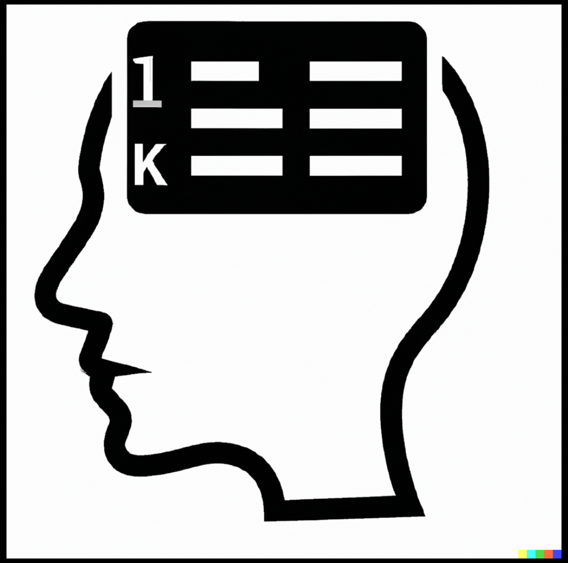
# Dynamic Programming




Tiziana Ligorio
Hunter College of The City University of New York

# Today's Plan

Recap

Motivation

Dynamic Programming

# Recap

Basic Recursion

- May have one or more recursive calls
- May reduce the problem by different amounts
- Base case implicit or explicit
- May or may not return a value

Recursive Backtracking

- Recursive call within loop (try all options at this state)
- Recursive call returns bool to determine whether current choice lead to solution

# Motivation

Recursion

Elegant / Intuitive solution

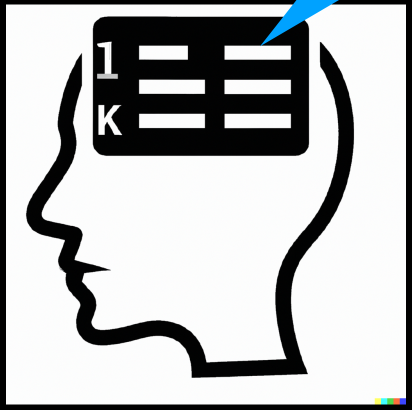
Overhead: repeated work - recursive calls on same subproblems

Solution: Dynamic Programming

Store solution to subproblems

Look-up instead of re-computing

Essentially, STORE what you compute for quick RETRIEVAL

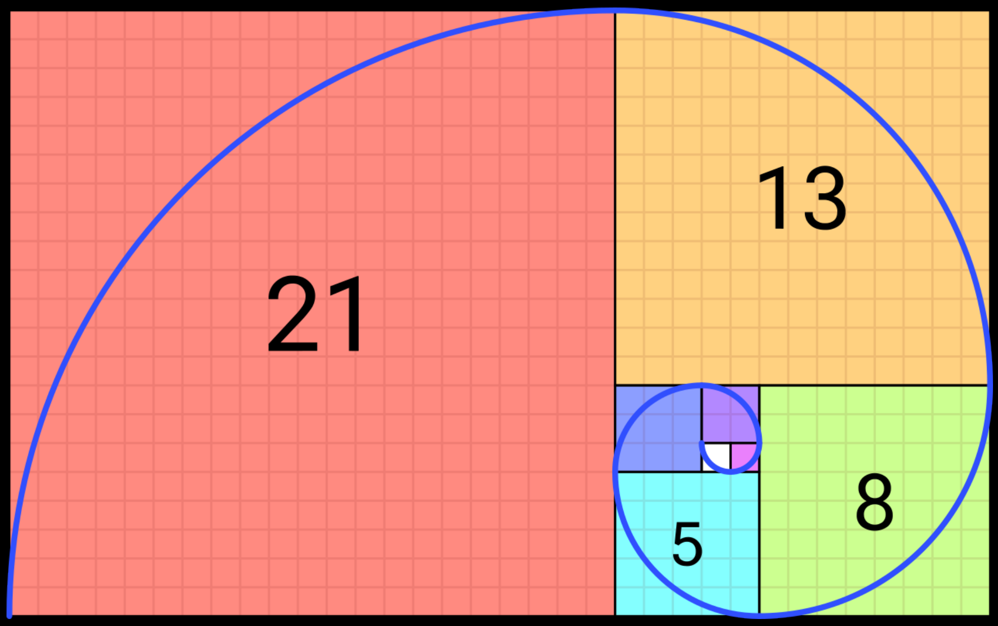


# Fibonacci

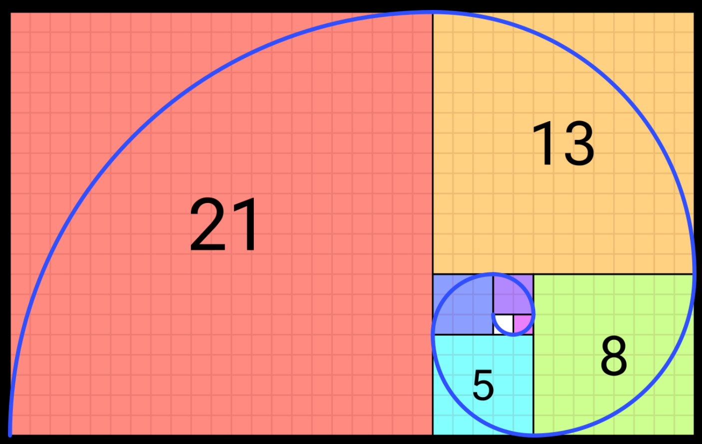
1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144

$F_n = F_{n-1} + F_{n-2}$

With $F_0 = 1$ and $F_1 = 1$



# Fibonacci - Recursion



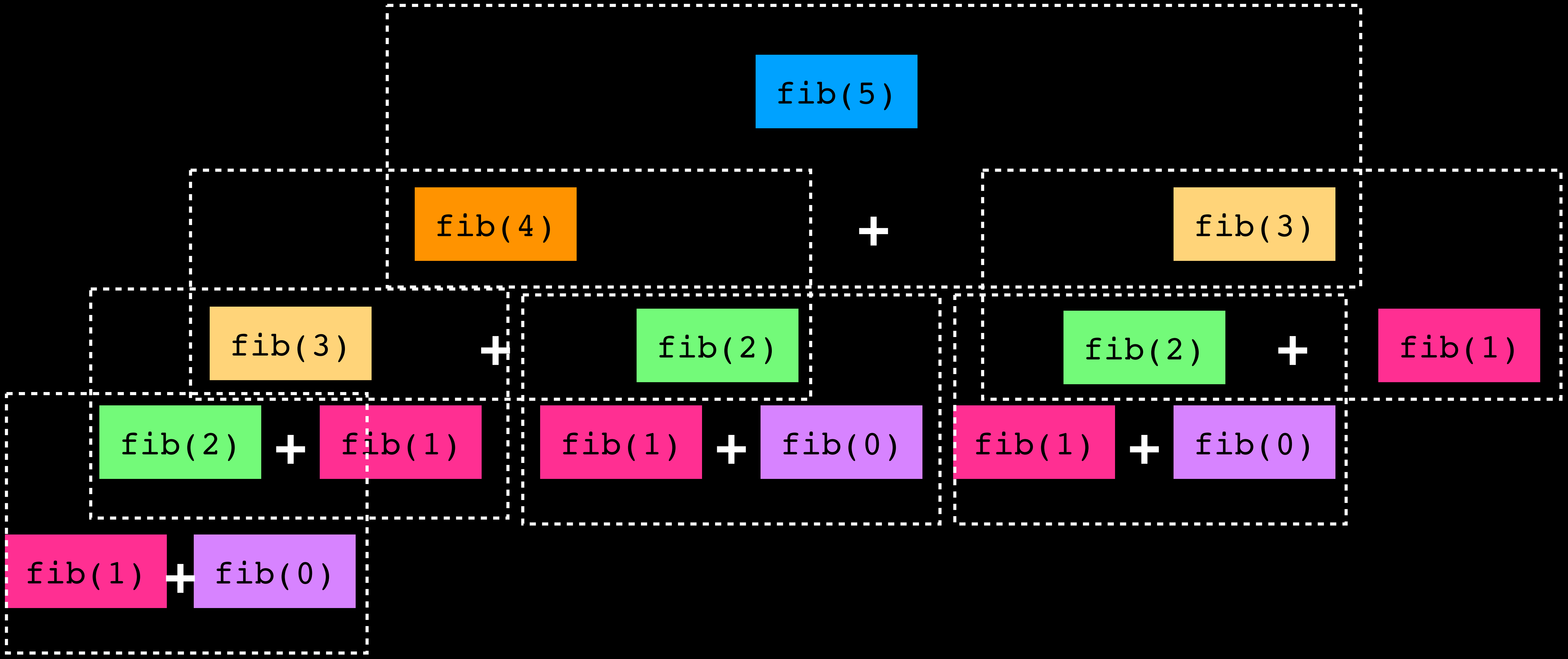

```
int fib(int n)
{
    // Base case
    if (n <= 1)
        return 1;

    // recursive calls
    return fib(n - 1) + fib(n - 2);
}
```

**Each `fib()` calls `fib()` twice**
**$T(n) = T(n-1) + T(n-2) + c$**
**$T(n) = 2 * 2 * 2 * \ldots * 2 = 2^n$**

**$O(2^n)$**

# Fibonacci - Recursion



# Dynamic Programming

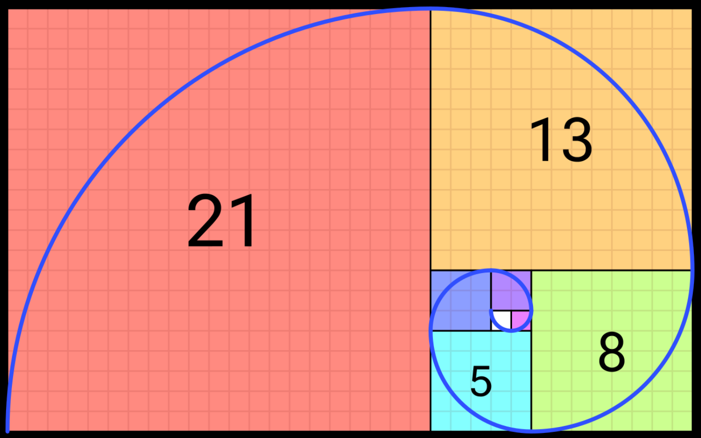
Remember instead of Recomputing

Two approaches:

Memoization: recursive approach but compute subproblem only once and store result in table.

