Dynamic Programming

Lecture 5 추가

Dynamic Programming

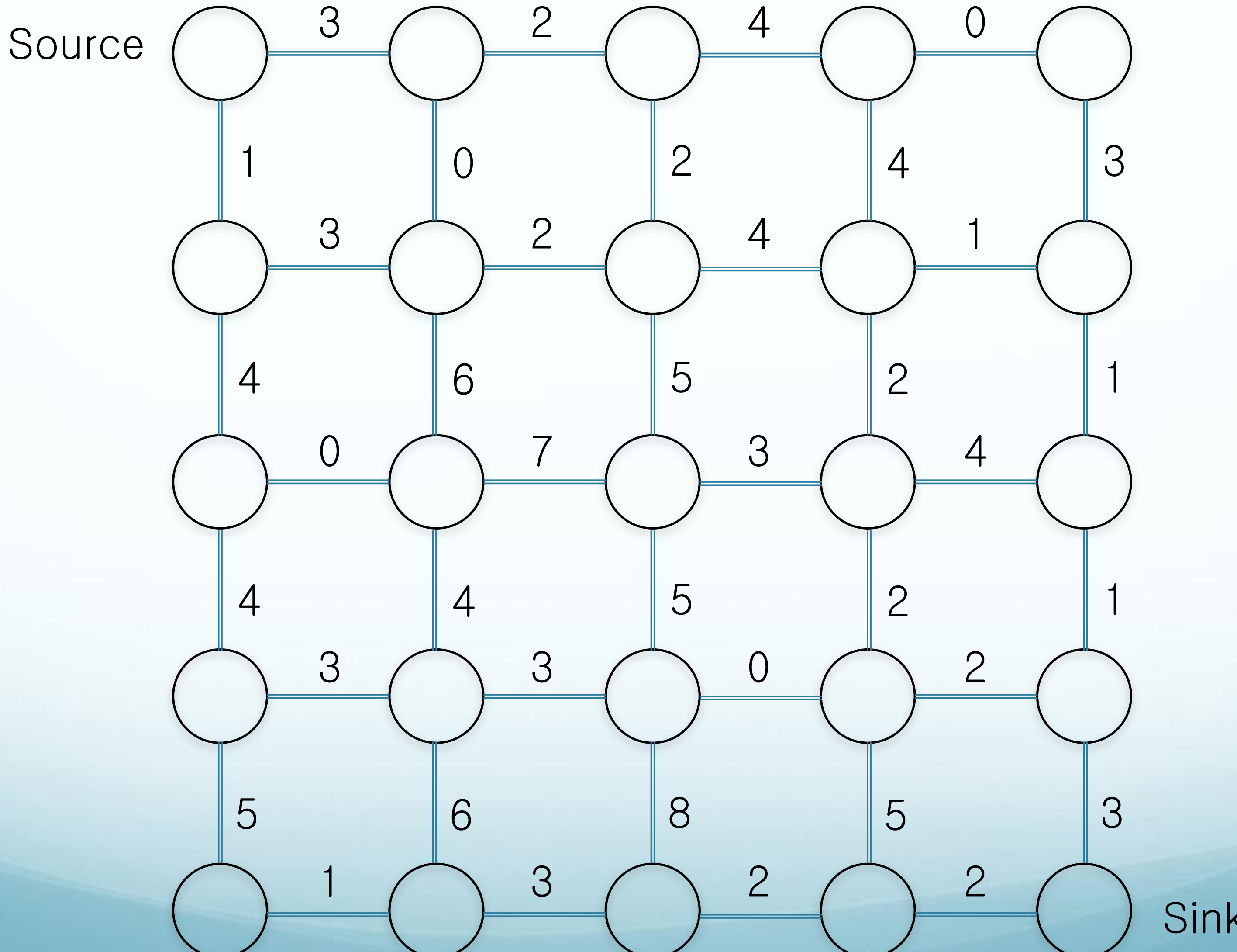
- Manhattan Tourist Problem – At Mahattan, a tourist wants to see as many attractions as possible.

Problem: Find a longest path in a weighted grid. Tourist can move either east and south.

Input : A weighted grid G with two distinguished vertices: a source and a sink.

Output: A longest path in G from source to sink.

Manhattan Tourist Problem



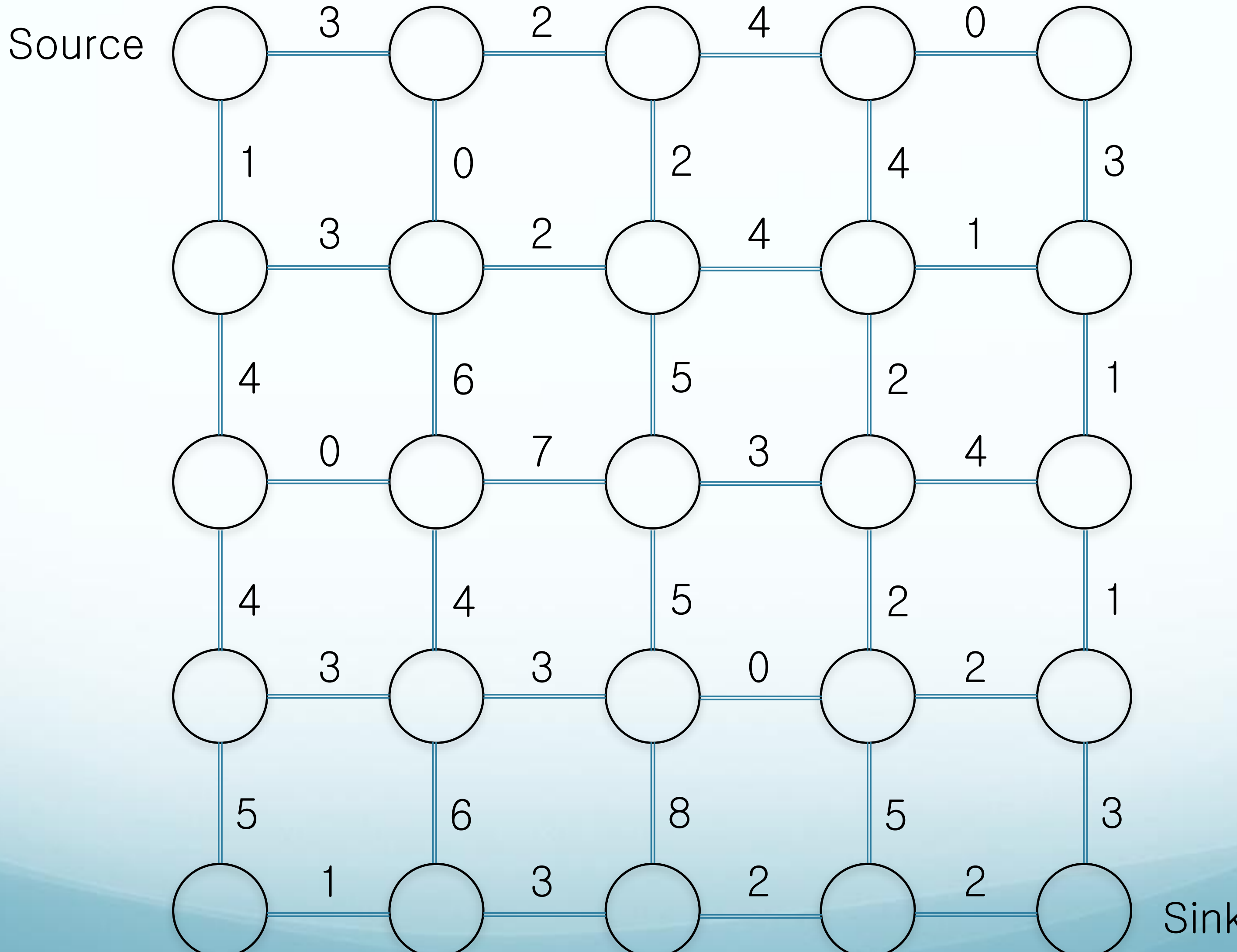
Exhaustive search/Brute force Algorithm

Examines every possible alternative to find one particular solution

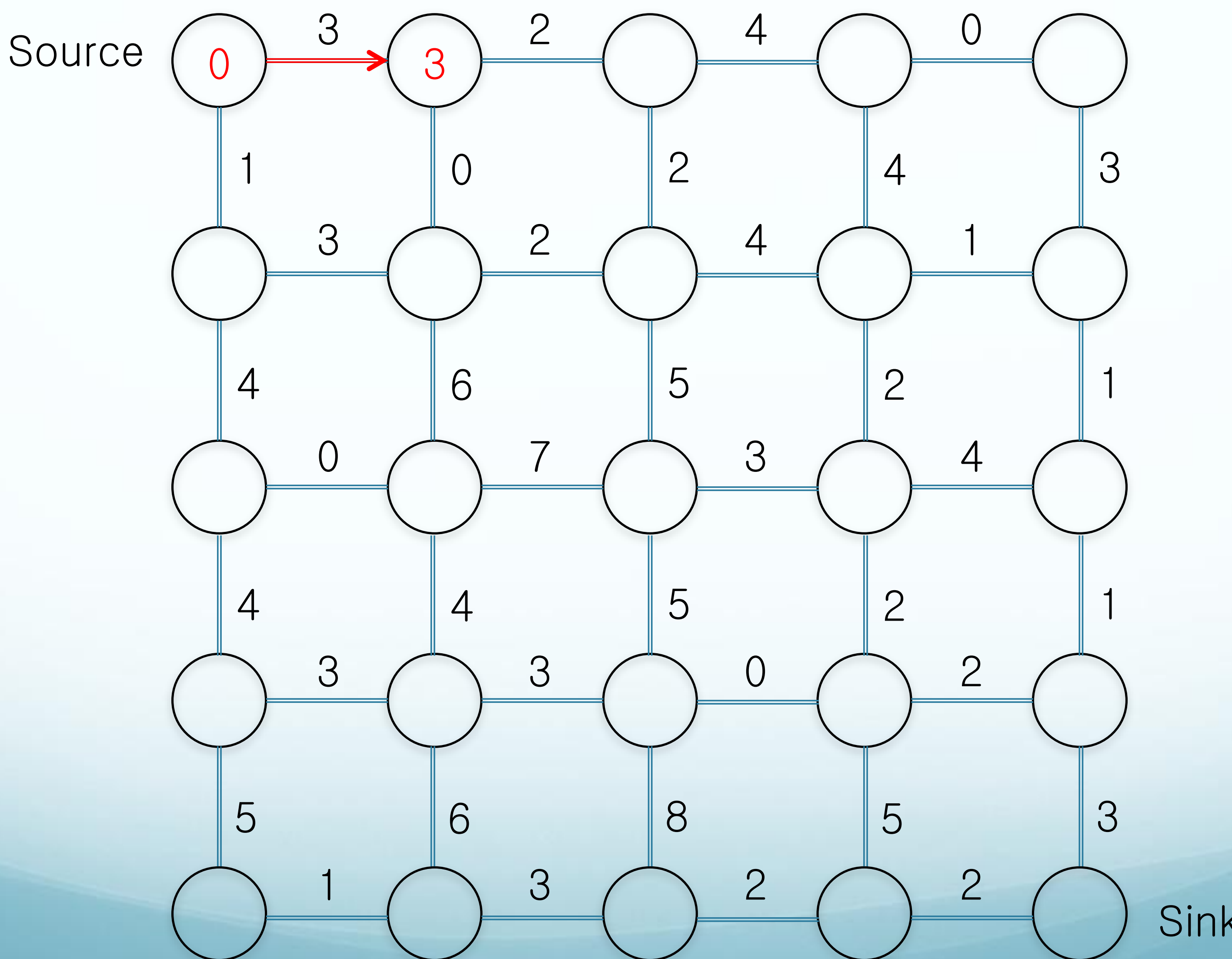
Greedy Algorithm

- Choose between east and south by comparing how many attractions tourist would see if he moves one block east or south.

