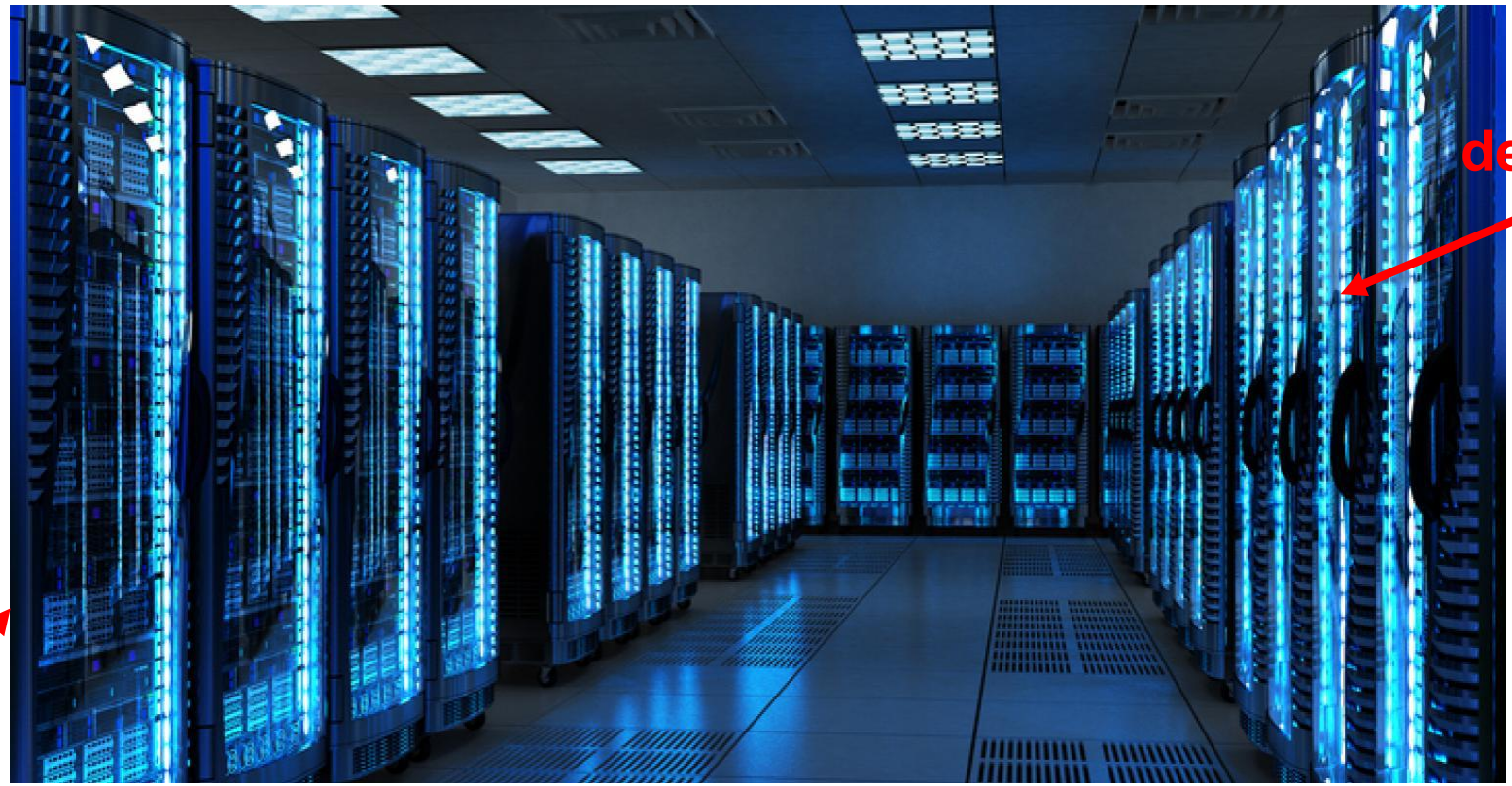
# Data Structures
# Programming Project #2

# Data Center

- A data center consists of multiple severs
- The servers are connected by switches in a local area network

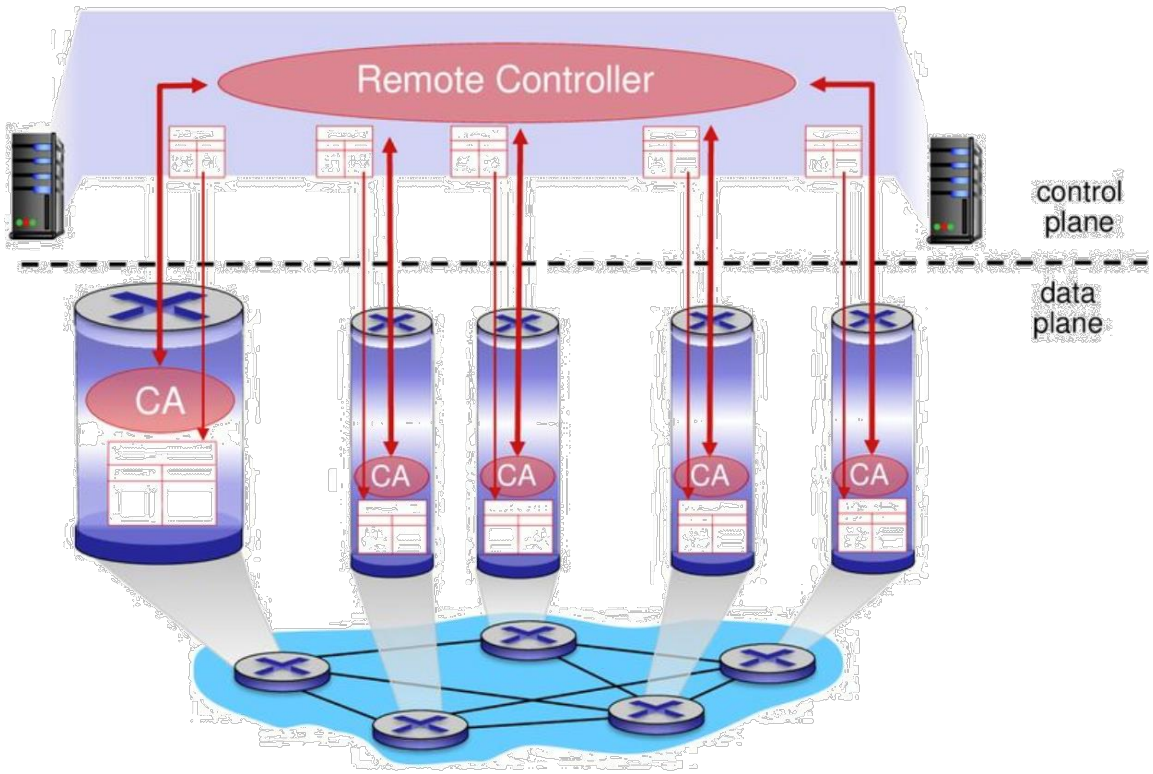


# Switches

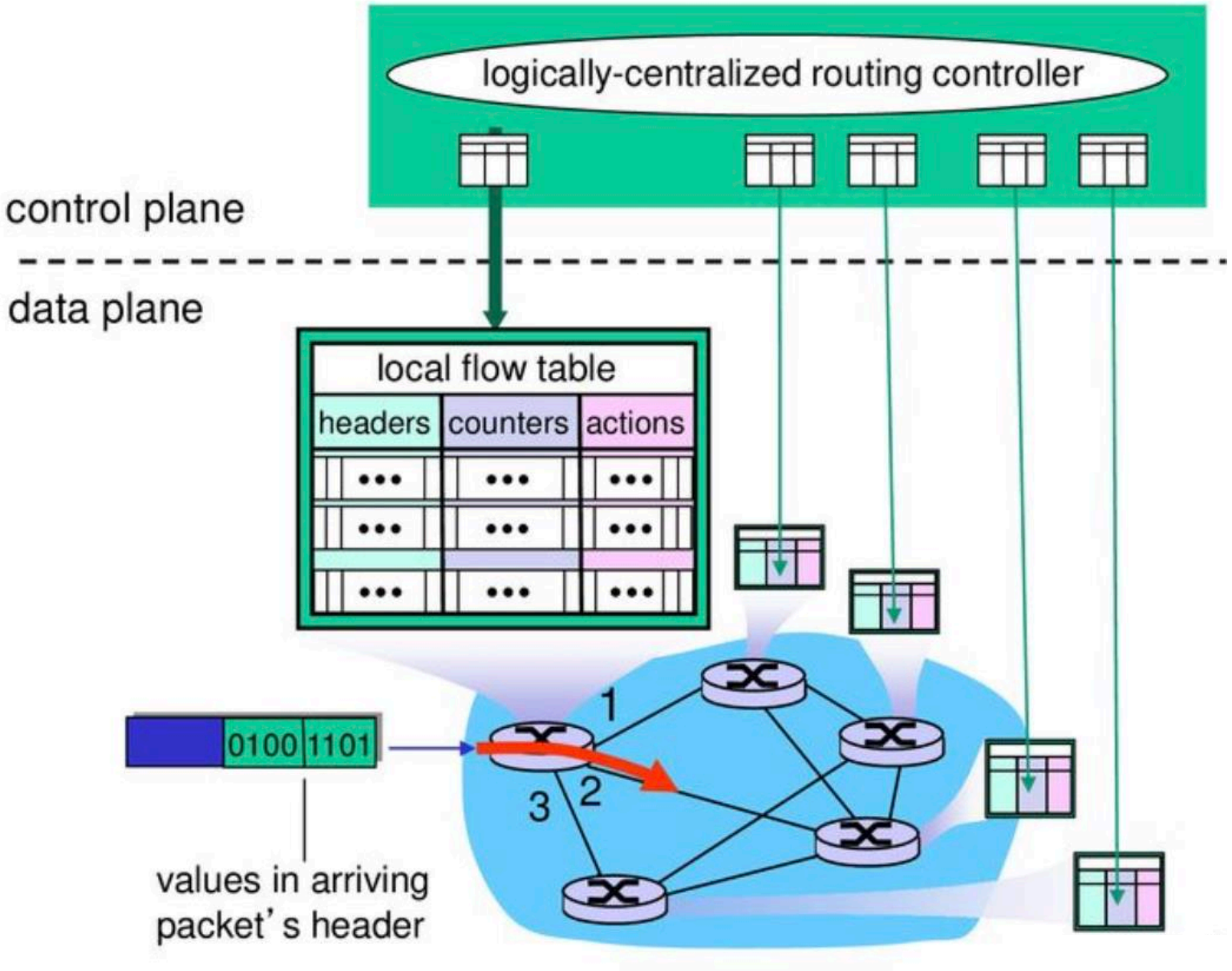
- Each switch has multiple ports
- Receive and forward the packets from a port to another port

