輸出$V_1$(假設$V_1$為起點) 。
- 將$V_1$的相鄰頂點$V_2$及$V_3$放入堆疊中。

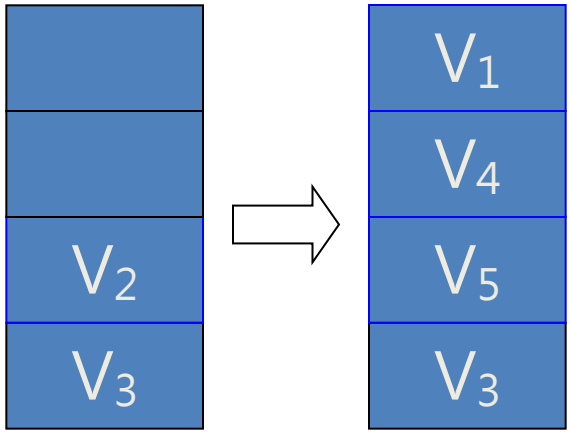


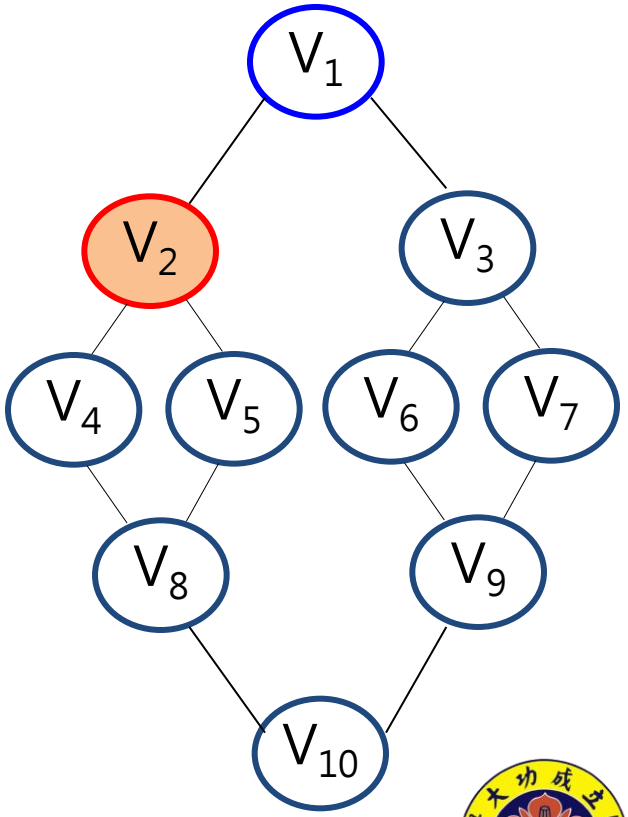




輸出：$V_1$

# DFS

- 彈出堆疊的第一個頂點$V_2$，然後將$V_2$的相鄰頂點$V_1$, $V_4$及$V_5$推入到堆疊。

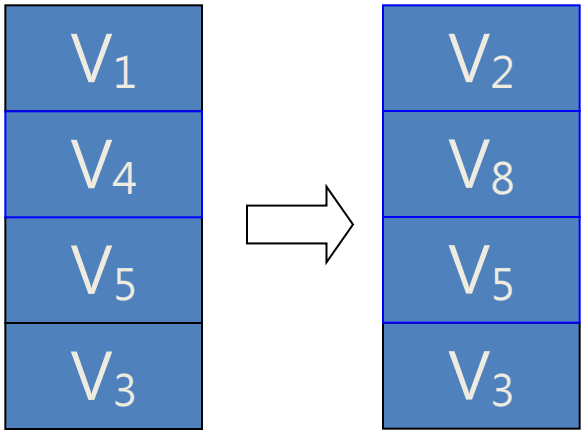


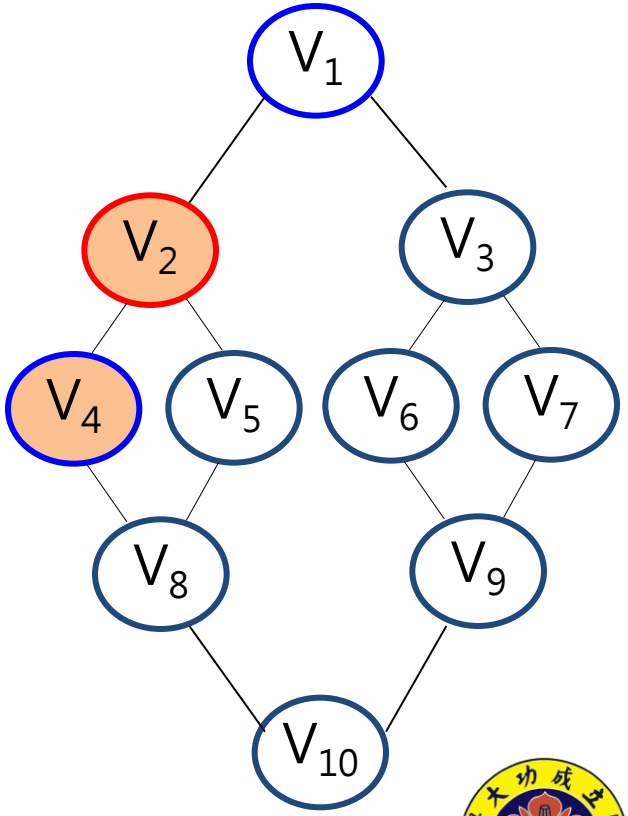




輸出：$V_1$ $V_2$

# DFS

- 彈出$V_1$，由於$V_1$已被輸出，故捨棄不用
- 接著再彈出$V_4$
- 將$V_4$的相鄰頂點$V_2$及$V_8$放入堆疊。

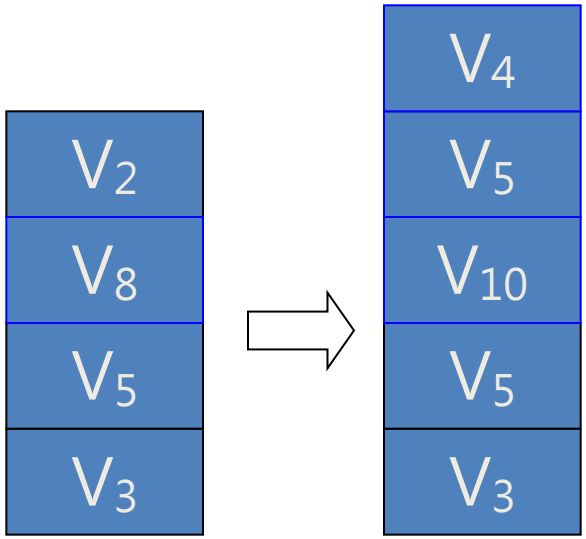


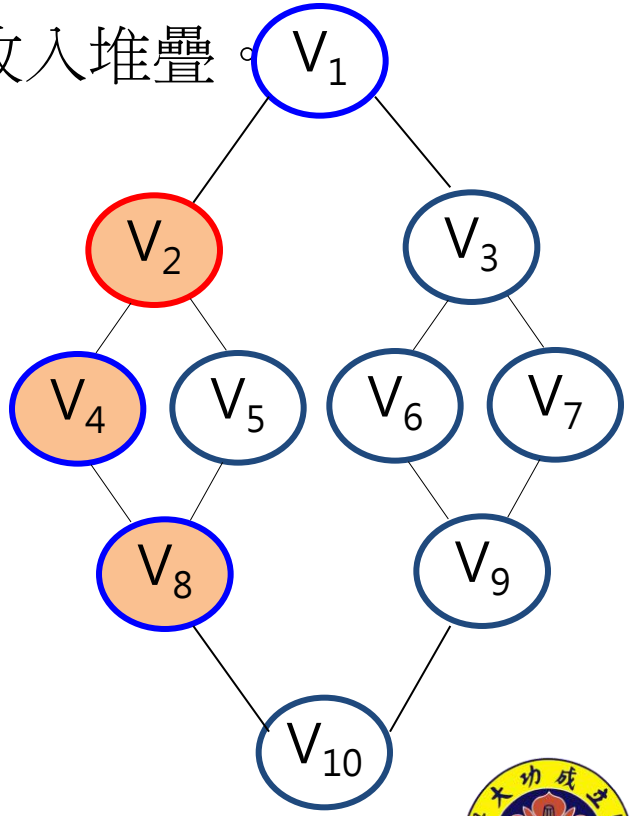




輸出：$V_1$ $V_2$ $V_4$

# DFS

- 彈出$V_2$，由於$V_2$已被輸出過，故捨棄不用
- 接著再彈出$V_8$
- 再將$V_8$的相鄰頂點$V_4$、$V_5$及$V_{10}$放入堆疊。