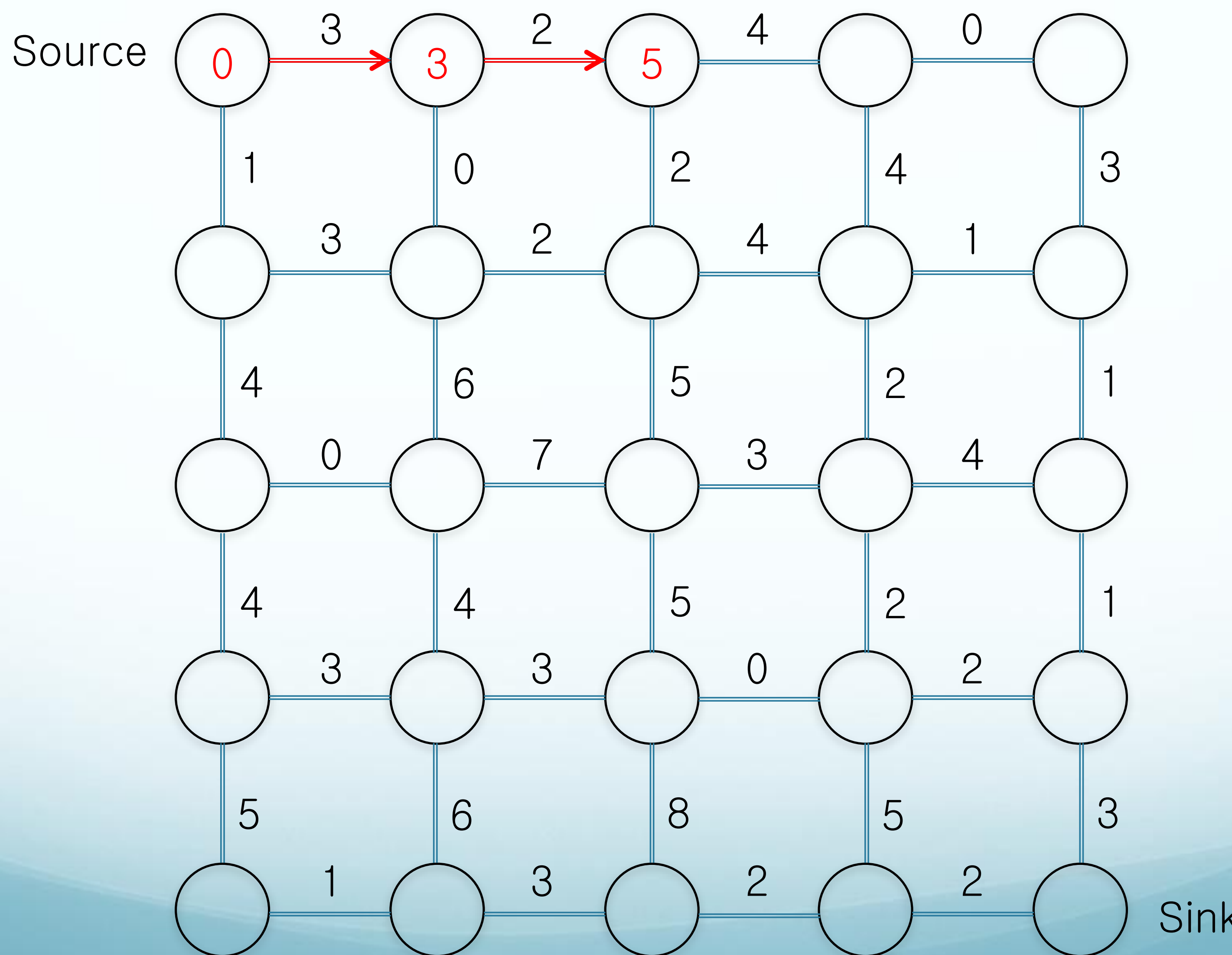
Greedy Algorithm



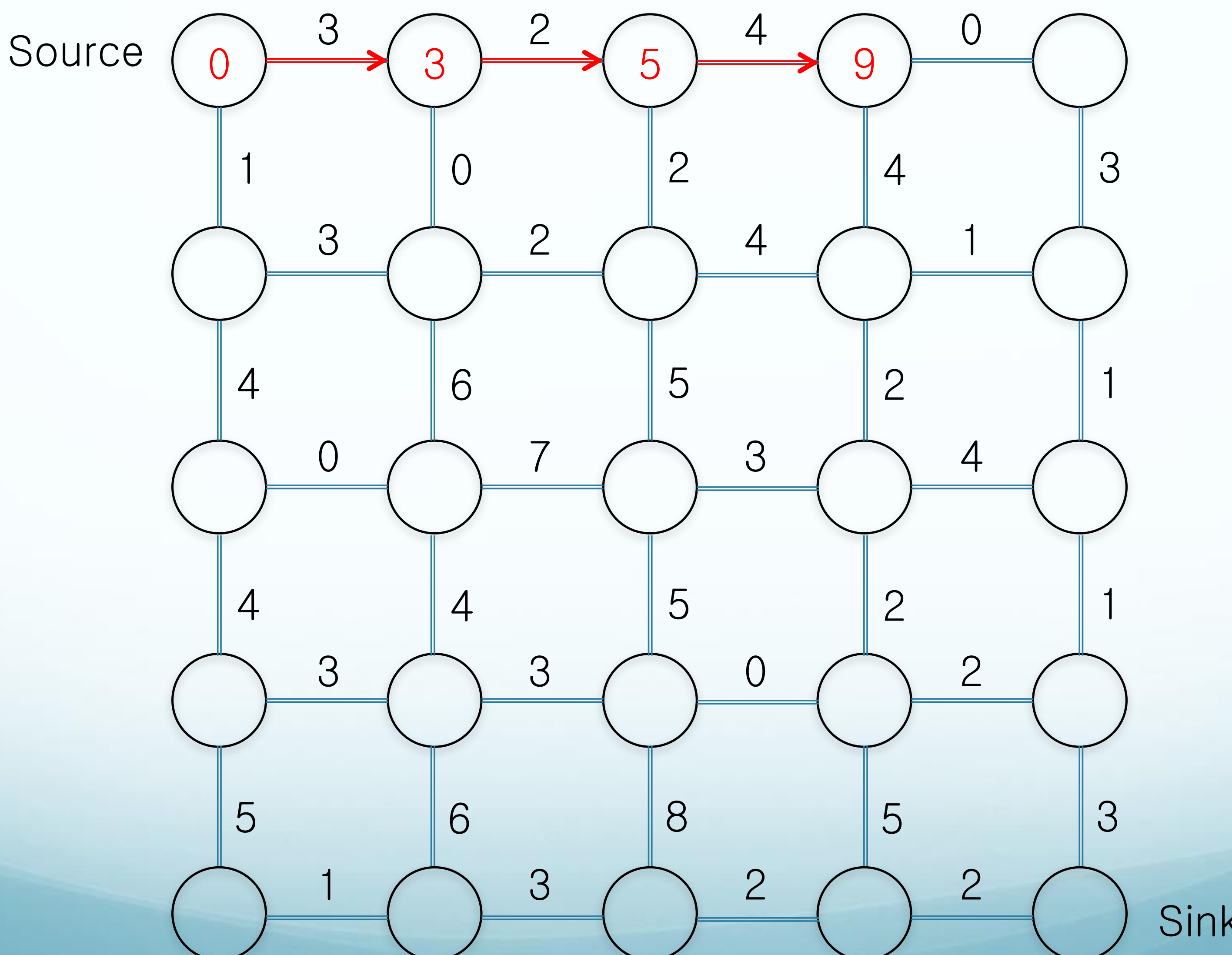
Greedy Algorithm



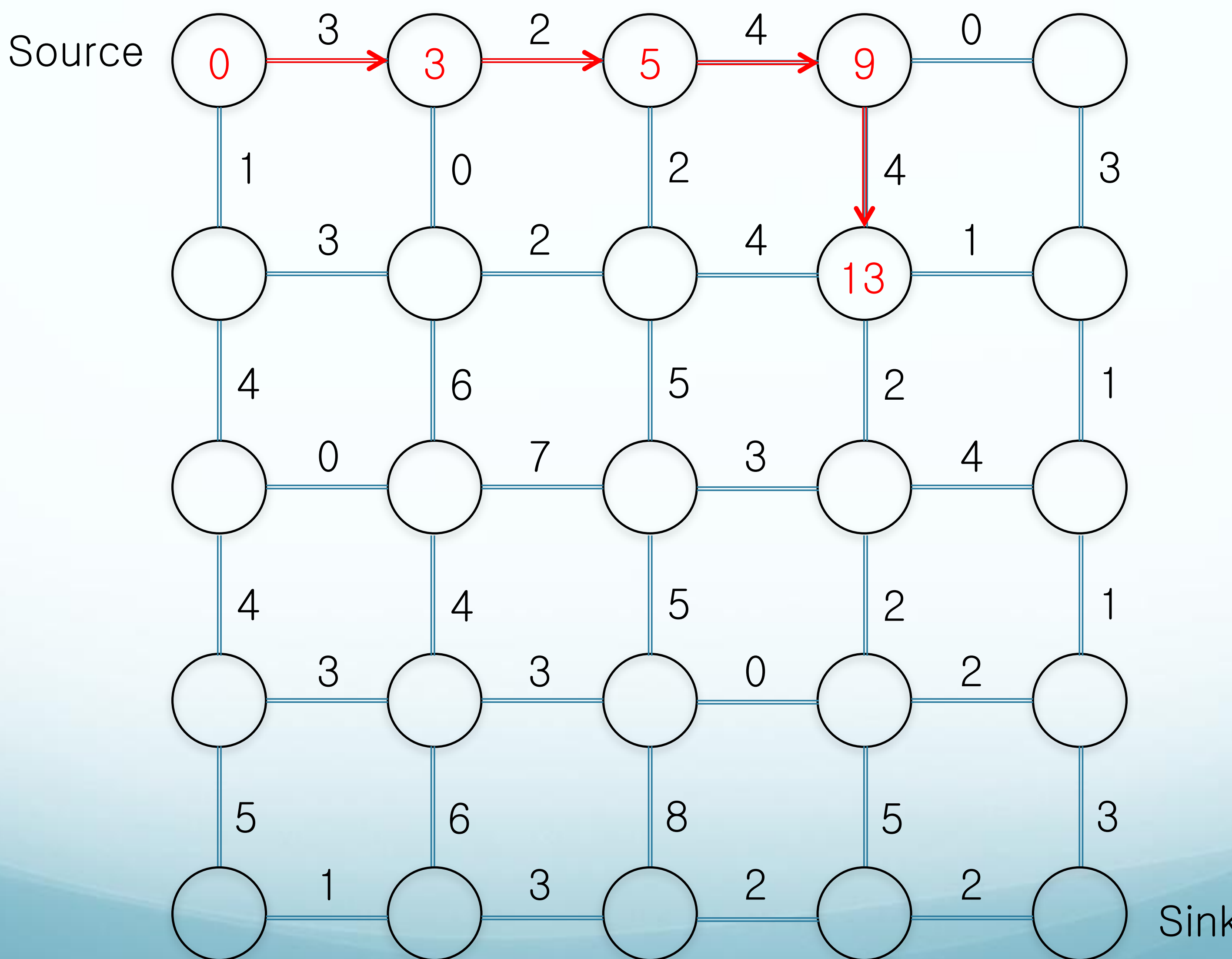
Greedy Algorithm



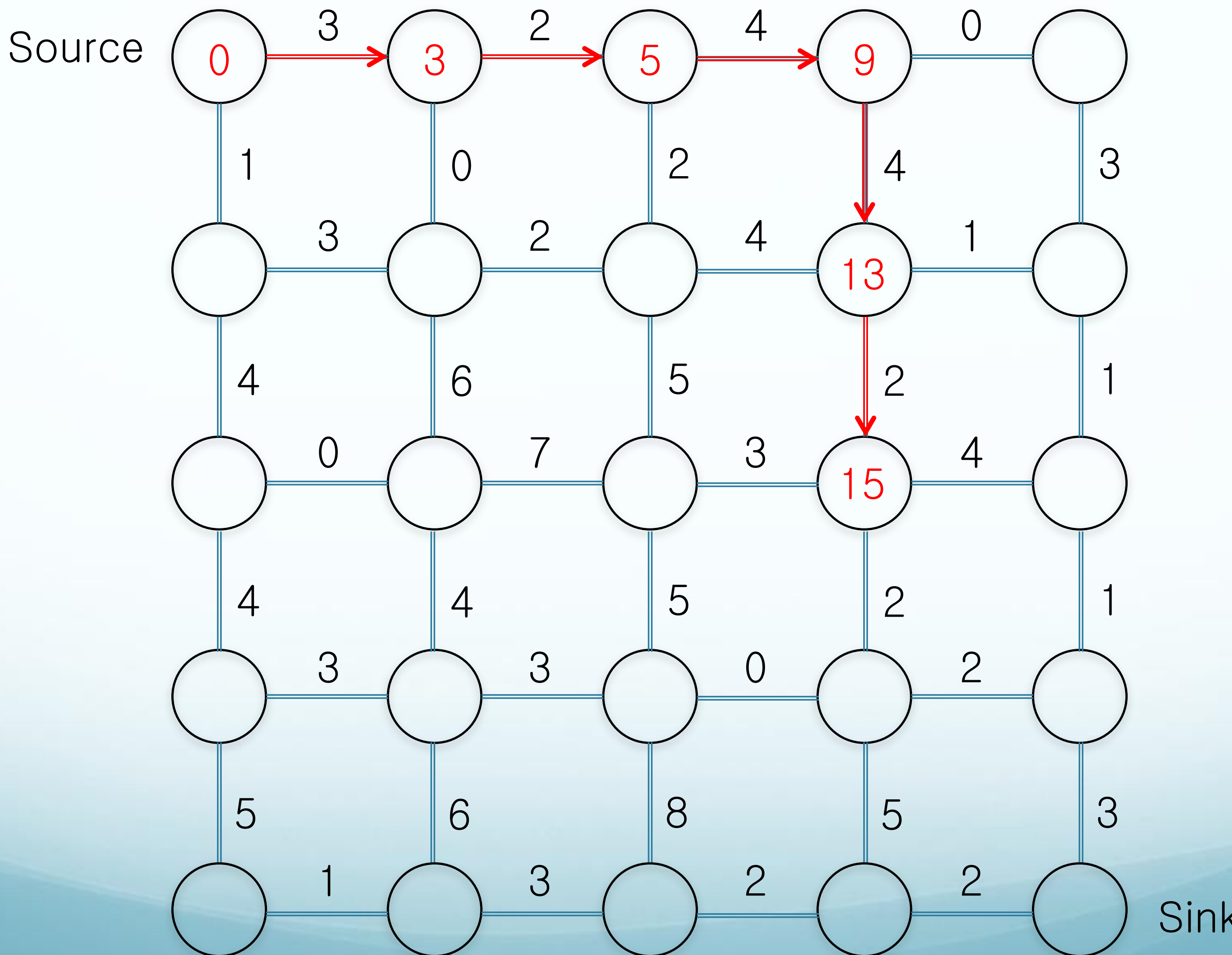
Greedy Algorithm



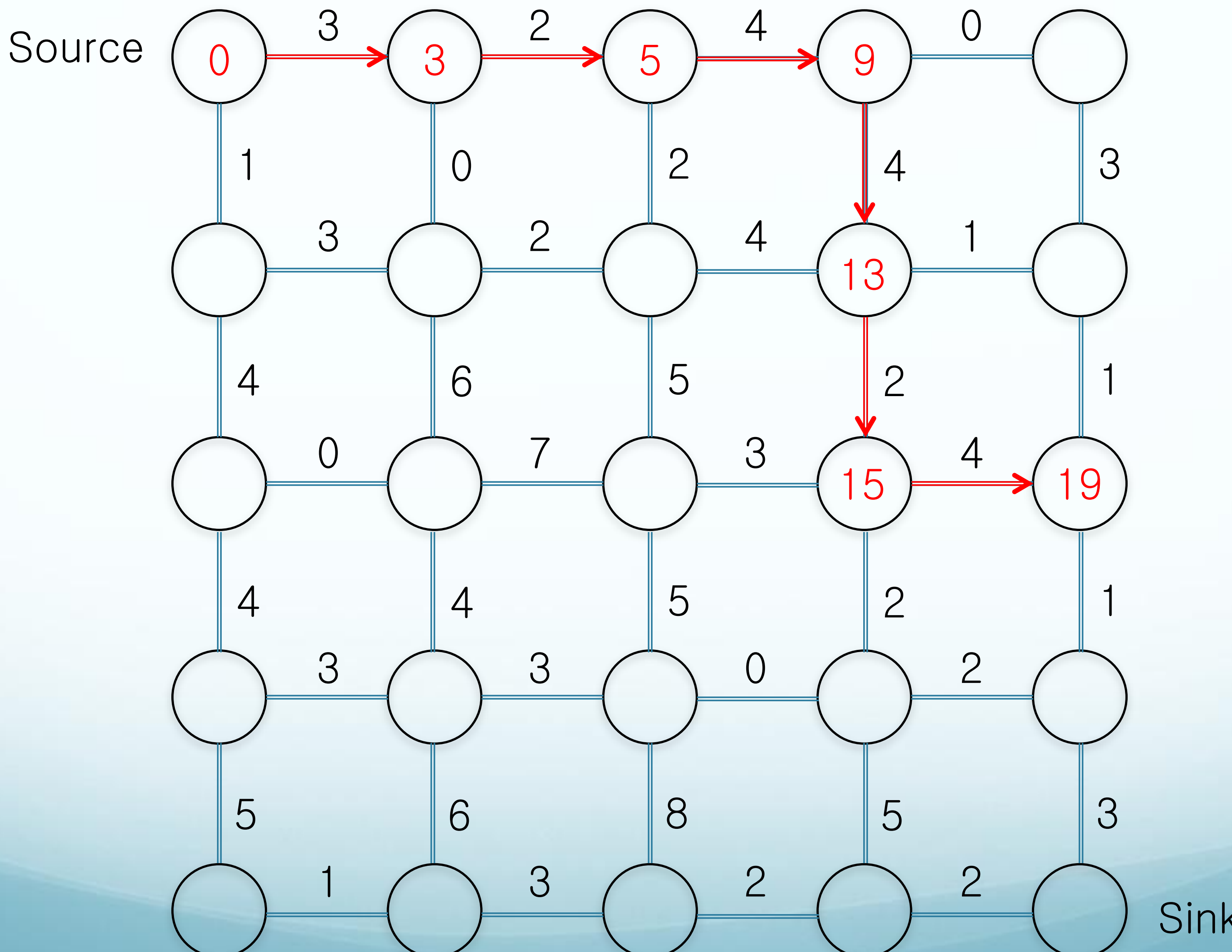
Greedy Algorithm



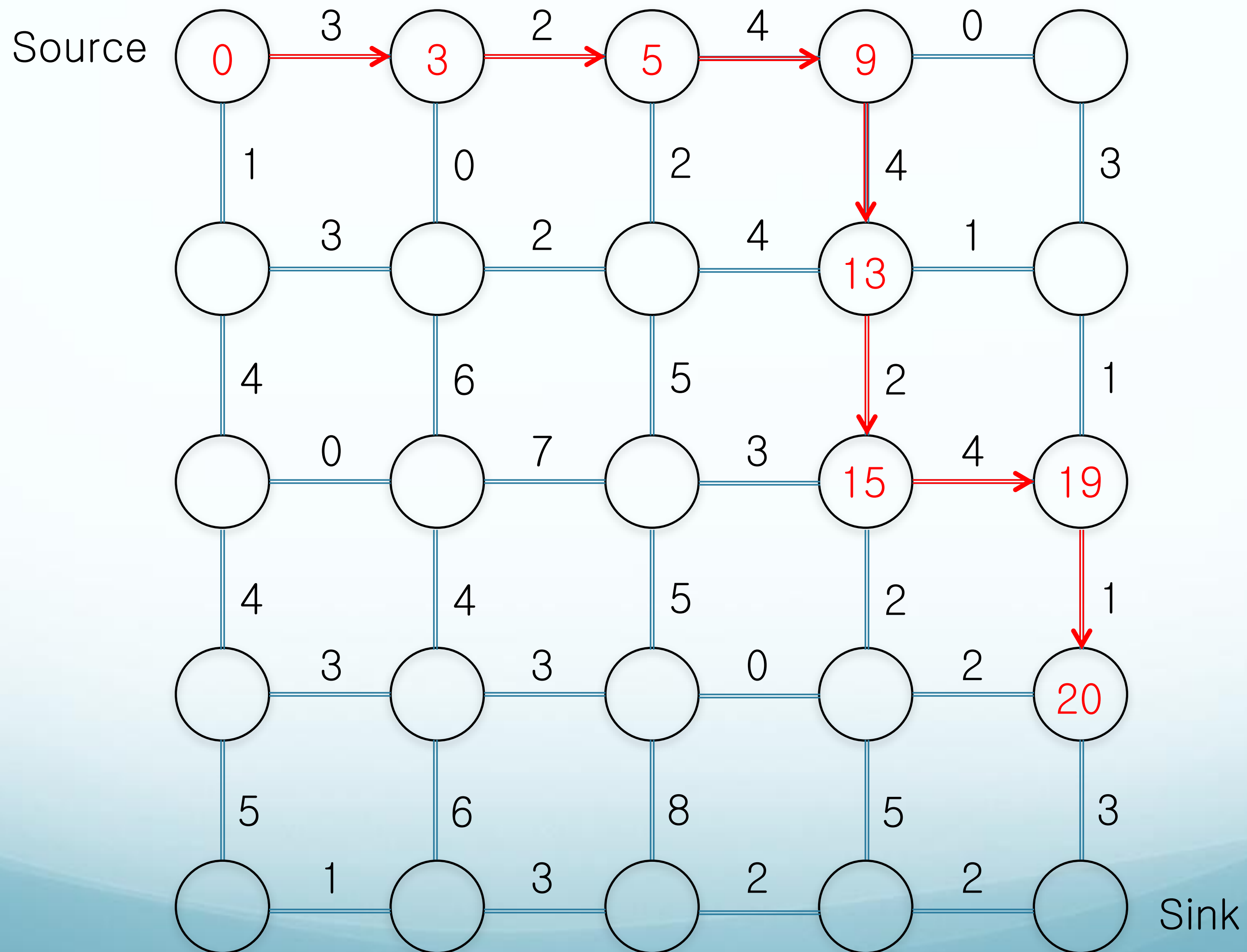
Greedy Algorithm



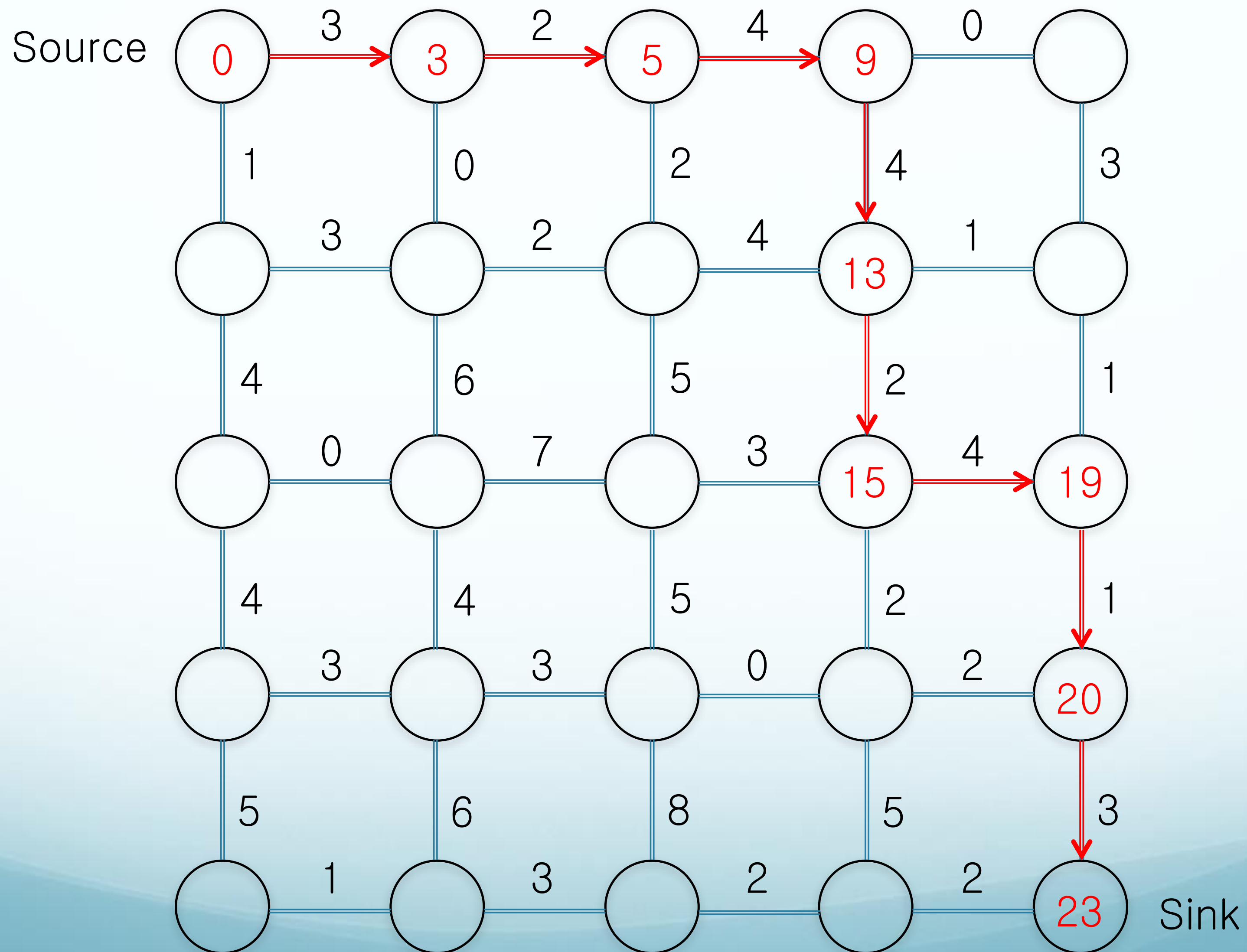