Tabulation: unravel problem and compute all sub-problems bottom-up, iteratively, and store results in table

# Fibonacci - Memoization




```
std::vector<int> res(n+1, 0);
//assume table accessible across calls
int fib(int n)
{
    // base case
    if (n <= 1)
        return 1;

    if (res[n] != 0) //look-up
        return res[n];
    else { //compute and store
        res[n] = fib(n - 1) + fib(n - 2);

        return res[n];
    }
}
```

**Each `fib()` computed only ONCE**
**n `fib()` computations in total**
**Table lookup is constant**

**O(n)**

# Fibonacci - Memoization

Same logic as recursion

Compute subproblem when encountered

Computes only necessary subproblems

Not all entry in lookup table may be filled in

Still, generally less efficient than iterative solution due to function-call overhead

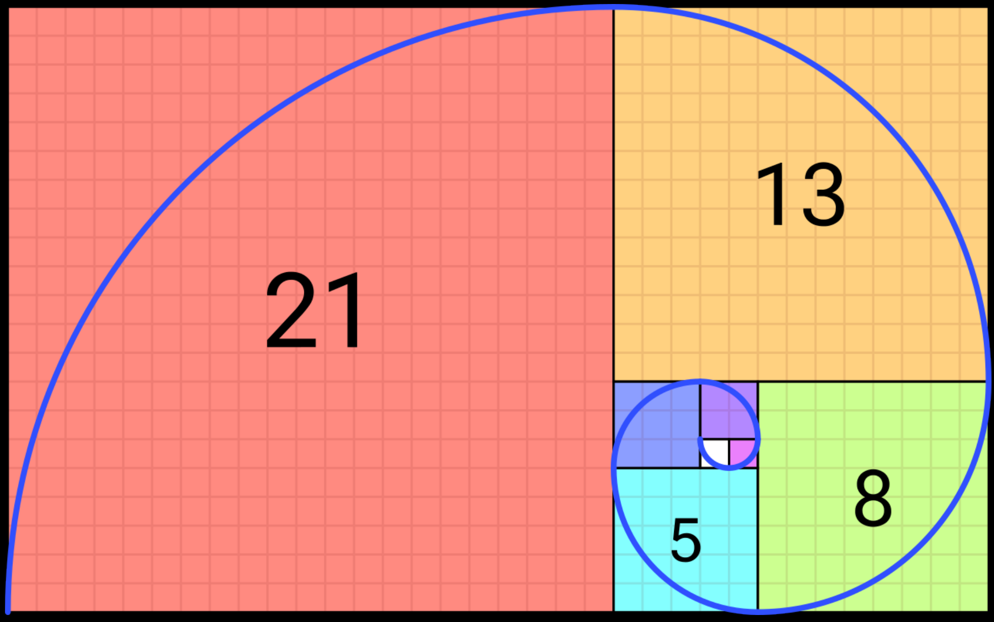
# Fibonacci

1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144

**Memoization**

$F_n = F_{n-1} + F_{n-2}$

With $F_0 = 1$ and $F_1 = 1$



# Fibonacci

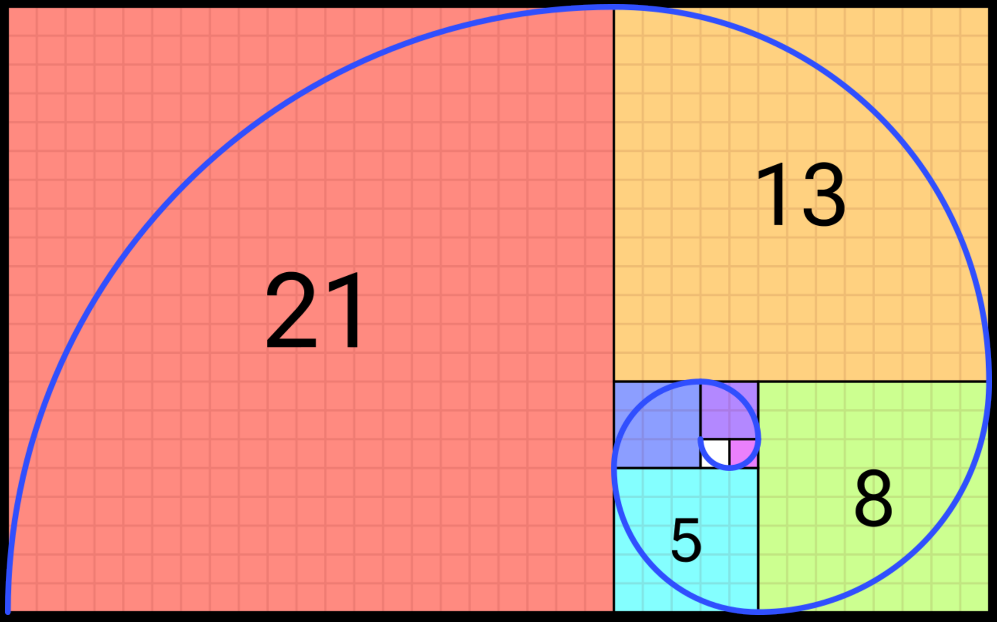
Tabulation

1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144

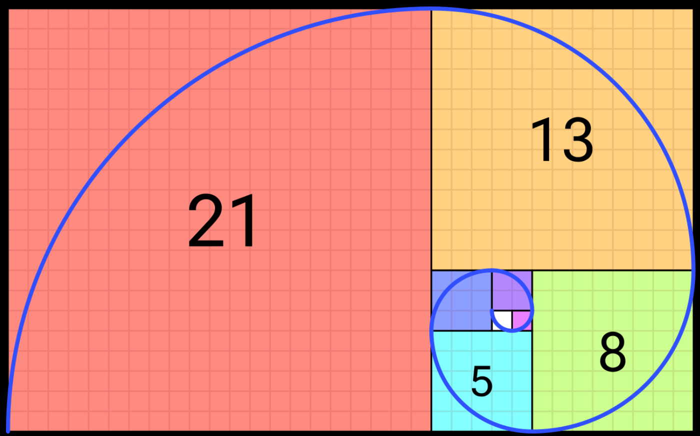
Memoization

$F_n = F_{n-1} + F_{n-2}$

With $F_0 = 1$ and $F_1 = 1$



# Fibonacci - Tabulation




```
int fib(int n)
{
    std::vector<int> res = {};
    res.push_back(1); //fib(0)
    res.push_back(1); //fib(1)
    for(int i = 2; i <= n; i++)
        res.push_back(res[i-1] + res[i-2]);

    return res[n];
}
```

**Bottom-up single iteration**

**O(n)**

# Fibonacci - Tabulation

Iterative (not recursive)

Start from smallest problem (base case) and work your way up

Computes ALL subproblems

Look-up table full from smallest (0 or 1) up to n, even if some subproblems are not needed

# Dynamic Programming Requirements

When can we apply dynamic programming solutions?