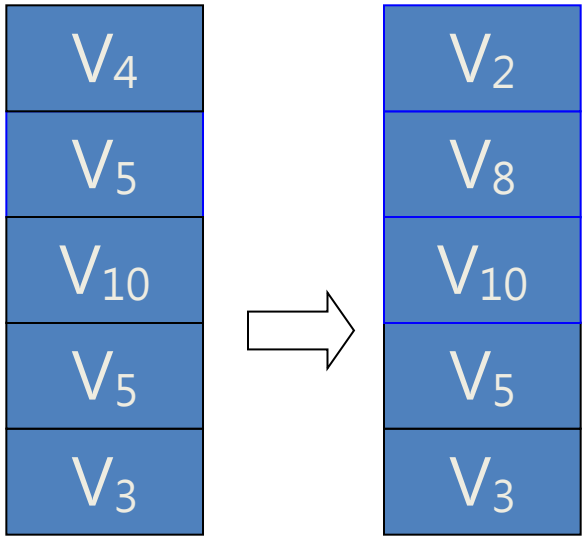


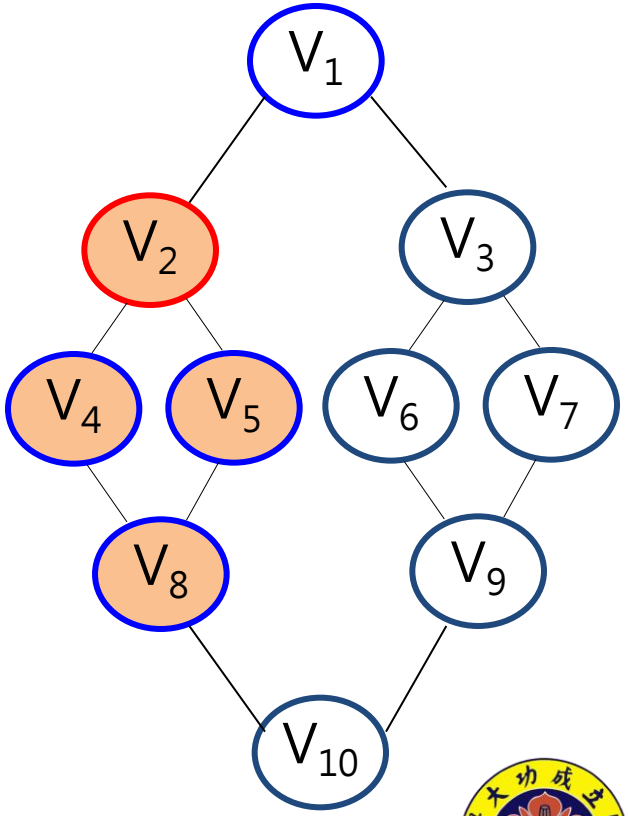




輸出：$V_1$ $V_2$ $V_4$ $V_8$

# DFS

- 彈出$V_4$，此頂點已被輸出，故捨棄不用
- 接著再彈出$V_5$ ,將 $V_5$的相鄰頂點$V_2$、$V_8$放入堆疊。

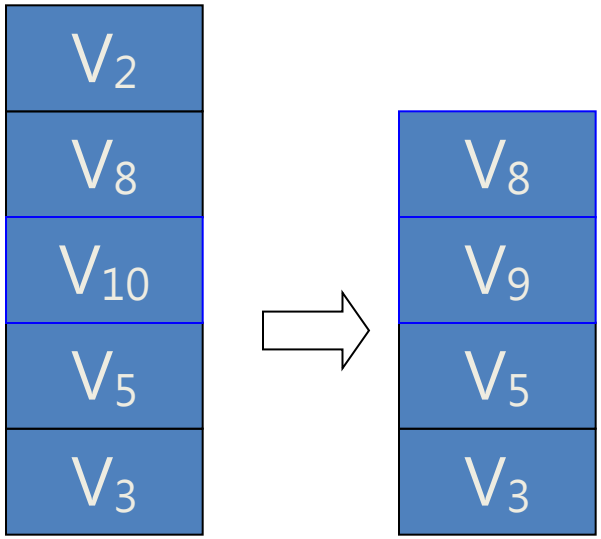


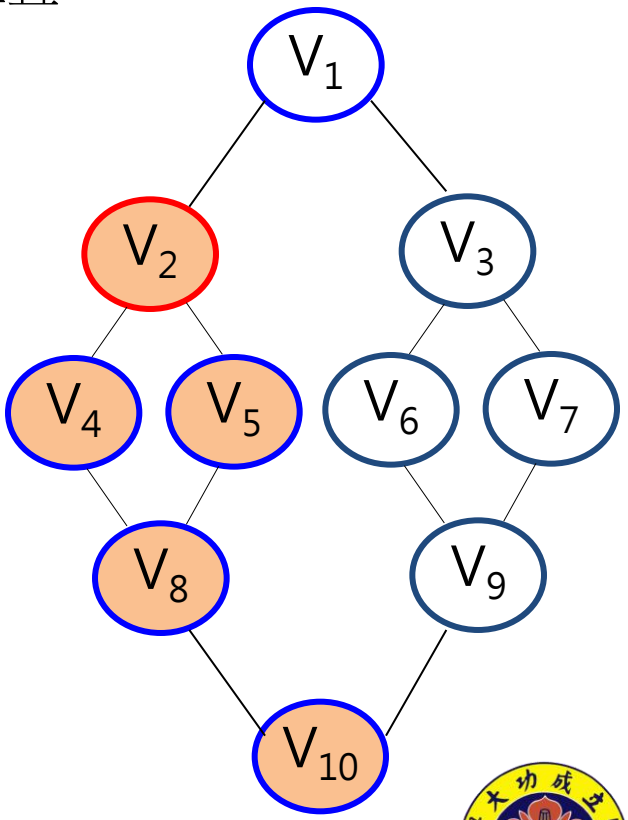




輸出：$V_1$ $V_2$ $V_4$ $V_8$ $V_5$

# DFS

- 彈出$V_2$及$V_8$，由於此二頂點已被輸出過， 故捨棄不用，接著再彈出$V_{10}$，再將$V_{10}$ 的相鄰點$V_8$及$V_9$放入堆疊。

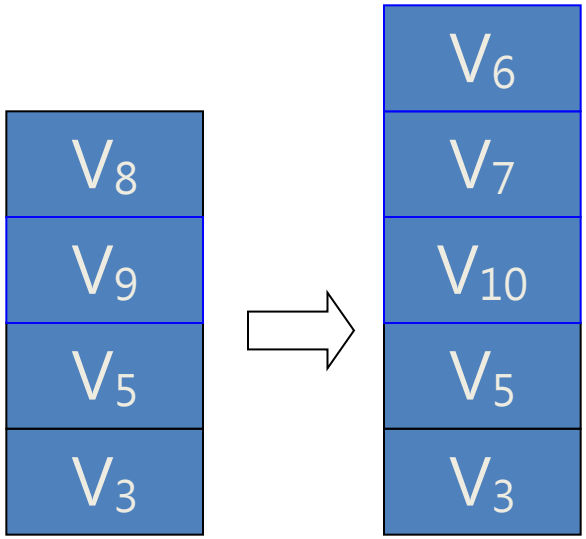


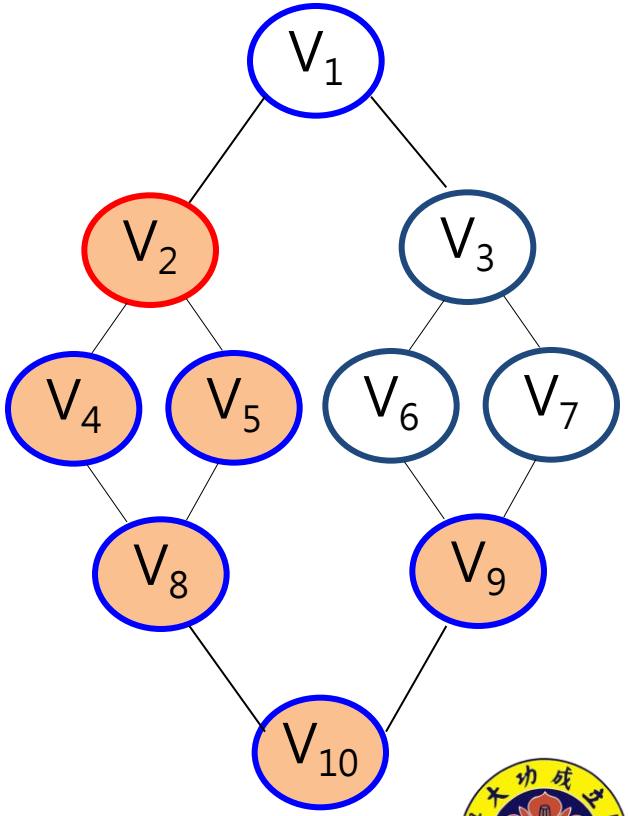




輸出：$V_1$ $V_2$ $V_4$ $V_8$ $V_5$ $V_{10}$

# DFS

- 彈出$V_8$，此頂點已被輸出，故捨棄不用，接著再彈出$V_9$，將$V_9$的相鄰頂點$V_6$、$V_7$及$V_{10}$放入堆疊。

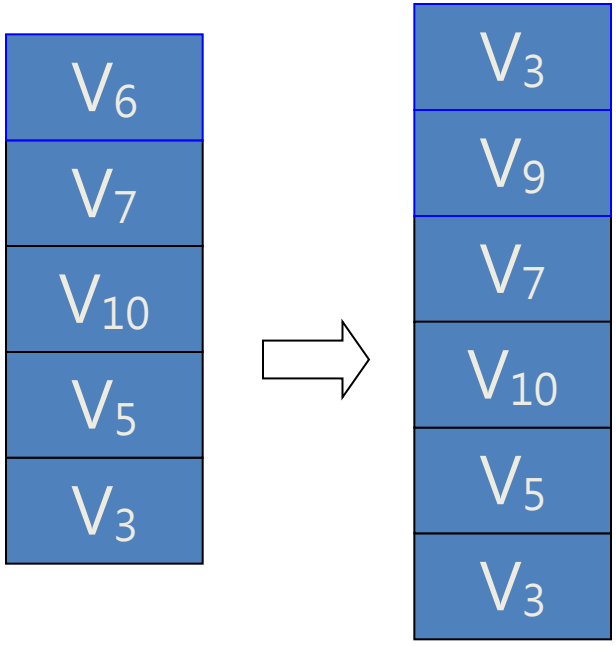


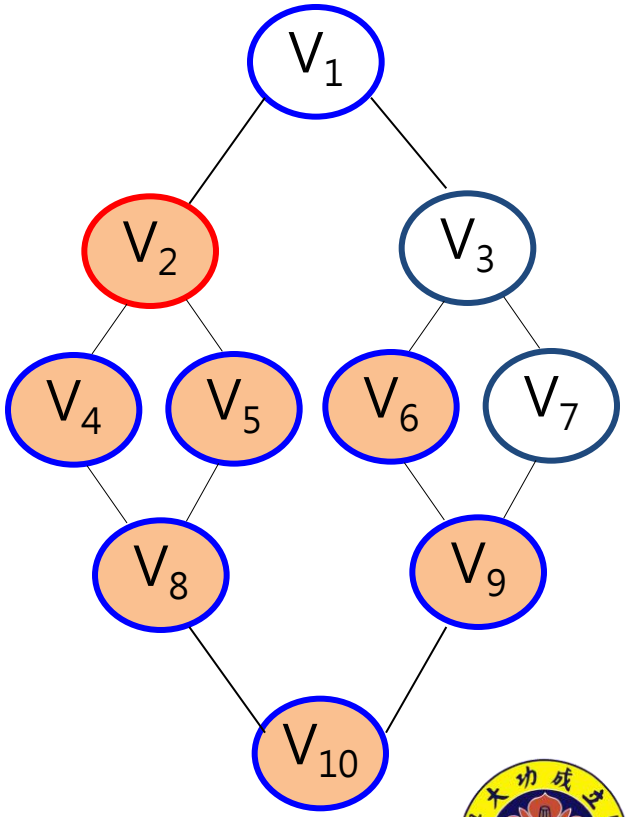




輸出：$V_1$ $V_2$ $V_4$ $V_8$ $V_5$ $V_{10}$ $V_9$

# DFS

- 彈出$V_6$，再將$V_6$的相鄰頂點$V_3$及$V_9$放入堆疊。

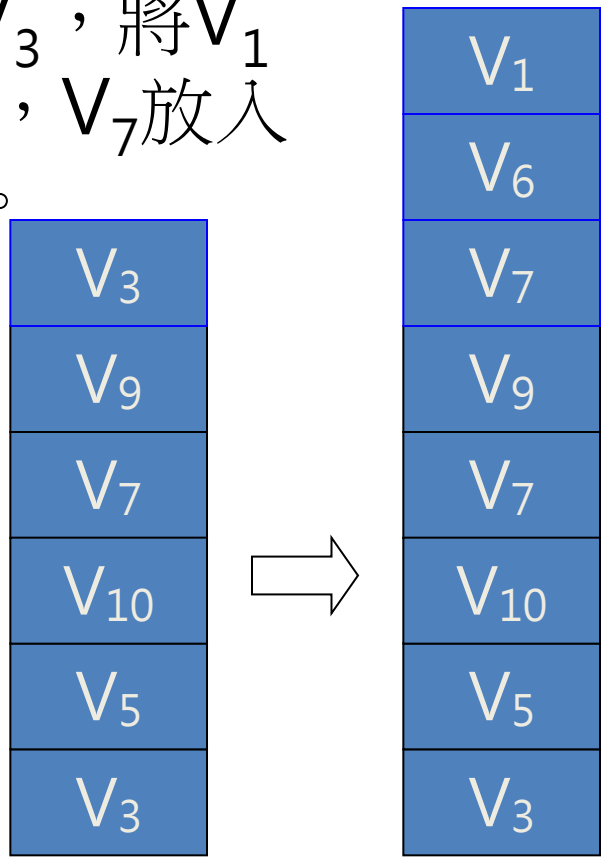


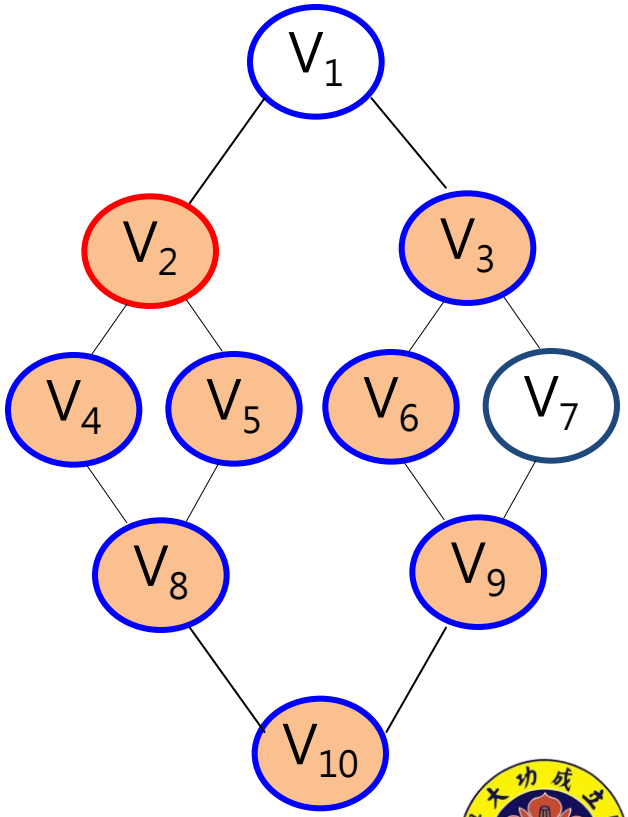




輸出：$V_1$ $V_2$ $V_4$ $V_8$ $V_5$ $V_{10}$ $V_9$ $V_6$

# DFS

- 彈出$V_3$，將$V_1$與$V_6$，$V_7$放入堆疊。

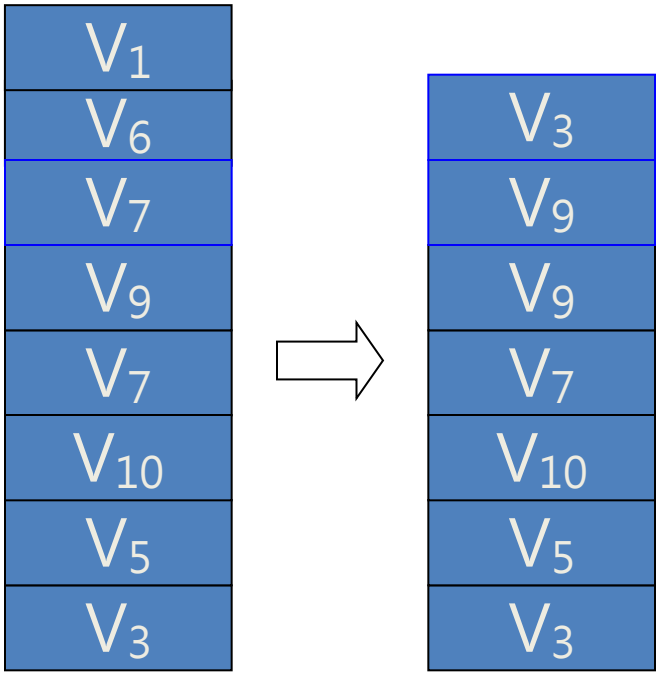


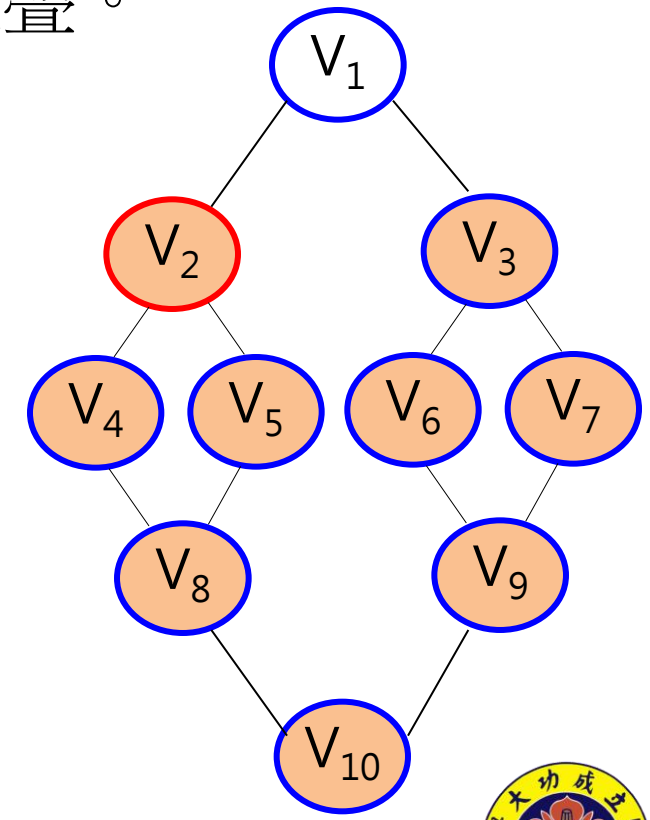




輸出：$V_1$ $V_2$ $V_4$ $V_8$ $V_5$ $V_{10}$ $V_9$ $V_6$ $V_3$

# DFS

- 彈出$V_1$，此頂點已輸出，故捨棄不用，接著再彈出$V_6$、$V_7$，再將$V_3$及$V_9$放入堆疊。




輸出：$V_1$ $V_2$ $V_4$ $V_8$ $V_5$ $V_{10}$ $V_9$ $V_6$ $V_3$ $V_7$

# DFS

- 最後彈出$V_3$, $V_9$, $V_9$, $V_7$, $V_{10}$, $V_5$, $V_3$，由於這些頂點皆已輸出過；此時堆疊是空的，表示搜尋已結束。
- 從上述的搜尋步驟可知其順序為：$V_1$, $V_2$, $V_4$, $V_8$, $V_5$, $V_{10}$, $V_9$, $V_6$, $V_3$, $V_7$。
- 需注意的是此順序**並不是唯一**，而是根據頂點放入堆疊的順序而定。

# DFS

```
void dfs(int now)
{
    visited[now] = true;
    for(int i=0; i<(int)adj[now].size(); i++)
    {
        int next = adj[now][i];
        if(!visited[next]) dfs(next);
    }
}
```

# DFS

- How can we Do with DFS ?
  - Check the Connectivity
  - Graph Traversal

# Example - 1

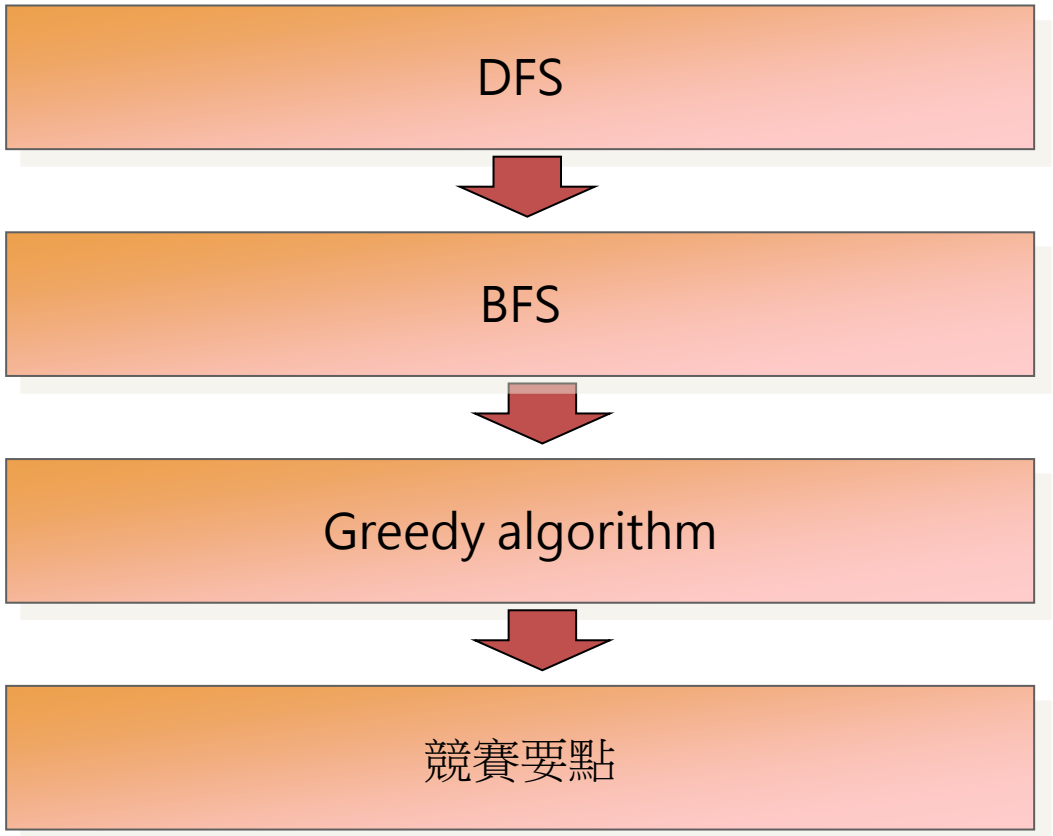
- UVA-10004
  - Bi-color Problem
- Input

```
3

3
0 1
1 2
2 0
9

8
0 1
0 2
0 3
0 4
0 5
0 6
0 7
0 8
0
```