Greedy Algorithm



Greedy Algorithm



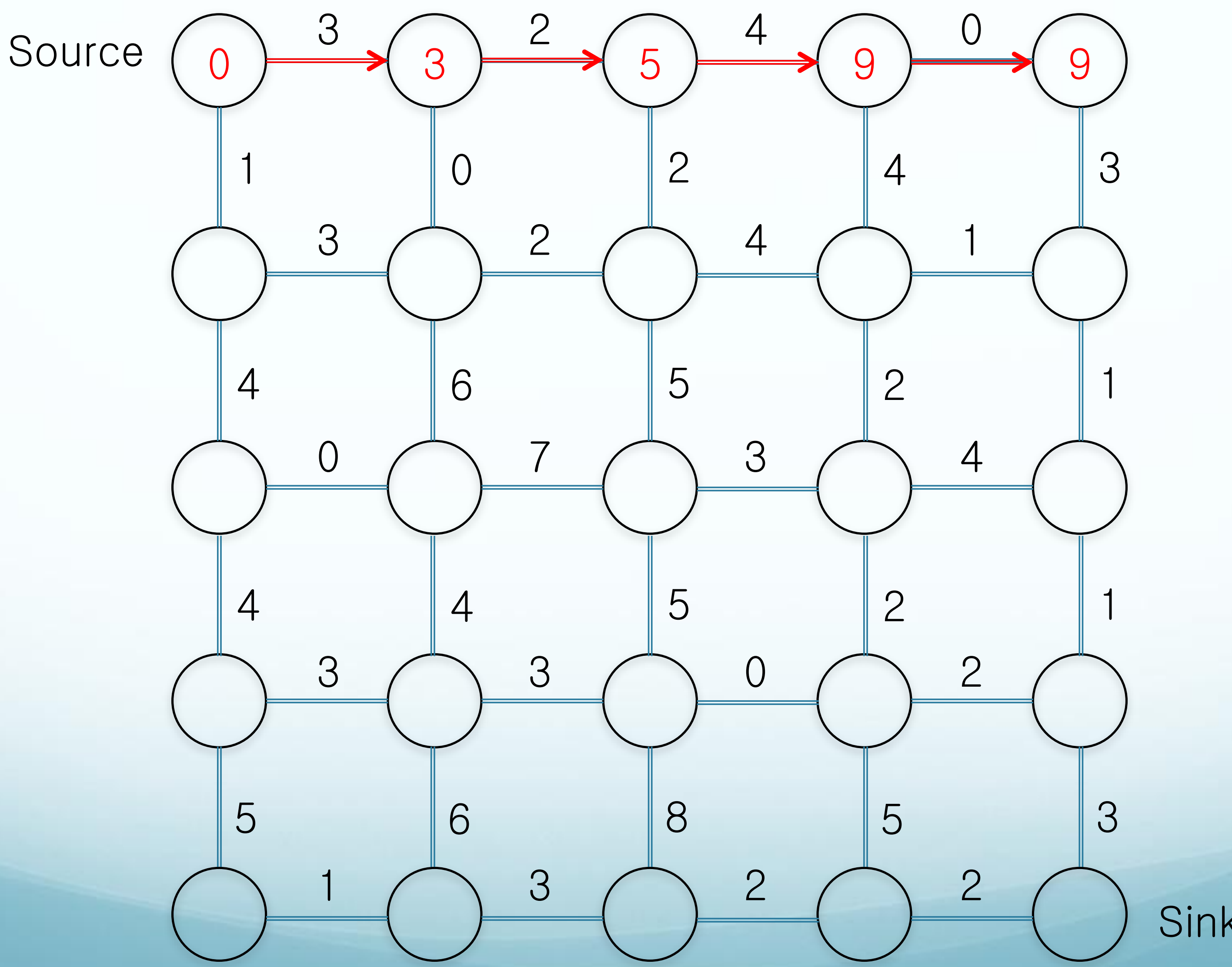
Greedy Algorithm



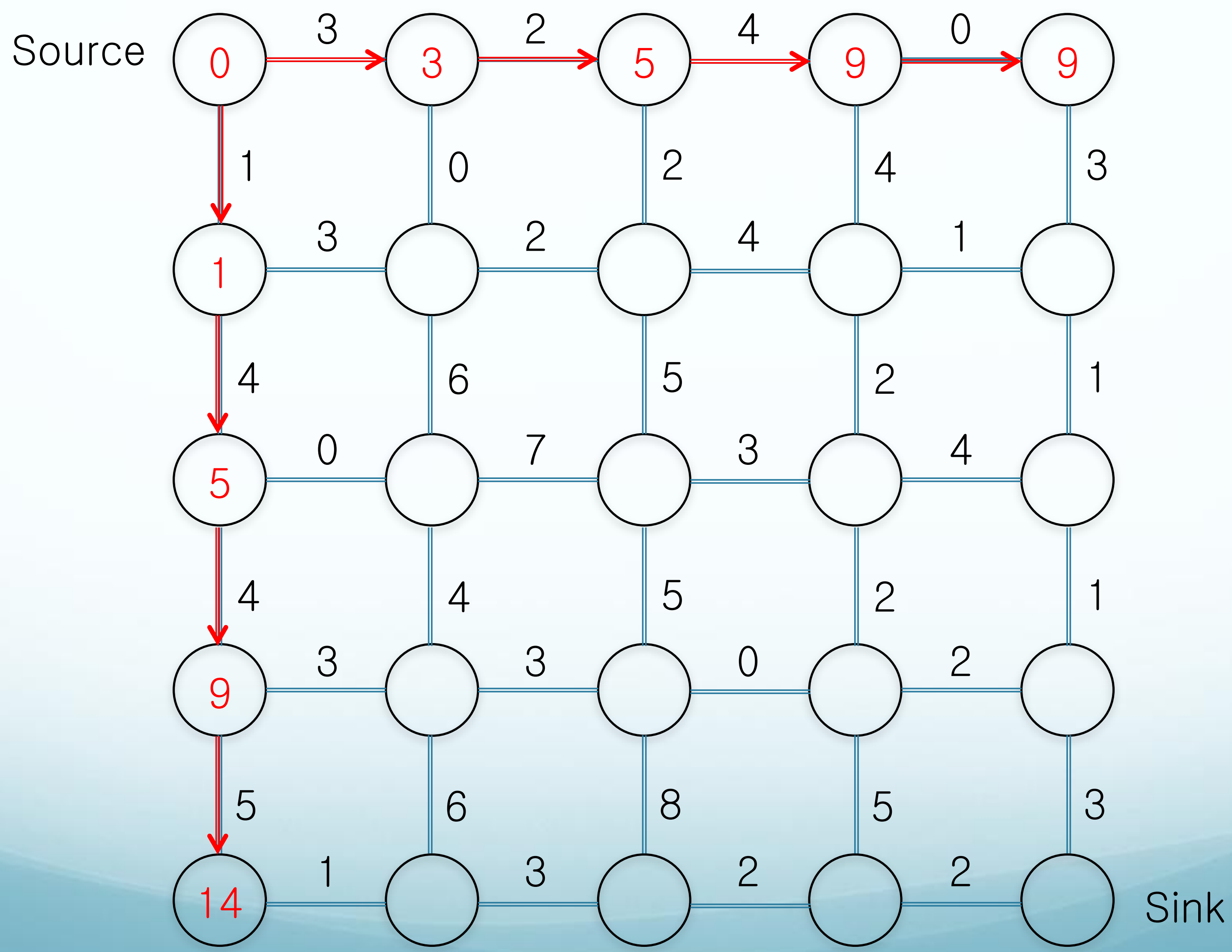
Dynamic Programming Algorithm

- Break a problem into smaller subproblems and use the solutions of the subproblems to construct the solution of the larger ones.
- Organizes computations to avoid recomputing values that you already know.