Original problem can be broken down into subproblems

Optimal substructure: optimal solution to subproblems contributes to optimal solution of original problem

When multiple solutions are possible, may not be possible to compute optimal subproblems backwards (difficult to memoize)

Sometimes it may be hard to unravel the problem into a bottom-up iteration (difficult to tabulate)

# Shortest Path Problem

Imagine traveling from S to E, but no direct path, must make stops in between

Cost or distance of each direct connection can be calculated

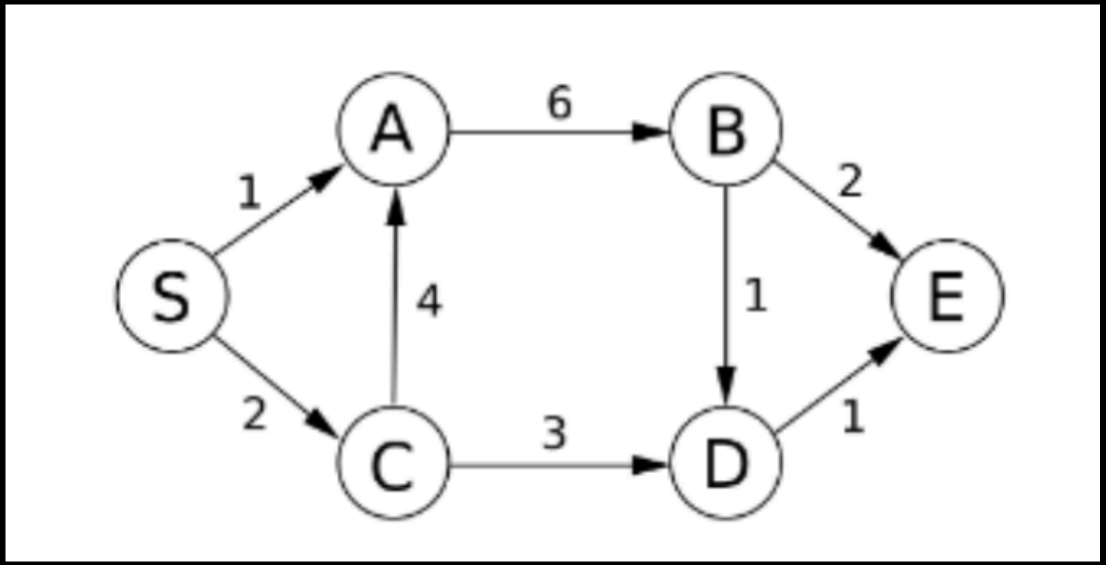
Which sequence (path) is cheapest/shortest?

Problem can be represented as a Directed Acyclic Graph (DAG)

# Shortest Path Problem

Approaches

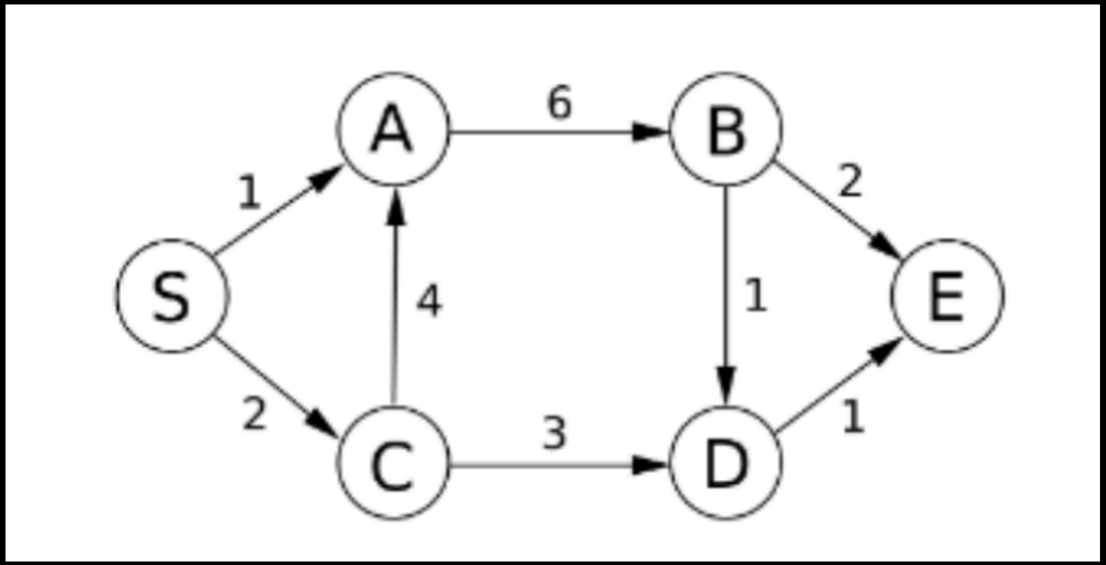
- Exhaustive search: try all paths and choose the best (shortest/cheapest) Exponential
- Greedy Search: start with cheapest route and make cheapest choice at each state. Not guaranteed to find optimal solution
- Can we break it down into subproblems?



# Shortest Path Problem

- Dynamic Programming:
  Let `d(X,Y)` be the distance/cost between nodes X and Y
  Subproblems: `sol(S) = min{sol(A) + d(S,A), sol(C) + d(S,C)}`

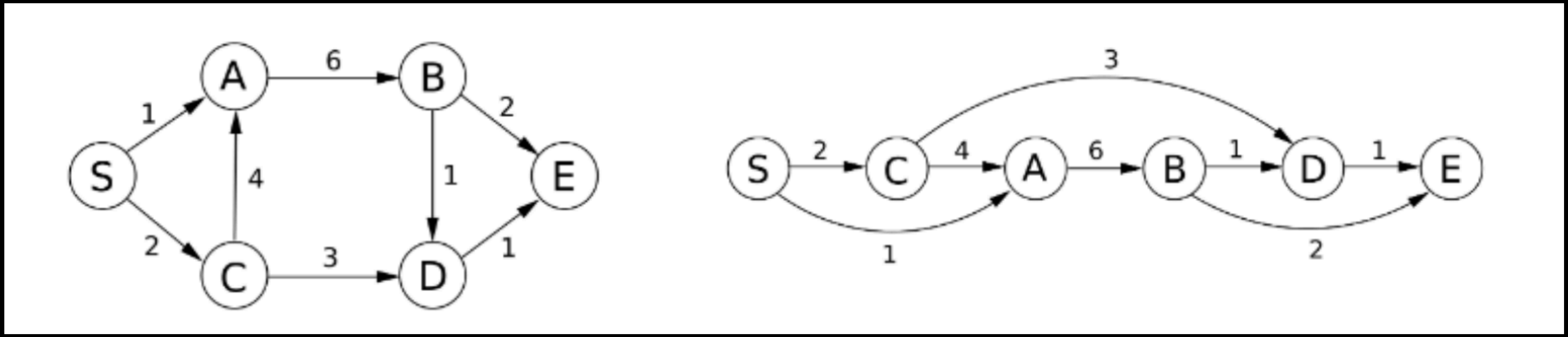
- Recursively this too is exponential, not guaranteed to have many repeated subproblems to exploit



# Shortest Path Problem

- Dynamic Programming:
  Let `d(X,Y)` be the distance/cost between nodes X and Y
  Subproblems: `sol(S) = min{sol(A) + d(S,A), sol(C) + d(S,C)}`

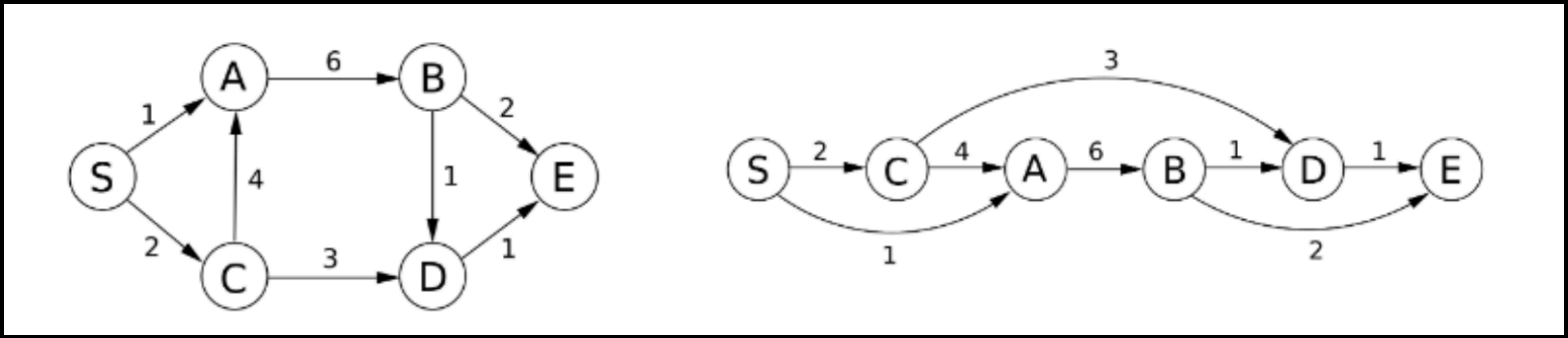
- Recursively this too is exponential, Memoization not guaranteed to have many repeated subproblems to exploit



# Shortest Path Problem

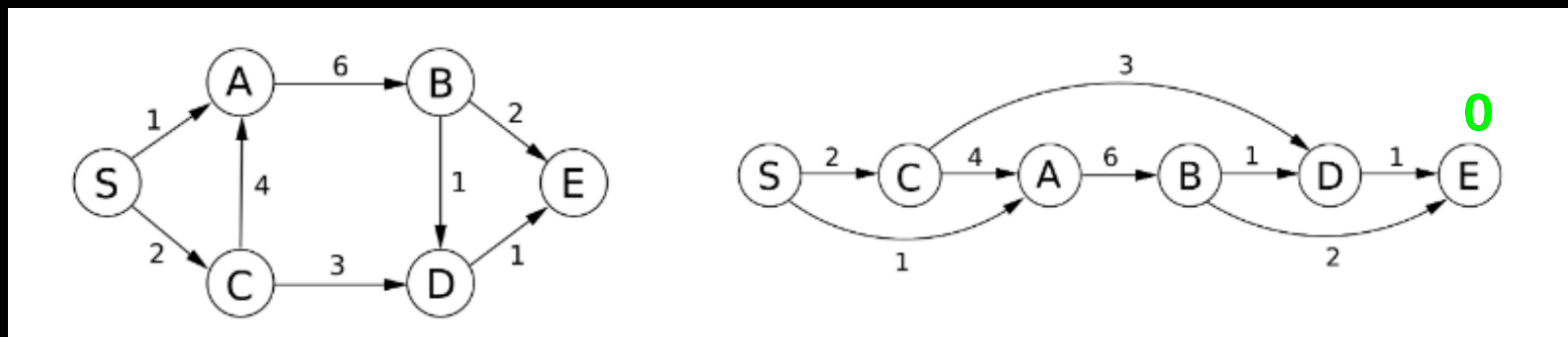
- Dynamic Programming:
  Let `d(X,Y)` be the distance/cost between nodes X and Y
  Subproblems: `sol(S) = min{sol(A) + d(S,A), sol(C) + d(S,C)}`