Output

```
NOT BICOLORABLE.
BICOLORABLE.
```

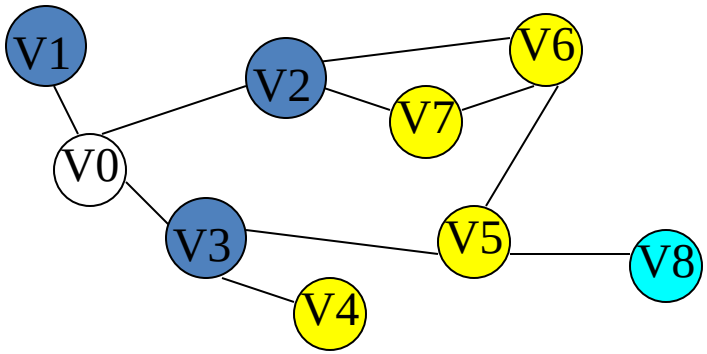
# Outline






made by electron & free999

# BFS




＊如果以V0為起點，其中一組 BFS 拜訪順序為：

V0, V1, V2, V3, V6, V7, V5, V4, V8

第一層　　第二層　　第三層

＊方法：從起點V0 開始，拜訪和起點V0 相鄰的所有頂點。再拜訪更外一層的頂點，也就是與起點V0 相鄰頂點的相鄰頂點。直到所有連通頂點都拜訪過為止



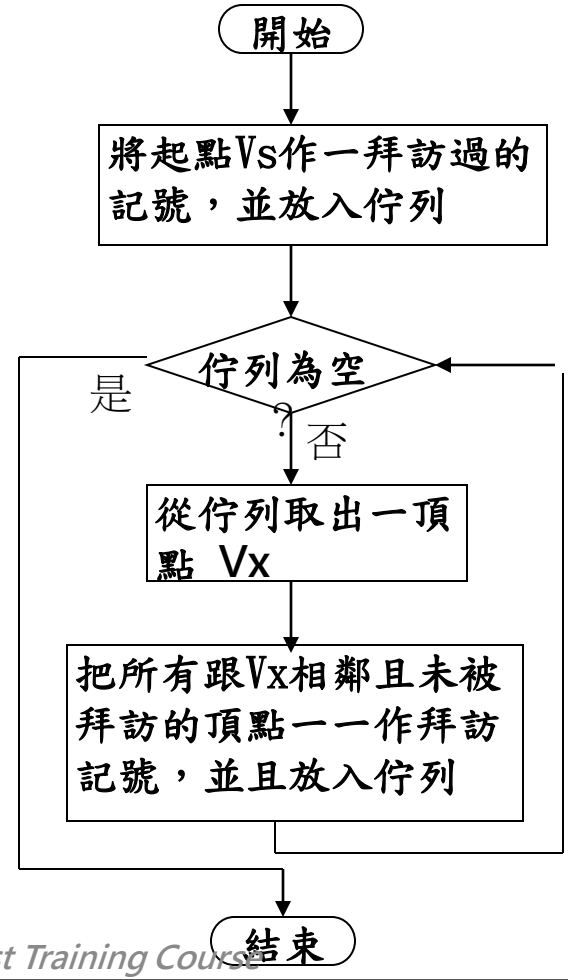
made by electron & free999

# BFS

- 由於先走訪的點，會先對其相鄰頂點進行廣度優先搜尋：後走訪者，則後考慮。
- 可以用佇列來存放以走訪的頂點。

# BFS

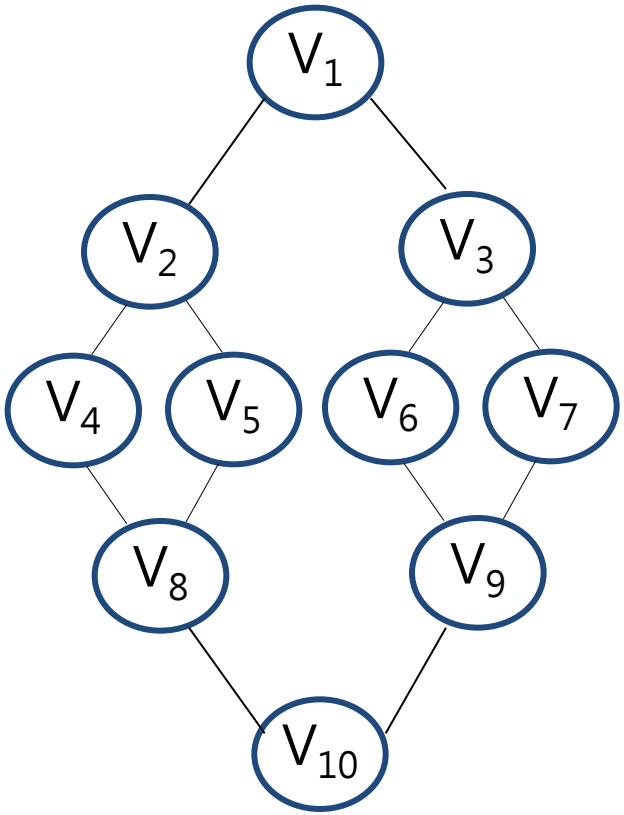
BFS 演算法可用下列流程圖表示：

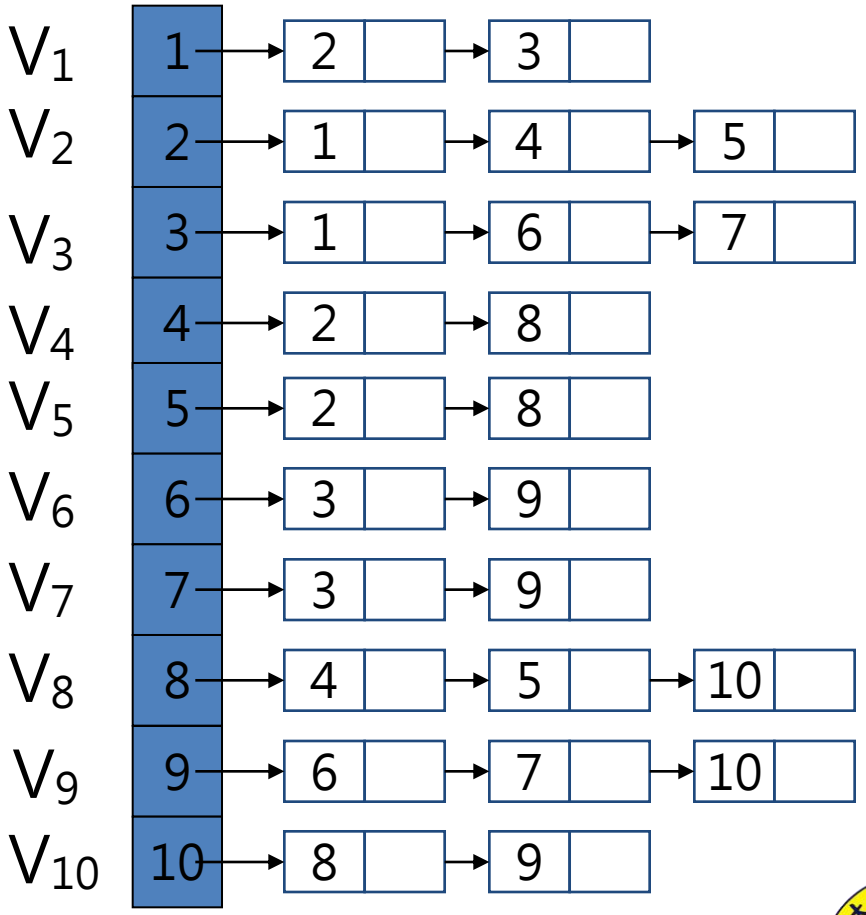





made by electron & free999

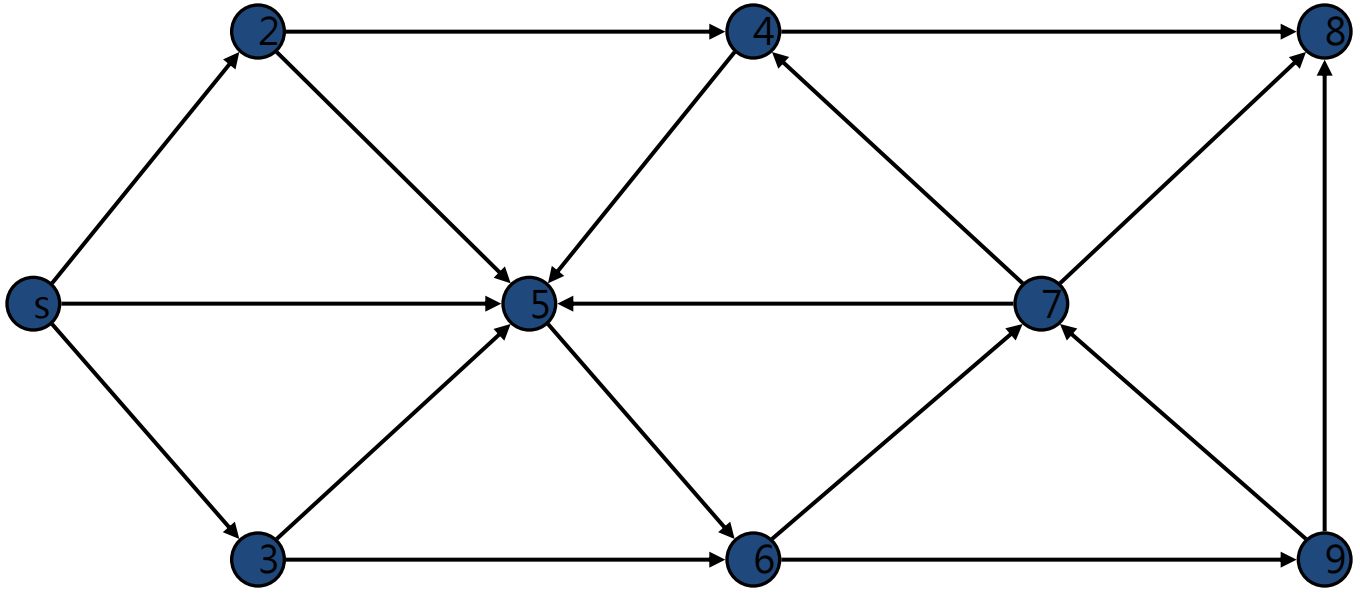
# BFS



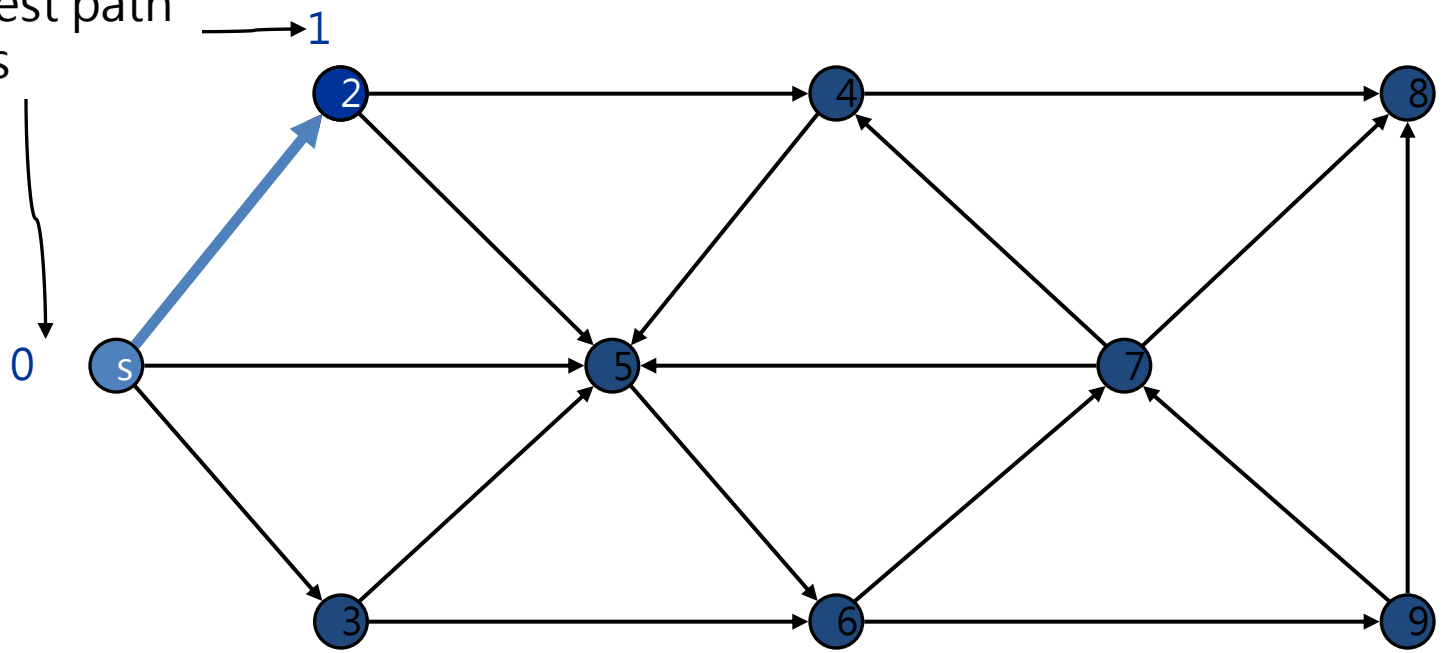




*made by electron & free999*

# BFS



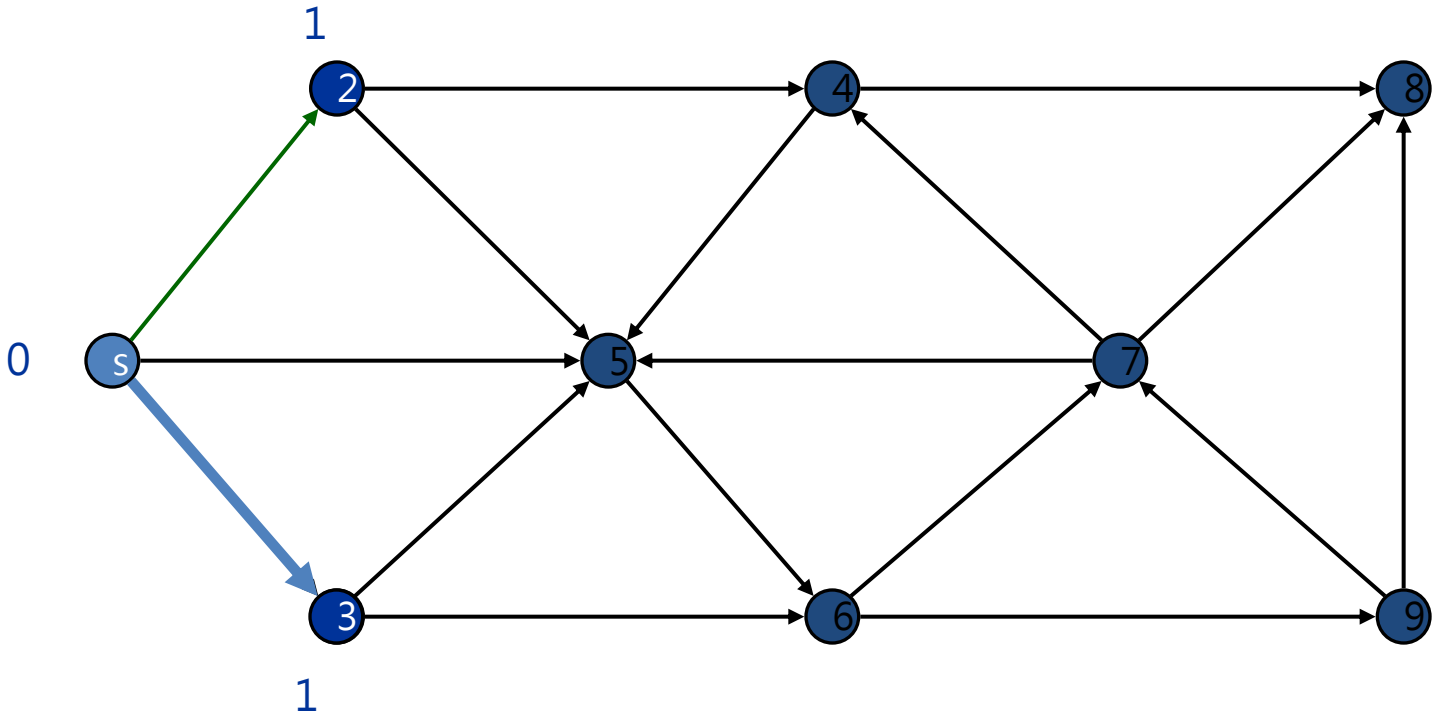
# BFS




| |
|---|
| Undiscovered |
| Discovered |
| Top of queue |
| Finished |

Queue: s

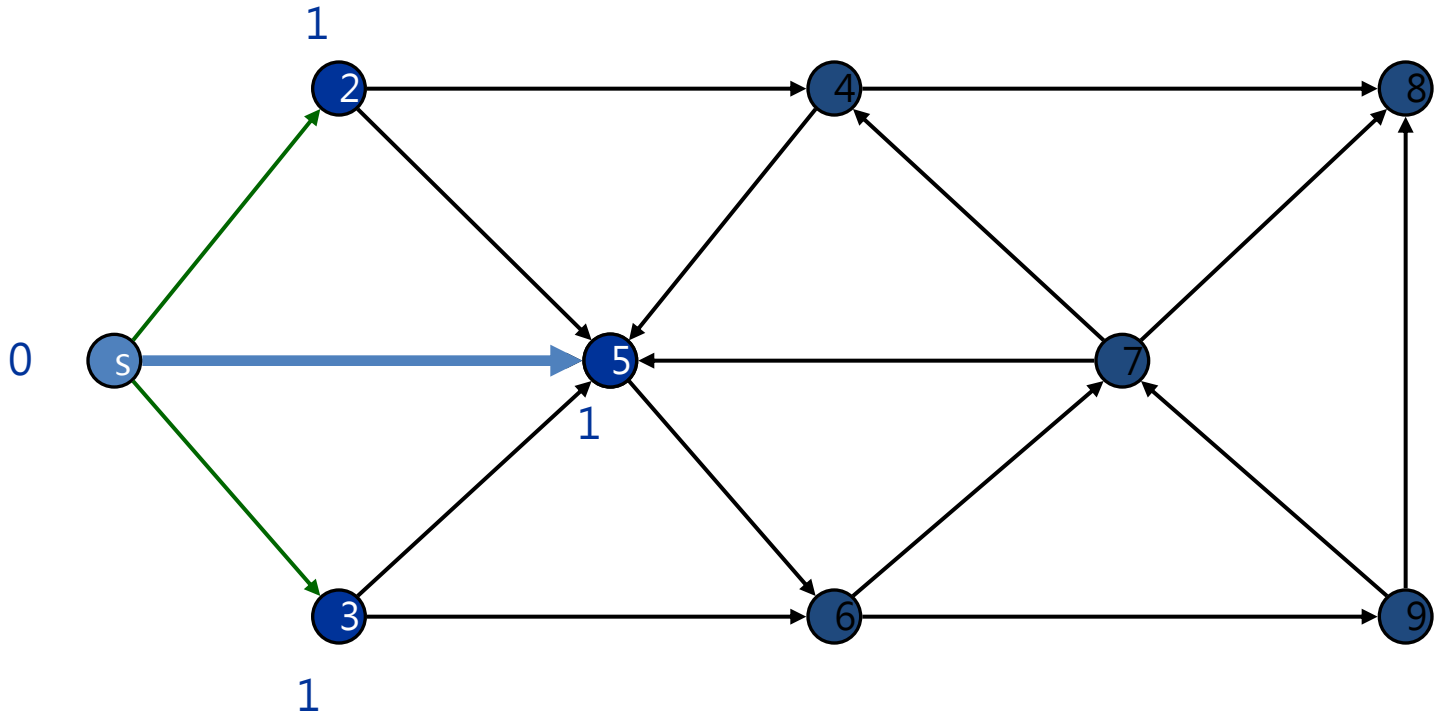
# BFS




| |
|---|
| Undiscovered |
| Discovered |
| Top of queue |
| Finished |

Queue: s 2

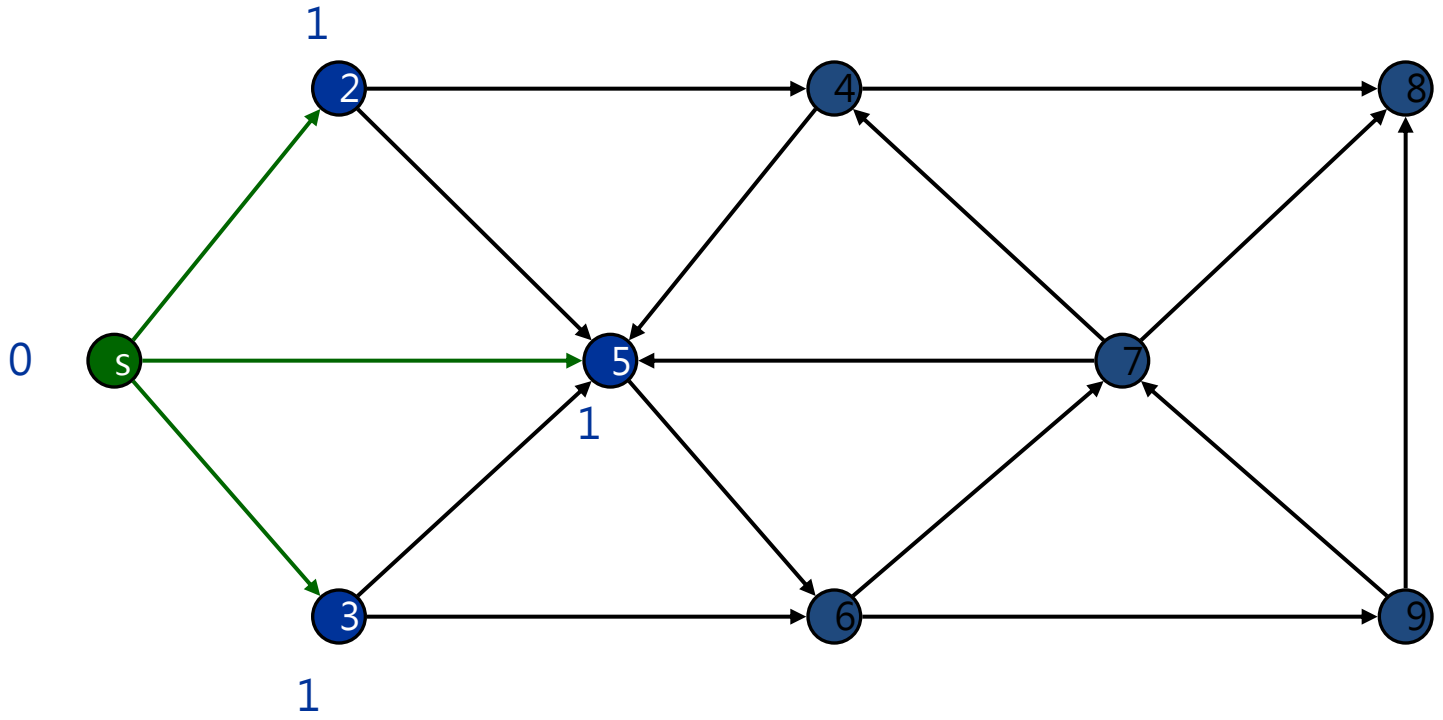
# BFS




| |