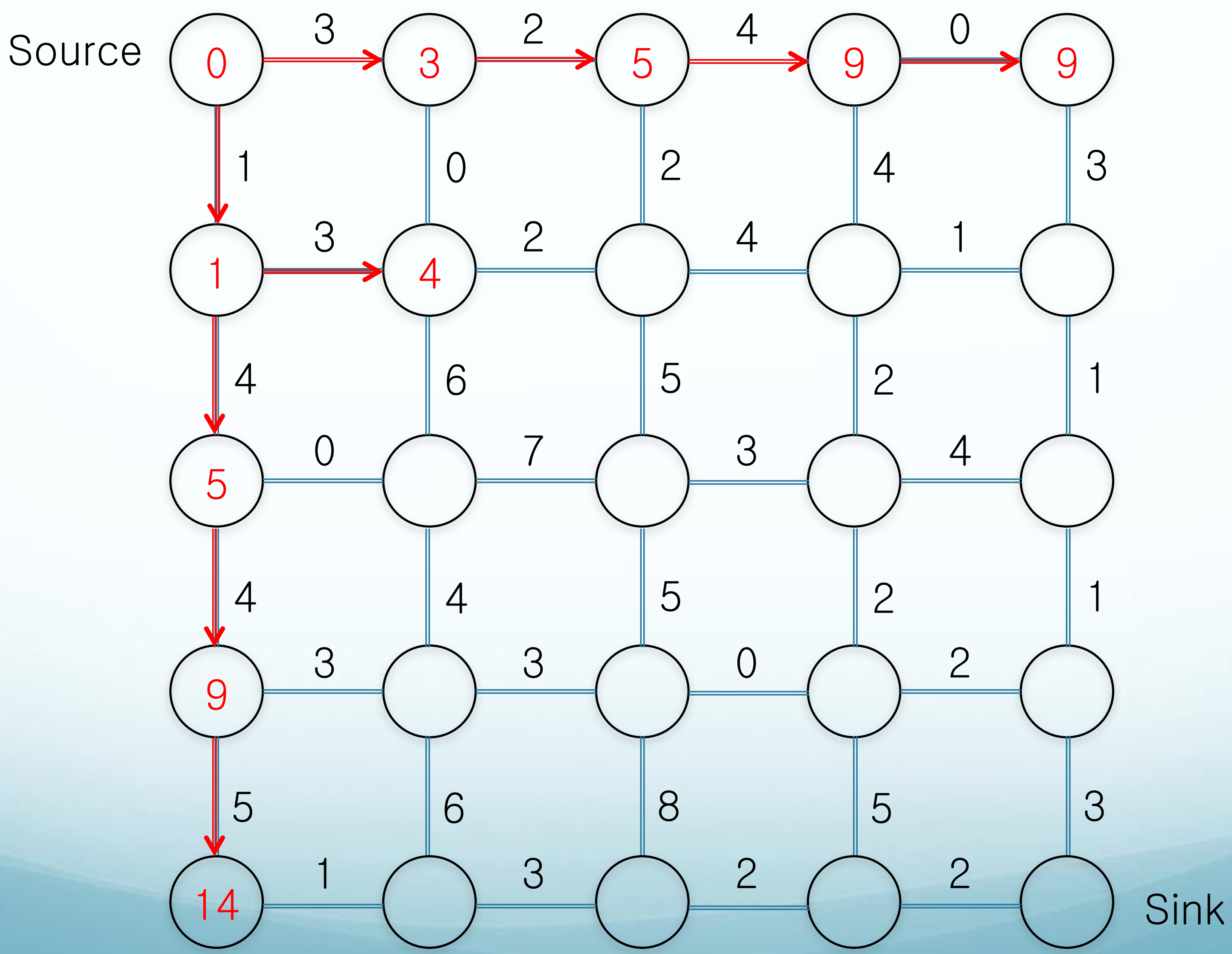
Dynamic Programming



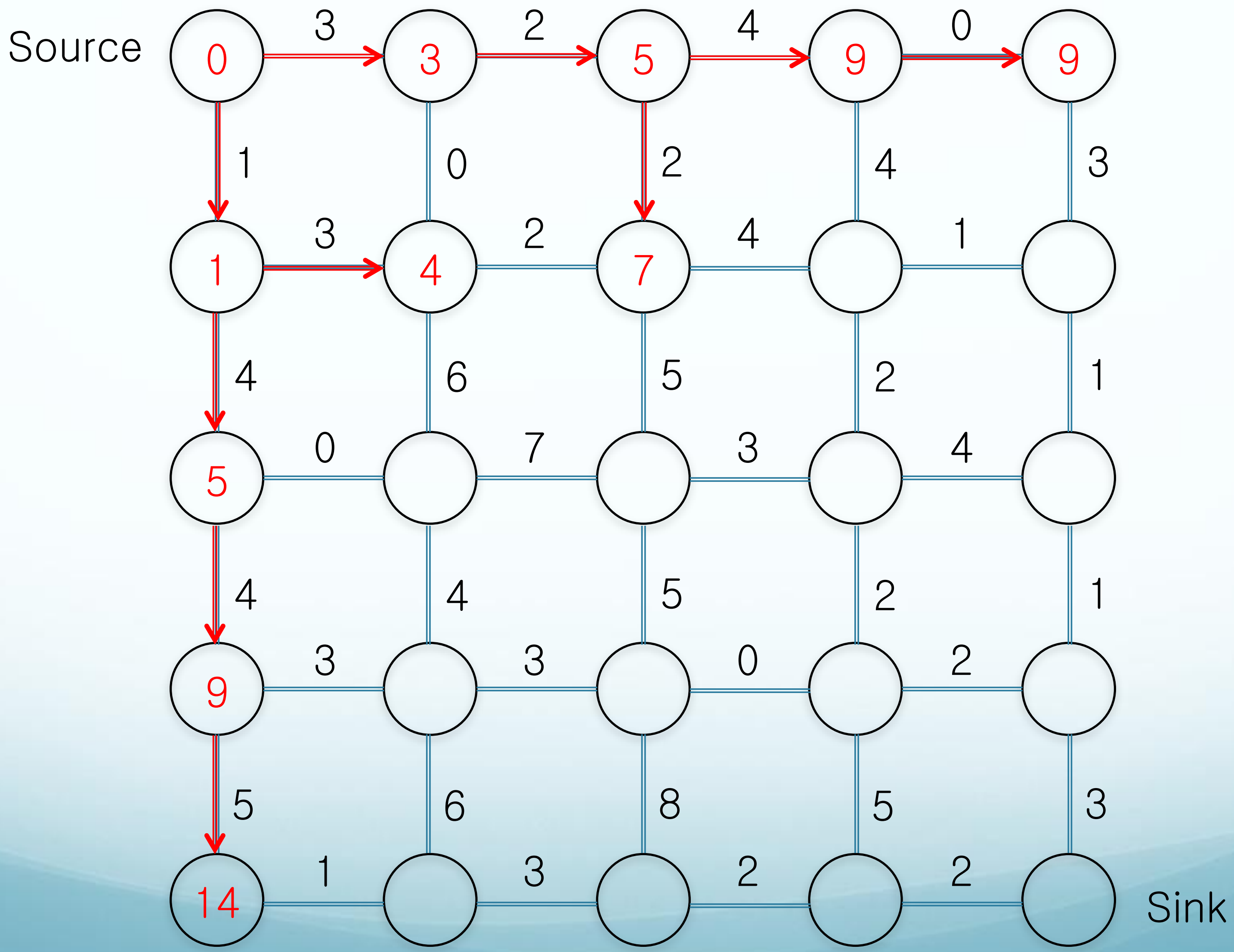
Dynamic Programming



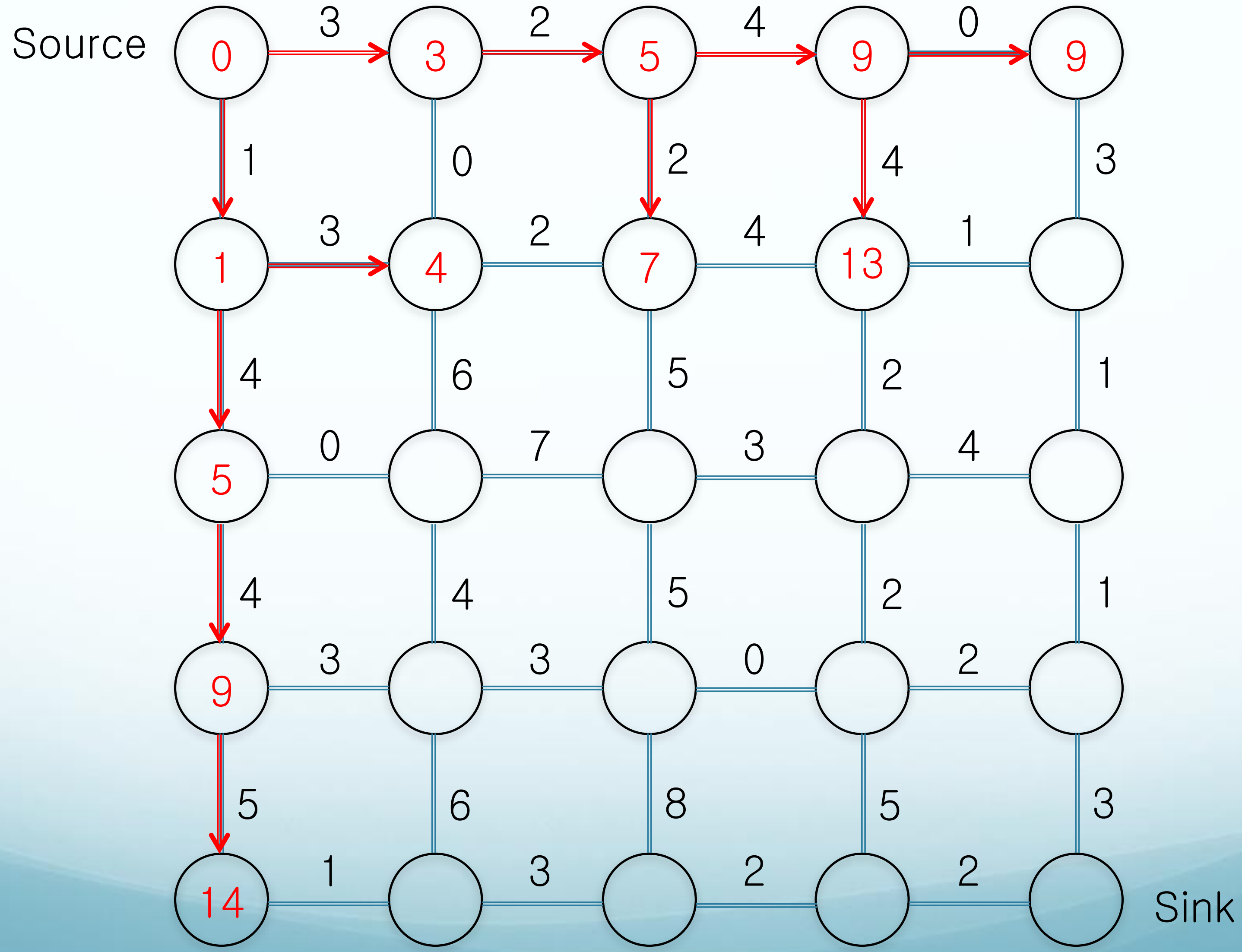
Dynamic Programming



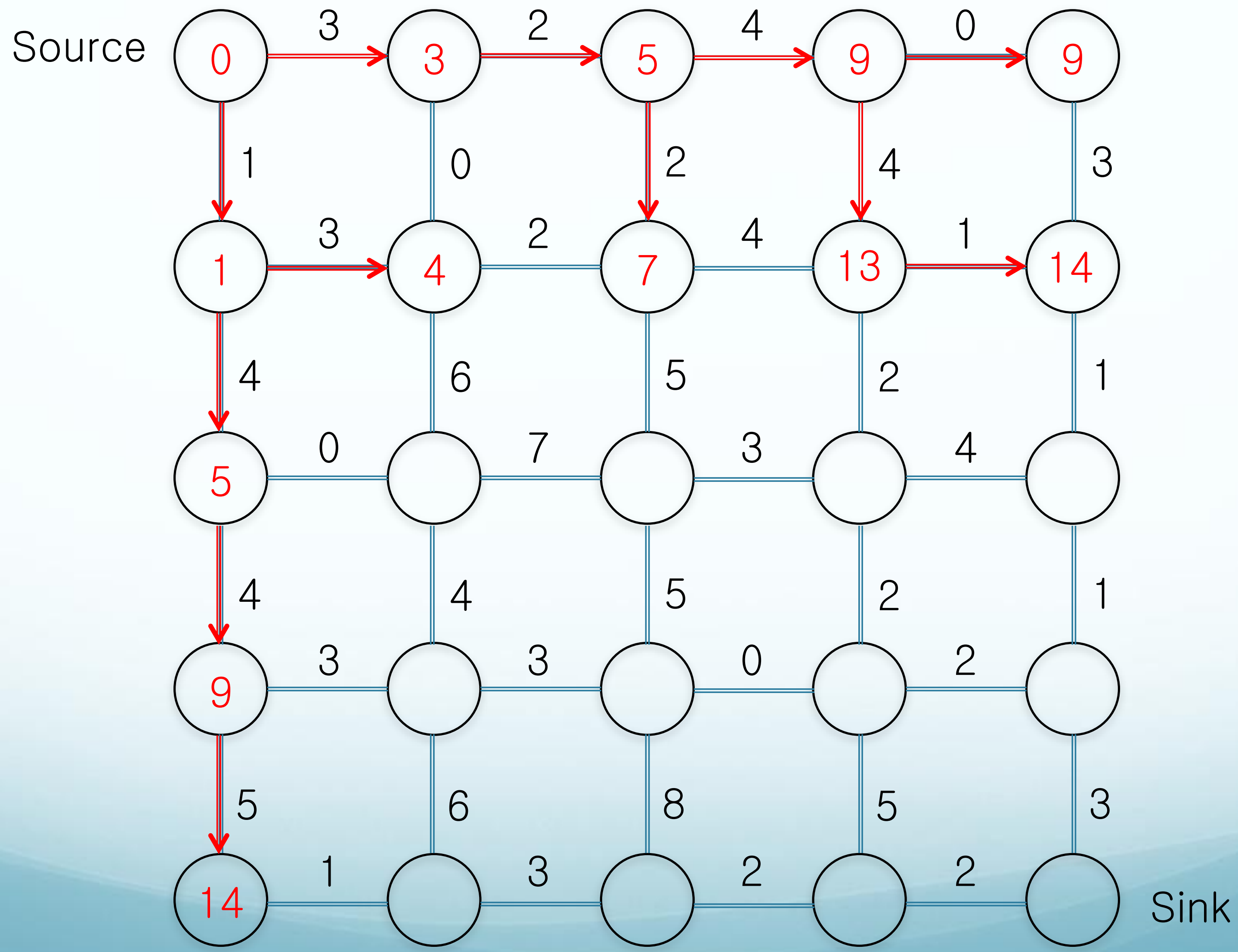
Dynamic Programming



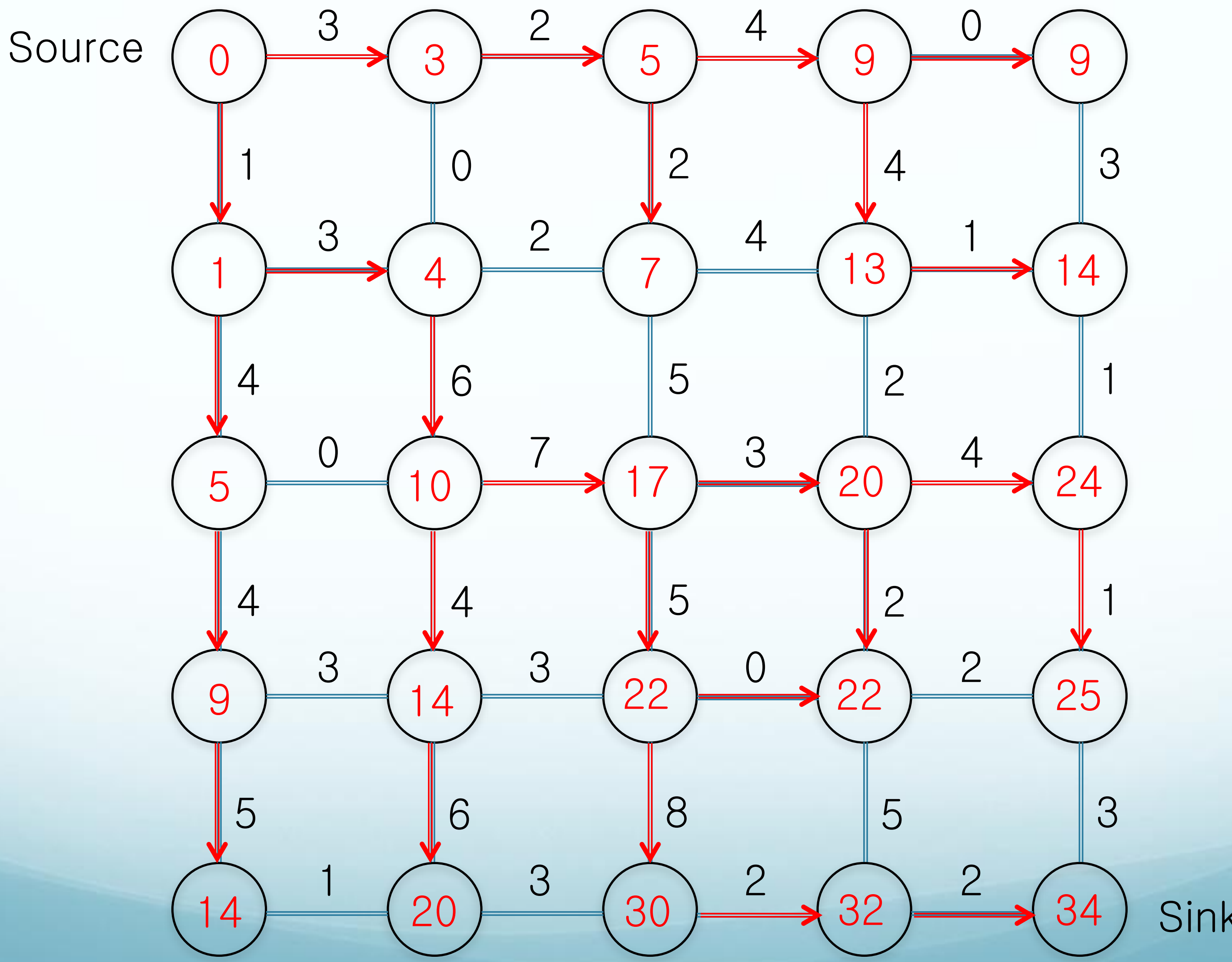
Dynamic Programming



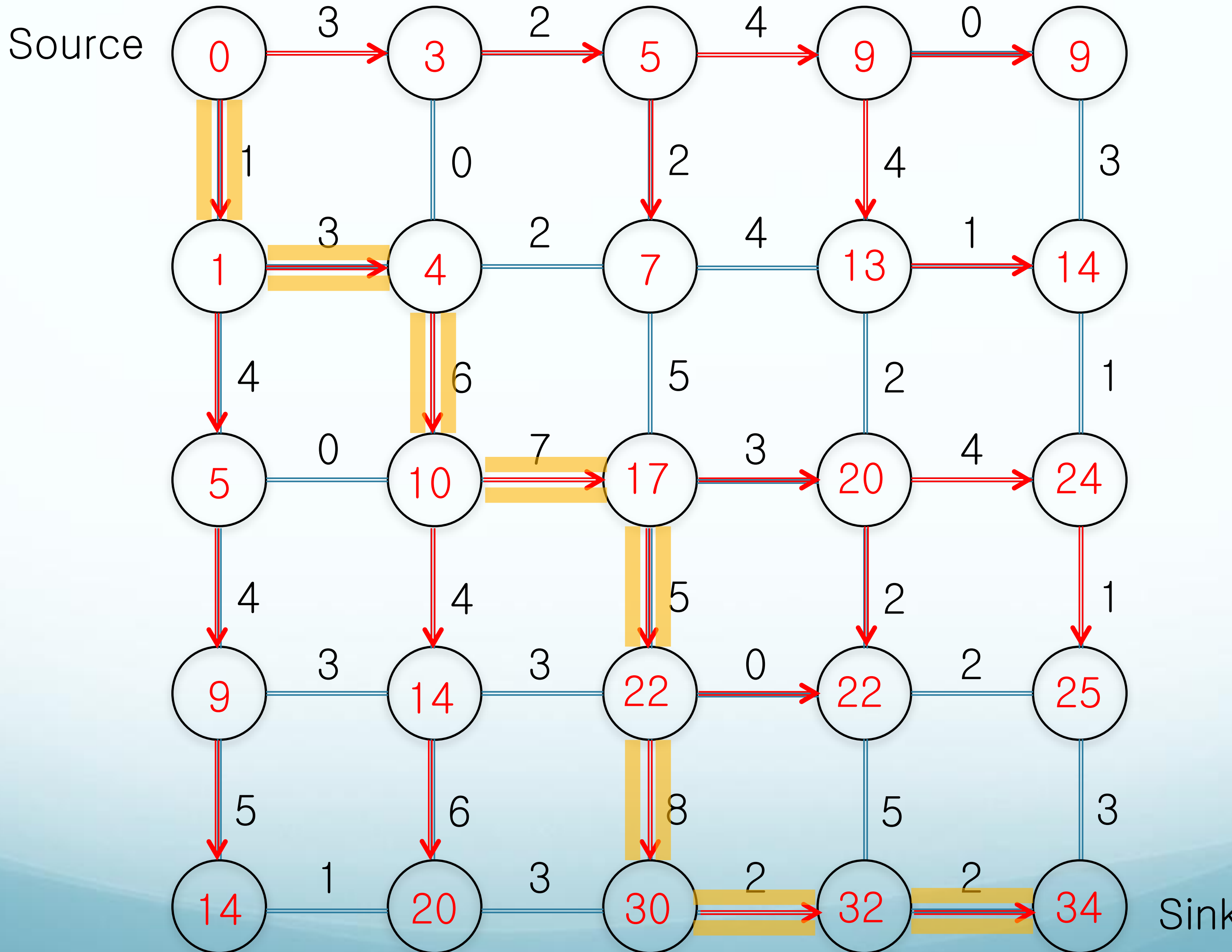
Dynamic Programming



Dynamic Programming



Dynamic Programming



동전 교환 문제
문제 정의

Coin Change Problem

- **Problem:** Convert some amount of money ***M*** into given denominations, using the smallest possible number of coins.

- **Input:** An amount of money, ***M***, and an array of ***d*** denominations $\boldsymbol{c}=(c_1, c_2, \ldots, c_d)$, in decreasing order of value ($c_1 > c_2 > \ldots > c_d$).

- **Output:** A list of ***d*** integers $\boldsymbol{k}=(k_1, k_2, \ldots, k_d)$ such that $c_1k_1 + c_2k_2 + \ldots + c_dk_d = \boldsymbol{M}$, and $k_1 + k_2 + \ldots + k_d$ is as small as possible.

동전 교환 문제

전역탐색 알고리즘(Exhaustive search/Brute force Algorithm)

Examines every possible alternative to find one particular solution

BruteForceChange (***M, c, d***)

힌트: 동전의 모든 조합을 다 생각해보자.

return (bestChange)

➡ 최악의 경우 M^d 만큼의 연산이 필요함, $\boldsymbol{O(M^d)}$

동전 교환 문제
탐욕 알고리즘 (Greedy Algorithm)

- 각 단계에서 가장 이익이 되는 방법을 쫓는 방법.
  1 단계 : 사용가능한 가장 큰 동전을 사용
  2 단계 : 1단계 이후 남은 금액에서 사용가능한 가장 큰 동전을 사용.
  3 단계 : 2단계 이후 남은 금액에서 사용가능한 가장 큰 동전을 사용.
  …
  남은 금액이 없을때 까지 계속 수행
- 예) **Input:** ***M***= 37, ***c***=(25, 20, 10, 5, 1), ***d***=5

Output: ***k*** =

동전 교환 문제
탐욕 알고리즘 (Greedy Algorithm)

- 각 단계에서 가장 이익이 되는 방법을 쫓는 방법.

  1 단계 : 사용가능한 가장 큰 동전을 사용

  2 단계 : 1단계 이후 남은 금액에서 사용가능한 가장 큰 동전을 사용.

  3 단계 : 2단계 이후 남은 금액에서 사용가능한 가장 큰 동전을 사용.

  …

  남은 금액이 없을때 까지 계속 수행

- 예) **Input:** ***M***= 37, ***c***=(25, 20, 10, 5, 1), ***d***=5

  25 사용, 남은 금액= 37-25=12

  10 사용, 남은 금액= 12-10=2

  1 사용, 남은 금액 2-1=1

  1 사용, 남은 금액 1-1=0

  Output: ***k*** = (1, 0, 1, 0, 2)

동전 교환 문제
탐욕 알고리즘 (Greedy Algorithm)

GreedyChange (***M, c, d***)

return ($k_1, k_2, \ldots, k_d$)

동전 교환 문제
탐욕 알고리즘 (Greedy Algorithm)

```
GreedyChange ( M, c, d )
    while (M > 0)
        for i=d to 1
            if M >= c_i