# SDN-enabled Switches

- A centralized controller is introduced – software-defined networking (SDN)

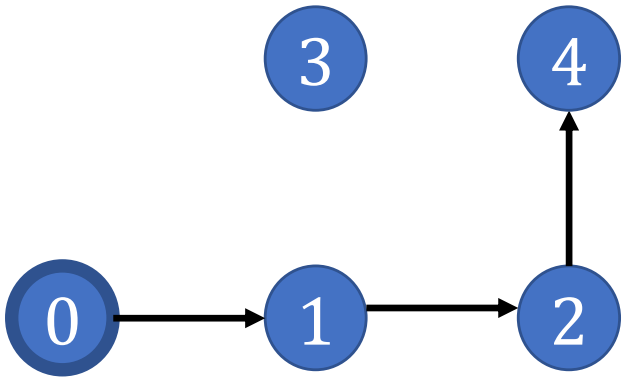


# Installing Rules in the SDN-enabled Switches

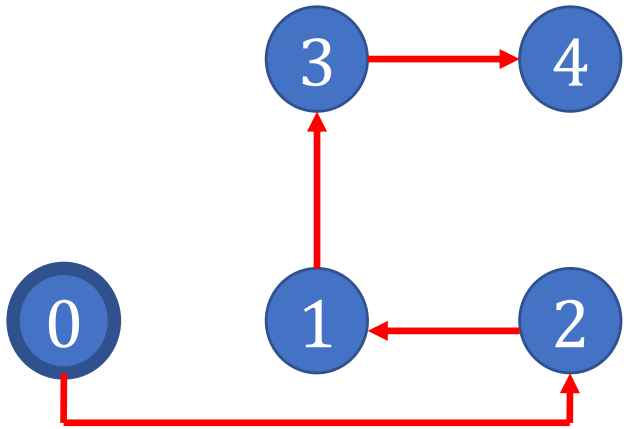


# Routing Path Update (aka Network Update)

- Given the **old** and **new** routing paths
- Update the routing paths



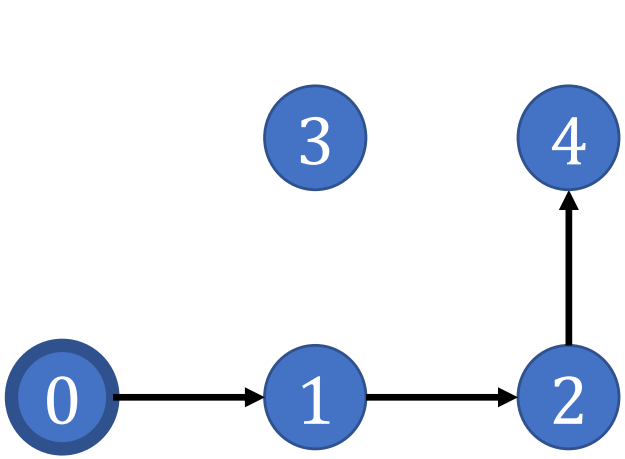

| 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 2 | 4 | -1 | -1 |



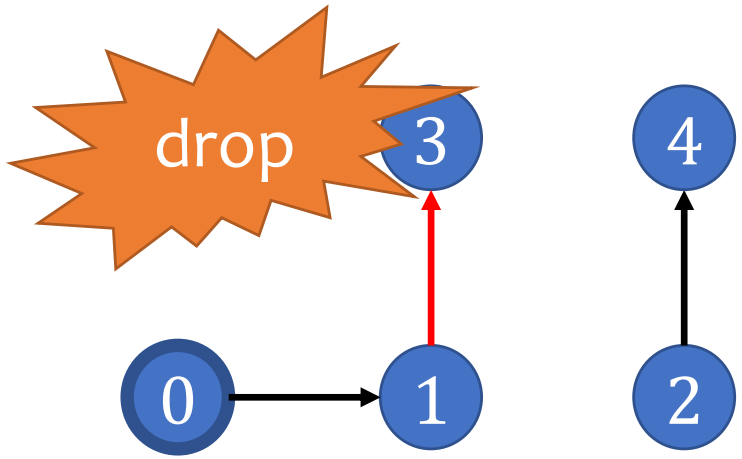

| 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 2 | 3 | 1 | 4 | -1 |

# Difficulty of Network Update in SDN

- The controller is logically-centralized
- However, the underlying mechanism is distributed
- Each switch receives the update message and updates its rule independently and asynchronously



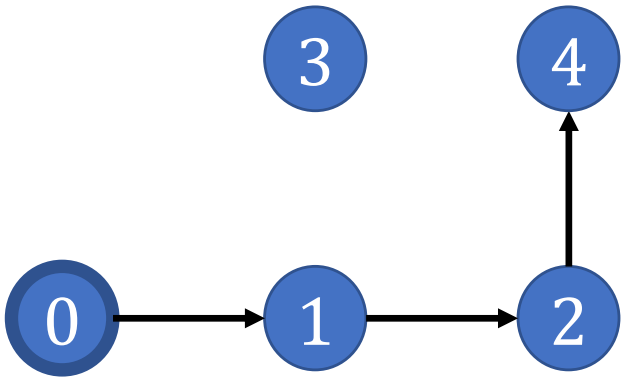

| 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 2 | 4 | -1 | -1 |



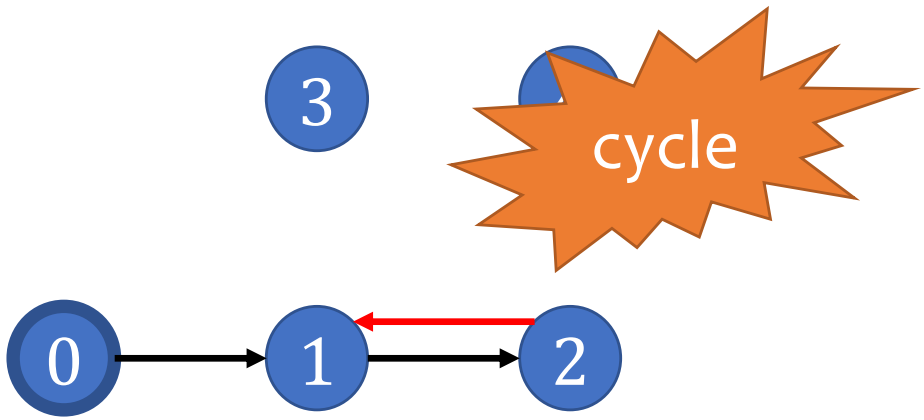

| 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 3 | 4 | -1 | -1 |

# Difficulty of Network Update in SDN

- The controller is logically-centralized
- However, the underlying mechanism is distributed
- Each switch receives the update message and updates its rule independently and asynchronously




| 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 2 | 4 | -1 | -1 |




| 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 3 | 4 | -1 | -1 |

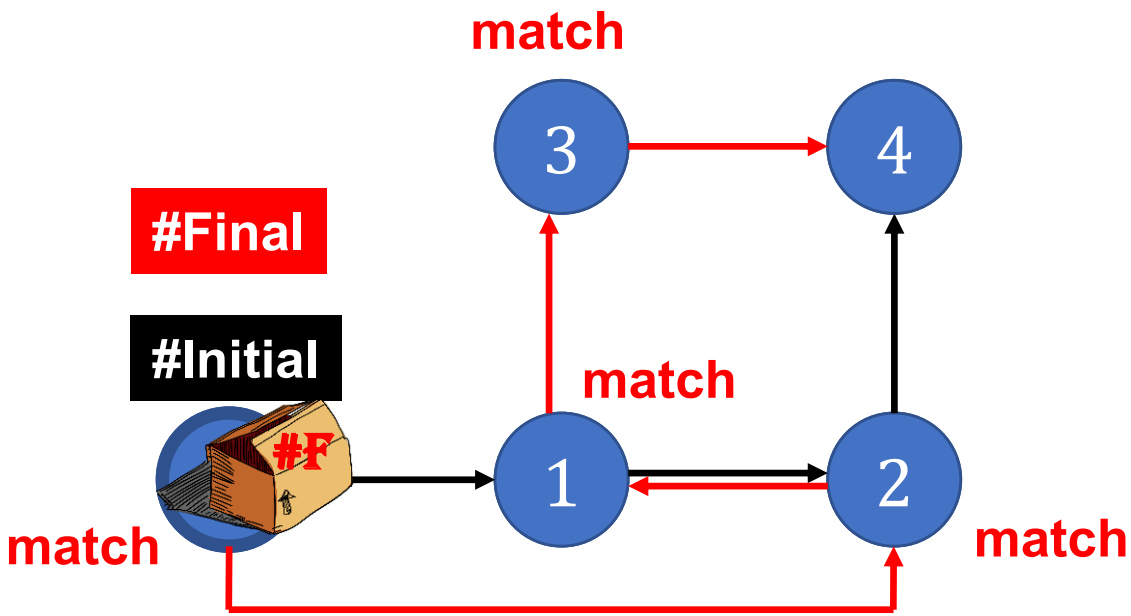
# Difficulty of Network Update in SDN

- The controller is logically-centralized
- However, the underlying mechanism is distributed
- Each switch receives the update message and updates its rule independently and asynchronously