|---|
| Undiscovered |
| Discovered |
| Top of queue |
| Finished |

Queue: s 2 3

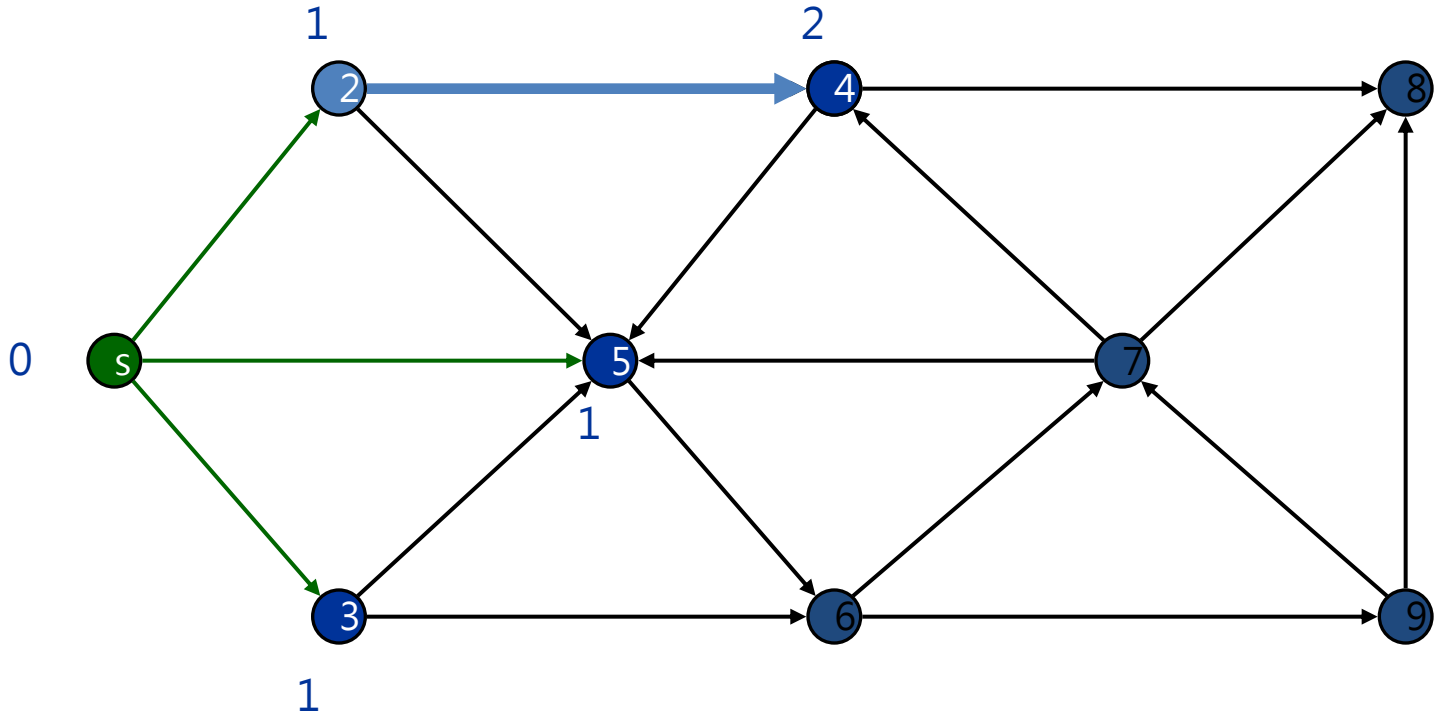
# BFS




| Undiscovered |
| --- |
| Discovered |
| Top of queue |
| Finished |

Queue: 2 3 5

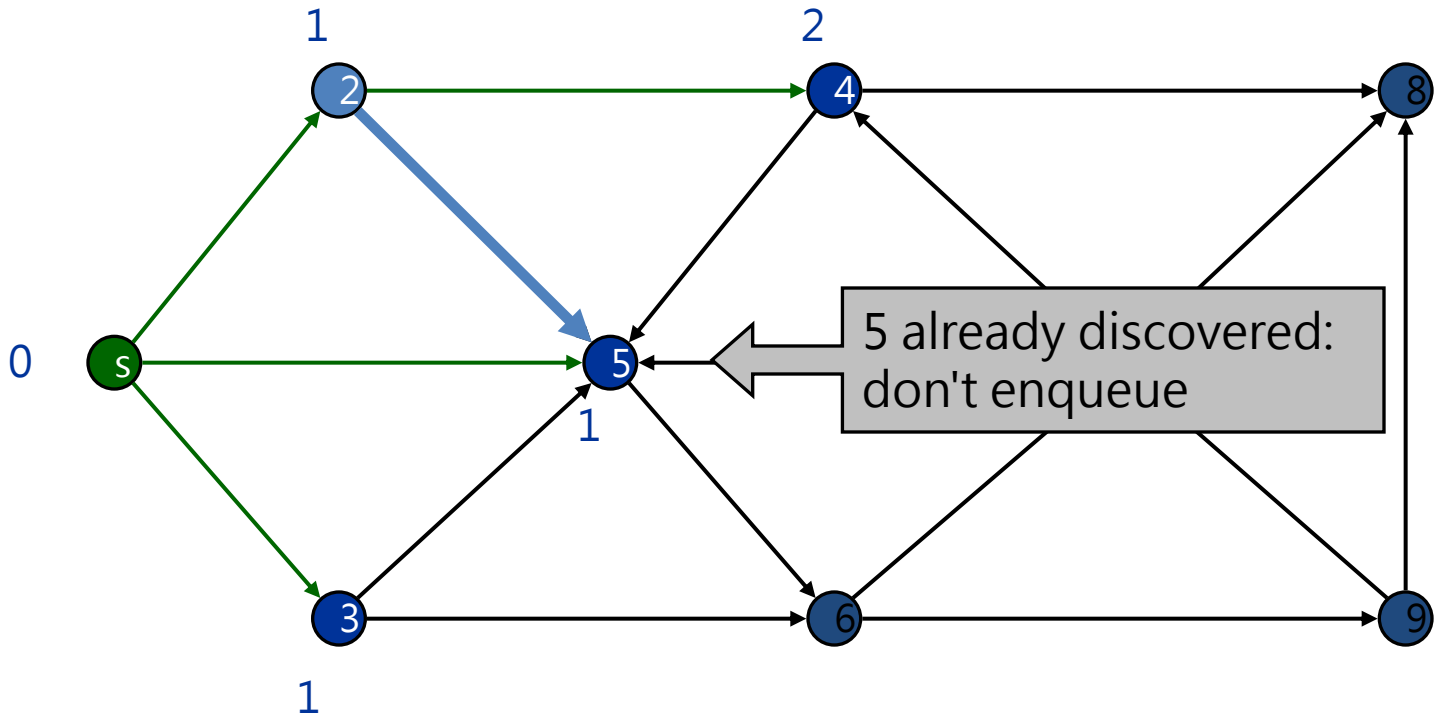
# BFS




| |
|---|
| Undiscovered |
| Discovered |
| Top of queue |
| Finished |

Queue: 2 3 5

# BFS




Undiscovered

Discovered

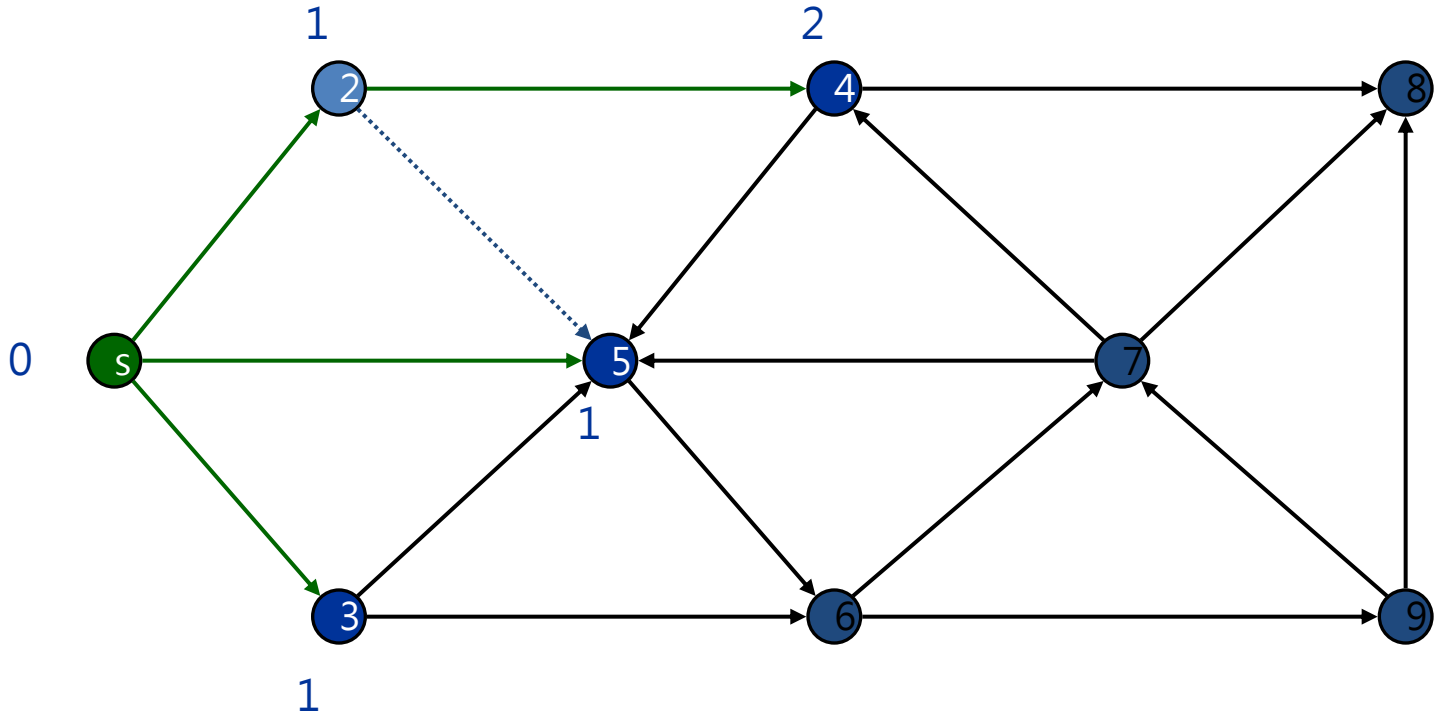
Top of queue

Finished

Queue: 2 3 5 4

*made by electron & free999*

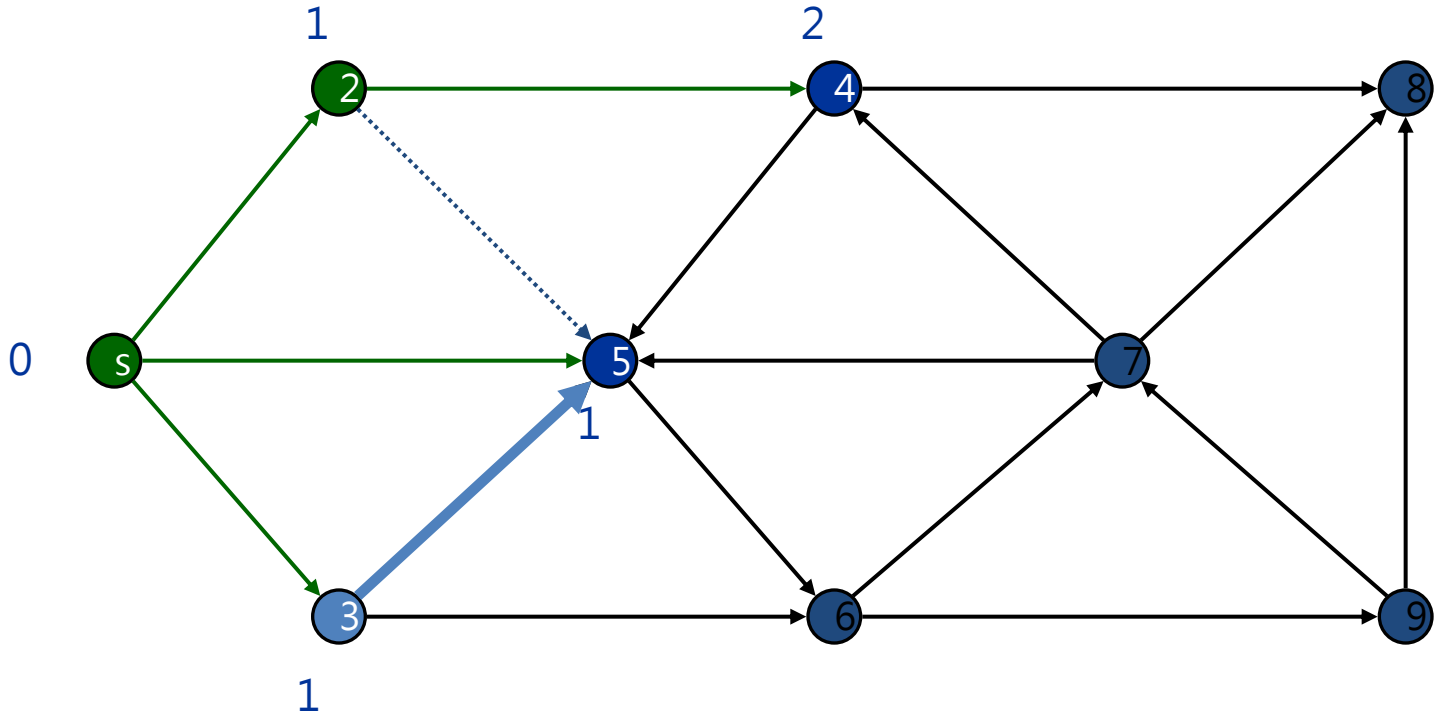
# BFS




| |
|---|
| Undiscovered |
| Discovered |
| Top of queue |
| Finished |

Queue: 2 3 5 4

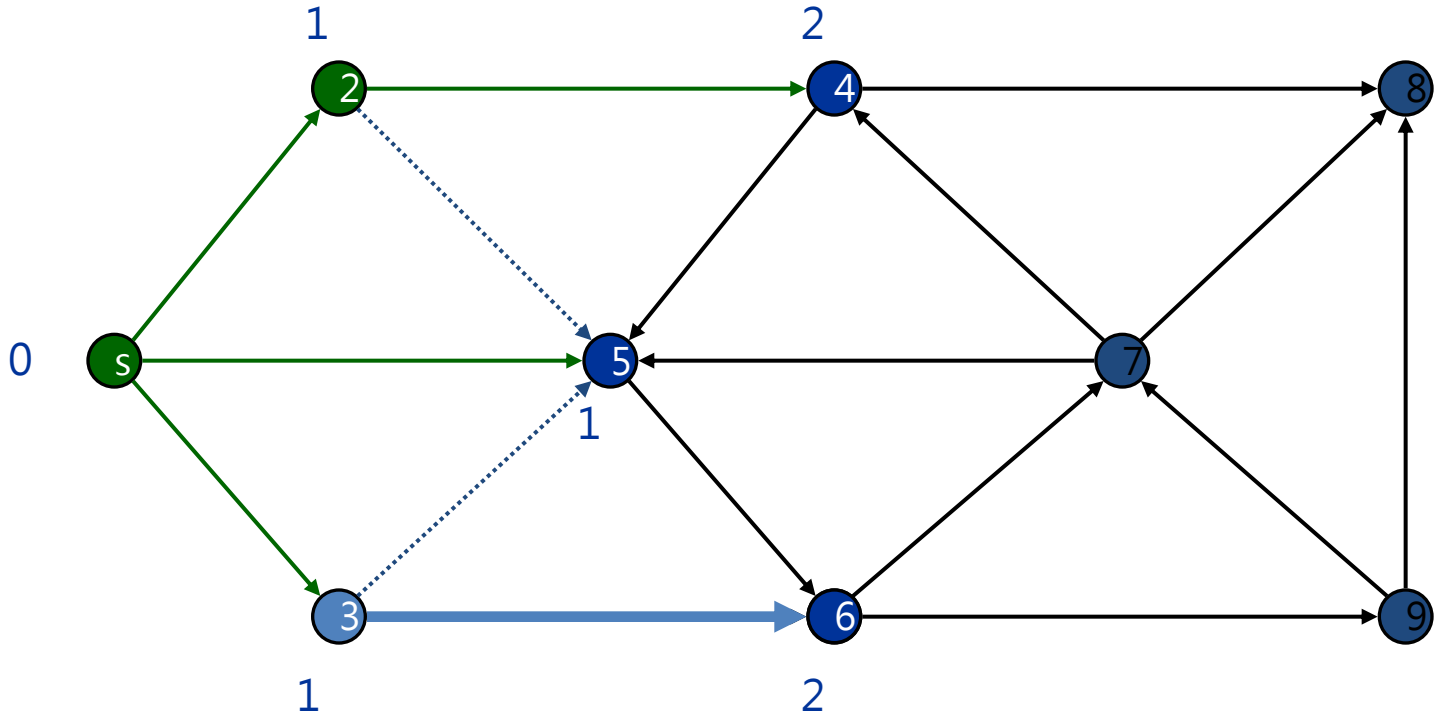
# BFS




| |
|---|
| Undiscovered |
| Discovered |
| Top of queue |
| Finished |

Queue: 3 5 4

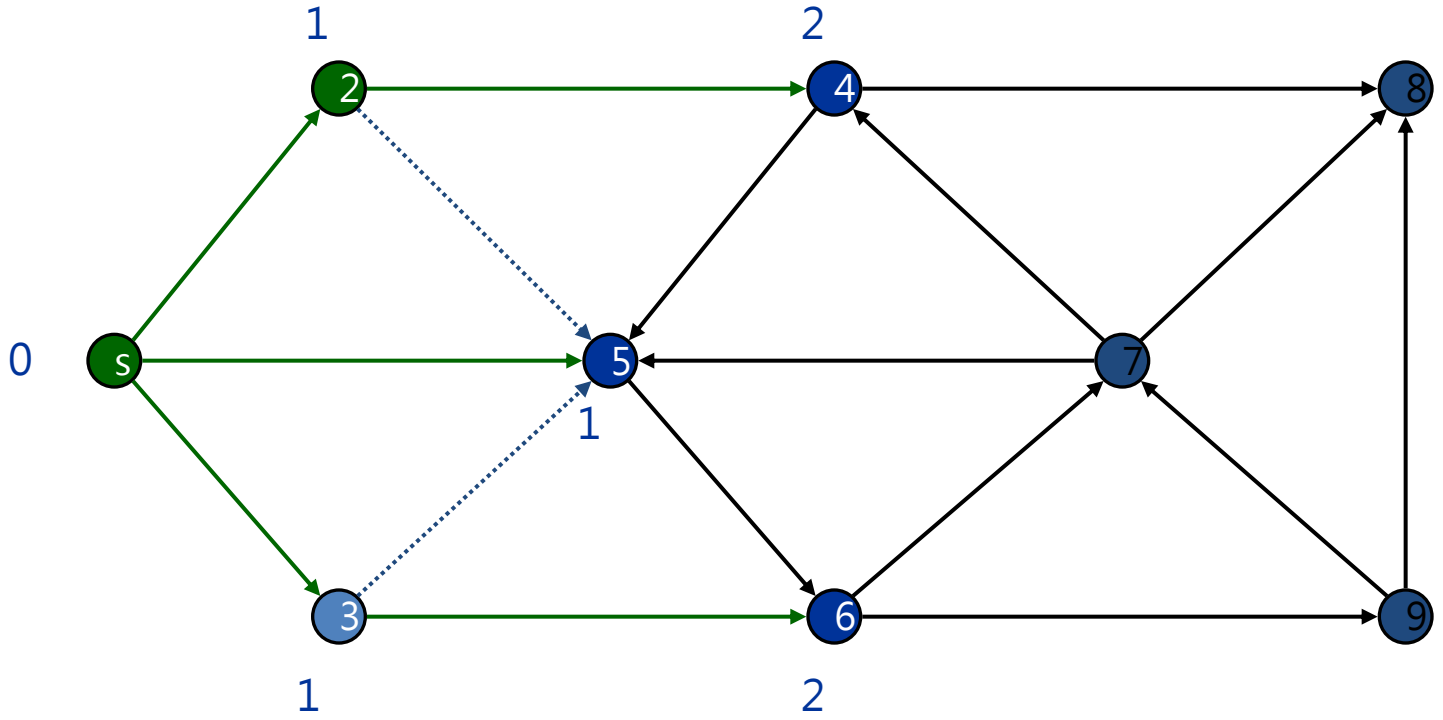
# BFS




| |
|---|
| Undiscovered |
| Discovered |
| Top of queue |
| Finished |

Queue: 3 5 4

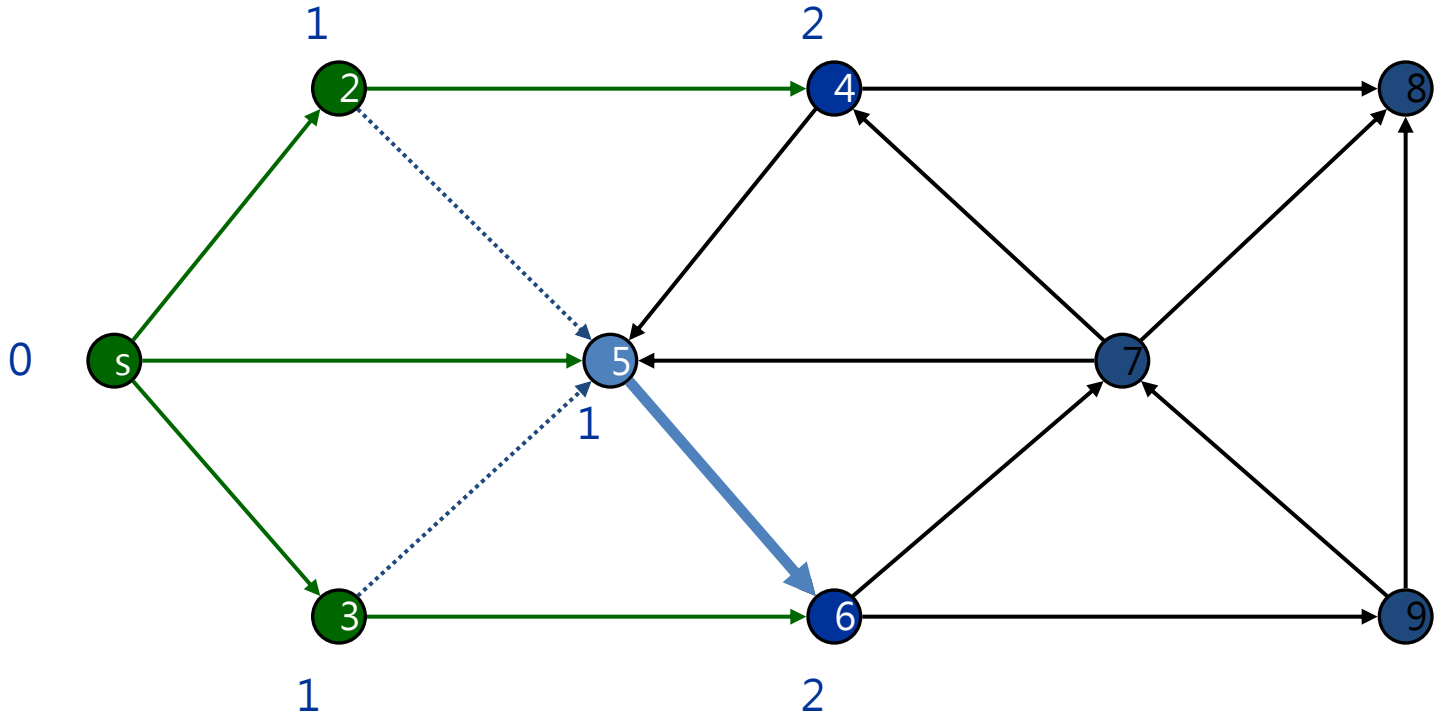
# BFS




| |
|---|
| Undiscovered |
| Discovered |
| Top of queue |