                M = M – c_i
                k_i = k_i + 1
                break
    return (k_1, k_2 , … , k_d)
```

동전 교환 문제
탐욕 알고리즘 (Greedy Algorithm)

```
GreedyChange ( M, c, d )
    while (M > 0)
        for i=d to 1
            if M >= c_i
                M = M – c_i
                k_i = k_i + 1
                break
    return (k_1, k_2 , … , k_d)
```

```
BetterGreedyChange ( M, c, d )
    for i=d to 1
        k_i = M / c_i
        M = M – c_i x k_i
    return (k_1, k_2 , … , k_d)
```

동전 교환 문제

탐욕 알고리즘 (Greedy Algorithm)

```
GreedyChange ( M, c, d )
    while (M > 0)
        for i=d to 1
            if M >= c_i
                M = M – c_i
                k_i = k_i + 1
                break
    return (k_1, k_2 , … , k_d)
```

최악의 경우 ***Md*** 만큼의 연산이 필요함, ***O(Md)***

```
BetterGreedyChange ( M, c, d )
    for i=d to 1
        k_i =    M / c_i
        M = M – c_i x k_i
    return (k_1, k_2 , … , k_d)
```

최악의 경우 ***d*** 만큼의 연산이 필요함, ***O(d)***

알고리즘의 평가기준
정당성 (correctness)

- 동전 교환 문제

  M=40, ***c*** = (25,20,10,5,1)

  탐욕 알고리즘: ***k*** = (1, 0, 1, 1, 0) -> $\sum_{i=1}^{d} k_i = 3$

  Correct answer: ***k*** = (0, 2, 0, 0, 0) -> $\sum_{i=1}^{d} k_i = 2$

동전 교환 문제

전역탐색 알고리즘(Exhaustive search/Brute force Algorithm)

Examines every possible alternative to find one particular solution

BruteForceChange (***M, c, d***)

힌트: 동전의 모든 조합을 다 생각해보자.

return (bestChange)

➡ 최악의 경우 M^d 만큼의 연산이 필요함, $O(M^d)$

동전 교환 문제

전역탐색 알고리즘(Exhaustive search/Brute force Algorithm)

Examines every possible alternative to find one particular solution

$M/c_1 \times M/c_2 \times \ldots M/c_d = M^d / (c_1 \times c_2 \times \ldots \times c_d)$

```
BruteForceChange ( M, c, d )
    smallestNumberOfCoins = ∞
        for each (k_1 , k_2 , … , k_d ) from (0,…,0) to (M/c_1,…,M/c_d)
            valueOfCoins = Σ_{i=1}^{d} c_i k_i
                if valueOfCoins = M
                    numberOfCoins = Σ_{i=1}^{d} k_i
                    if numberOfCoins < smallestNumberOfCoins
                        smallestNumberOfCoins = numberOfCoins
                        bestChange = (k_1 , k_2 , … , k_d )
    return (bestChange)
```

➡ 최악의 경우 ***M^d*** 만큼의 연산이 필요함, ***O(M^d)***

동전 교환 문제

?