- How to solve the issue?
- Two-phase commit
- Round-based update

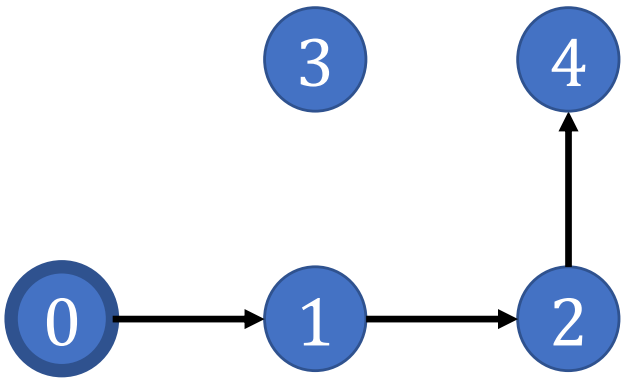
# Solution of Network Update in SDN

- Two-phase commit
- Drawback: waste the TCAM size during the update

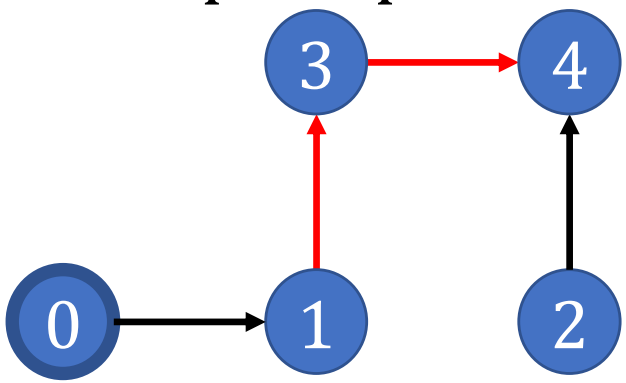


# Solution of Network Update in SDN

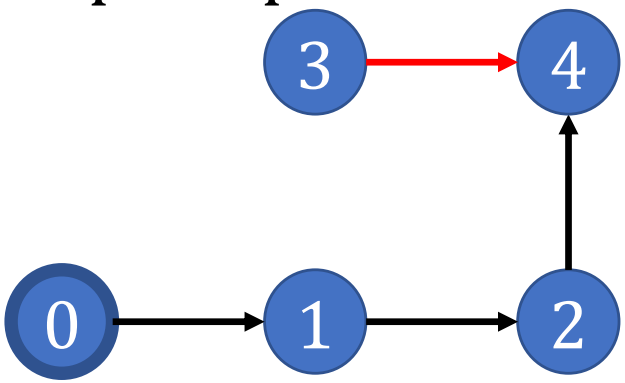
- Round-based update (1st attempt)



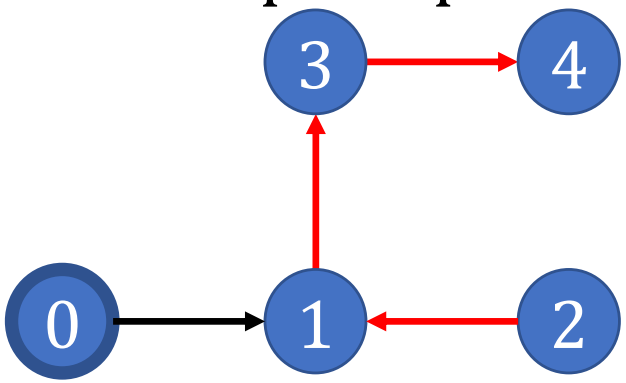

Step 2: Update 1



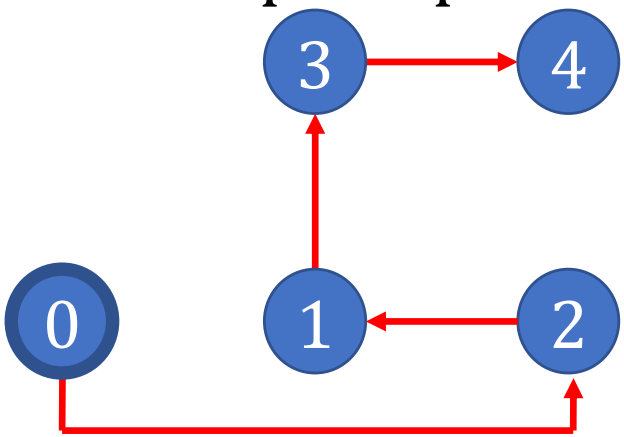

Step 1: Update 3




Step 3: Update 2

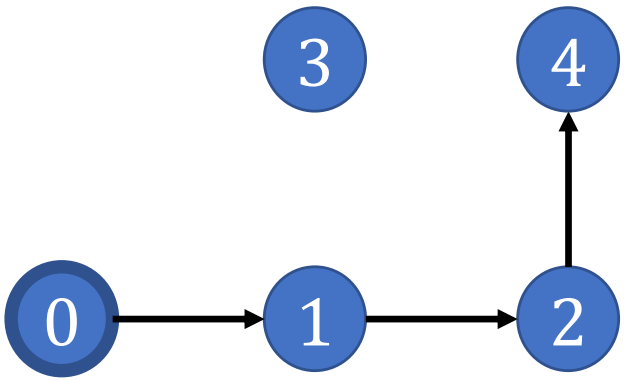



Step 4: Update 0

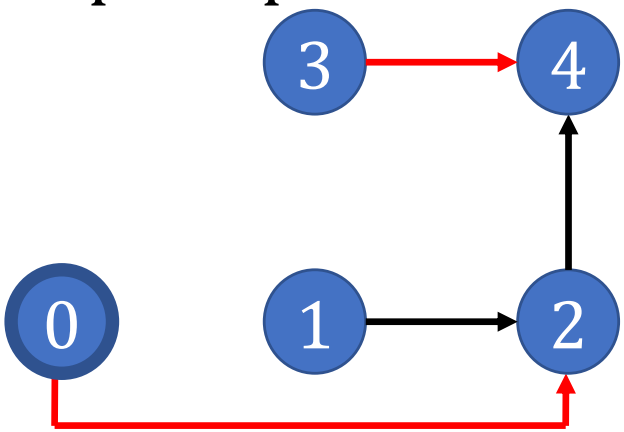


# Solution of Network Update in SDN

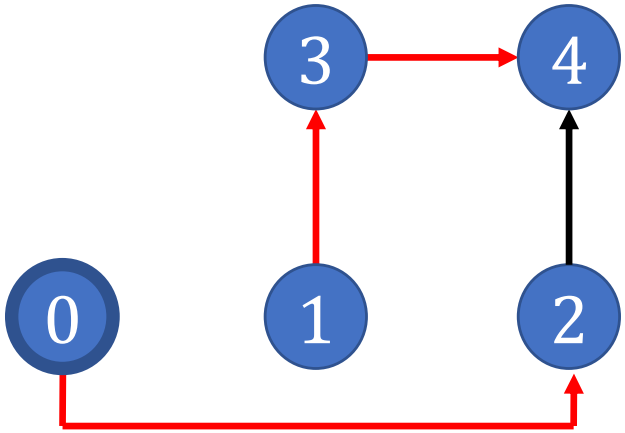
- Round-based update (2nd attempt)



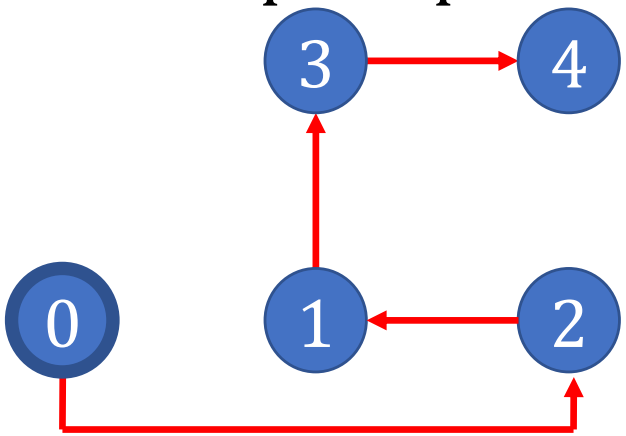

Step 1: Update 0 and 3




Step 2: Update 1

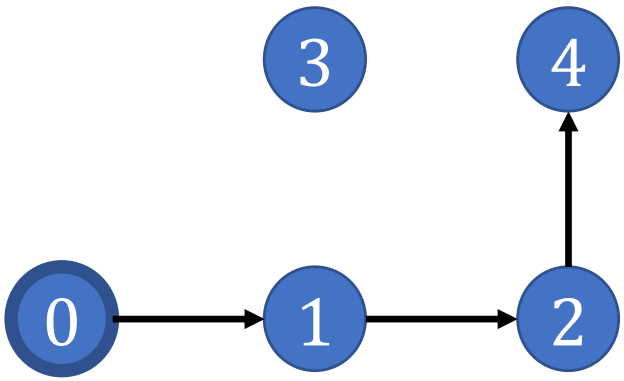



Step 3: Update 2

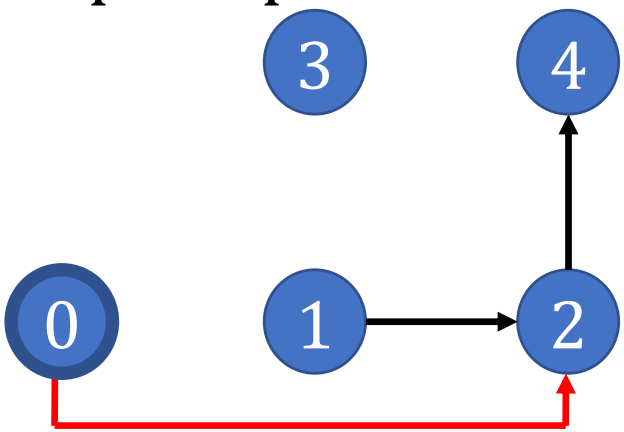


# Solution of Network Update in SDN

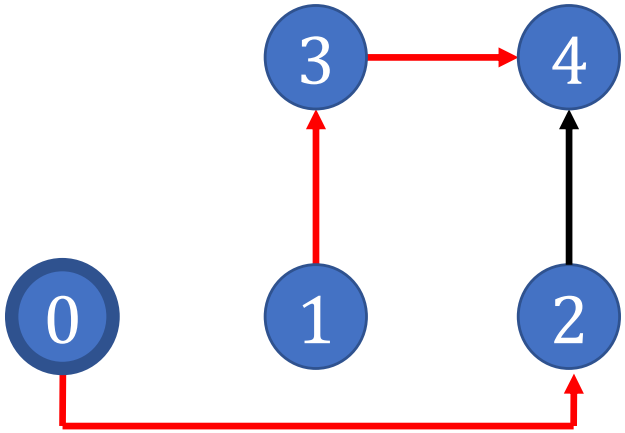
- Round-based update (3rd attempt)