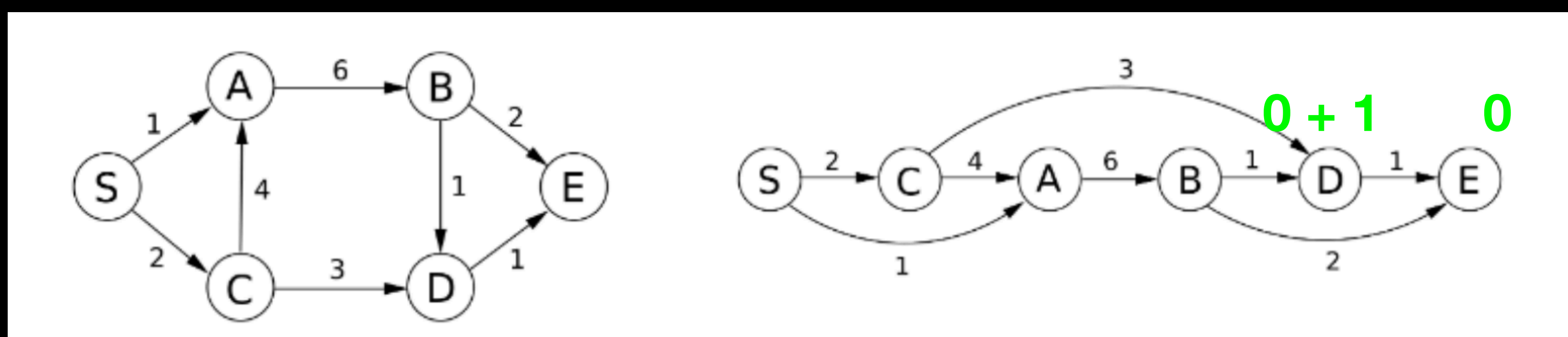
- Tabulation: fill out table starting from smaller subproblem