➡ ***O(Md)*** 알고리즘도 존재함.

Divide and Conquer Algorithm

- Break into smaller sub-problems
- Use recursive algorithm
- Ex) M=11, c=1,3,5

Best Combination for 11

- Best for 11-1 + 1cent
- Best for 11-3 + 3cent
- Best for 11-5 + 5cent

Recursive Algorithm

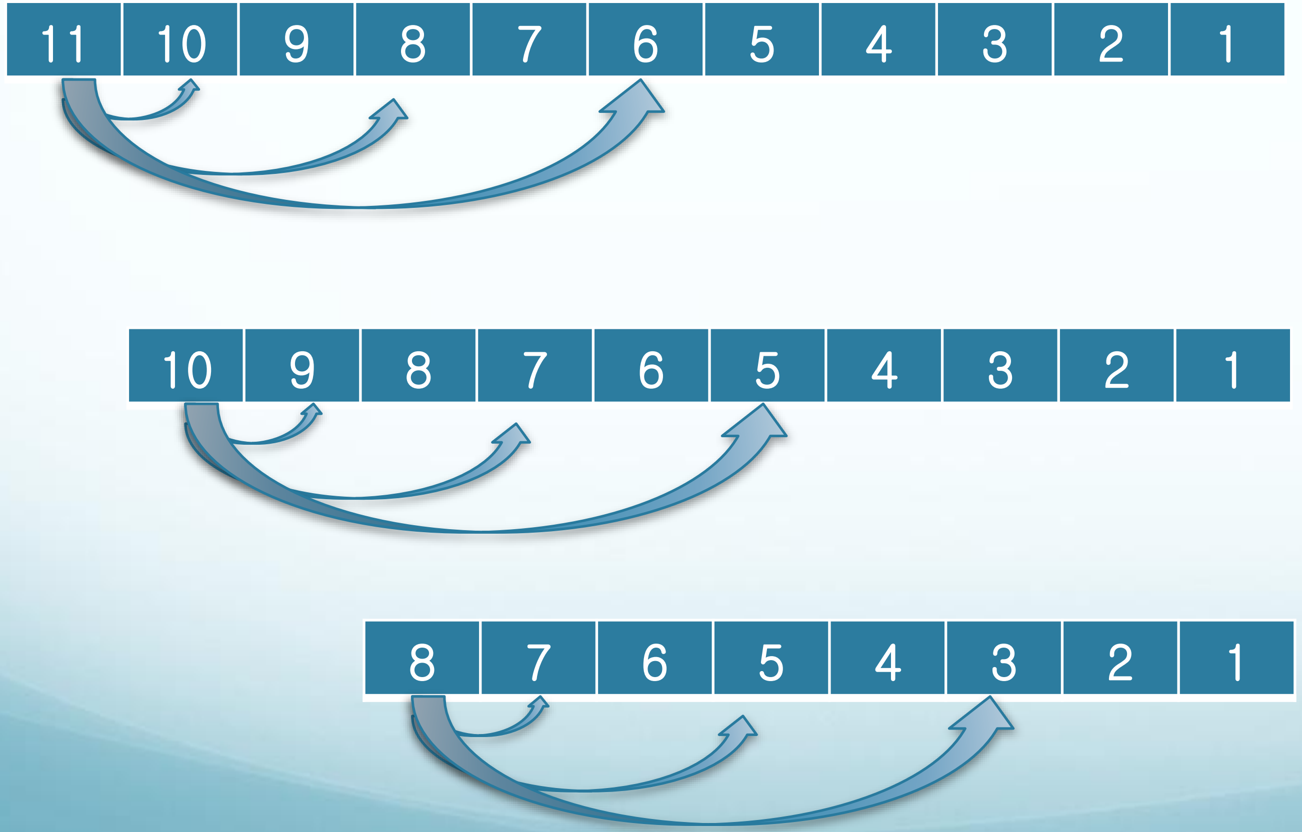
- Break into smaller sub-problems
- Use recursive alrogithm
- Ex) M=11, c=1,3,5

Best Combination for M

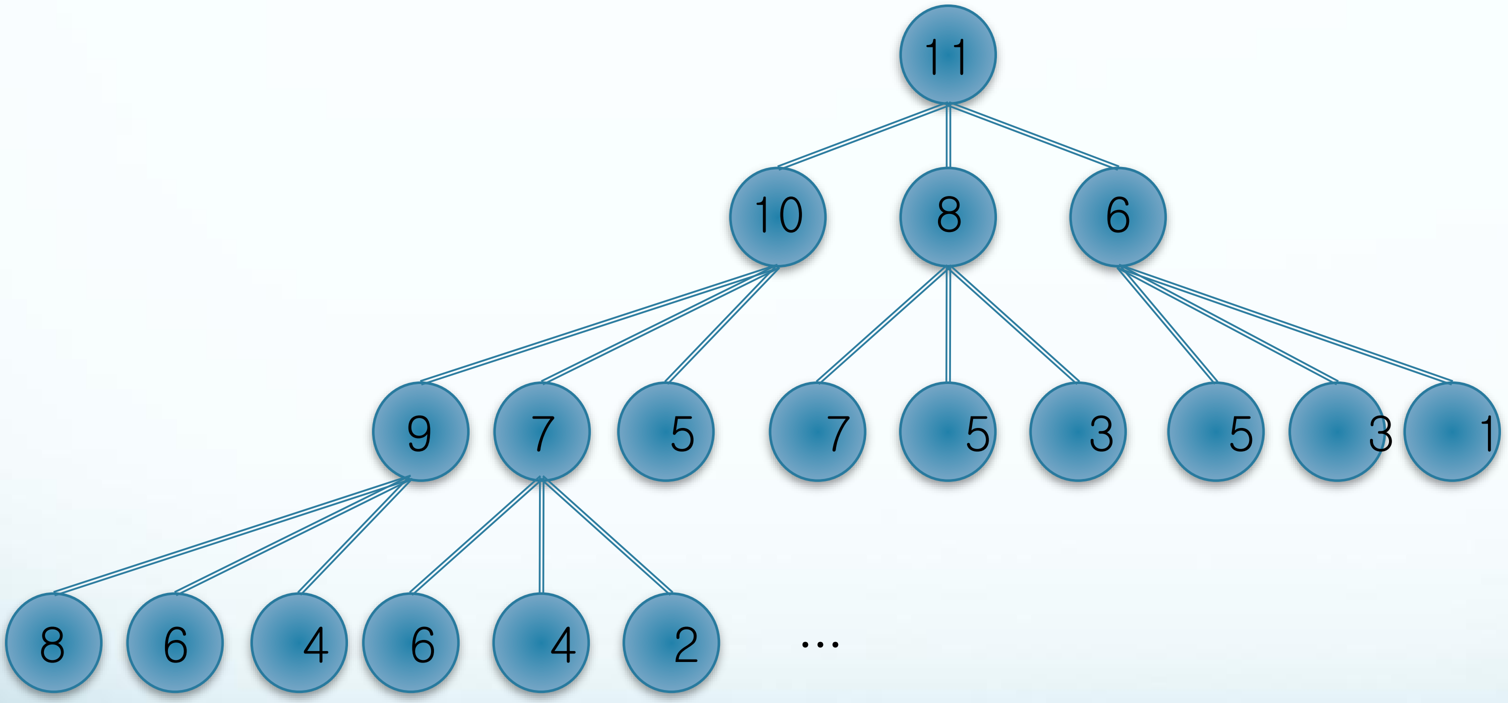
min

- Best for M−c1+ 1
- …
- Best for M−cd + 1

Recursive Algorithm



Recursive Algorithm



Dynamic Programming

- Reverse the order in which we solve the problem

1	2	3	4	5	6	7	8	9	10	11
1		1		1						

Dynamic Programming

- M=11, C={1,3,5}

1	2	3	4	5	6	7	8	9	10	11
1		1		1						

1	2	3	4	5	6	7	8	9	10	11
1	2	1		1						

1

1	2	3	4	5	6	7	8	9	10	11
1	2	1	2	1						

3 1

M=11, C={1,3,5}

1	2	3	4	5	6	7	8	9	10	11
1	2	1	2	1	2					

1	2	3	4	5	6	7	8	9	10	11
1	2	1	2	1	2	3				

1	2	3	4	5	6	7	8	9	10	11
1	2	1	2	1	2	3	2			

M=11, C={1,3,5}

1	2	3	4	5	6	7	8	9	10	11
1	2	1	2	1	2	3	2	3		

1	2	3	4	5	6	7	8	9	10	11
1	2	1	2	1	2	3	2	3	2	

1	2	3	4	5	6	7	8	9	10	11
1	2	1	2	1	2	3	2	3	2	3

Move Rocks problem

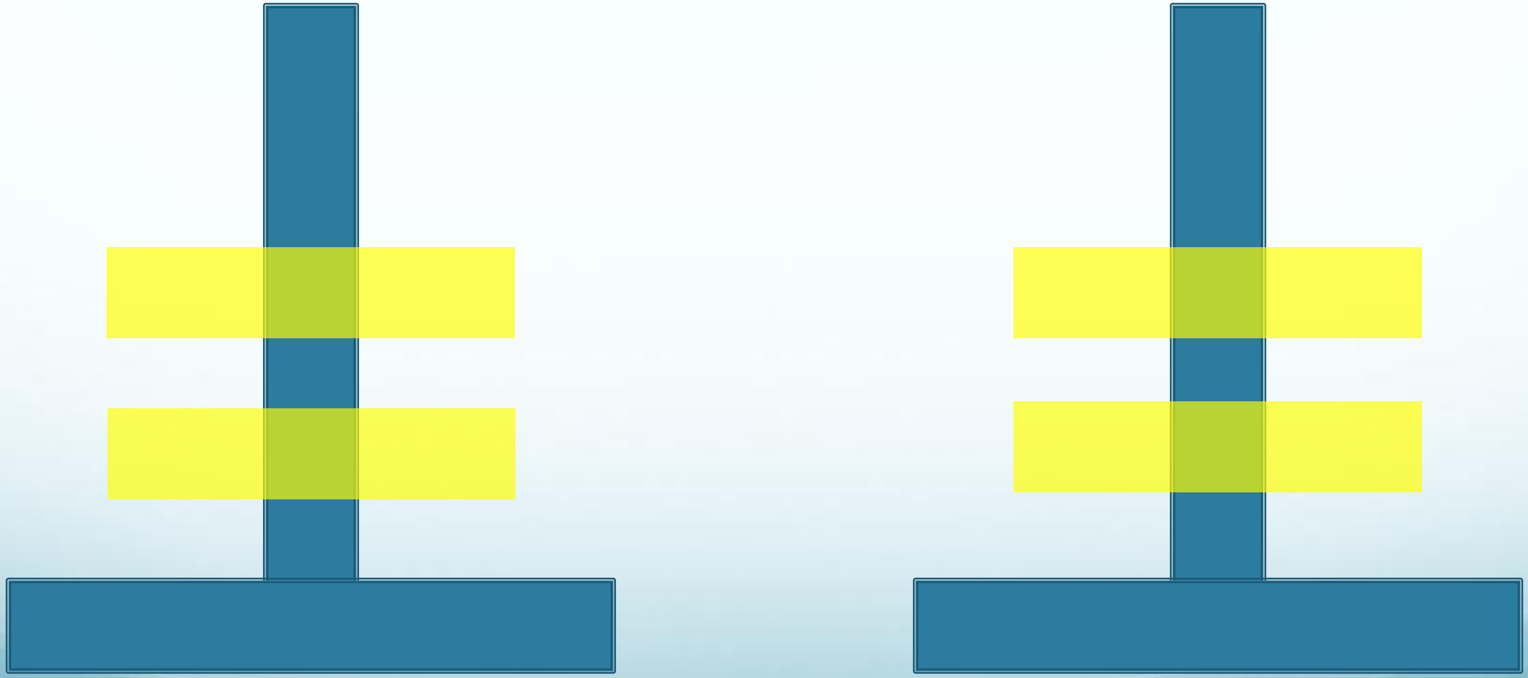
- Marry and Tomas is playing with rocks. There are two piles with m rocks and n rocks, respectively. At each turn, one can remove one rock from one of the piles or remove one rock from each of the piles. The rocks once removed from the pile cannot move back to the pile. The one who takes the last rock wins the game.

Move Rocks problem

- Marry and Tomas is playing with rocks. There are two piles with m rocks and n rocks, respectively. At each turn, one can remove one rock from one of the piles or remove one rock from each of the piles. The rocks once removed from the pile cannot move back to the pile. The one who takes the last rock wins the game.
- Who wins when n=m=2?
- Who wins when n=2 and m=4?
- Who wins when n=m=5?