| Finished |

Queue: 3 5 4 6

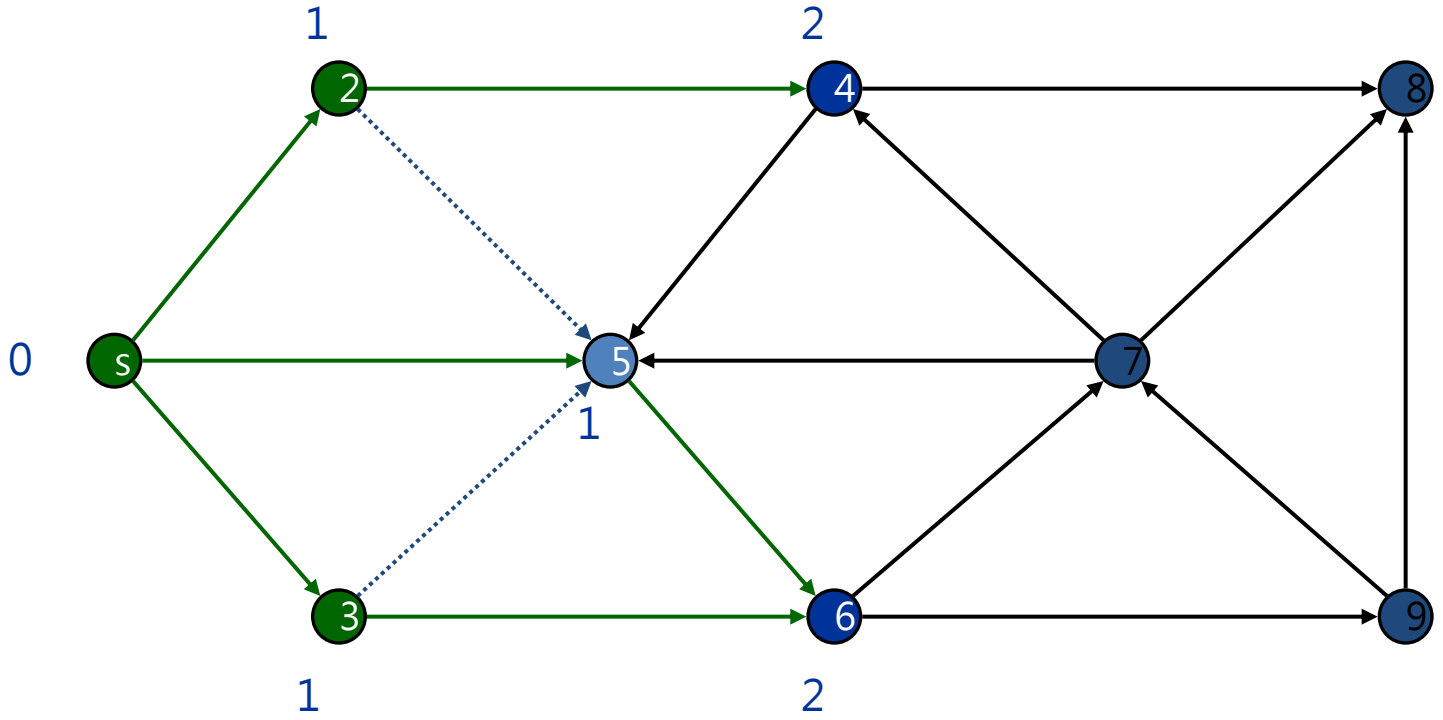
# BFS




| |
|---|
| Undiscovered |
| Discovered |
| Top of queue |
| Finished |

Queue: 5 4 6

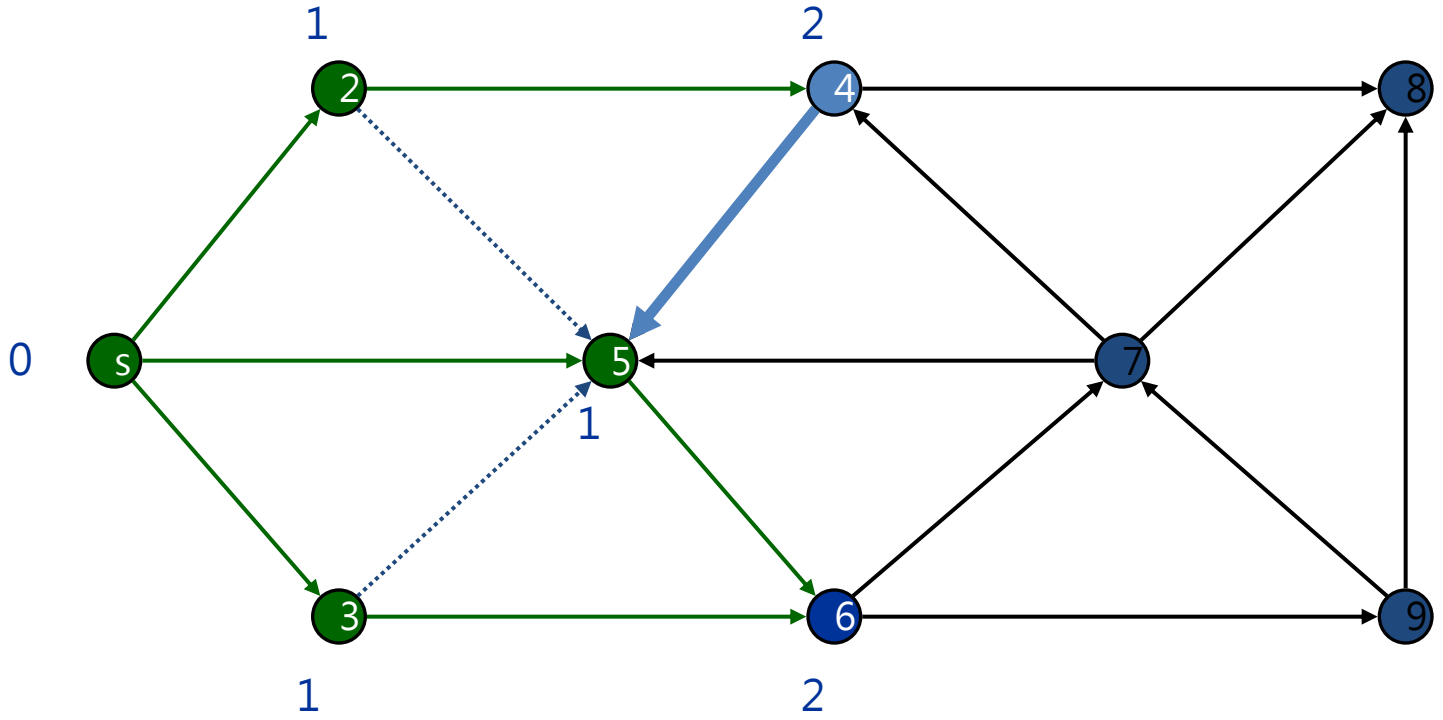
# BFS




| |
|---|
| Undiscovered |
| Discovered |
| Top of queue |
| Finished |

Queue: 5 4 6

# BFS

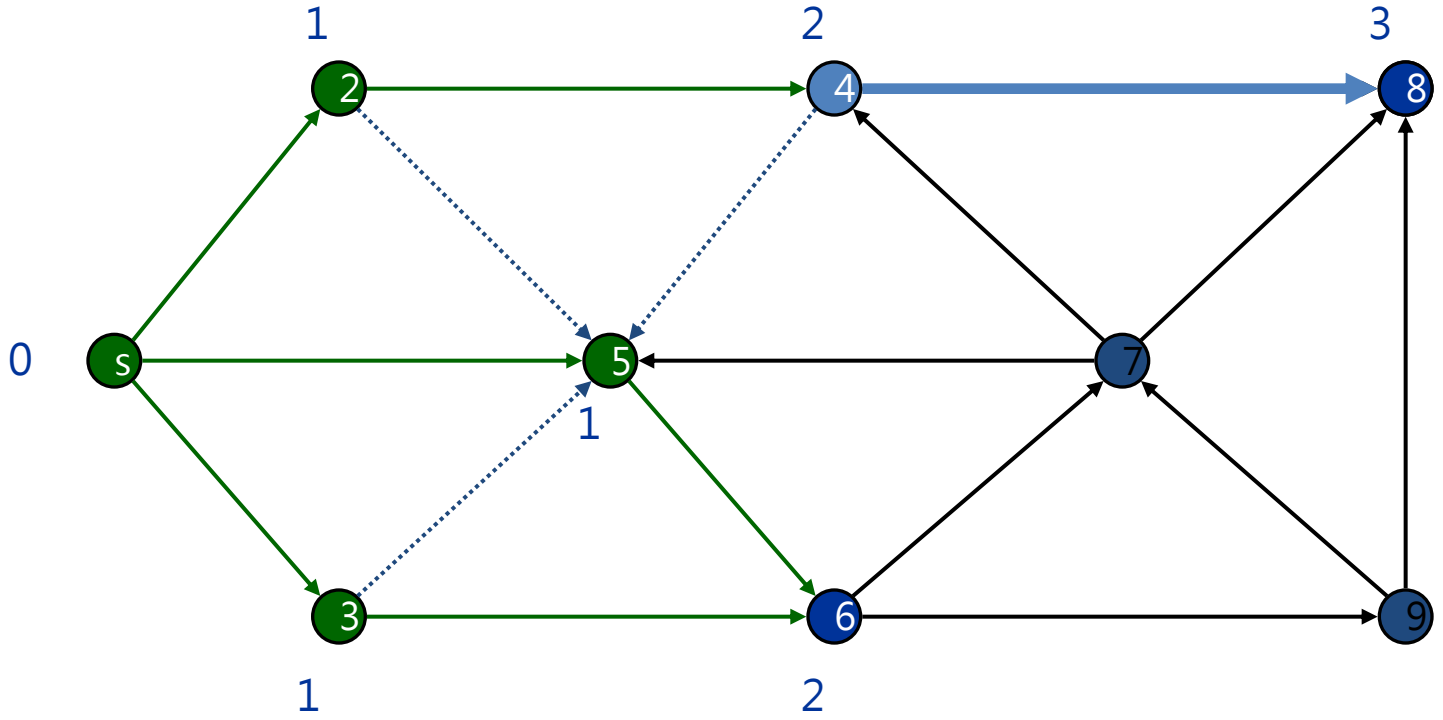



| |
|---|
| Undiscovered |
| Discovered |
| Top of queue |
| Finished |

Queue: 4 6

*made by electron & free999*

# BFS

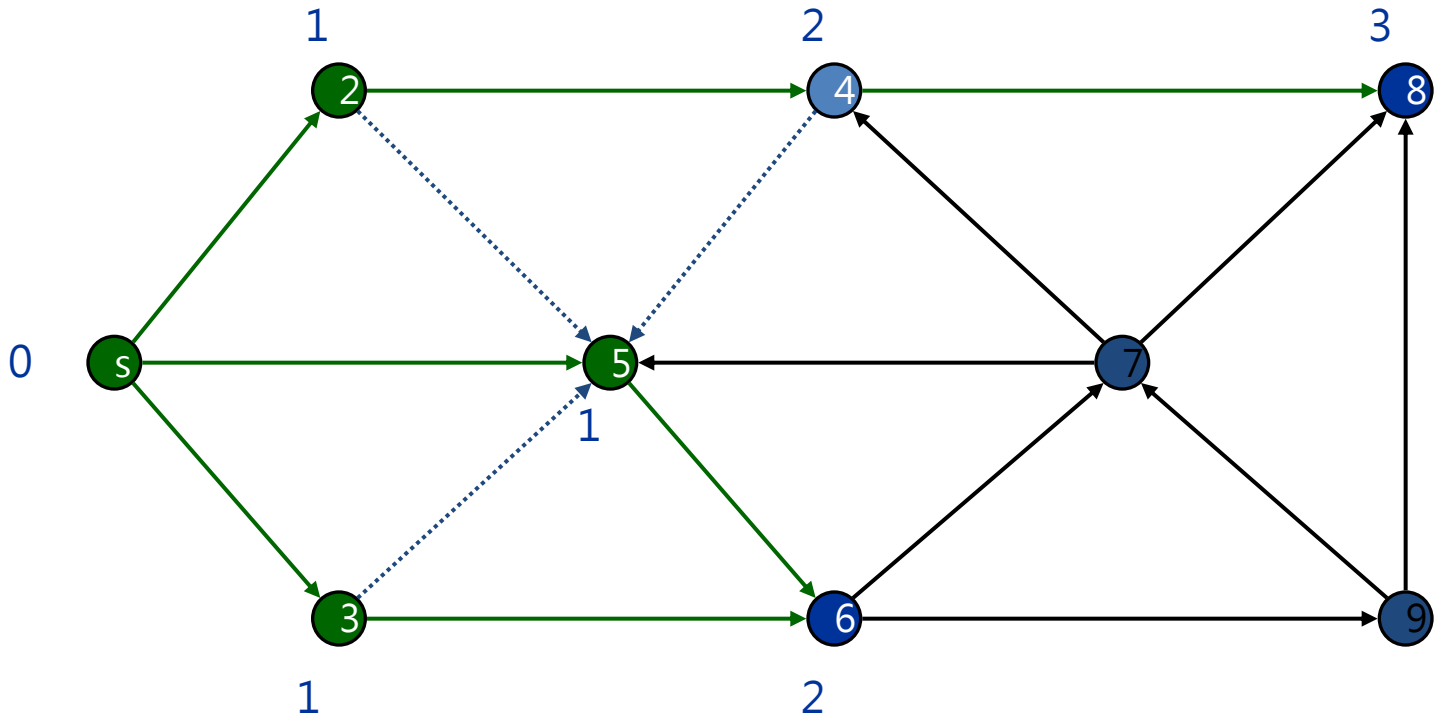



| |
|---|
| Undiscovered |
| Discovered |
| Top of queue |
| Finished |

Queue: 4 6

*made by electron & free999*

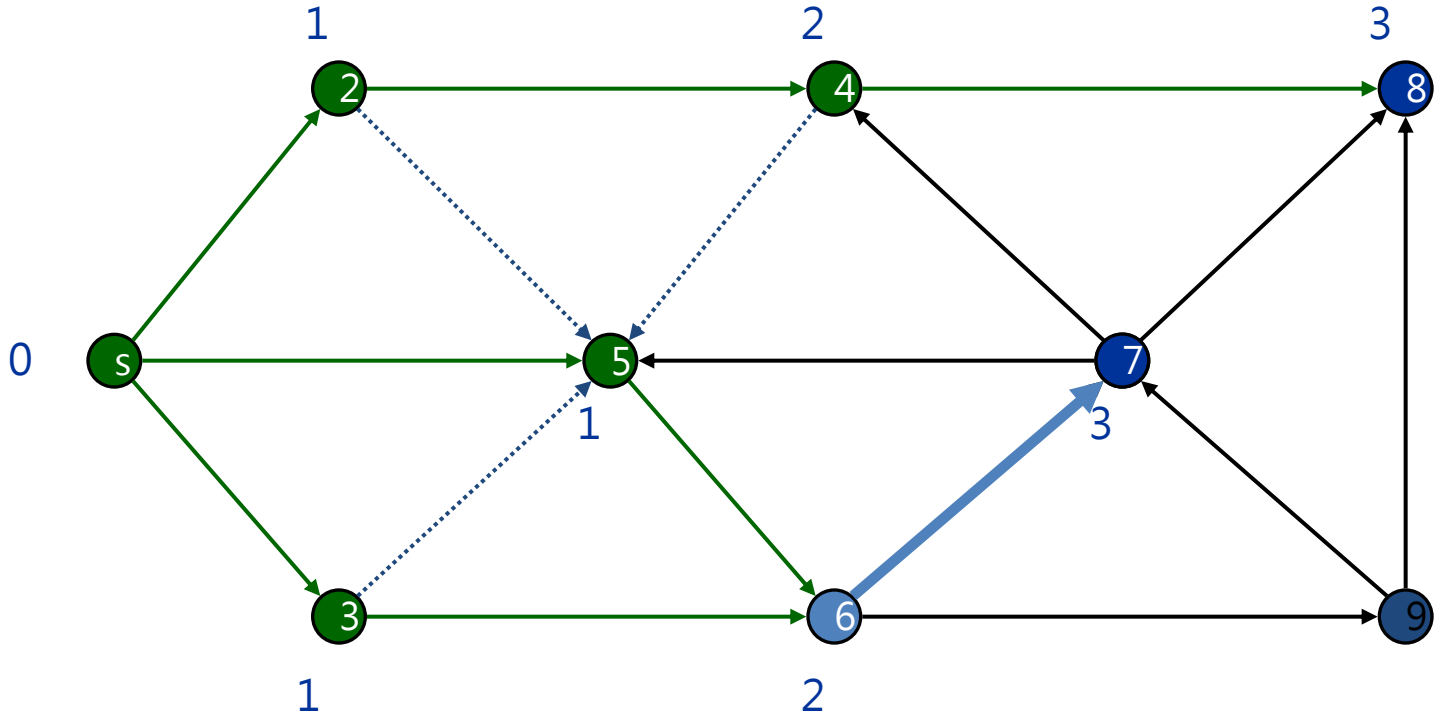
# BFS




| |
|---|
| Undiscovered |
| Discovered |
| Top of queue |
| Finished |

Queue: 4 6 8

# BFS




Undiscovered

Discovered

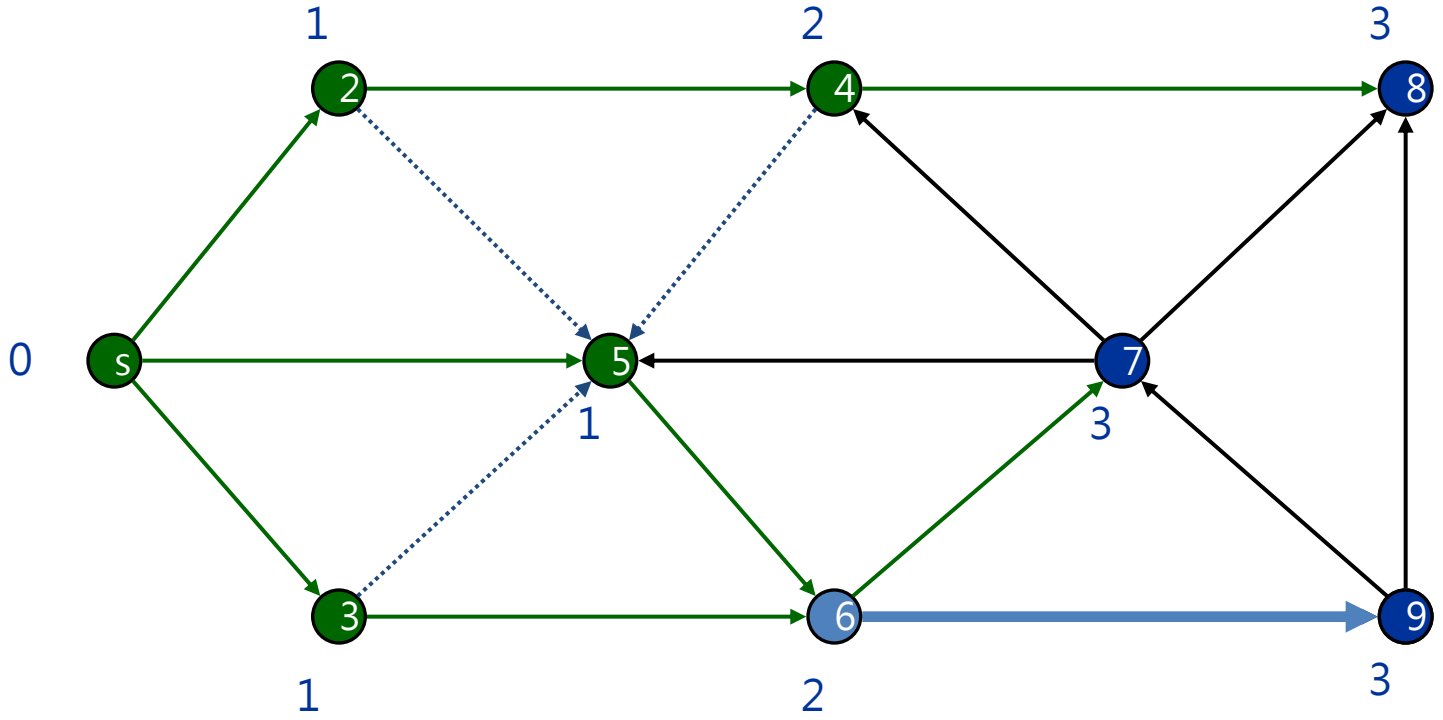
Top of queue

Finished

Queue: 6 8

*made by electron & free999*

# BFS




Undiscovered

Discovered

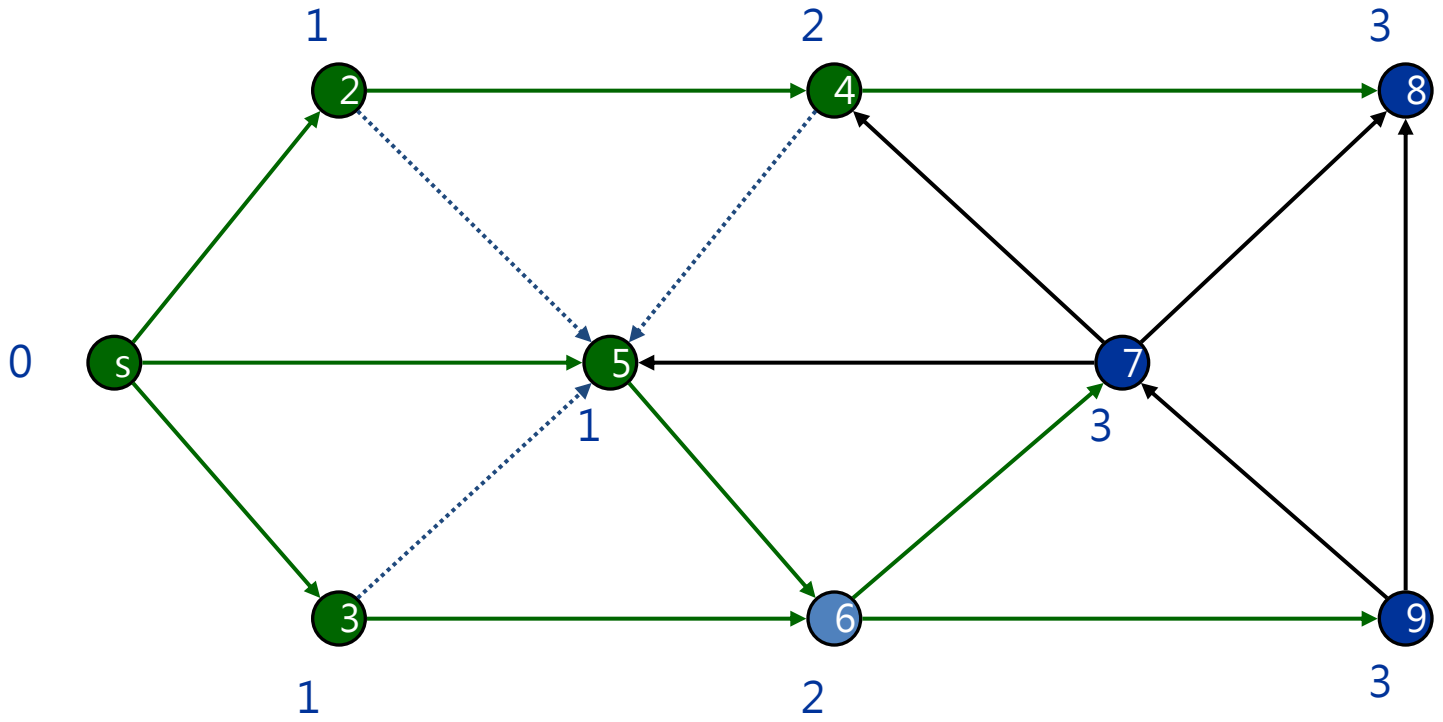
Top of queue

Finished