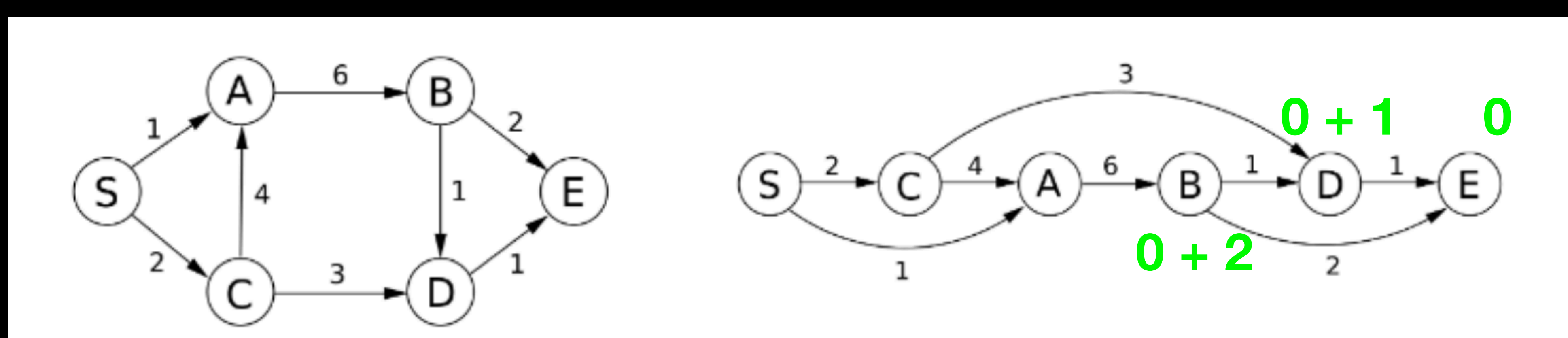


# Shortest Path Problem

`sol(S) = min{sol(A) + d(S,A), sol(C) + d(S,C)}`

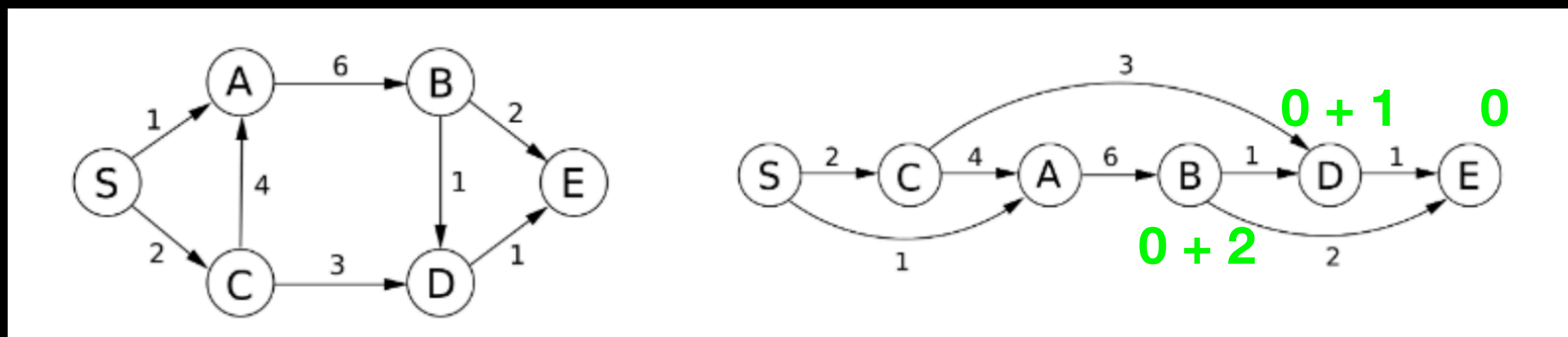


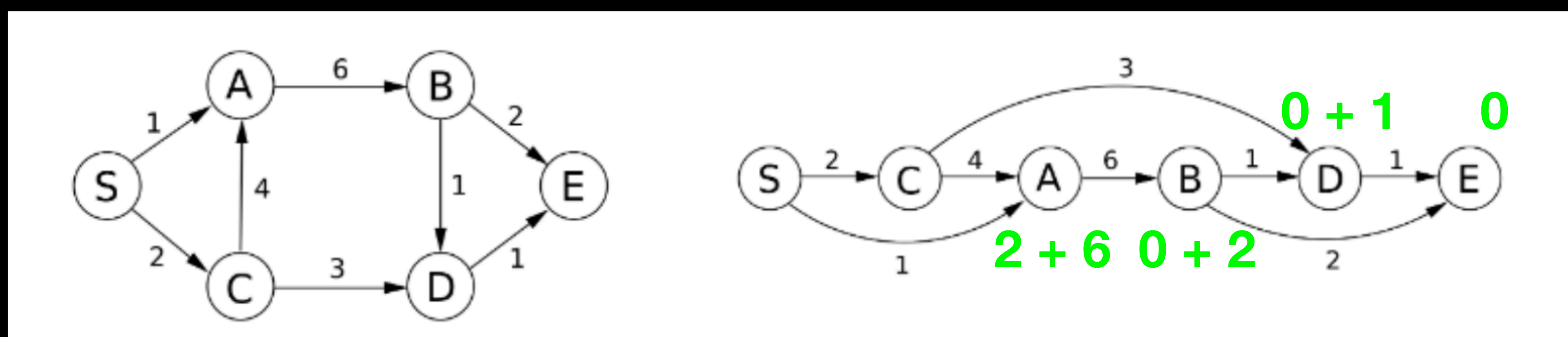



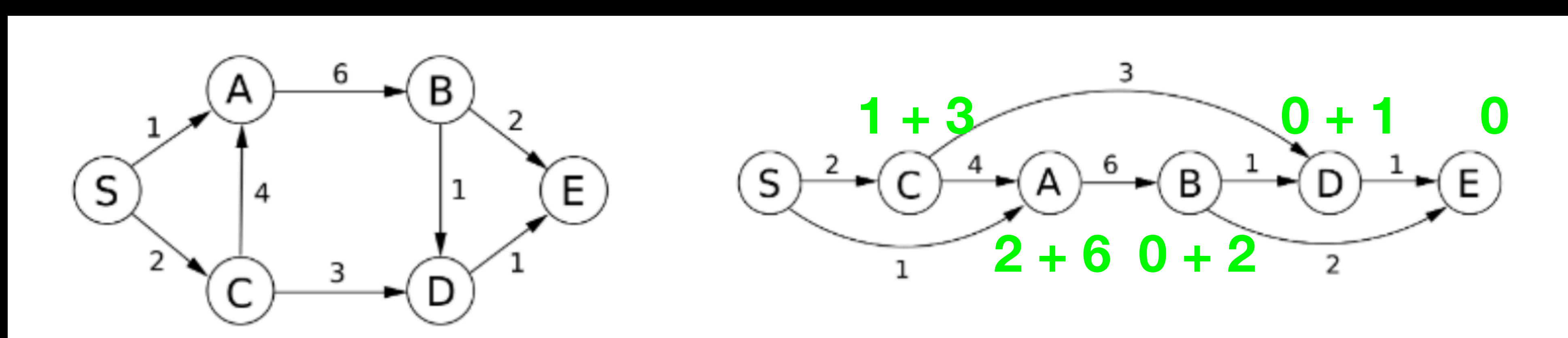



# Shortest Path Problem

```
sol(S) = min{sol(A) + d(S,A), sol(C) + d(S,C)}
```

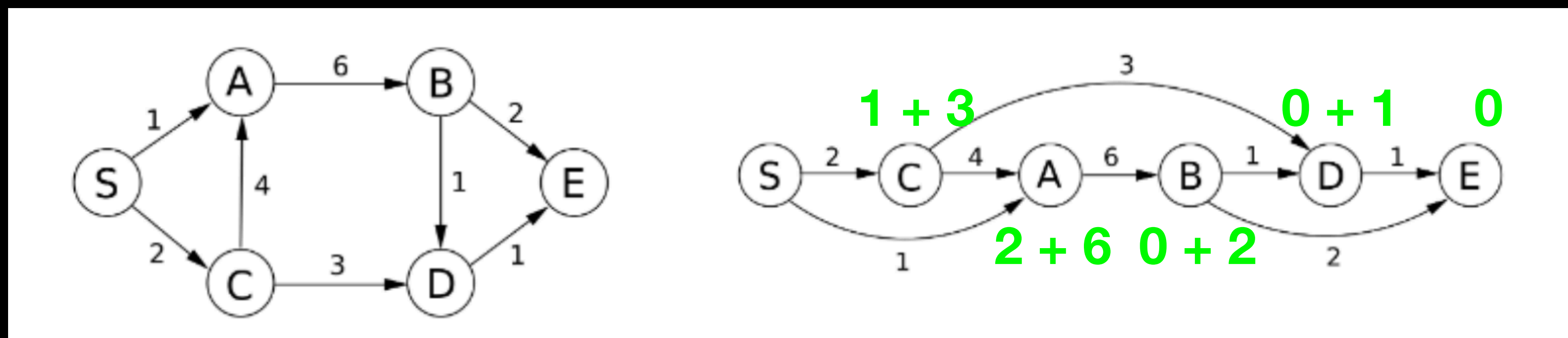


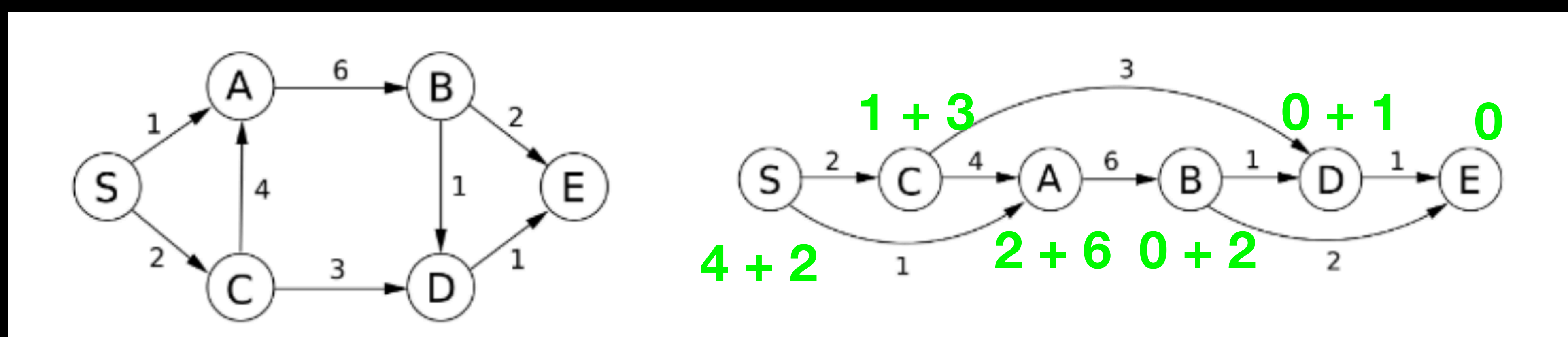



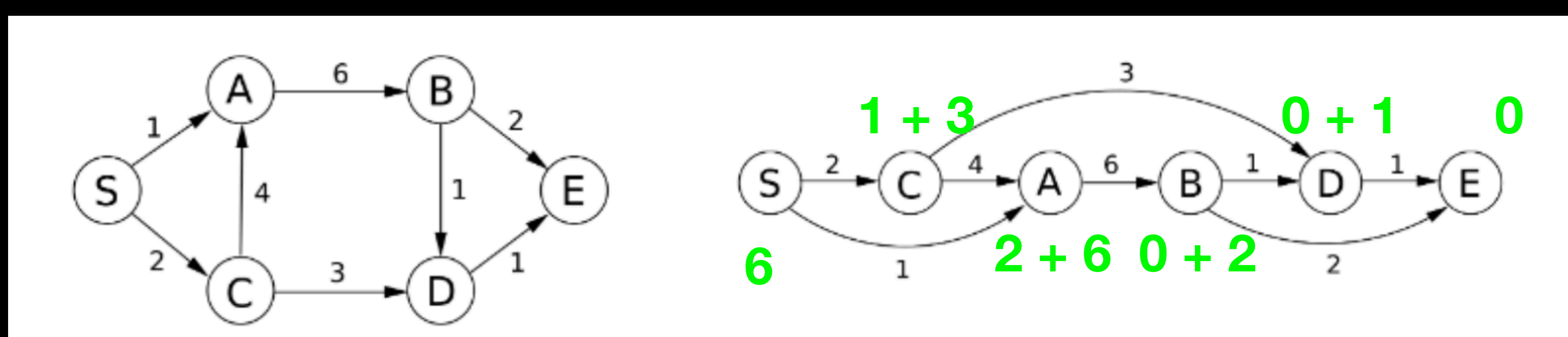



# Shortest Path Problem

```
sol(S) = min{sol(A) + d(S,A), sol(C) + d(S,C)}
```

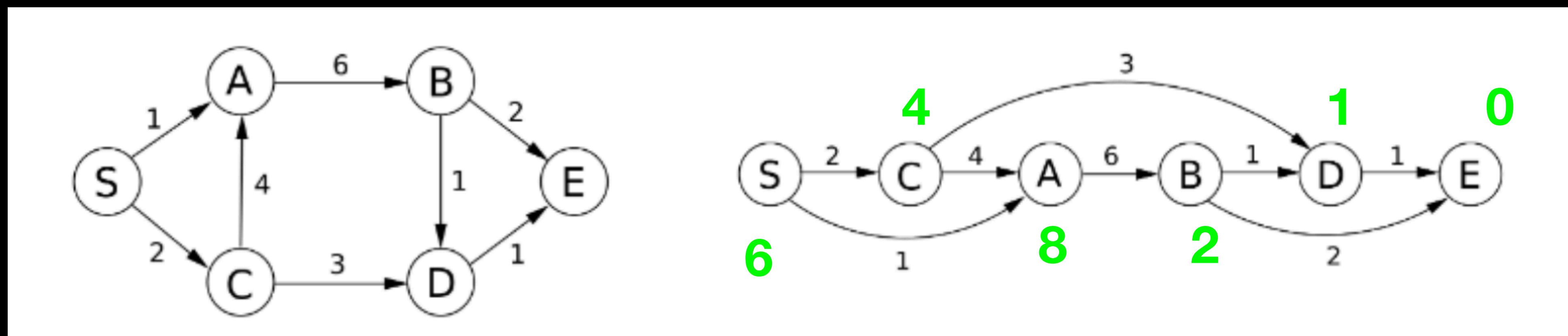


# Shortest Path Problem

`sol(S) = min{sol(A) + d(S,A), sol(C) + d(S,C)}`



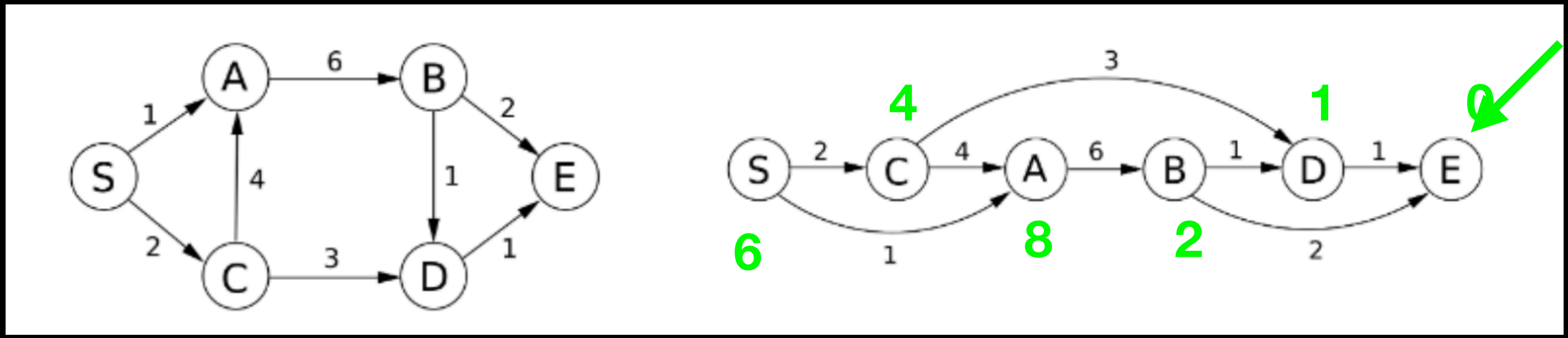

**Adjacency Matrix**

| | S | C | A | B | D | E |
|---|---|---|---|---|---|---|
| **S** | 0 | 2 | 1 | -1 | -1 | -1 |
| **C** | -1 | 0 | 4 | -1 | 3 | -1 |
| **A** | -1 | -1 | 0 | 6 | -1 | -1 |
| **B** | -1 | -1 | -1 | 0 | 1 | 2 |
| **D** | -1 | -1 | -1 | -1 | 0 | 1 |
| **E** | -1 | -1 | -1 | -1 | -1 | 0 |

No negative edges.
-1 means no edge.