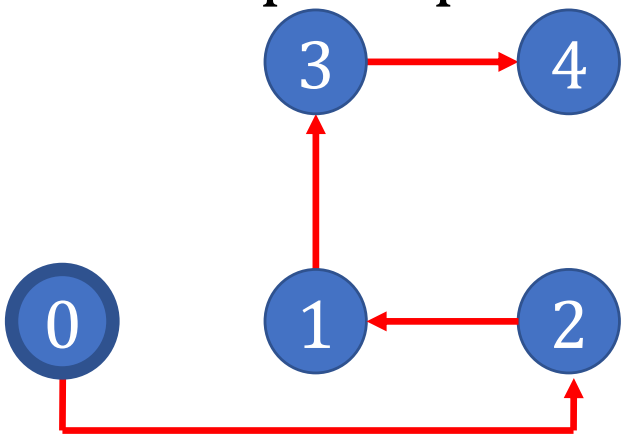

Step 1: Update 0




Step 2: Update 1 and 3

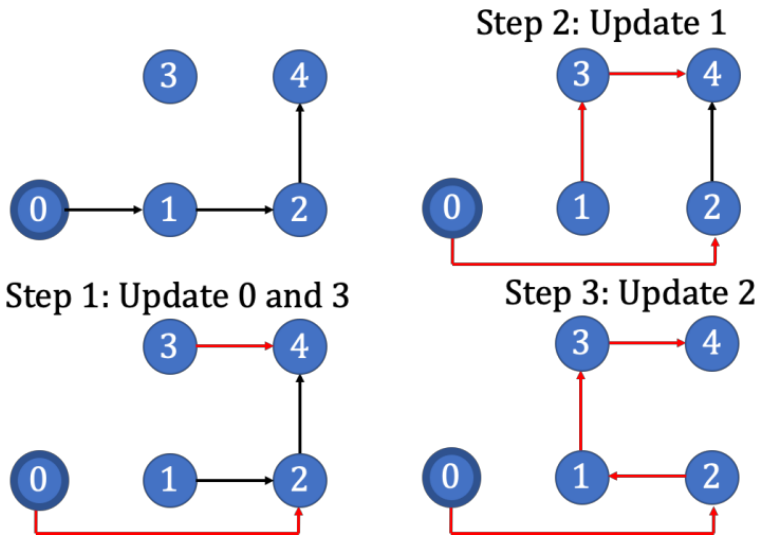



Step 3: Update 2



# Programming Project #2: Minimize the number of update rounds

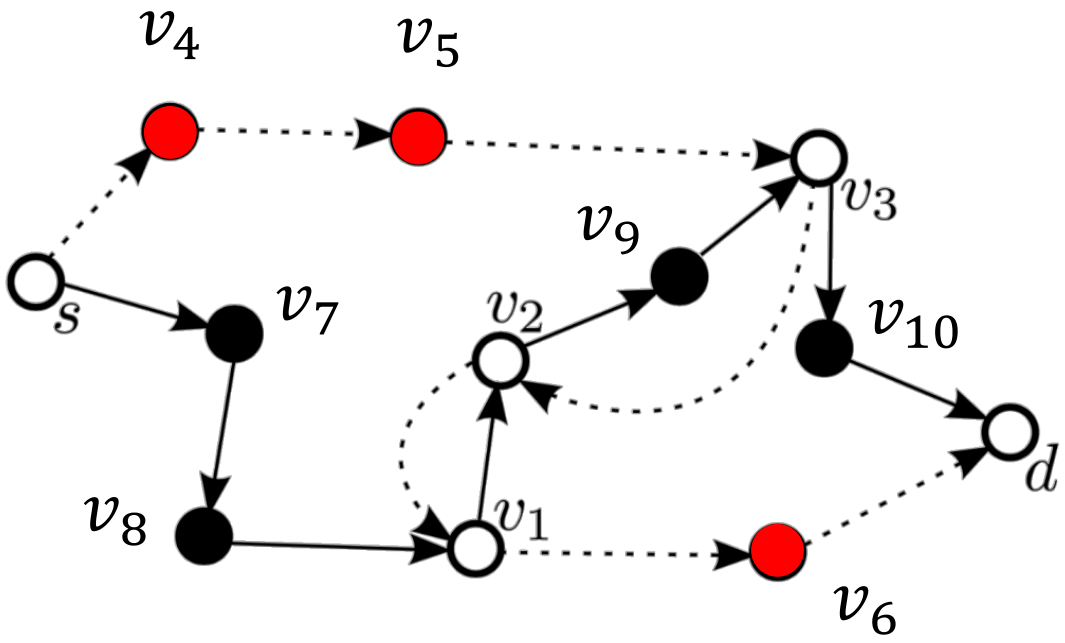
- Input:
  - Numbers of nodes in old and new paths
  - Nodes in old and new paths
- Procedure:
  - Minimize the rounds of update
- Output:
  - Rules of each switch in each round
- The grade is inversely proportional to the number of rounds



# Round-Based Update Algorithm

Solid line: old path  Dashed line: new path

| s | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | d |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 7 | 2 | 9 | 10 | -1 | -1 | -1 | 8 | 1 | 3 | d | -1 |



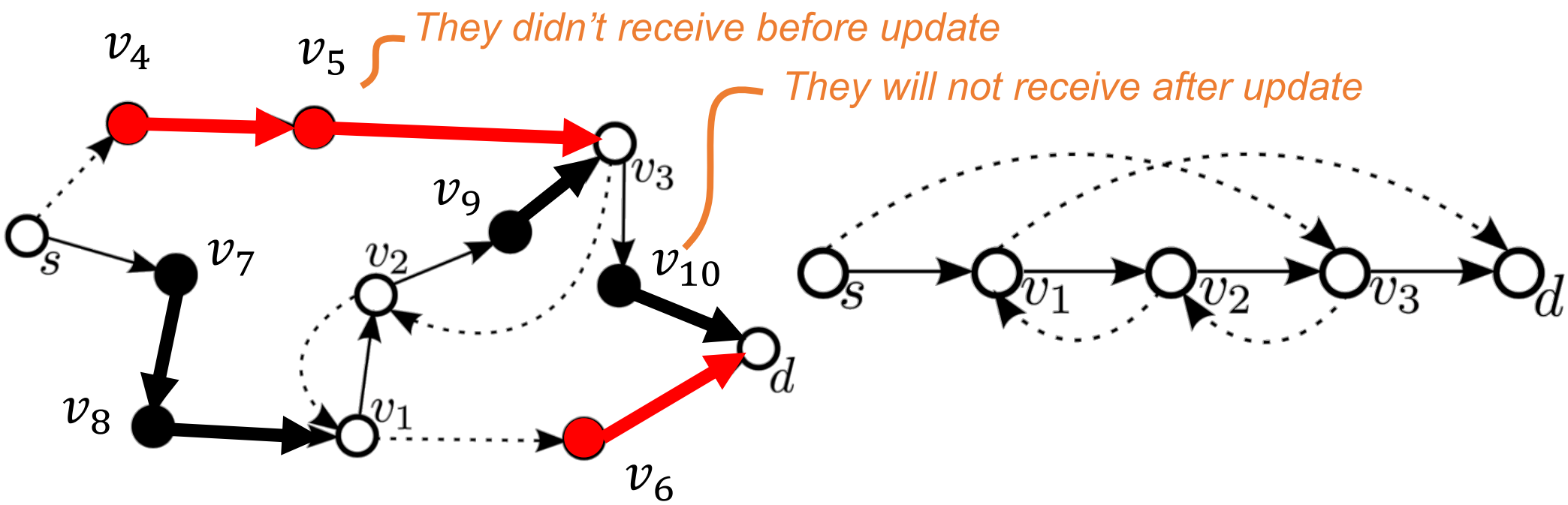

| s | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | d |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 4 | 6 | 1 | 2 | 5 | 3 | d | -1 | -1 | -1 | -1 | -1 |

# Round-Based Update Algorithm

Solid line: old path  Dashed line: new path

- Add the rules in red nodes in the first round
- Remove the rules in **black nodes** in the **last** round

➔ Reduce the the network to the line representation



# Round-Based Update Algorithm

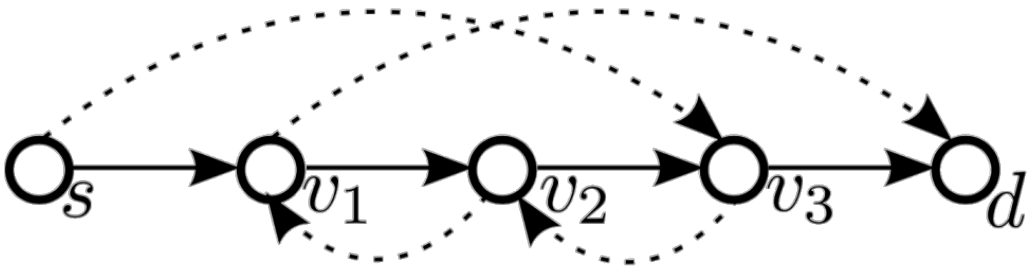
Solid line: old path  Dashed line: new path

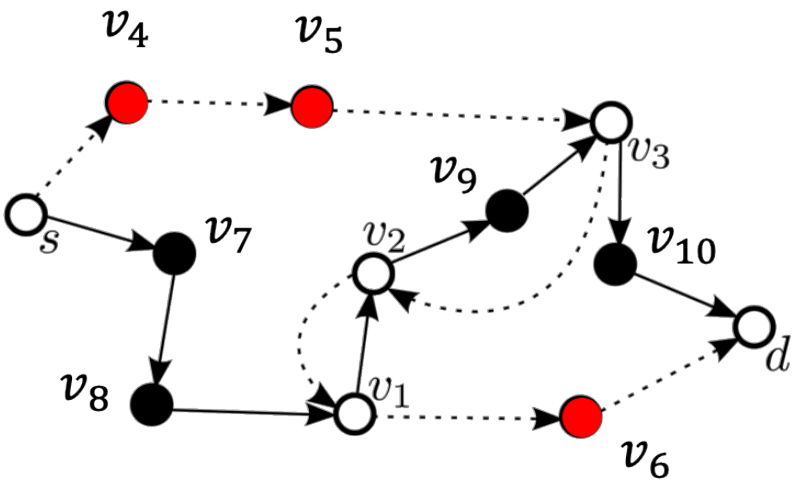
- Add the rules in red nodes in the first round
- Remove the rules in **black nodes** in the **last** round

➔ Reduce the the network to the line representation

| s | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | d |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 7 | 2 | 9 | 10 | 5 | 3 | d | 8 | 1 | 3 | d | -1 |