Queue: 6 8 7

*made by electron & free999*

# BFS




Undiscovered

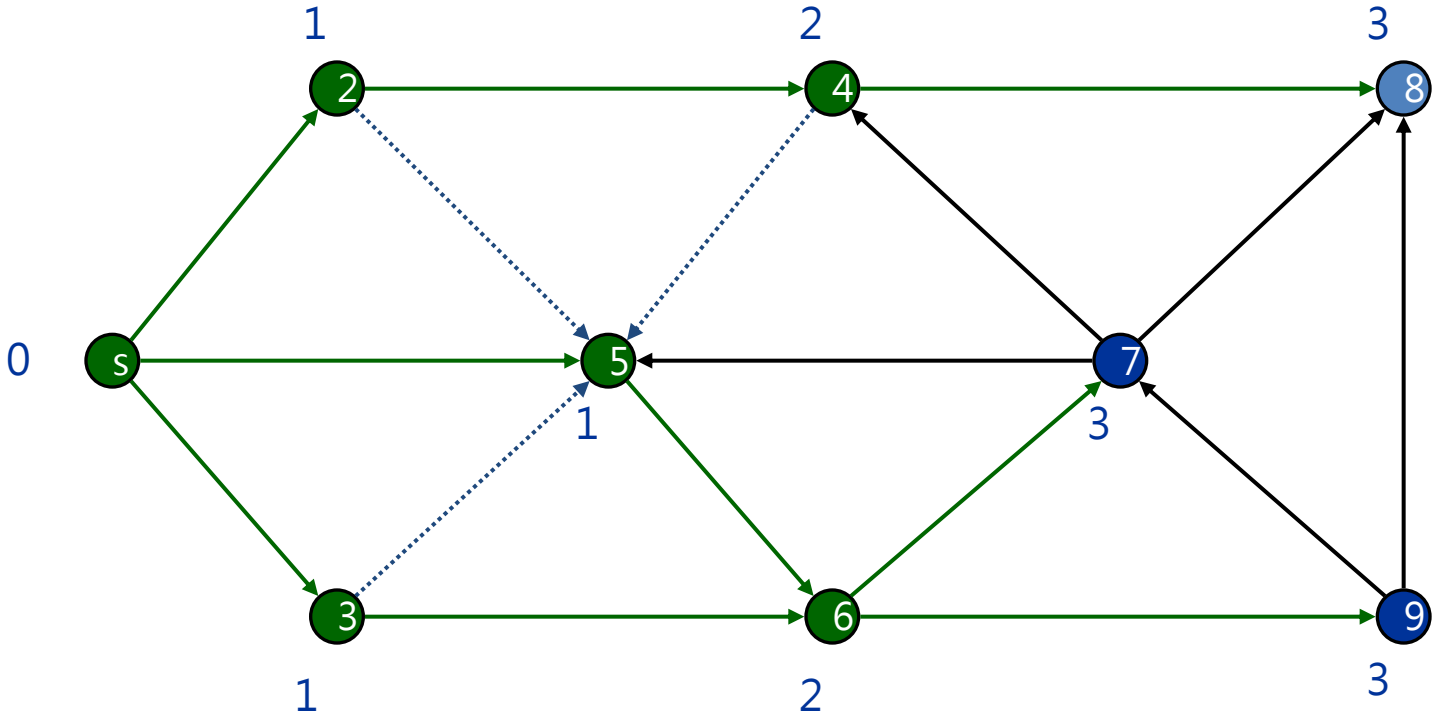
Discovered

Top of queue

Finished

Queue: 6 8 7 9

# BFS




Undiscovered

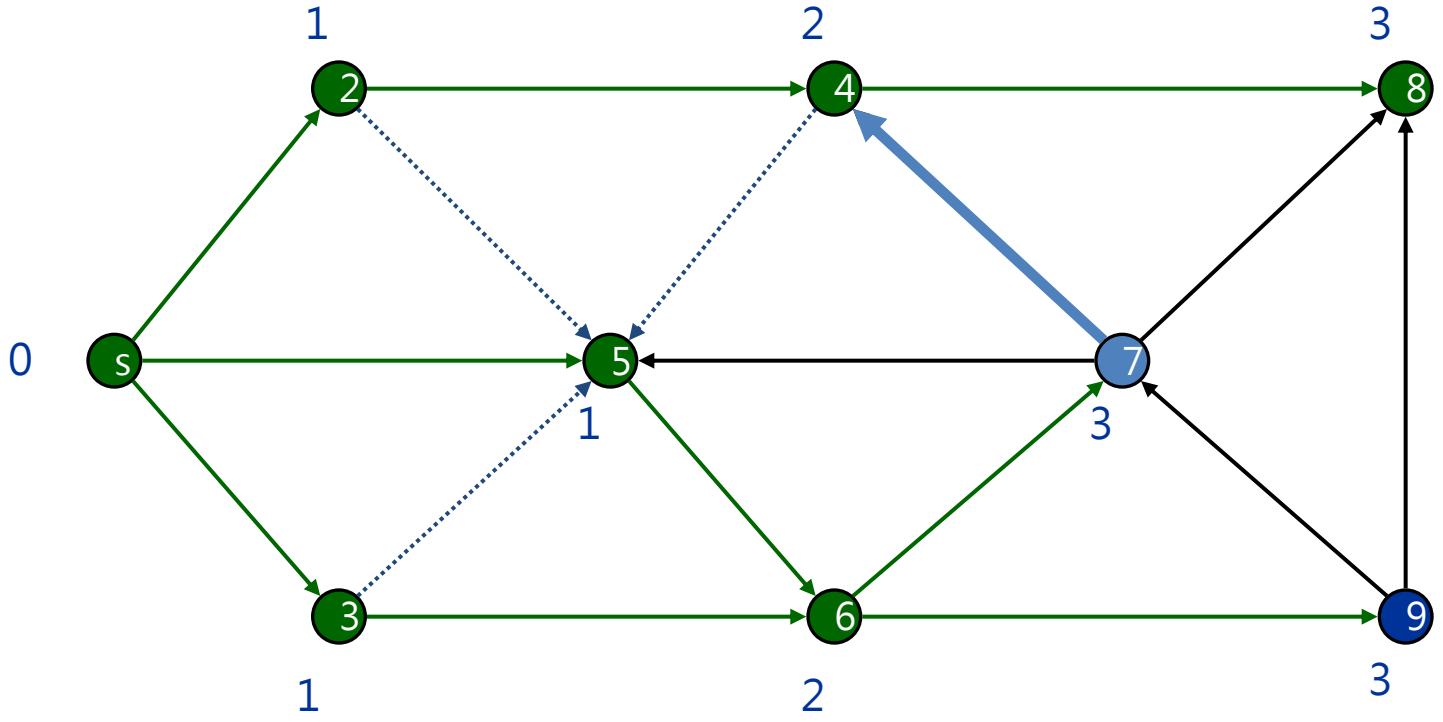
Discovered

Top of queue

Finished

Queue: 8 7 9

# BFS




Undiscovered

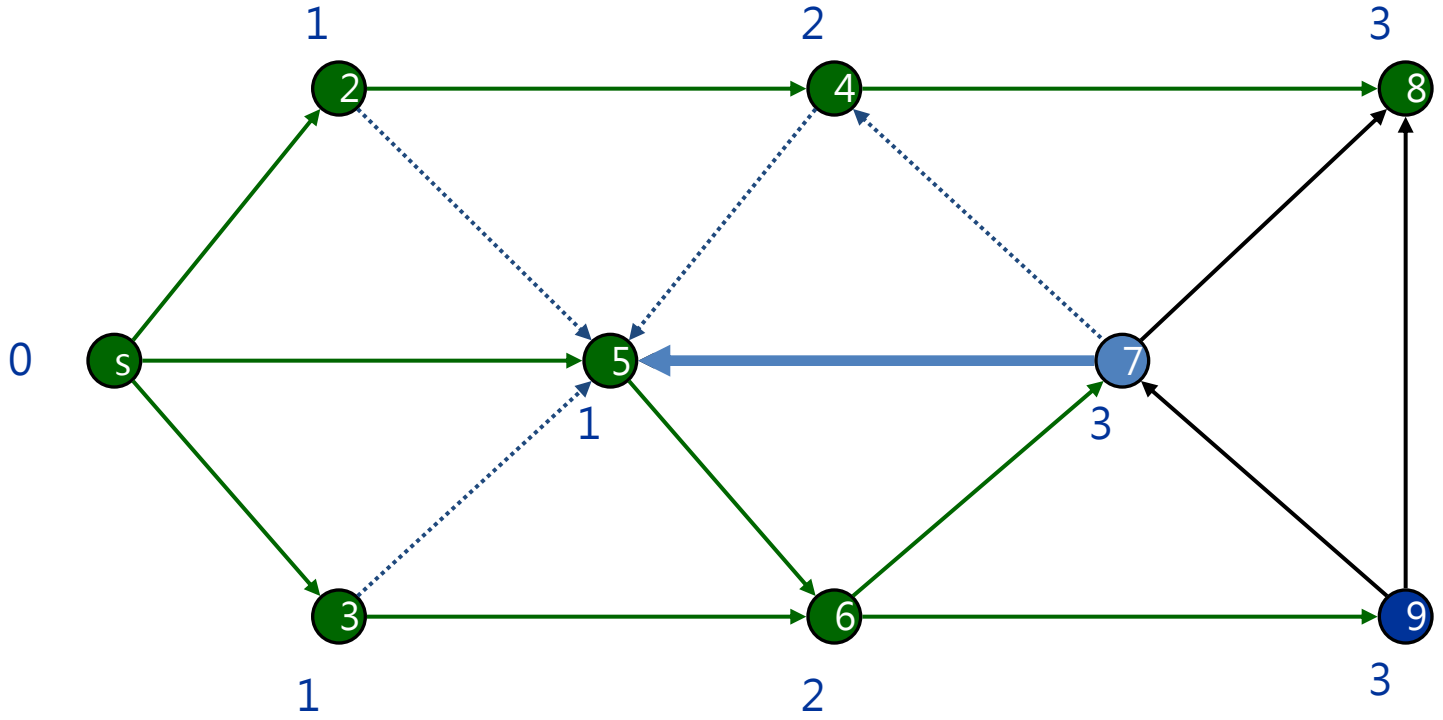
Discovered

Top of queue

Finished

Queue: 7 9

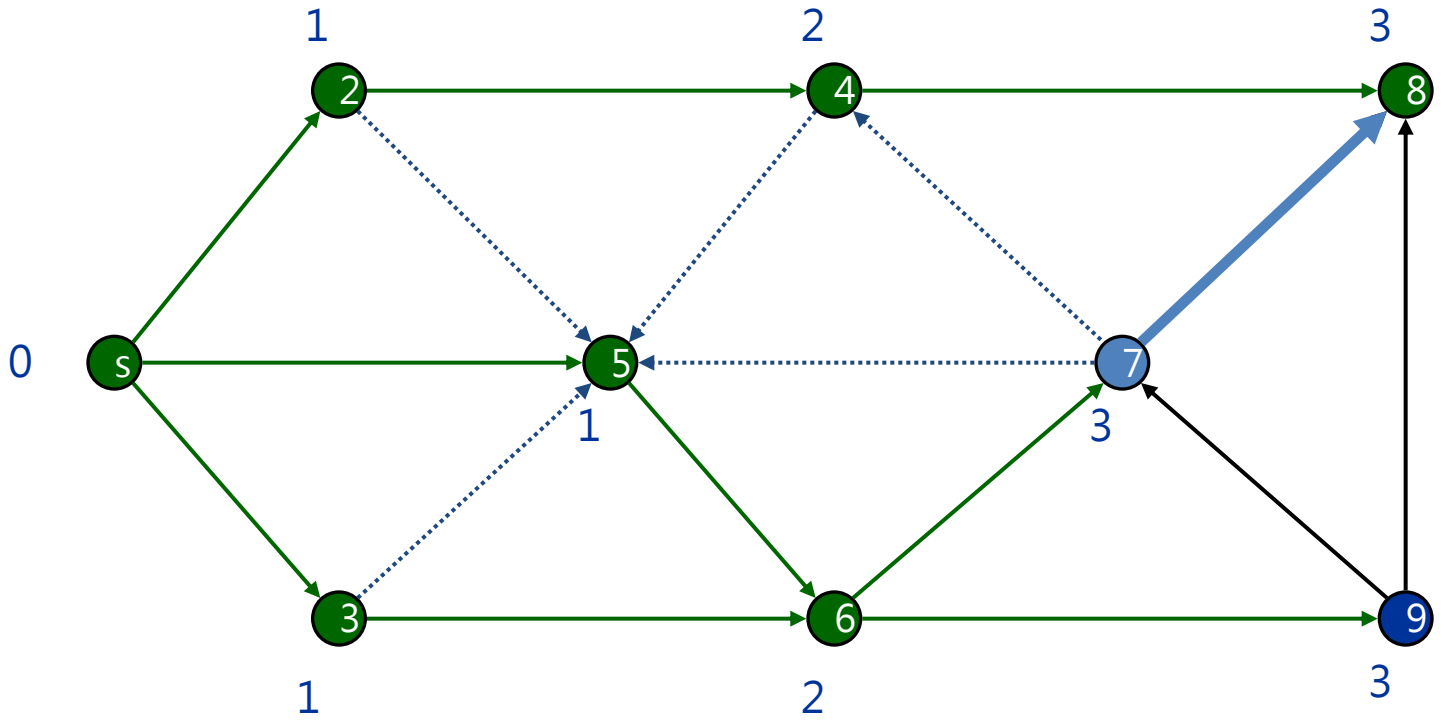
# BFS




| |
|---|
| Undiscovered |
| Discovered |
| Top of queue |
| Finished |

Queue: 7 9

# BFS




Undiscovered

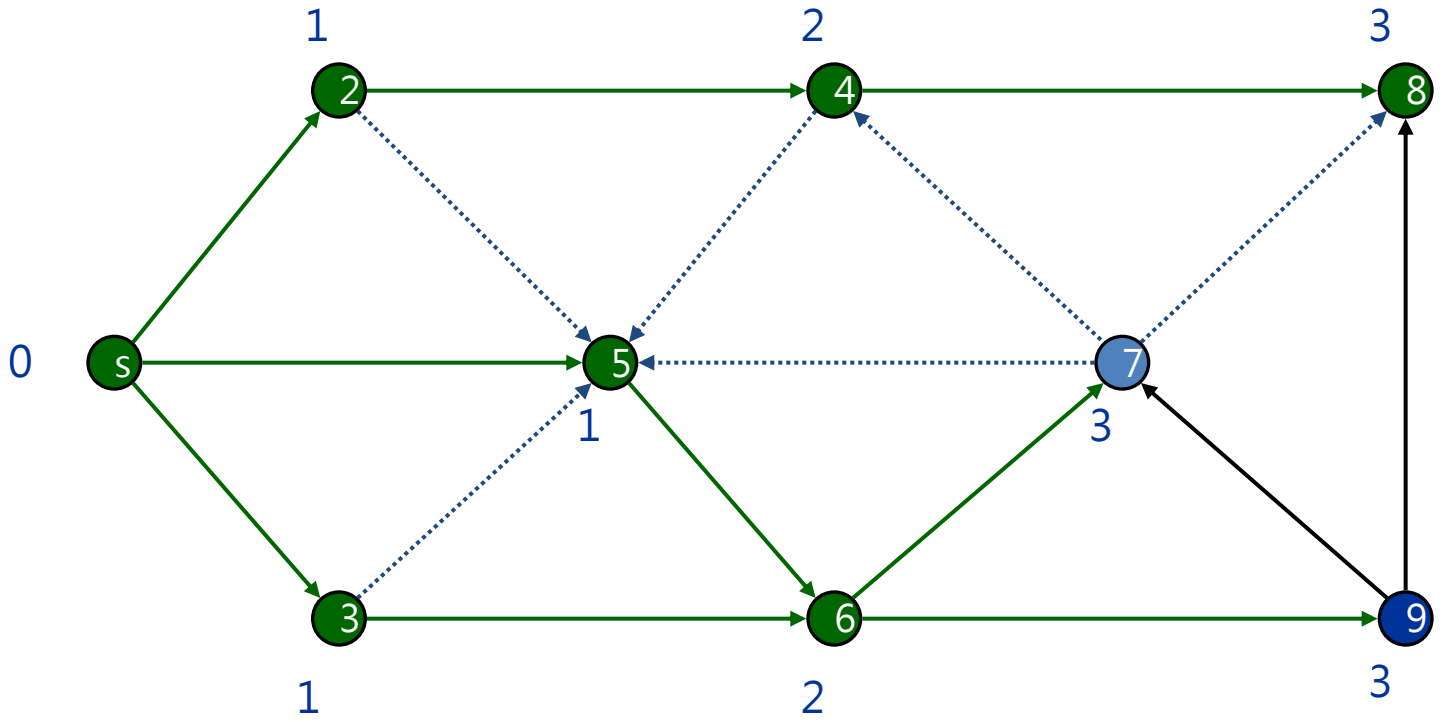
Discovered

Top of queue

Finished

Queue: 7 9

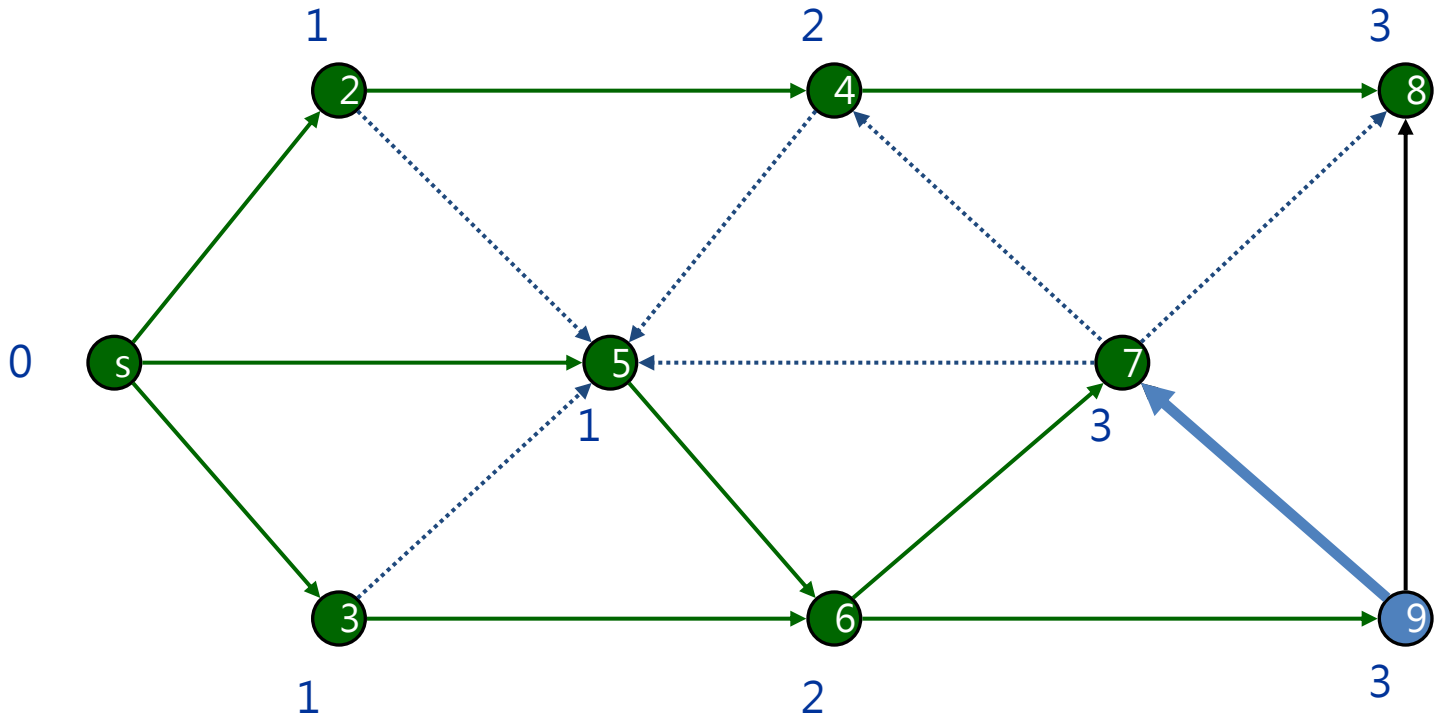
# BFS




| |
|---|
| Undiscovered |
| Discovered |
| Top of queue |
| Finished |

Queue: 7 9

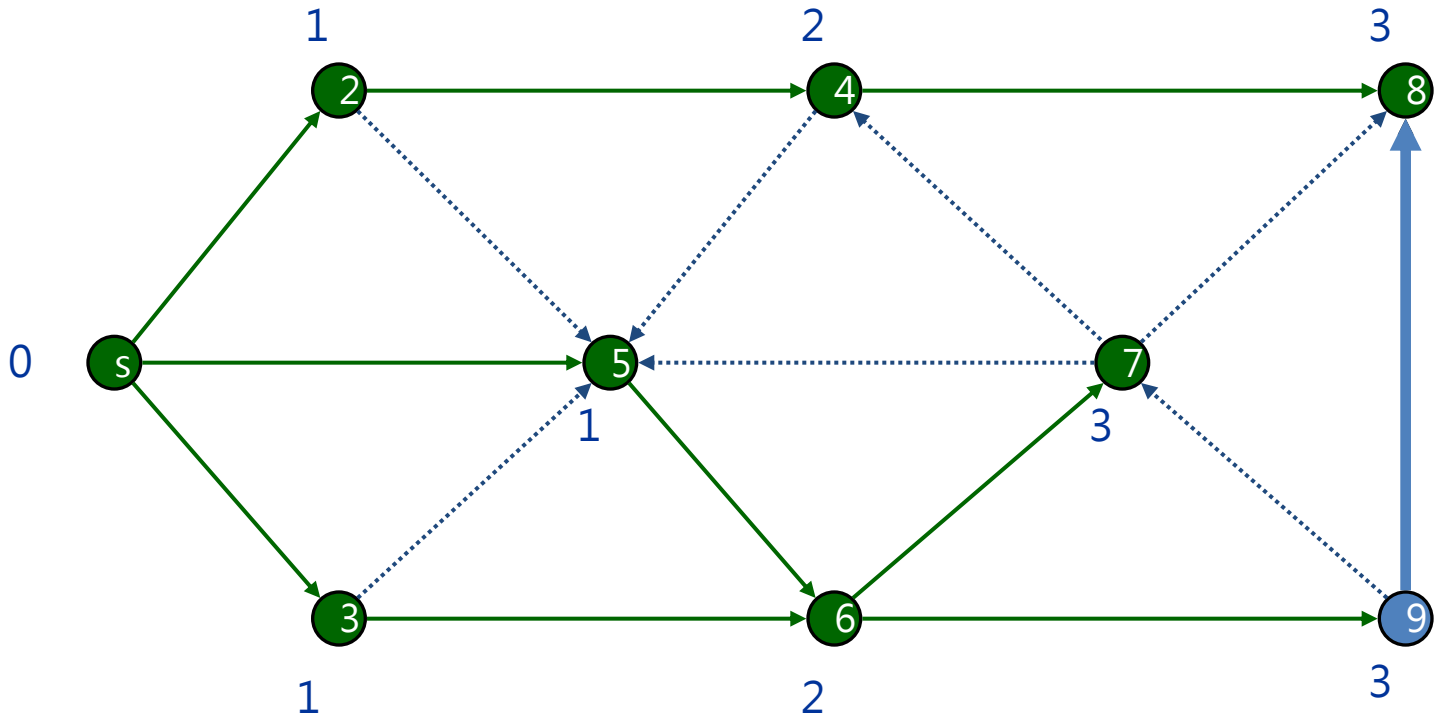
# BFS




| |
|---|
| Undiscovered |
| Discovered |
| Top of queue |
| Finished |

Queue: 9

# BFS




Undiscovered

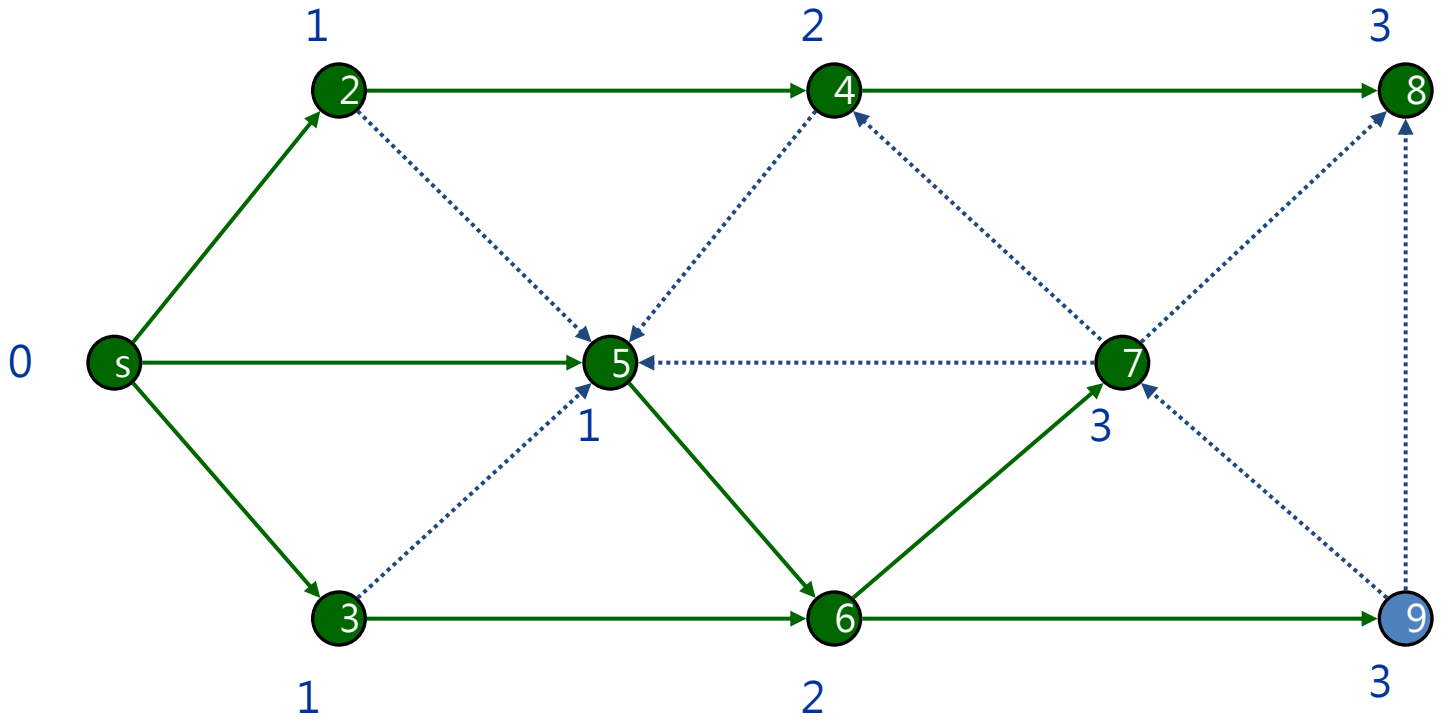
Discovered

Top of queue

Finished

Queue: 9