# Shortest Path Problem

```
sol(S) = min{sol(A) + d(S,A), sol(C) + d(S,C)}
```




**Adjacency Matrix**

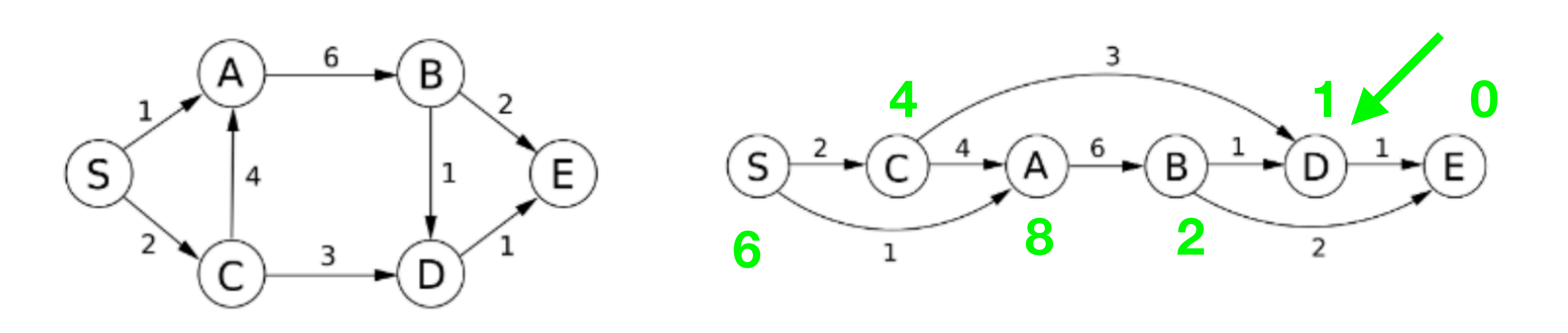
| | S | C | A | B | D | E |
|---|---|---|---|---|---|---|
| **S** | 0 | 2 | 1 | -1 | -1 | -1 |
| **C** | -1 | 0 | 4 | -1 | 3 | -1 |
| **A** | -1 | -1 | 0 | 6 | -1 | -1 |
| **B** | -1 | -1 | -1 | 0 | 1 | 2 |
| **D** | -1 | -1 | -1 | -1 | 0 | 1 |
| **E** | -1 | -1 | -1 | -1 | -1 | 0 |

**Shortest Path**

| | S | C | A | B | D | E |
|---|---|---|---|---|---|---|
| **Sp** | | | | | | 0 |

# Shortest Path Problem

```
sol(S) = min{sol(A) + d(S,A), sol(C) + d(S,C)}
```




**Adjacency Matrix**

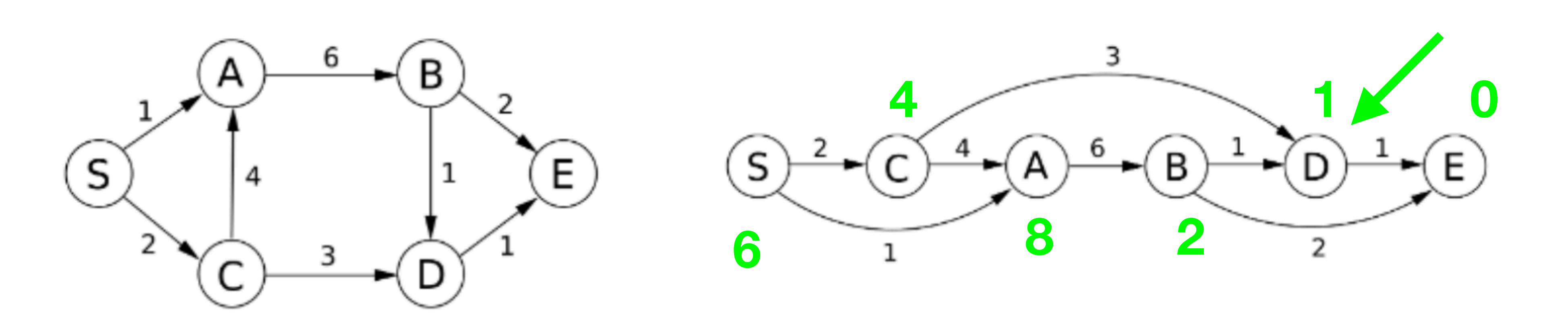
| | S | C | A | B | D | E |
|---|---|---|---|---|---|---|
| **S** | 0 | 2 | 1 | -1 | -1 | -1 |
| **C** | -1 | 0 | 4 | -1 | 3 | -1 |
| **A** | -1 | -1 | 0 | 6 | -1 | -1 |
| **B** | -1 | -1 | -1 | 0 | 1 | 2 |
| **D** | -1 | -1 | -1 | -1 | 0 | 1 |
| **E** | -1 | -1 | -1 | -1 | -1 | 0 |

**Shortest Path**

| | S | C | A | B | D | E |
|---|---|---|---|---|---|---|
| **Sp** | | | | | | 0 |

# Shortest Path Problem

```
sol(S) = min{sol(A) + d(S,A), sol(C) + d(S,C)}
```




**Adjacency Matrix**

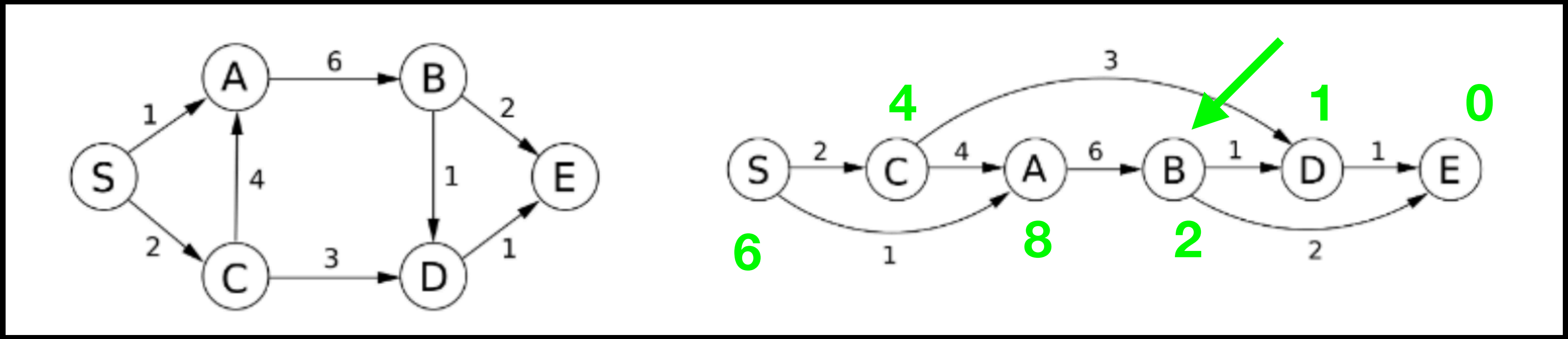
| | S | C | A | B | D | E |
|---|---|---|---|---|---|---|
| **S** | 0 | 2 | 1 | -1 | -1 | -1 |
| **C** | -1 | 0 | 4 | -1 | 3 | -1 |
| **A** | -1 | -1 | 0 | 6 | -1 | -1 |
| **B** | -1 | -1 | -1 | 0 | 1 | 2 |
| **D** | -1 | -1 | -1 | -1 | 0 | 1 |
| **E** | -1 | -1 | -1 | -1 | -1 | 0 |

**Shortest Path**

| | S | C | A | B | D | E |
|---|---|---|---|---|---|---|
| **Sp** | | | | | 1 | 0 |

# Shortest Path Problem

`sol(S) = min{sol(A) + d(S,A), sol(C) + d(S,C)}`




**Adjacency Matrix**

| | S | C | A | B | D | E |
|---|---|---|---|---|---|---|
| **S** | 0 | 2 | 1 | -1 | -1 | -1 |
| **C** | -1 | 0 | 4 | -1 | 3 | -1 |
| **A** | -1 | -1 | 0 | 6 | -1 | -1 |
| **B** | -1 | -1 | -1 | 0 | 1 | 2 |
| **D** | -1 | -1 | -1 | -1 | 0 | 1 |
| **E** | -1 | -1 | -1 | -1 | -1 | 0 |