0th round



# Round-Based Update Algorithm

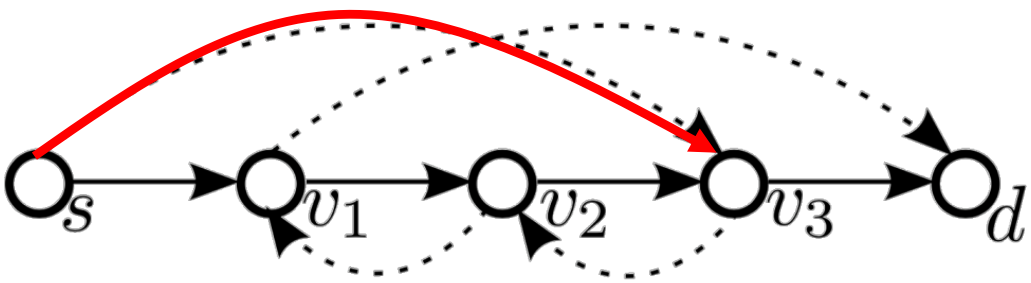
Solid line: old path  Dashed line: new path

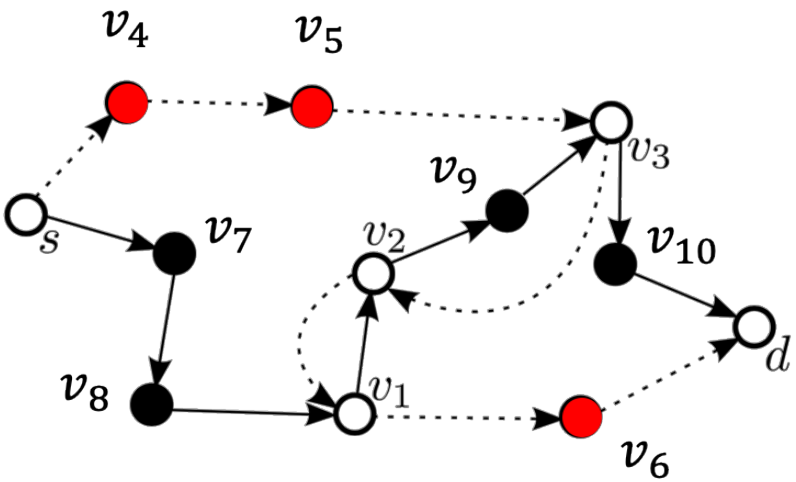
- Add the rules in red nodes in the first round
- Remove the rules in **black nodes** in the **last** round

➔ Reduce the the network to the line representation

| s | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | d |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **4** | 2 | 9 | 10 | 5 | 3 | d | 8 | 1 | 3 | d | -1 |

1nd round



# Round-Based Update Algorithm

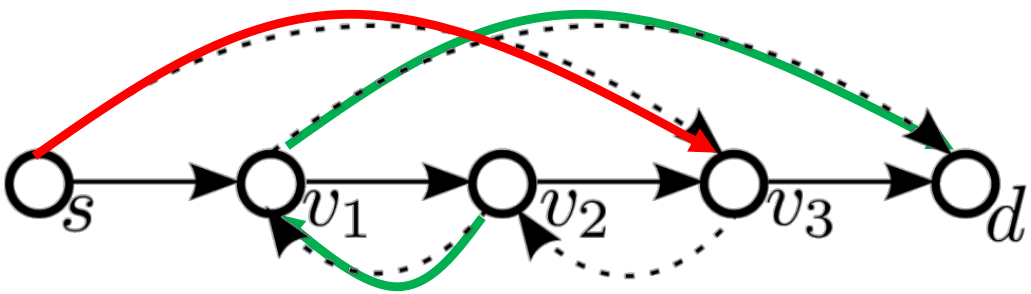
Solid line: old path  Dashed line: new path

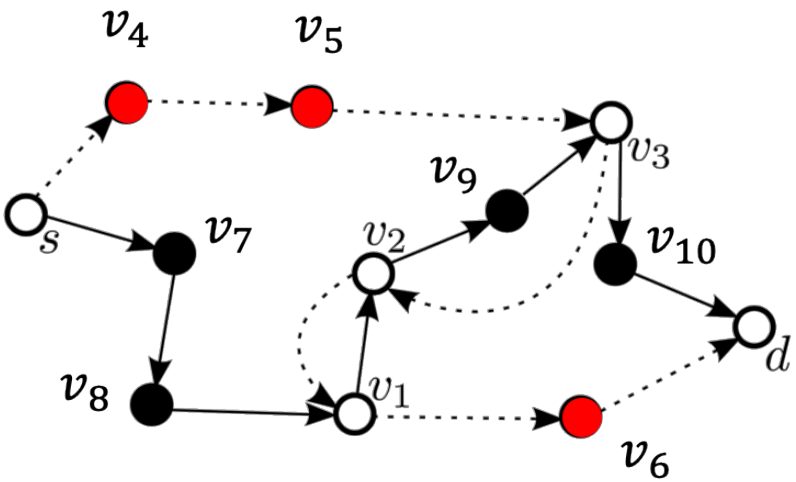
- Add the rules in red nodes in the first round
- Remove the rules in **black nodes** in the **last** round

➔ Reduce the the network to the line representation

| s | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | d |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 4 | 6 | 1 | 10 | 5 | 3 | d | 8 | 1 | 3 | d | -1 |

2nd round



# Round-Based Update Algorithm

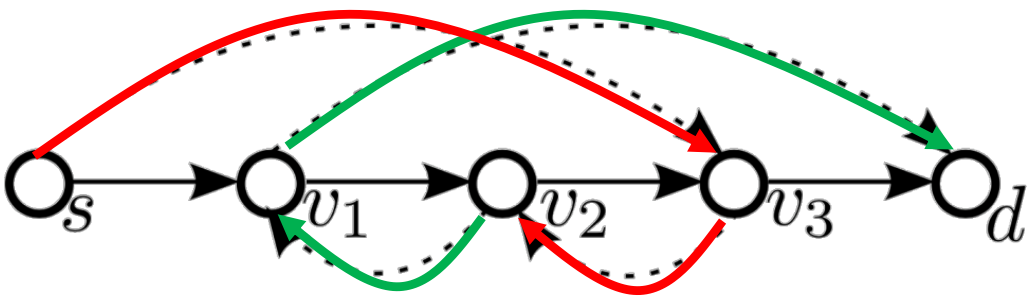
Solid line: old path  Dashed line: new path

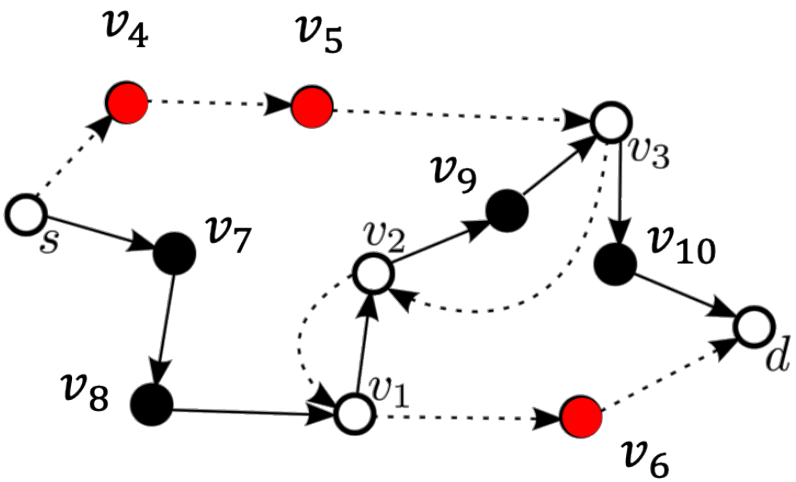
- Add the rules in red nodes in the first round
- Remove the rules in **black nodes** in the **last** round

➔ Reduce the the network to the line representation

| s | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | d |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 4 | 6 | 1 | **2** | 5 | 3 | d | 8 | 1 | 3 | d | -1 |

3rd round



# Round-Based Update Algorithm

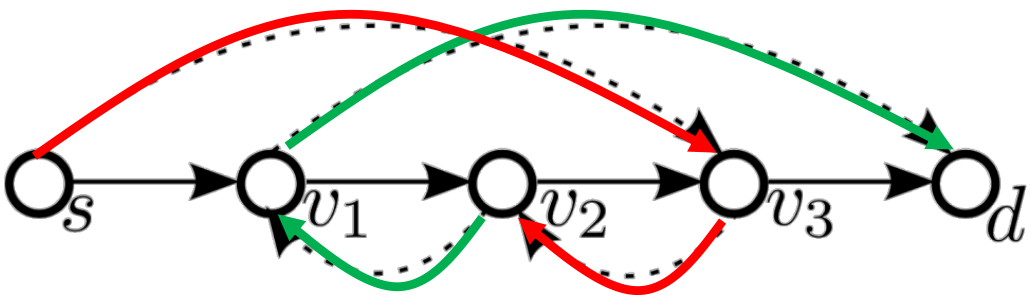
Solid line: old path  Dashed line: new path

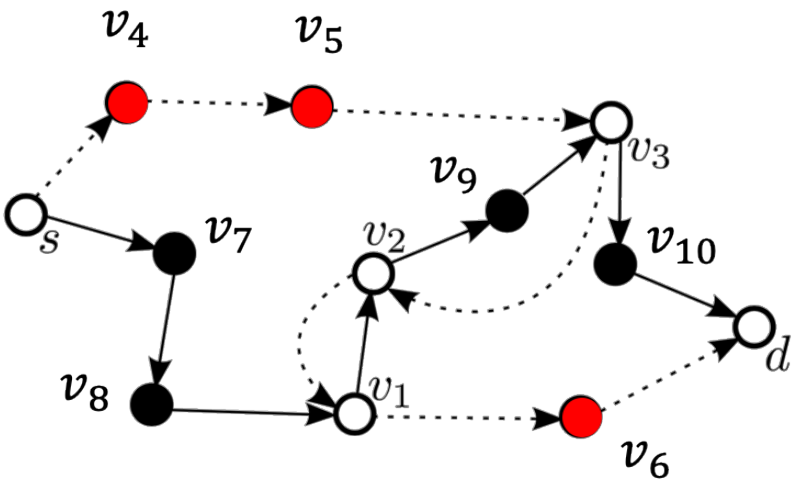
- Add the rules in red nodes in the first round
- Remove the rules in **black nodes** in the **last** round