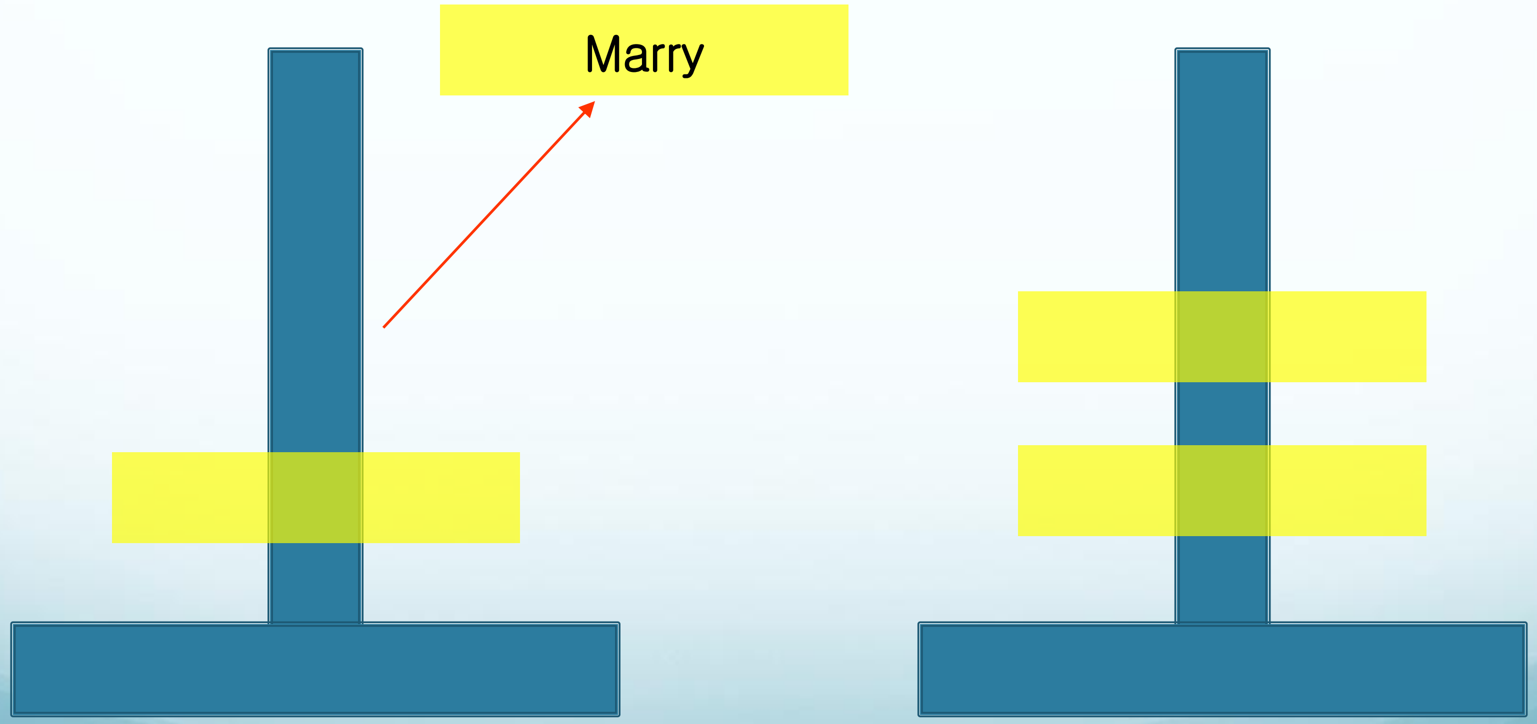
Move Rocks problem

- Who wins when n=m=2?