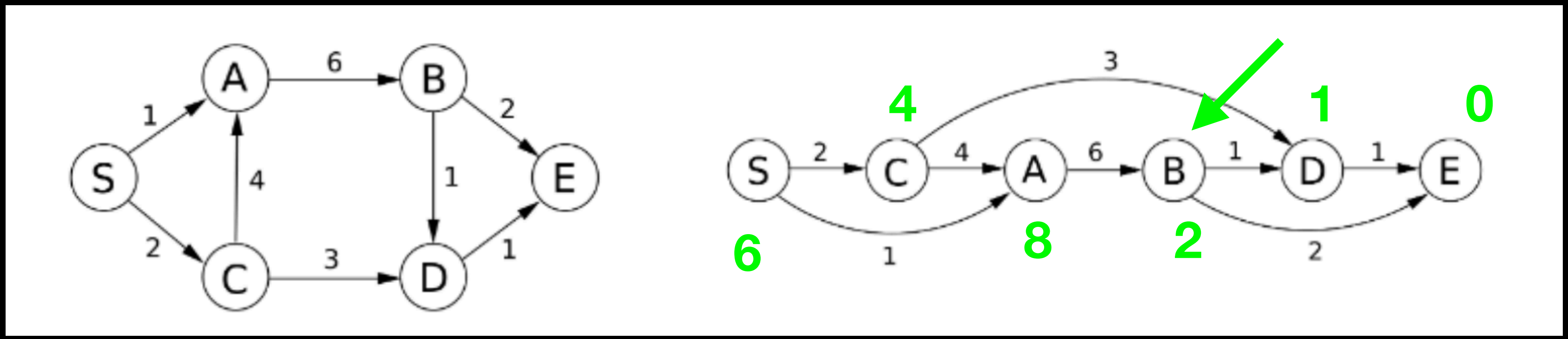
**Shortest Path**

| | S | C | A | B | D | E |
|---|---|---|---|---|---|---|
| **Sp** | | | | | 1 | 0 |

min(2+0,

# Shortest Path Problem

```
sol(S) = min{sol(A) + d(S,A), sol(C) + d(S,C)}
```




**Adjacency Matrix**

| | S | C | A | B | D | E |
|---|---|---|---|---|---|---|
| **S** | 0 | 2 | 1 | -1 | -1 | -1 |
| **C** | -1 | 0 | 4 | -1 | 3 | -1 |
| **A** | -1 | -1 | 0 | 6 | -1 | -1 |
| **B** | -1 | -1 | -1 | 0 | 1 | 2 |
| **D** | -1 | -1 | -1 | -1 | 0 | 1 |
| **E** | -1 | -1 | -1 | -1 | -1 | 0 |

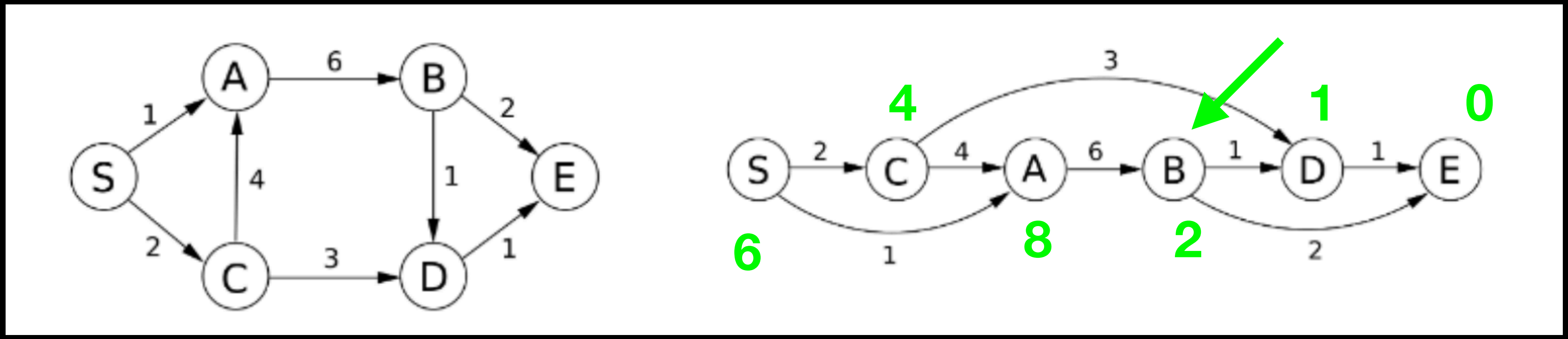
**Shortest Path**

| | S | C | A | B | D | E |
|---|---|---|---|---|---|---|
| **Sp** | | | | | 1 | 0 |

min(2+0, 1+1)

# Shortest Path Problem

```
sol(S) = min{sol(A) + d(S,A), sol(C) + d(S,C)}
```




**Adjacency Matrix**

| | S | C | A | B | D | E |
|---|---|---|---|---|---|---|
| **S** | 0 | 2 | 1 | -1 | -1 | -1 |
| **C** | -1 | 0 | 4 | -1 | 3 | -1 |
| **A** | -1 | -1 | 0 | 6 | -1 | -1 |
| **B** | -1 | -1 | -1 | 0 | 1 | 2 |
| **D** | -1 | -1 | -1 | -1 | 0 | 1 |
| **E** | -1 | -1 | -1 | -1 | -1 | 0 |

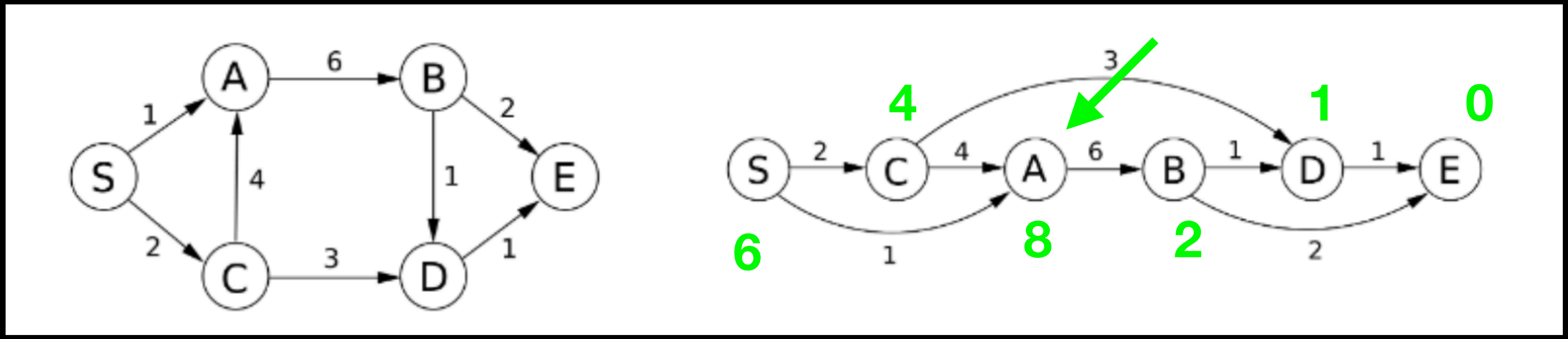
**Shortest Path**

| | S | C | A | B | D | E |
|---|---|---|---|---|---|---|
| **Sp** | | | | 2 | 1 | 0 |

**min(2+0, 1+1)**

# Shortest Path Problem

```
sol(S) = min{sol(A) + d(S,A), sol(C) + d(S,C)}
```




**Adjacency Matrix**

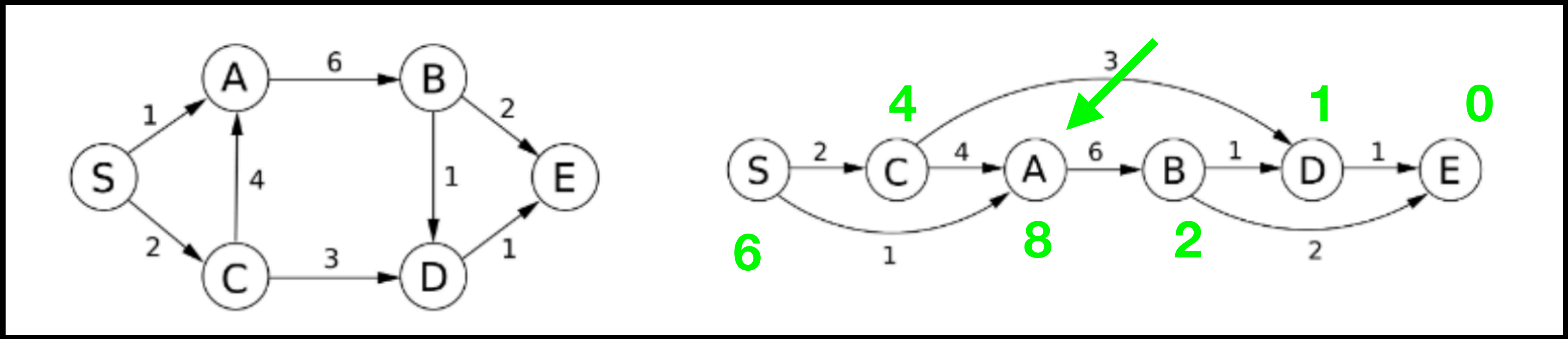
| | S | C | A | B | D | E |
|---|---|---|---|---|---|---|
| **S** | 0 | 2 | 1 | -1 | -1 | -1 |
| **C** | -1 | 0 | 4 | -1 | 3 | -1 |
| **A** | -1 | -1 | 0 | 6 | -1 | -1 |
| **B** | -1 | -1 | -1 | 0 | 1 | 2 |
| **D** | -1 | -1 | -1 | -1 | 0 | 1 |
| **E** | -1 | -1 | -1 | -1 | -1 | 0 |