➔ Reduce the the network to the line representation

| s | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | d |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 4 | 6 | 1 | 2 | 5 | 3 | d | -1 | -1 | -1 | -1 | -1 |

4th round



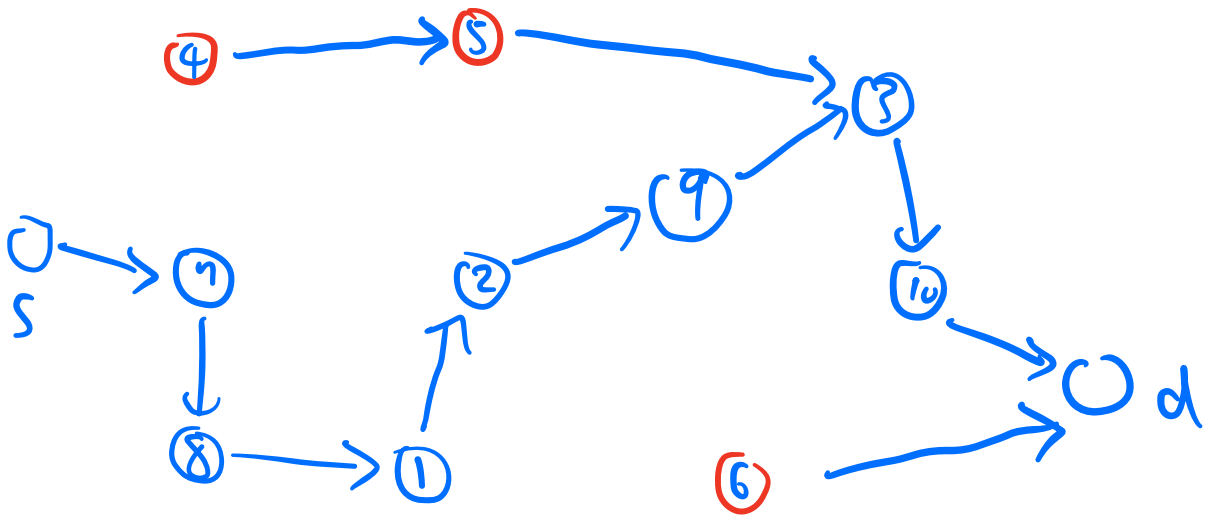
# Implementation Rules

- You have to use th tructure "li ed list" to implement the routing path ext-ho )
- Next-hop table
- A positive integer i resents node ID
- -1 in the examp ll (0)

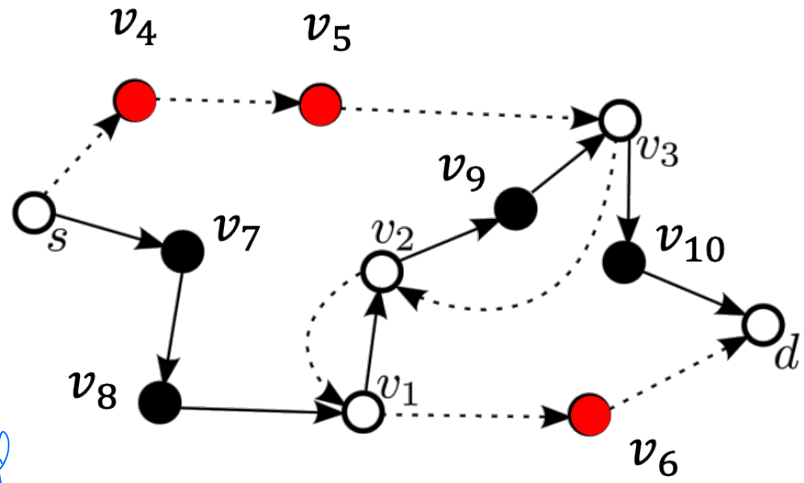
```
struct node {
        int id;
        struct node *link;
}
struct node table[nodeNum];
```

# Discussion

- Minimizing the number of update rounds is NP-hard
- You may not find the minimum number of update rounds for this problem unless NP = P



# Input Sample: input.txt




Format:

#Nodes

Path1

Path2

| s | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | d |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 7 | 2 | 9 | 10 | -1 | -1 | -1 | 8 | 1 | 3 | d | -1 |

e.g.,

| s | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | d |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 4 | 6 | 1 | 2 | 5 | 3 | d | -1 | -1 | -1 | -1 | -1 |

12

7 2 9 10 -1 -1 -1 8 1 3 11 -1

4 6 1 2 5 3 11 -1 -1 -1 -1 -1

# Output Sample:
## use printf

Format:
#Rounds
Path1
Path2
…

| s | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | d |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 7 | 2 | 9 | 10 | -1 | -1 | -1 | 8 | 1 | 3 | d | -1 |

| s | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | d |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 7 | 2 | 9 | 10 | 5 | 3 | d | 8 | 1 | 3 | d | -1 |

| s | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | d |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 4 | 2 | 9 | 10 | 5 | 3 | d | 8 | 1 | 3 | d | -1 |

| s | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | d |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 4 | 6 | 1 | 10 | 5 | 3 | d | 8 | 1 | 3 | d | -1 |

| s | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | d |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 4 | 6 | 1 | 2 | 5 | 3 | d | 8 | 1 | 3 | d | -1 |

| s | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | d |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 4 | 6 | 1 | 2 | 5 | 3 | d | -1 | -1 | -1 | -1 | -1 |

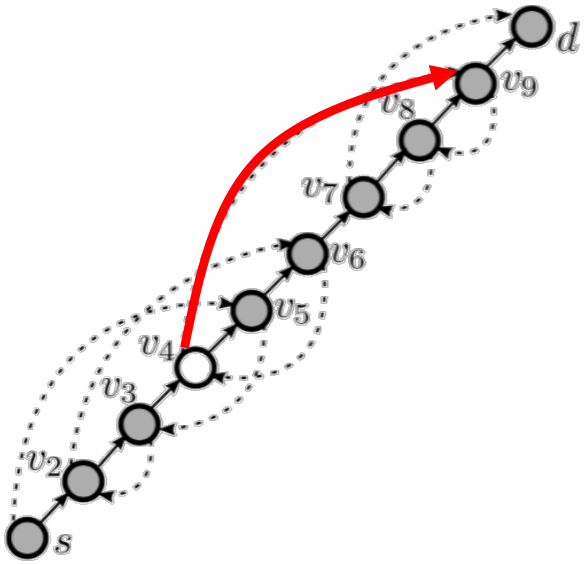
e.g.,
6
7 2 9 10 -1 -1 -1 8 1 3 11 -1
7 2 9 10 5 3 11 8 1 3 11 -1
4 2 9 10 5 3 11 8 1 3 11 -1
4 6 1 10 5 3 11 8 1 3 11 -1
4 6 1 2 5 3 11 8 1 3 11 -1
4 6 1 2 5 3 11 -1 -1 -1 -1 -1

# Note

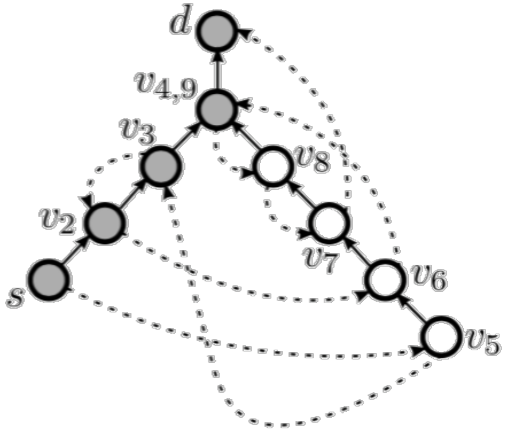
- Superb deadline: 10/31 Tue
- Deadline: 11/7 Tue
- Pass the test of our online judge platform
- Submit your code to E-course2
- Demonstrate your code remotely or in person with TA
- **C Source code (i.e., only .c)**
- Show a good programming style

# Note: Round-Based Update Algorithm (1/3)

- Shortcut phase: used in odd rounds
- In each round, we iteratively select the edge that has the farthest reaching distance and does not interfere with the selected edge until there is no such edge
- # selected edges >= 1 → Update the selected edges



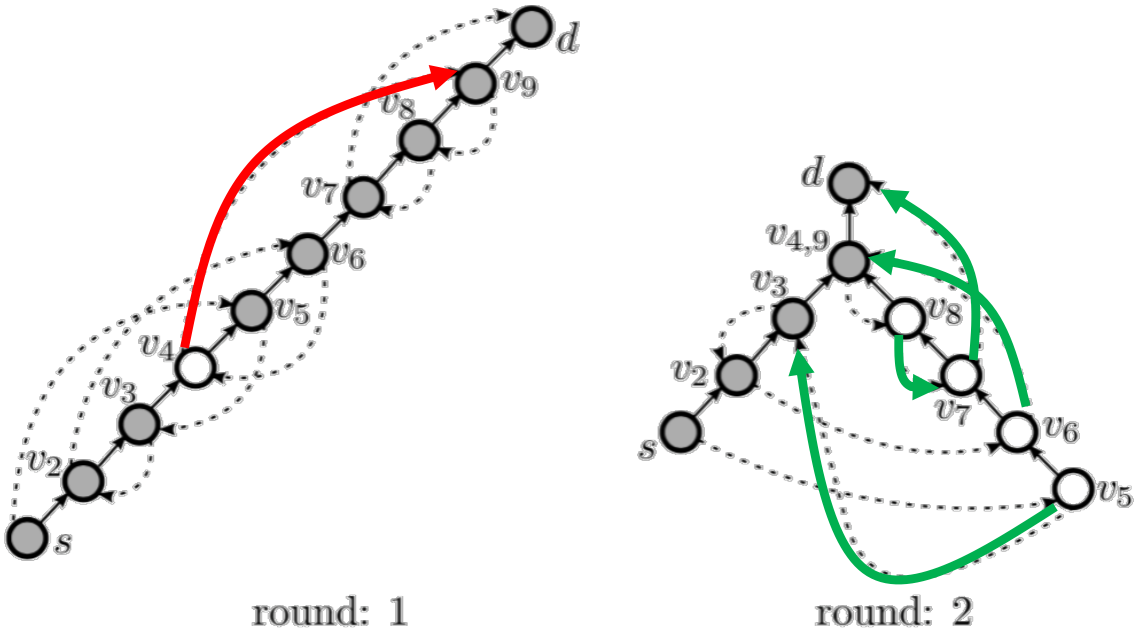

round: 1




round: 2

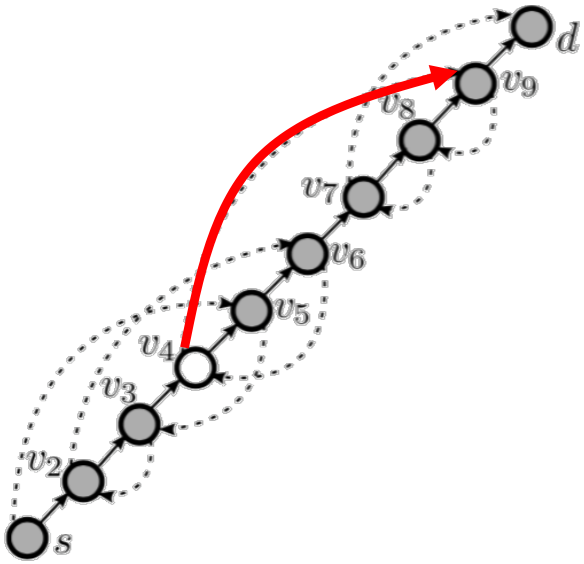
# Note: Round-Based Update Algorithm (2/3)