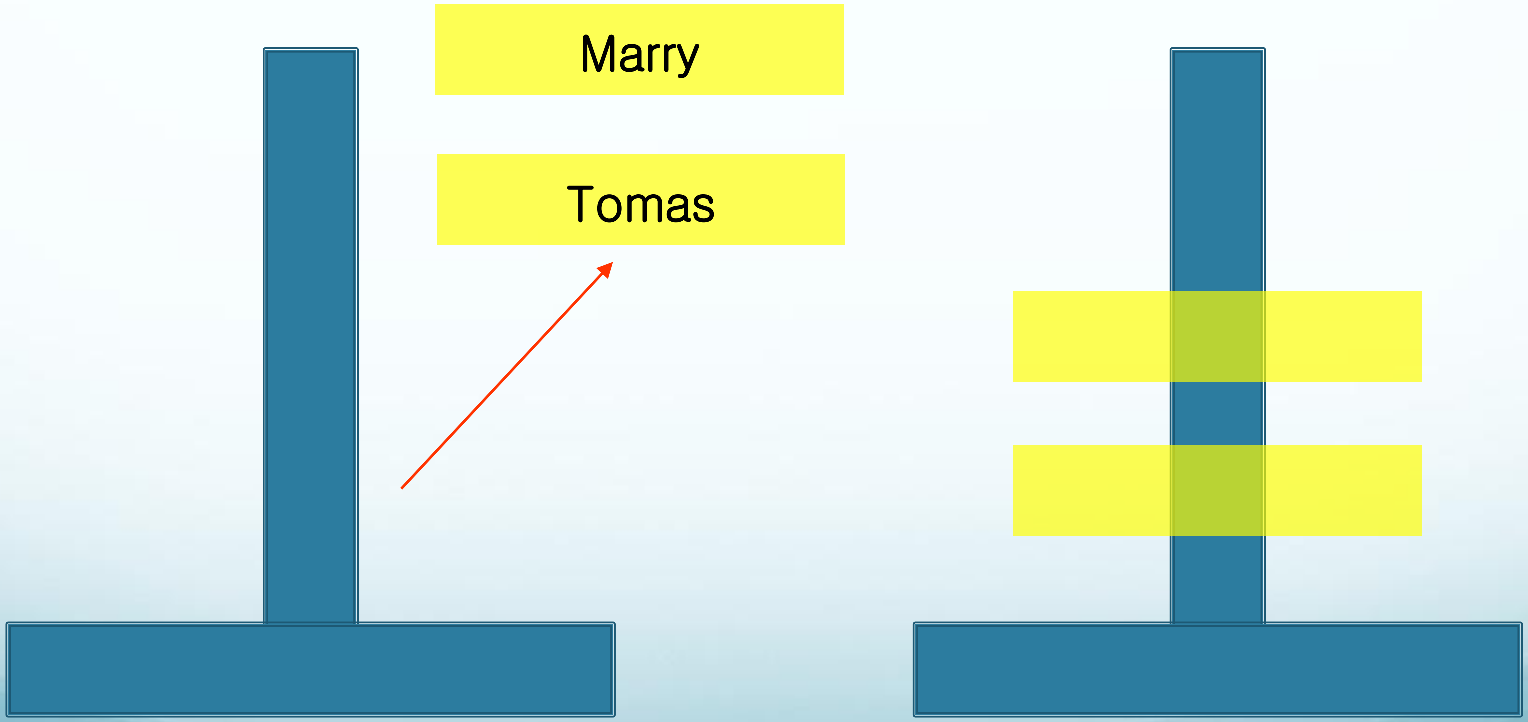


Move Rocks problem

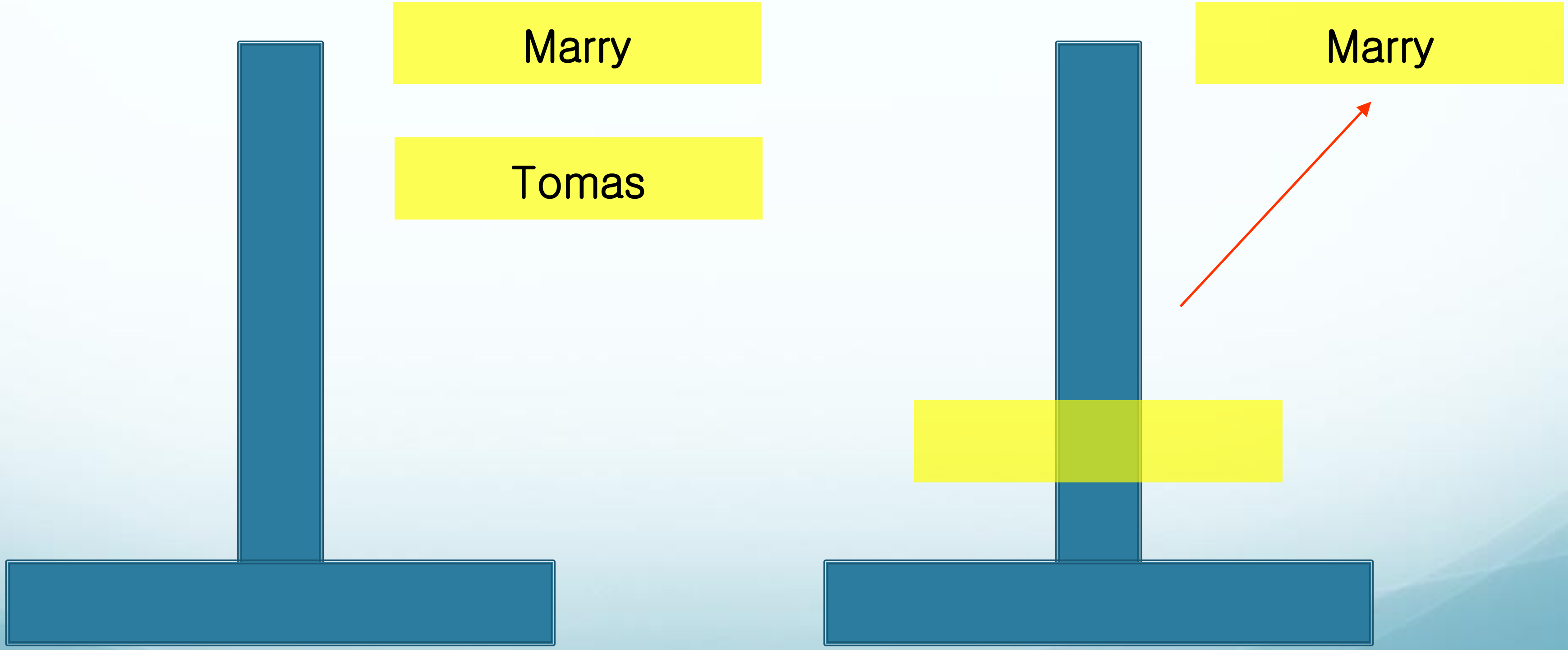
- Marry starts



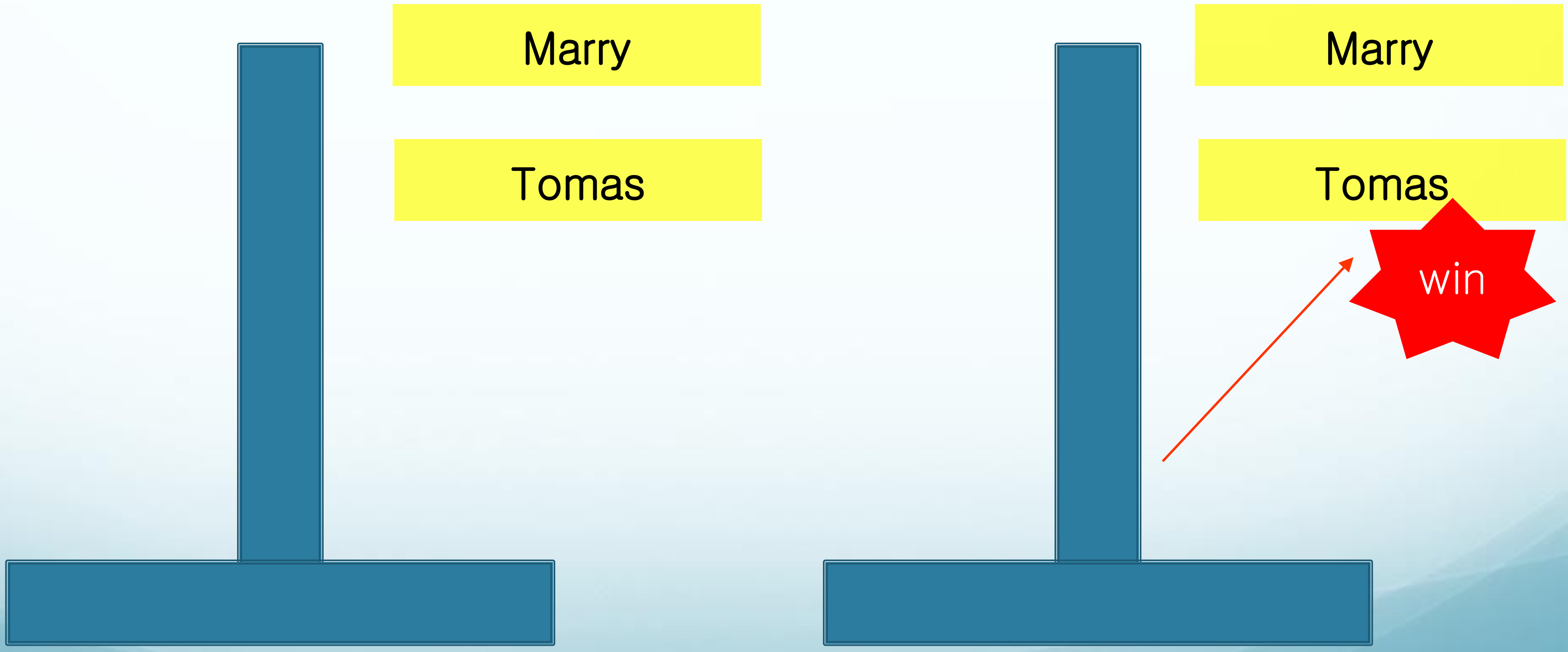
Move Rocks problem



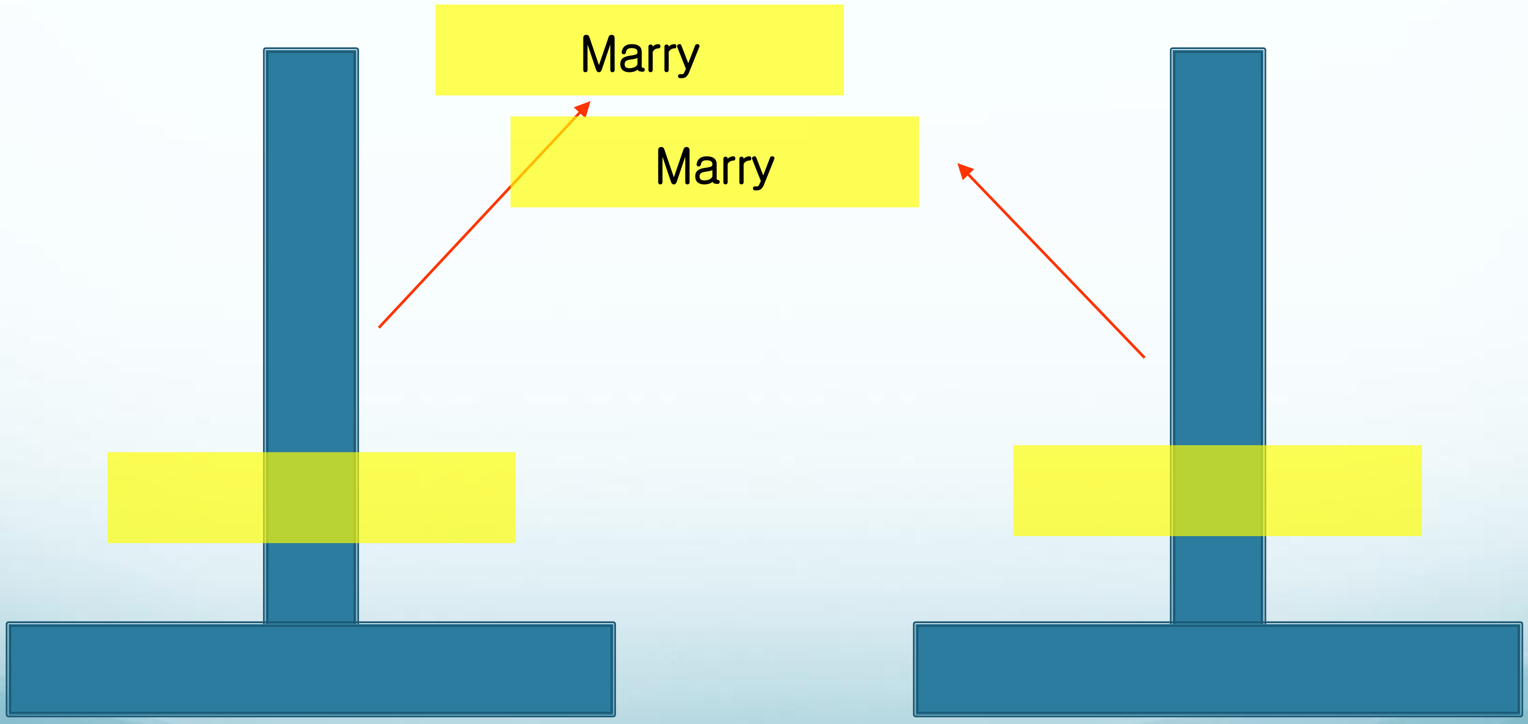
Move Rocks problem



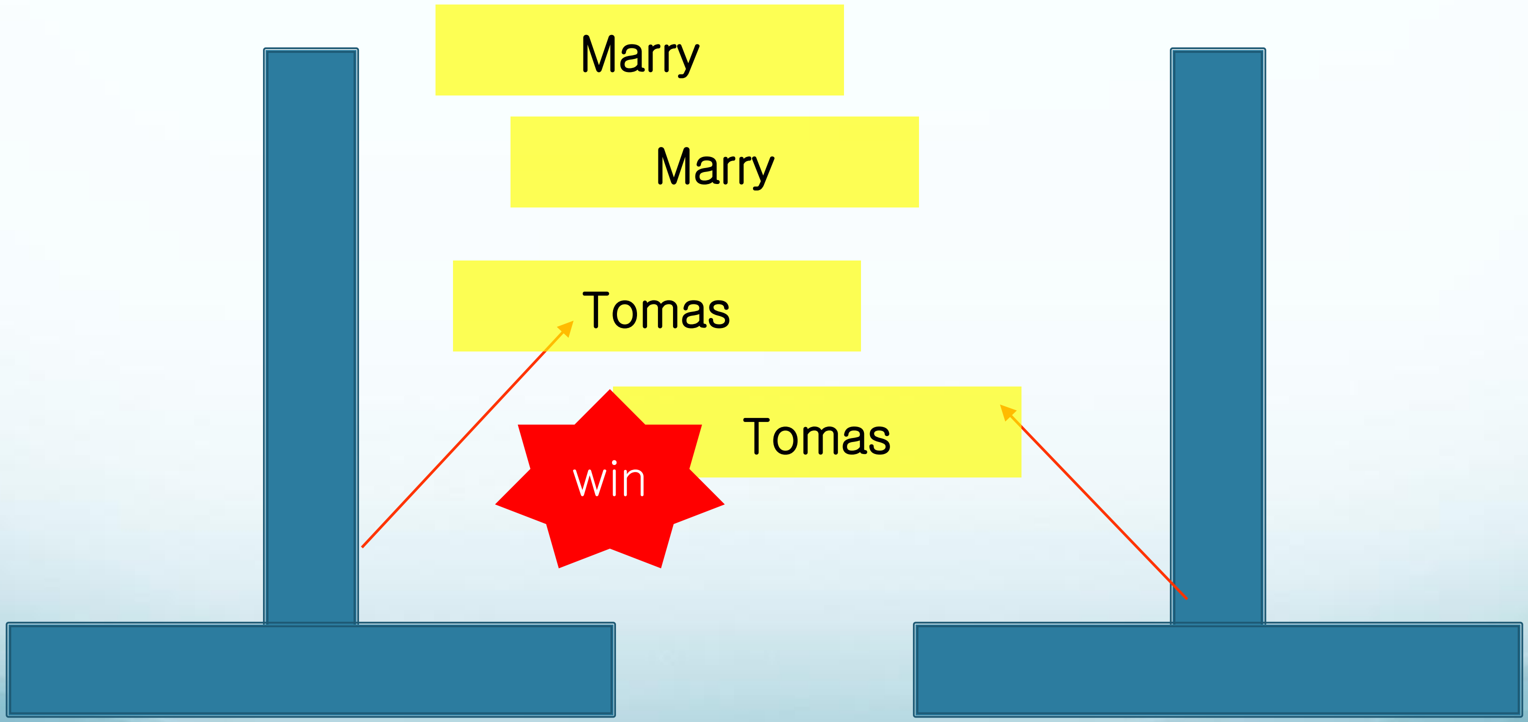
Move Rocks problem



Move Rocks problem



Move Rocks problem



Move Rocks problem

- Who wins when m=n=2
  - If the one starts wins, write W
  - If the one starts loses, write L

	0	1	2
0			
1			
2			

Move Rocks problem

- Who wins when m=n=2
  - If the one starts wins, write W
  - If the one starts loses, write L

	0	1	2
0		W	
1	W	W	
2			

There are only three ways to go

Move Rocks problem

- Who wins when m=n=2
  - If the one starts wins, write W
  - If the one starts loses, write L

	0	1	2
0		W	L
1	W	W	
2	L		

Move Rocks problem

- Who wins when m=n=2
  - If the one starts wins, write W
  - If the one starts loses, write L

	0	1	2
0		W	L
1	W	W	W
2	L	W	

Move Rocks problem

- Who wins when m=n=2
  - If the one starts wins, write W
  - If the one starts loses, write L

	0	1	2
0		W	L
1	W	W	W
2	L	W	L

Move Rocks problem

- Who wins when m=n=2
  - If the one starts wins, write W
  - If the one starts loses, write L

	0	1	2
0		W	L
1	W	W	W
2	L	W	L

Move Rocks problem

- Who wins when m=n=2
  - If the one starts wins, write W
  - If the one starts loses, write L

	0	1	2
0		W	L
1	W	W	W
2	L	W	L

Move Rocks problem

- Who wins when m=n=5
  - If the one starts wins, write W
  - If the one starts loses, write L

	0	1	2	3	4	5
0						
1						
2						
3						
4						
5						

Move Rocks problem

- Who wins when m=n=5
  - If the one starts wins, write W
  - If the one starts loses, write L

	0	1	2	3	4	5
0	L	W	L	W	L	W
1	W	W	W	W	W	W
2	L	W	L	W	L	W
3	W	W	W	W	W	W
4	L	W	L	W	L	W
5	W	W	W	W	W	W

Move Rocks problem

- Who wins when m=n=5
  - If the one starts wins, write W
  - If the one starts loses, write L

	0	1	2	3	4	5
0		W	L	W	L	W
1	W	W	W	W	W	W
2	L	W	L	W	L	W
3	W	W	W	W	W	W
4	L	W	L	W	L	W
5	W	W	W	W	W	W

Move Rocks problem

- Let's write a pseudo code for this algorithm

	0	1	2	3	4	5
0	L	W	L	W	L	W
1	W	W	W	W	W	W
2	L	W	L	W	L	W
3	W	W	W	W	W	W
4	L	W	L	W	L	W
5	W	W	W	W	W	W

Move Rocks problem

- Let's write a pseudo code for this algorithm

Rocks(5,5)

R	0	1	2	3	4	5
0	R(0,0)	R(0,1)	R(0,2)	R(0,3)	R(0,4)	R(0,5)
1	R(1,0)	R(1,1)	R(1,2)	R(1,3)	R(1,4)	R(1,5)
2	R(2,0)	R(2,1)	R(2,2)	R(2,3)	R(2,4)	R(2,5)
3	R(3,0)	R(3,1)	R(3,2)	R(3,3)	R(3,4)	R(3,5)
4	R(4,0)	R(4,1)	R(4,2)	R(4,3)	R(4,4)	R(4,5)
5	R(5,0)	R(5,1)	R(5,2)	R(5,3)	R(5,4)	R(5,5)

Move Rocks problem

- Let's write a pseudo code for this algorithm

Rocks(n,m)

?

Move Rocks problem

- Let's write a pseudo code for this algorithm

Rocks(n,m)

R	0	1	2	3	…	m
0	R(0,0)	R(0,1)	R(0,2)	R(0,3)	…	R(0,m)
1	R(1,0)	R(1,1)	R(1,2)	R(1,3)	…	R(1,m)
2	R(2,0)	R(2,1)	R(2,2)	R(2,3)	…	R(2,m)
3	R(3,0)	R(2,1)	R(3,2)	R(3,3)	…	R(3,m)
…	…	…	…	…	…	…
n	R(n,0)	R(n,1)	R(n,2)	R(n,3)	…	R(n,m)

Move Rocks problem

- Let's write a pseudo code for this algorithm

Rocks(n,m)

R	0	1	…	j	…	m
0	R(0,0)	…	…	…	…	R(0,m)
1	R(1,0)	…	…	…	…	…
…	…	…	?	?	…	…
i	…	…	?	R(i,j)	…	…
…	…	…	…	…	…	…
n	R(n,0)	…	…	…	…	R(n,m)

Move Rocks problem

- Let's write a pseudo code for this algorithm

Rocks(n,m)

R	0	1	…	j	…	m
0	R(0,0)	…	…	…	…	R(0,m)
1	R(1,0)	…	…	…	…	…
…	…	…	?	R(i-1,j)	…	…
i	…	…	?	R(i,j)	…	…
…	…	…	…	…	…	…
n	R(n,0)	…			…	R(n,m)

Move Rocks problem

- Let's write a pseudo code for this algorithm

Rocks(n,m)

R	0	1	…	j	…	m
0	R(0,0)	…	…	…	…	R(0,m)
1	R(1,0)	…	…	…	…	…
…	…	…	R(i−1, j−1)	R(i−1,j)	…	…
i	…	…	R(i,j−1)	R(i,j)	…	…
…	…	…	…	…	…	…
n	R(n,0)	…			…	R(n,m)

Move Rocks problem

- Let's write a pseudo code for this algorithm

```
Rocks(n,m)

        R(0,0) <- L        //initialize

    for i = 1 to n      //fill the first row
                if R(i-1,0) = W , R(i,0) <- L
                else              , R(i,0) <- W

            ?            //fill the first column
```

	0	1	2	3	4	5
0	L	W	L	W	L	W
1	W	W	W	W	W	W
2	L	W	L	W	L	W
3	W	W	W	W	W	W
4	L	W	L	W	L	W
5	W	W	W	W	W	W

Move Rocks problem

- Let's write a pseudo code for this algorithm

```
Rocks(n,m)

        R(0,0) <- L        //initilaize

    for i = 1 to n //fill the first row
                if R(i-1,0) = W , R(i,0) <- L
                else              , R(i,0) <- W

    for j =1 to m //fill the first column
        if R(0,j-1) = W , R(0,j) <- L
        else              , R(0,j) <- W

    ?           //fill the rest
```

	0	1	2	3	4	5
0	L	W	L	W	L	W
1	W	W	W	W	W	W
2	L	W	L	W	L	W
3	W	W	W	W	W	W
4	L	W	L	W	L	W
5	W	W	W	W	W	W

Move Rocks problem

- Let's write a pseudo code for this algorithm

```
Rocks(n,m)

        R(0,0) <- L

    for i = 1 to n
                if R(i-1,0) = W , R(i,0) <- L
                else             , R(i,0) <- W

    for j =1 to m
        if R(0,j-1) = W , R(0,j) <- L
        else             , R(0,j) <- W

    for i = 1 to n
        for j <- 1 to m
            ?
```

	0	1	2	3	4	5
0	L	W	L	W	L	W
1	W	W	W	W	W	W
2	L	W	L	W	L	W
3	W	W	W	W	W	W
4	L	W	L	W	L	W
5	W	W	W	W	W	W

Move Rocks problem

- Let's write a pseudo code for this algorithm

```
Rocks(n,m)

        R(0,0) <- L

    for i = 1 to n
                    if R(i-1,0) = W , R(i,0) <- L
                    else            , R(i,0) <- W

    for j =1 to m
        if R(0,j-1) = W , R(0,j) <- L
        else            , R(0,j) <- W

    for i = 1 to n
        for j = 1 to m
            if R(i-1,j-1) = W and R(I,j-1)=W and R(i-1,j)=W
                R(i,j) <-L
            else
                R(I,j) <-W
```

	0	1	2	3	4	5
0	L	W	L	W	L	W
1	W	W	W	W	W	W
2	L	W	L	W	L	W
3	W	W	W	W	W	W
4	L	W	L	W	L	W
5	W	W	W	W	W	W

Move Rocks problem

- Write this in a simpler way(cannot be generalized)

	0	1	2	3	4	5
0	L	W	L	W	L	W
1	W	W	W	W	W	W
2	L	W	L	W	L	W
3	W	W	W	W	W	W
4	L	W	L	W	L	W
5	W	W	W	W	W	W

```
fastRocks(n,m)
    if n and m are even,      return L
    else                      return W
```