# BFS




Undiscovered

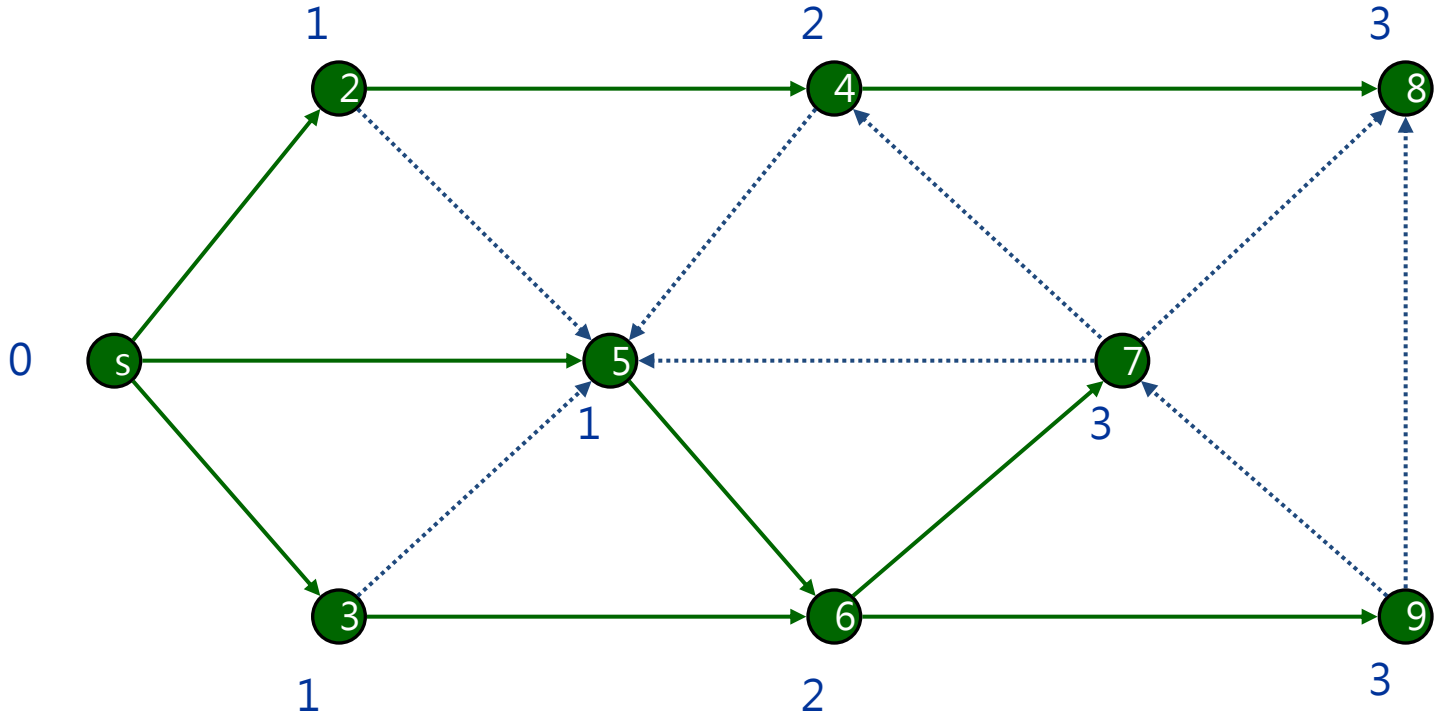
Discovered

Top of queue

Finished

Queue: 9

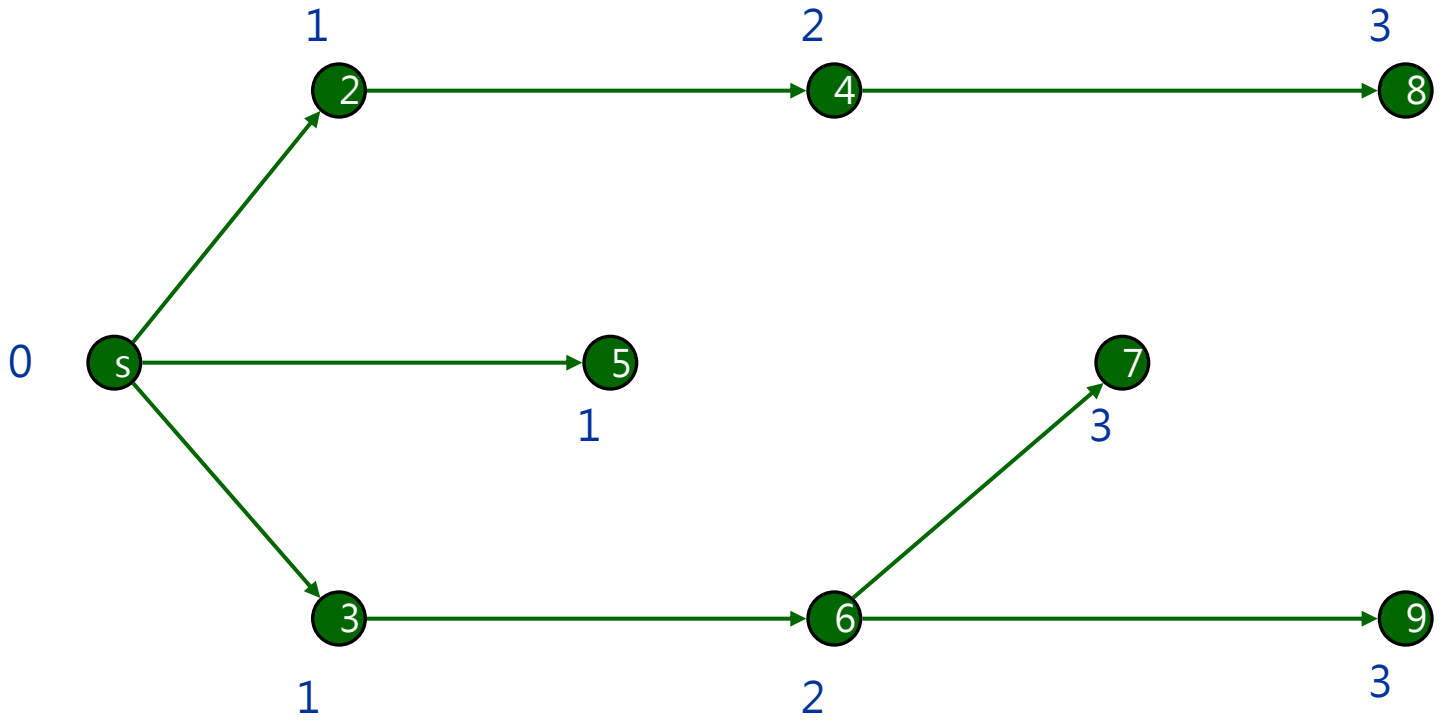
# BFS




| |
|---|
| Undiscovered |
| Discovered |
| Top of queue |
| Finished |

Queue:

# BFS




Level Graph

# BFS

```
1  int visited[n], i;
2  main()
3  {   int v;
4      for (i=0; i<n; i++) visited[i] = 0;
5      visited[0] = 1;
6      Add_Queue(0);
7      while (!Queue.empty())
8      {      v = Queue.front(), Queue.pop();
9             印出v;
10            for (所有與v相鄰的頂點w)
11            {      if (visited(w) == 0)
12                   {      visited(w) = 1;
13                          Add_Queue[w];}
14            }
14     }
15 }
```

# BFS



## Uva 532

You are trapped in a 3D dungeon and need to find the quickest way out! The dungeon is composed of unit cubes which may or may not be filled with rock. It takes one minute to move one unit north, south, east, west, up or down. You cannot move diagonally and the maze is surrounded by solid rock on all sides.