- Prune phase: used in even rounds
- Update all nodes that are not on the current path from the source to the destination
- They can be updated in the same round since they don't receive any packet after the 1st round

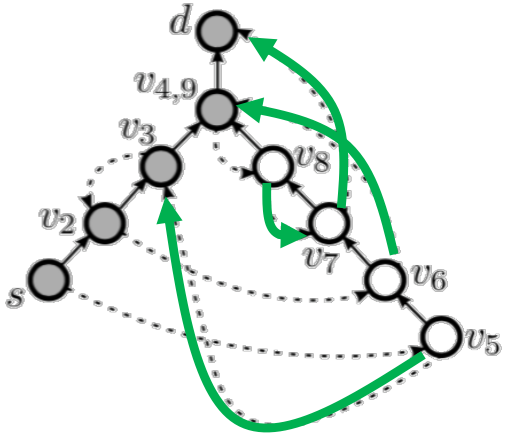


# Note: Round-Based Update Algorithm (3/3)

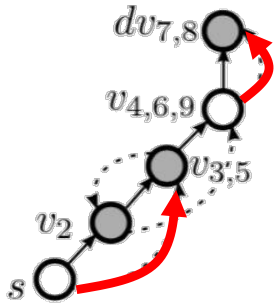
- The algorithm
  Repeat the two phases until all nodes are updated
- Shortcut phase: used in odd rounds
- Prune phase: used in even rounds



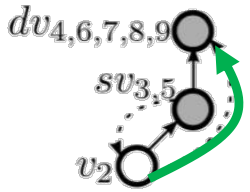

round: 1




round: 2

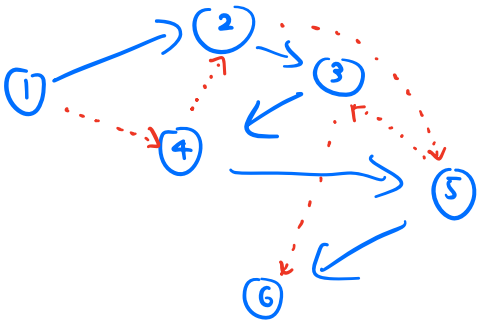


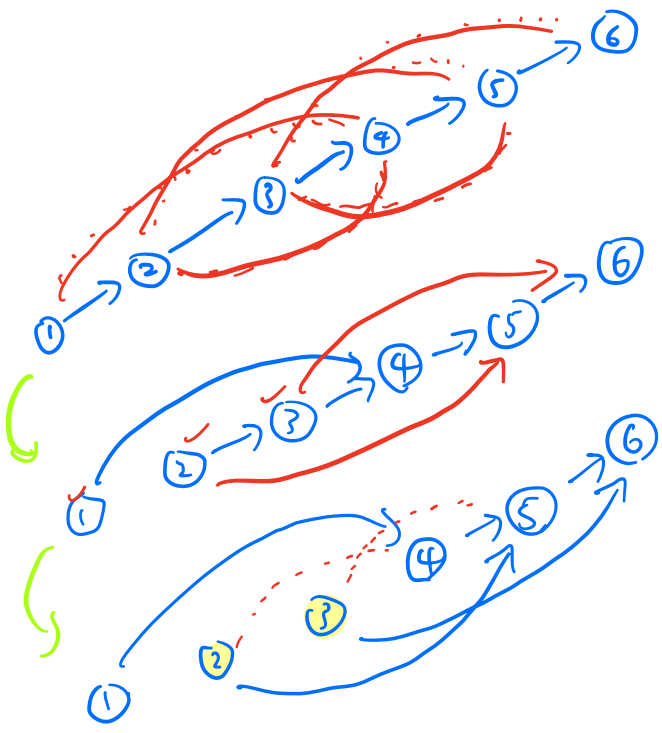

round: 3

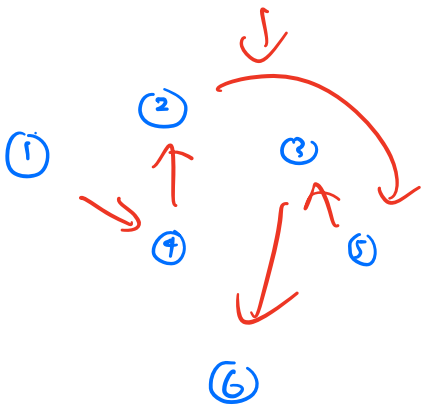


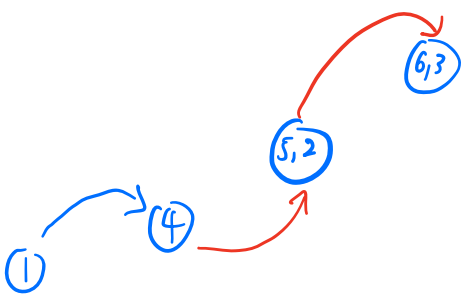

round: 4

| 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|
| 2 | 3 | 4 | 5 | 6 | 0 |
| S | | | | | E |
| 4 | 5 | 6 | 2 | 3 | 0 |



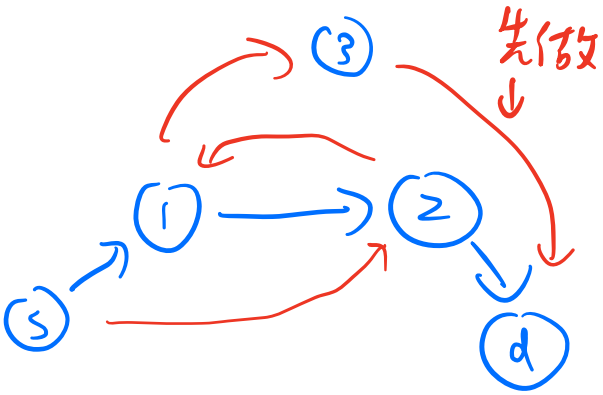







shrink 條件


(前提是⇒該路徑為update路徑)



odd time

新開 table 檢查 →

① 更新後 非route點數量

| s | 1 | 2 | 3 | d |
|---|---|---|---|---|
| 1 | 1 | X | X | X |

↳開table檢查新 route會不會 ↺ →X→ ↳不需更新

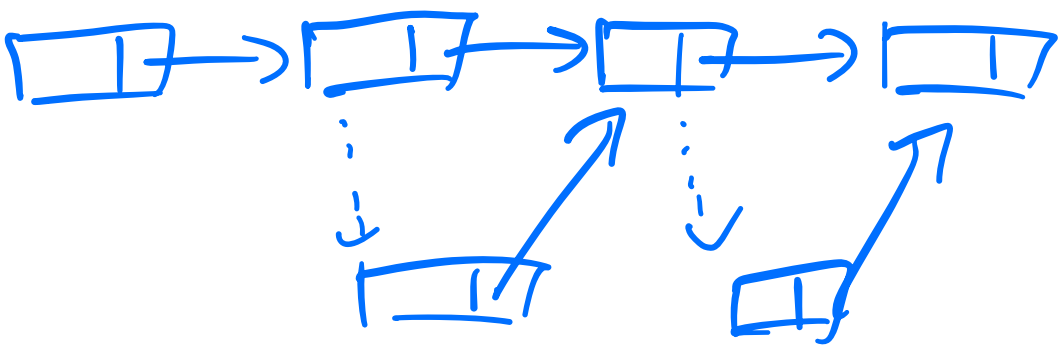
↳如果先做 even time 是不是不用檢查

even time

更新 非路徑點

↓

再把非路徑縮點

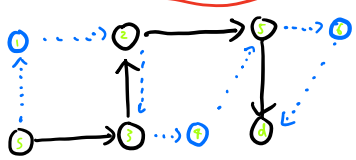


## odd time

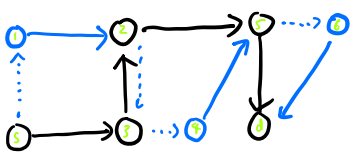
table_node_skip 要檢查 shrink

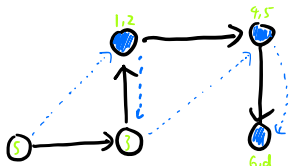
| s | 1 | 2 | 3 | 4 | 5 | 6 | d |
|---|---|---|---|---|---|---|---|
| 1 | \ | E | 1 | \ | 0 | \ | \ |

error 可更新的路徑



shrink





要, table更新問題 =)

不要, TLE的可能性

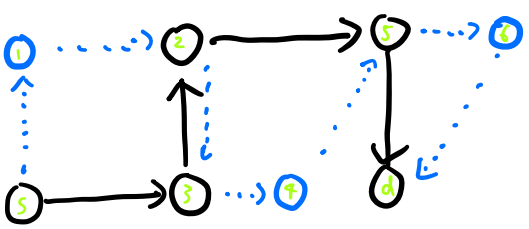
似乎不用 shrink, 計算被跳過的點就好(是否要shrink?)

## even time

table_unused

| s | 1 | 2 | 3 | 4 | 5 | 6 | d |
|---|---|---|---|---|---|---|---|
| \ | ✓ | \ | \ | ✓ | \ | ✓ | \ |

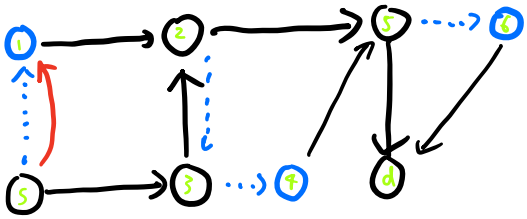
可直接更新



table_check

| 5 | 1 | 2 | 3 | 4 | 5 | 6 | d |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | X | X | 3 | X | 4 |

↳ 無 loop
無斷



實作

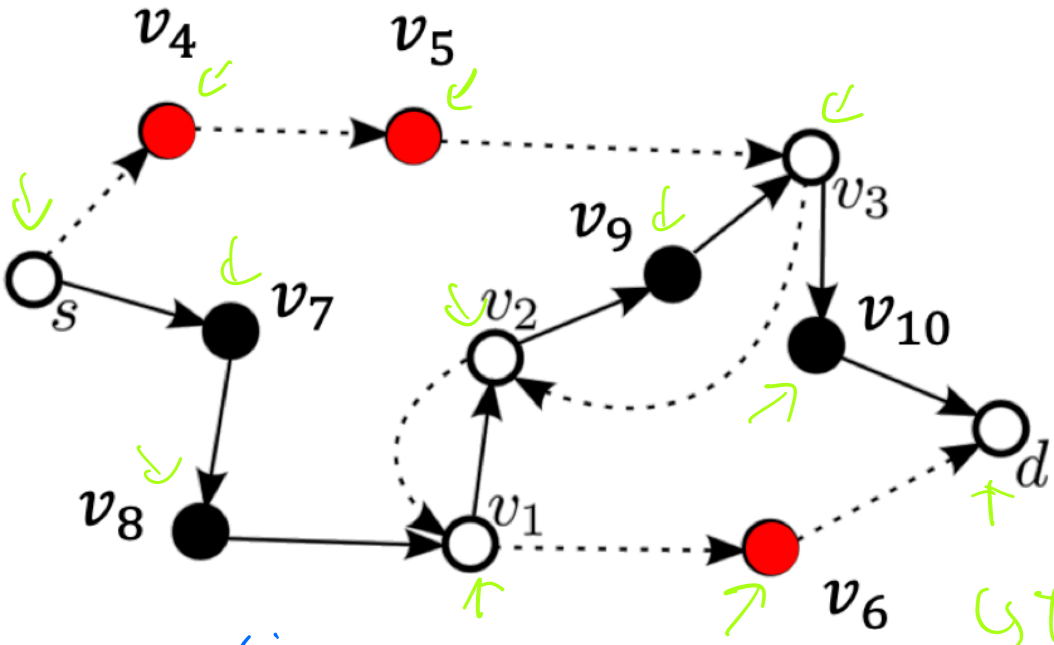
table (update)

current path

| s | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | d |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 4 | 6 | 1 | 2 | 5 | 3 | 11 | -1 | -1 | -1 | -1 | -1 |

table (current)

| s | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | d |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 7 | 2 | 9 | 10 | -1 | -1 | -1 | 8 | 1 | 3 | 11 | -1 |




↳ table_record_node

even time:

table_unused

| s | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | d |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | -1 |

↳ current ~~path~~ => even time 不會動到

table_record_node

- table
- finaloutput

更新

table => 也等於 output [0]

| 5 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | d |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 7 | 2 | 9 | 10 | 5 | 3 | 11 | 8 | 1 | 3 | 11 | -1 |

next node : 0

i = 1 => 1

Odd time:

table node left

| 5 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | d |
|---|---|---|---|---|---|---|---|---|---|---|---|
| ~~1~~ | 1 | 1 | ~~1~~ | 0 | 0 | 0 | 1 | 1 | 1 | ~~1~~ 0 | 0 |

0 0 ✓ ✓ 0 ✓

7 -> ④ -> record [4] finaltable [5] test-stored

table check

| 5 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | d |
|---|---|---|---|---|---|---|---|---|---|---|---|
| ~~0~~ | 0 | 0 | ~~0~~ | ~~0~~ | ~~0~~ | 0 | 0 | 0 | 0 | ~~0~~ | ~~0~~ 1 |

table node skip

| 5 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | d |
|---|---|---|---|---|---|---|---|---|---|---|---|
| ~~0~~ | ~~0~~ | ~~0~~ | ~~0~~ | -1 | -1 | -1 | 0 | 0 | 0 | 0 | -1 |

5 4

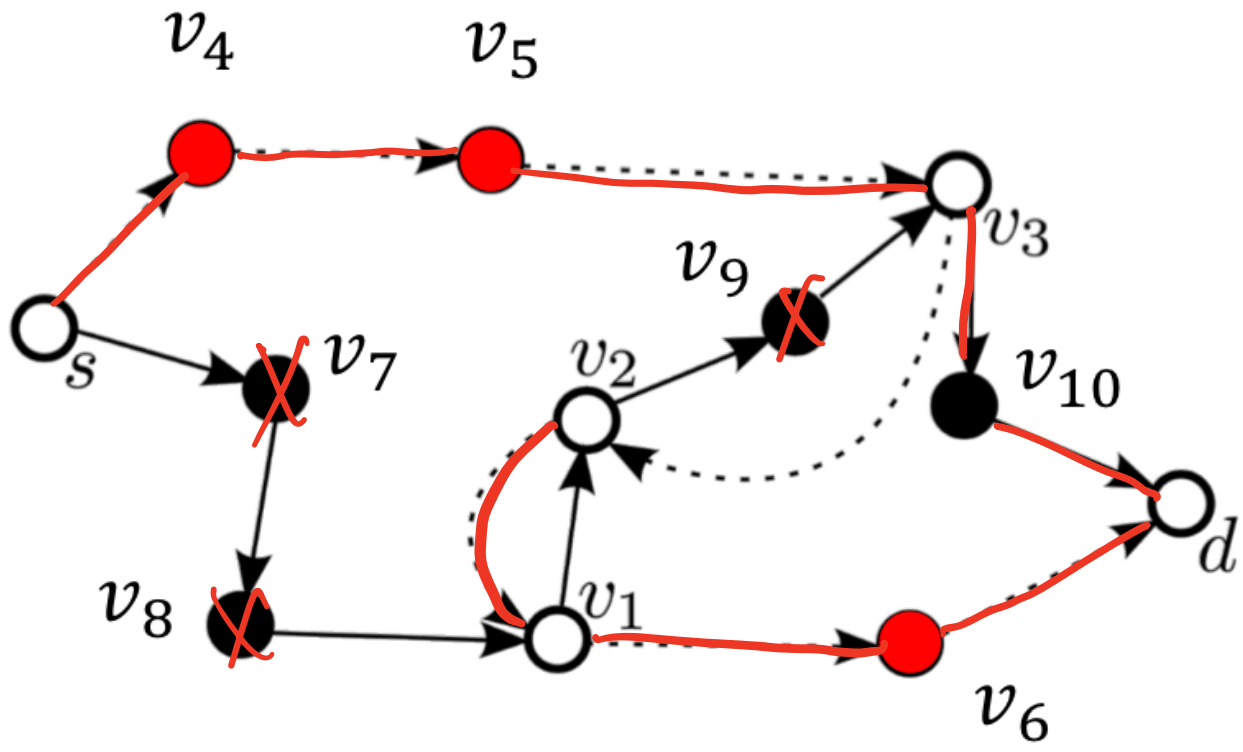
test-stored test = 8

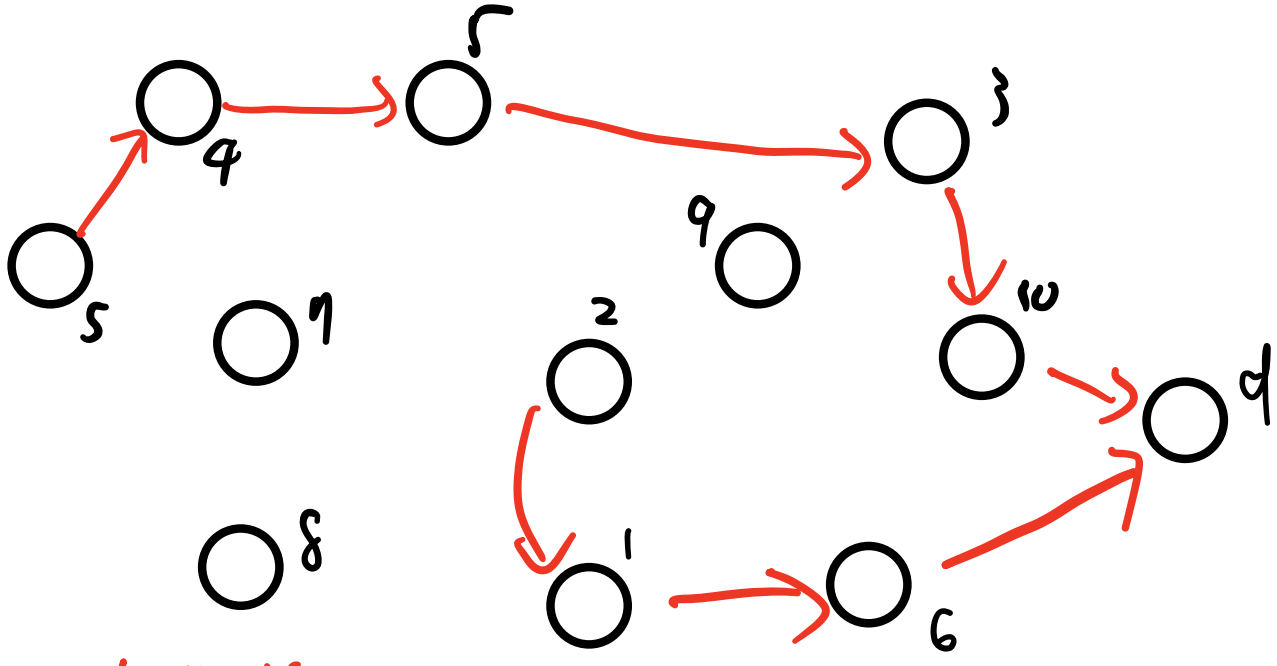
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | d |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 5 | 1 | 2 | 3 | ~~4~~ | ~~5~~ | ~~9~~ | ~~10~~ | X | X | X | X |

不能用 unused 去 initialize，得用 chunk 去改

+

問題，要改掉




最後一步判斷可開 array 存final移動