Is an escape possible? If yes, how long will it take?

# BFS

**Uva 532**

- **Input Specification**
  The input file consists of a number of dungeons. Each dungeon description starts with a line containing three integers *L*, *R* and *C* (all limited to 30 in size).
  *L* is the number of levels making up the dungeon.
  *R* and *C* are the number of rows and columns making up the plan of each level.

  Then there will follow *L* blocks of *R* lines each containing *C* characters. Each character describes one cell of the dungeon. A cell full of rock is indicated by a `#' and empty cells are represented by a `.'. Your starting position is indicated by `S' and the exit by the letter 'E'. There's a single blank line after each level. Input is terminated by three zeroes for *L*, *R* and *C*.
- **Output Specification**
  Each maze generates one line of output. If it is possible to reach the exit, print a line of the form

  Escaped in *x* minute(s).

  where *x* is replaced by the shortest time it takes to escape.
  If it is not possible to escape, print the line

  Trapped!

# BFS

## Uva 532

- Sample Input

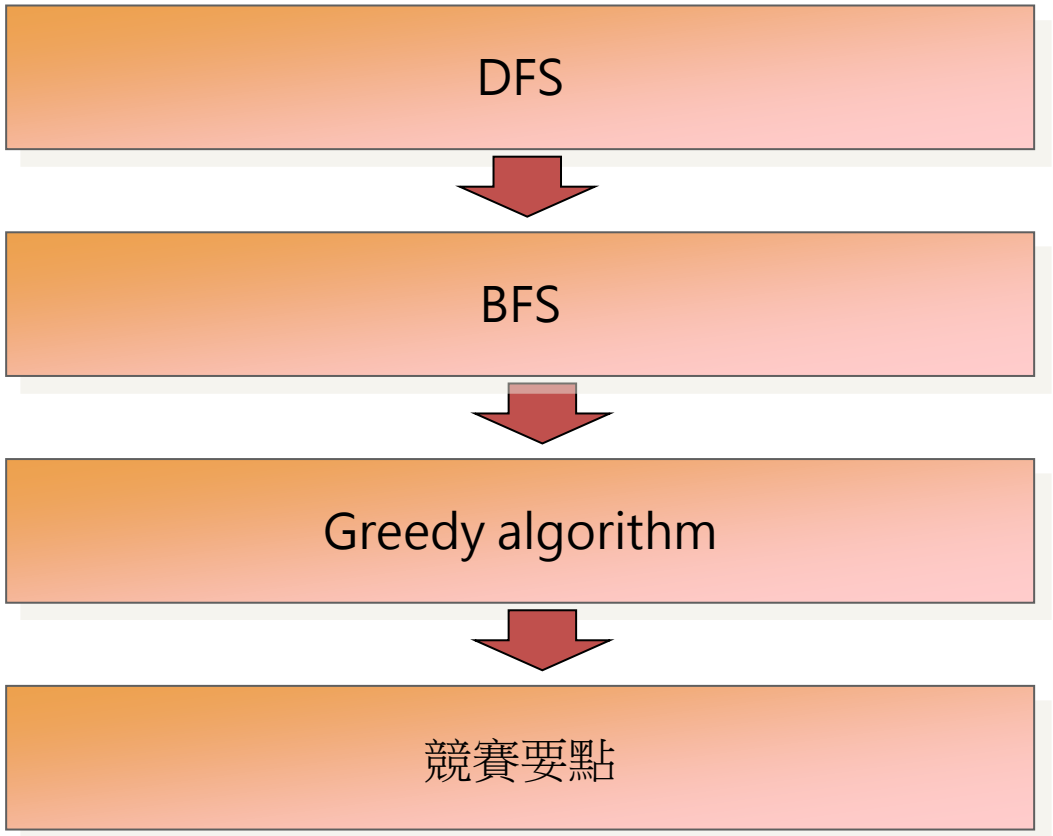
```
3 4 5
S....
.###.
.##..
###.#

#####
#####
##.##
##...

#####
#####
#.###
####E

1 3 3
S##
#E#
###

0 0 0
```

- Sample Output

```
Escaped in 11 minute(s).
Trapped!
```

*made by electron & free999*

# Outline






made by electron & free999

# Greedy Strategy

- Greedy Strategy
  - Choose an Optimal Solution in Now-Running Step
  - **Greedy algorithms are simple and straightforward**. They are shortsighted in their approach in the sense that they take decisions on the basis of information at hand without worrying about the effect these decisions may have in the future. They are easy to invent, easy to implement and most of the time quite efficient. Many problems cannot be solved correctly by greedy approach. Greedy algorithms are used to solve optimization problems

# Greedy Strategy

- Example
  - Minimum Number of Changing Problem
    - Make a change of a given amount using the smallest possible number of coins.
  - Available Coins
    - dollars (100 cents)
    - quarters (25 cents)
    - dimes (10 cents)
    - nickels (5 cents)
    - pennies (1 cent)
  - **Make a change of a given amount using the smallest possible number of coins**

# Example - 3

## POJ 3617 Best Cow Line

### Problem Description

FJ is about to take his $N$ ($1 \le N \le 2{,}000$) cows to the annual"Farmer of the Year" competition. In this contest every farmer arranges his cows in a line and herds them past the judges.

The contest organizers adopted a new registration scheme this year: simply register the initial letter of every cow in the order they will appear (i.e., If FJ takes Bessie, Sylvia, and Dora in that order he just registers BSD). After the registration phase ends, every group is judged in increasing lexicographic order according to the string of the initials of the cows' names.

FJ is very busy this year and has to hurry back to his farm, so he wants to be judged as early as possible. He decides to rearrange his cows, who have already lined up, before registering them.

FJ marks a location for a new line of the competing cows. He then proceeds to marshal the cows from the old line to the new one by repeatedly sending either the first or last cow in the (remainder of the) original line to the end of the new line. When he's finished, FJ takes his cows for registration in this new order.

Given the initial order of his cows, determine the least lexicographic string of initials he can make this way.

# Example - 3

## POJ 3617 Best Cow Line

### IO Description

The number of participants, n: 3 ≤ n ≤ 10000.
The distance (measured in centimeters), d: 500 ≤ d ≤ 200000.
The running speed (centimeters per second) of each participant, ri: 50 ≤ ri ≤ 1000.

### Input

Line 1: A single integer: *N*
Lines 2..*N*+1: Line *i*+1 contains a single initial ('A'..'Z') of the cow in the *i*th position in the original line

### Output

The least lexicographic string he can make. Every line (except perhaps the last one) contains the initials of 80 cows ('A'..'Z') in the new line.

# Example - 3

## POJ 3617 Best Cow Line

### Sample I/O

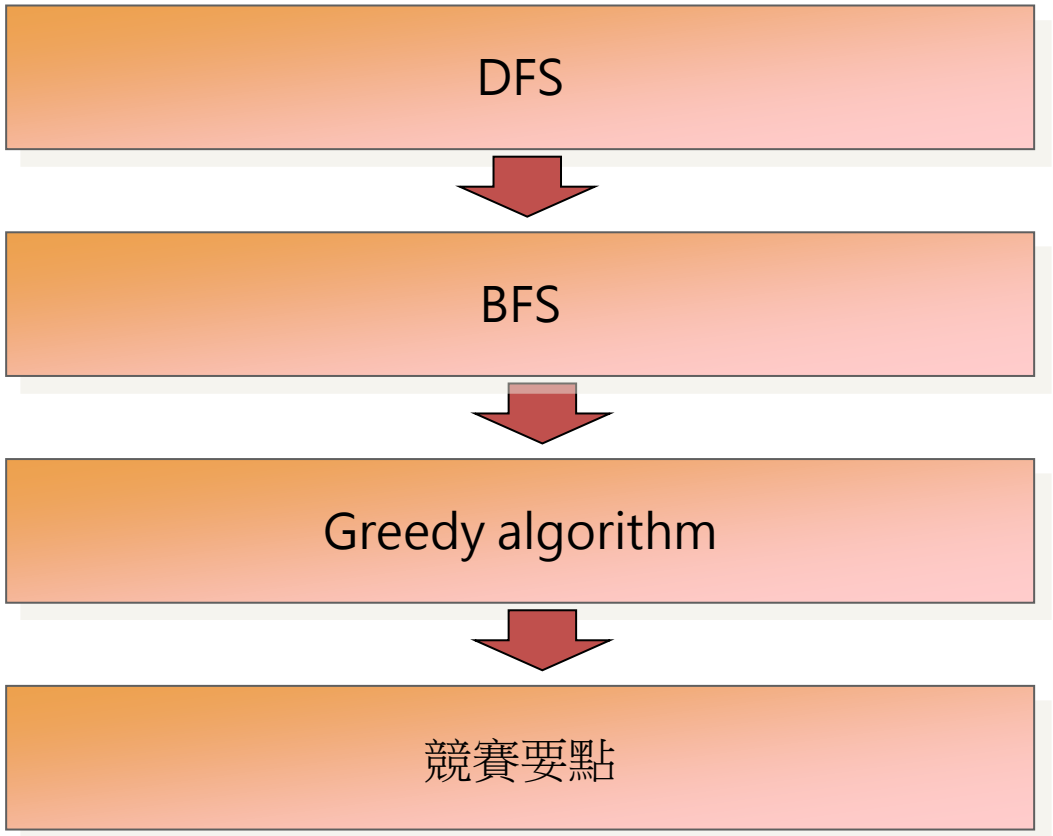
**Sample Input**

```
6
A
C
D
B
C
B
```

**Sample Output**

```
ABCBCD
```

# Outline






made by electron & free999

# Library

- 25頁A4單面
- Library可以有什麼？
  - 程式碼。EX : 語法、演算法、等等
  - 問題的解法
  - 目錄、頁碼
  - 注意事項
- 製作技巧 ( 為了紀錄更多資料 )
  - 單面印 2~3欄 (可直式、橫式)
  - 字體縮小 (可到6)
  - 手寫 (例如：目錄、頁碼、注意事項)
  - 平常要養成製作個人Library的習慣



made by electron & free999

# 比賽一開始

- 誰看題目
- 誰setting
- 誰負責撰題目、錠題目

# 如何選題

- 比賽一開始
  - 第一題先看
  - 題目短的先看
  - 秒數短的先看

- 盡快把所有的題目看完
- 看 Scoreboard 哪題最多人解出來、最快解出來

# 解讀Scoreboard


| Rank | Name | Solved | Time | A | B | C | D | E | F | G | H | I | J | Total a |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | +1 ironwood branch | 8 | 792 | 1/16 | 1/33 | 1/113 | 1/39 | 0/-- | 1/243 | 1/63 | 1/58 | 2/207 | 36/-- | 45/8 |
| 2 | Musou | 8 | 1180 | 1/125 | 1/94 | 1/78 | 1/112 | 0/-- | 3/296 | 1/60 | 2/169 | 1/186 | 0/-- | 11/8 |
| 3 | University of Agitsune | 7 | 575 | 1/21 | 1/14 | 3/151 | 1/37 | 3/-- | 18/-- | 1/66 | 1/97 | 1/149 | 27/-- | 57/7 |
| 4 | TwT514 | 7 | 776 | 1/93 | 1/67 | 1/128 | 1/28 | 3/-- | 7/-- | 1/81 | 3/106 | 1/233 | 5/-- | 24/7 |
| 5 | Reaper | 7 | 795 | 1/30 | 1/67 | 1/209 | 2/116 | 0/-- | 3/-- | 1/46 | 1/88 | 3/179 | 10/-- | 23/7 |
| 6 | Taipei-Hot | 7 | 852 | 1/152 | 1/49 | 2/175 | 1/64 | 0/-- | 14/-- | 1/32 | 1/108 | 1/252 | 0/-- | 22/7 |
| 7 | haskell-lover | 7 | 1053 | 1/146 | 1/30 | 1/66 | 1/110 | 0/-- | 2/279 | 2/139 | 2/223 | 0/-- | 3/-- | 13/7 |
| 8 | shms | 7 | 1128 | 1/66 | 1/147 | 1/282 | 1/77 | 18/-- | 1/-- | 1/29 | 1/235 | 1/292 | 0/-- | 26/7 |
| 9 | ->undefined-> | 7 | 1323 | 1/95 | 1/104 | 3/289 | 4/152 | 0/-- | 0/-- | 1/43 | 1/228 | 3/272 | 0/-- | 14/7 |
| 10 | SHY | 6 | 954 | 2/33 | 1/102 | 8/-- | 2/294 | 0/-- | 0/-- | 2/46 | 1/169 | 1/250 | 0/-- | 17/6 |
| 11 | hki.kih.ihk | 6 | 1072 | 1/66 | 1/197 | 1/170 | 1/59 | 0/-- | 0/-- | 6/226 | 3/214 | 0/-- | 0/-- | 13/6 |
| 12 | _(:3 _ /_ )_ | 6 | 1126 | 1/152 | 1/88 | 4/280 | 1/170 | 0/-- | 0/-- | 2/127 | 1/229 | 0/-- | 0/-- | 10/6 |
| 13 | Pandawa | 6 | 1227 | 1/151 | 1/76 | 4/263 | 1/160 | 0/-- | 0/-- | 3/138 | 3/299 | 0/-- | 0/-- | 13/6 |
| 14 | dontbullyusQAQ | 5 | 611 | 2/124 | 1/26 | 4/103 | 1/171 | 0/-- | 1/-- | 1/107 | 2/-- | 1/-- | 0/-- | 13/5 |
| 15 | Otoshigami | 5 | 660 | 1/129 | 1/79 | 3/162 | 0/-- | 0/-- | 1/179 | 1/71 | 3/-- | 0/-- | 0/-- | 10/5 |
| 16 | 1 < 37122 | 5 | 704 | 1/166 | 1/85 | 4/-- | 2/236 | 0/-- | 0/-- | 1/22 | 2/155 | 0/-- | 0/-- | 11/5 |
| 17 | ( = w=)-c< -_-) | 5 | 733 | 1/86 | 1/112 | 4/160 | 1/174 | 0/-- | 0/-- | 1/141 | 0/-- | 0/-- | 0/-- | 8/5 |
| 18 | Hermit | 4 | 466 | 1/169 | 1/43 | 7/-- | 1/156 | 0/-- | 0/-- | 1/98 | 1/-- | 0/-- | 0/-- | 12/4 |
| 19 | THE 2DM@STER | 4 | 526 | 1/185 | 1/52 | 1/170 | 1/-- | 0/-- | 0/-- | 1/119 | 0/-- | 0/-- | 0/-- | 5/4 |
| 20 | oshieteZukky | 4 | 547 | 2/211 | 1/34 | 1/168 | 0/-- | 0/-- | 0/-- | 1/114 | 0/-- | 0/-- | 0/-- | 5/4 |
| 21 | TJU_On | 4 | 583 | 1/60 | 1/145 | 0/-- | 3/-- | 0/-- | 0/-- | 1/86 | 1/292 | 0/-- | 0/-- | 7/4 |
| 22 | Ha...ha~chu~ | 4 | 586 | 1/111 | 1/35 | 5/225 | 0/-- | 0/-- | 0/-- | 1/135 | 4/-- | 0/-- | 1/-- | 13/4 |
| 23 | ACEasy | 4 | 754 | 1/287 | 1/233 | 2/59 | 0/-- | 0/-- | 0/-- | 1/155 | 0/-- | 0/-- | 0/-- | 5/4 |



made by electron & free999

# 上傳結果WA?

- 測試極端測資
- 檢查題目
- 檢查程式碼

# 卡題了

- 寫的人判斷多久可以解決，超過10分鐘就換人寫別題吧~
- 系內比賽沒有印表機，可利用雙視窗，一個人coding、一個人debug
- 有印表機的比賽，要換人就直接印了、印不用錢！
- 發現bug了，記得把電腦搶回來~~

# 其他....

- 不要三個人一起盯著電腦，除非比賽時間快到了，要多thinking!!
- 不要讓電腦空著、可以先打測資打完
- 千萬不要在比賽中看到題目，知道演算法是什麼卻不會寫。所以一定要確保上課有教的部分都有人會寫若在練習時間不足的情況下，可以考慮進行分工，誰負責練圖論、誰負責練DP、等等。

# 團練

- 團練可以解決比賽中的很多問題。
- 團練的幫助之一是有助於了解隊友的在比賽時的特性，畢竟在比賽中時間是很重要的，一題寫的時間愈短 不僅是Score方面的影響，更重要的是 你可以有更多時間去解決其他題目。
- 當然團練也可以增強團隊的程式能力，可以看隊友的code怎麼寫，跟比賽中誰來debug有關，如果隊友的寫法比較好，就可以改變自己的寫法~

# 團練-1

- 看到題目知道解法不知道誰要寫？
- 一個題目誰要寫的問題可能會跟題目的類型是誰練的有關，都有練的話，團練中就可以知道誰寫比較好(打字比較快?)之類的。

# 團練-2

- 某一題卡題
- 誰要來幫忙debug? 誰負責想可能錯的測資?

# 團練-3

- 誰看題目比較快？
- 比賽一開始通常是一人setting(之後看題)、一人撕題目紙(之後看題)、一人看題，第三者通常是給看最快的人做，他必須知道題目要給誰解(所謂的推坑)，或是給自己。

# 最後


acm International Collegiate Programming Contest
IBM event sponsor

不要忘記吃點心!!!

# Homework 5

- UVA (20 Problems)
  - 260, 336, 352, 383, 439, 532, 567, 571, 601, 705, 762, 10004, 10009, 10039, 10308, 10505, 11597, 10672, 10939, 11706


# Thank for Your Attention



***made by electron & free999***