**Shortest Path**

| | S | C | A | B | D | E |
|---|---|---|---|---|---|---|
| **Sp** | | | | 2 | 1 | 0 |

# Shortest Path Problem

```
sol(S) = min{sol(A) + d(S,A), sol(C) + d(S,C)}
```




**Adjacency Matrix**

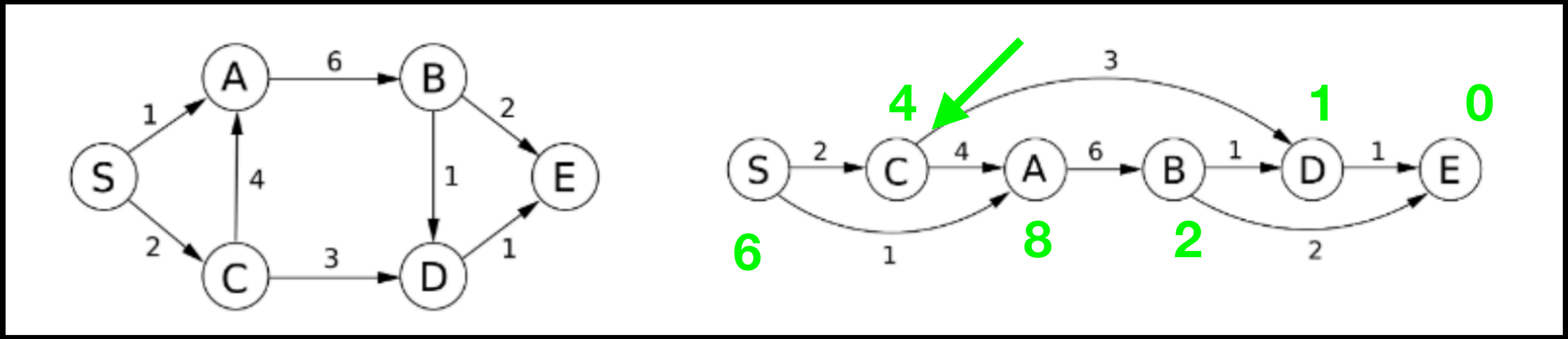
| | S | C | A | B | D | E |
|---|---|---|---|---|---|---|
| **S** | 0 | 2 | 1 | -1 | -1 | -1 |
| **C** | -1 | 0 | 4 | -1 | 3 | -1 |
| **A** | -1 | -1 | 0 | 6 | -1 | -1 |
| **B** | -1 | -1 | -1 | 0 | 1 | 2 |
| **D** | -1 | -1 | -1 | -1 | 0 | 1 |
| **E** | -1 | -1 | -1 | -1 | -1 | 0 |

**Shortest Path**

| | S | C | A | B | D | E |
|---|---|---|---|---|---|---|
| **Sp** | | | 8 | 2 | 1 | 0 |

# Shortest Path Problem

`sol(S) = min{sol(A) + d(S,A), sol(C) + d(S,C)}`




**Adjacency Matrix**

| | S | C | A | B | D | E |
|---|---|---|---|---|---|---|
| **S** | 0 | 2 | 1 | -1 | -1 | -1 |
| **C** | -1 | 0 | 4 | -1 | 3 | -1 |
| **A** | -1 | -1 | 0 | 6 | -1 | -1 |
| **B** | -1 | -1 | -1 | 0 | 1 | 2 |
| **D** | -1 | -1 | -1 | -1 | 0 | 1 |
| **E** | -1 | -1 | -1 | -1 | -1 | 0 |

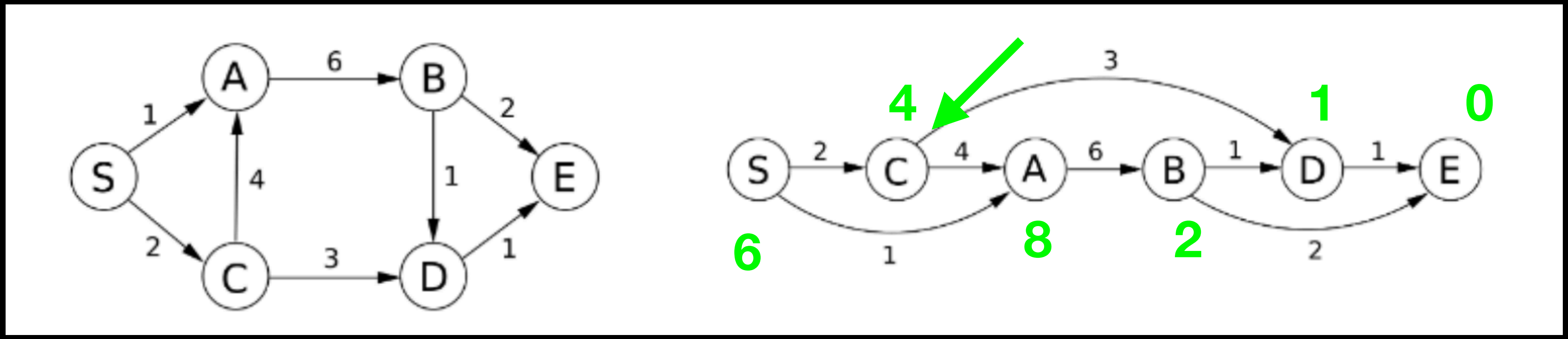
**Shortest Path**

| | S | C | A | B | D | E |
|---|---|---|---|---|---|---|
| **Sp** | | | 8 | 2 | 1 | 0 |

min(3+1,

# Shortest Path Problem

```
sol(S) = min{sol(A) + d(S,A), sol(C) + d(S,C)}
```




**Adjacency Matrix**

| | S | C | A | B | D | E |
|---|---|---|---|---|---|---|
| **S** | 0 | 2 | 1 | -1 | -1 | -1 |
| **C** | -1 | 0 | 4 | -1 | 3 | -1 |
| **A** | -1 | -1 | 0 | 6 | -1 | -1 |
| **B** | -1 | -1 | -1 | 0 | 1 | 2 |
| **D** | -1 | -1 | -1 | -1 | 0 | 1 |
| **E** | -1 | -1 | -1 | -1 | -1 | 0 |

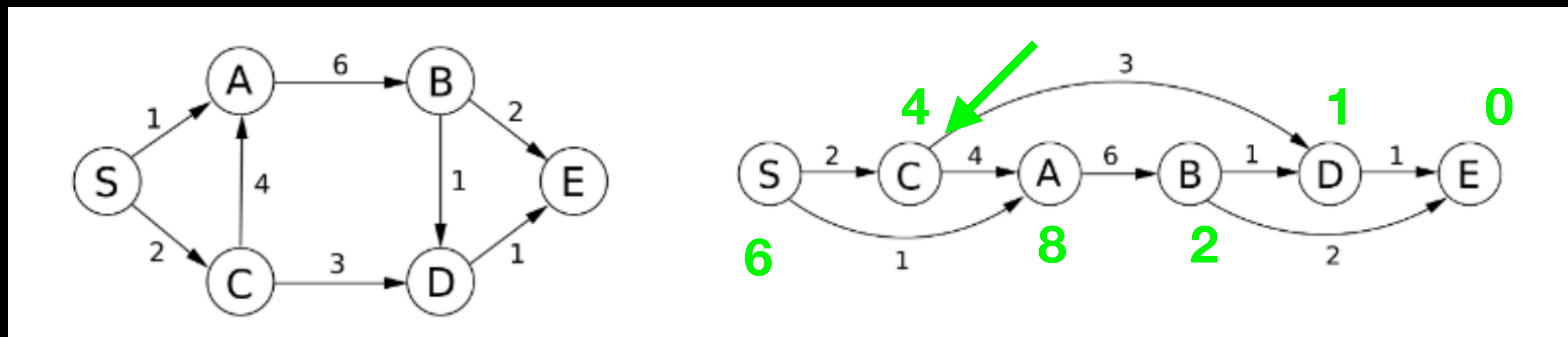
**Shortest Path**

| | S | C | A | B | D | E |
|---|---|---|---|---|---|---|
| **Sp** | | | 8 | 2 | 1 | 0 |

min(3+1,  4+8)

# Shortest Path Problem

`sol(S) = min{sol(A) + d(S,A), sol(C) + d(S,C)}`




**Adjacency Matrix**

| | S | C | A | B | D | E |
|---|---|---|---|---|---|---|
| **S** | 0 | 2 | 1 | -1 | -1 | -1 |
| **C** | -1 | 0 | 4 | -1 | 3 | -1 |
| **A** | -1 | -1 | 0 | 6 | -1 | -1 |
| **B** | -1 | -1 | -1 | 0 | 1 | 2 |
| **D** | -1 | -1 | -1 | -1 | 0 | 1 |
| **E** | -1 | -1 | -1 | -1 | -1 | 0 |

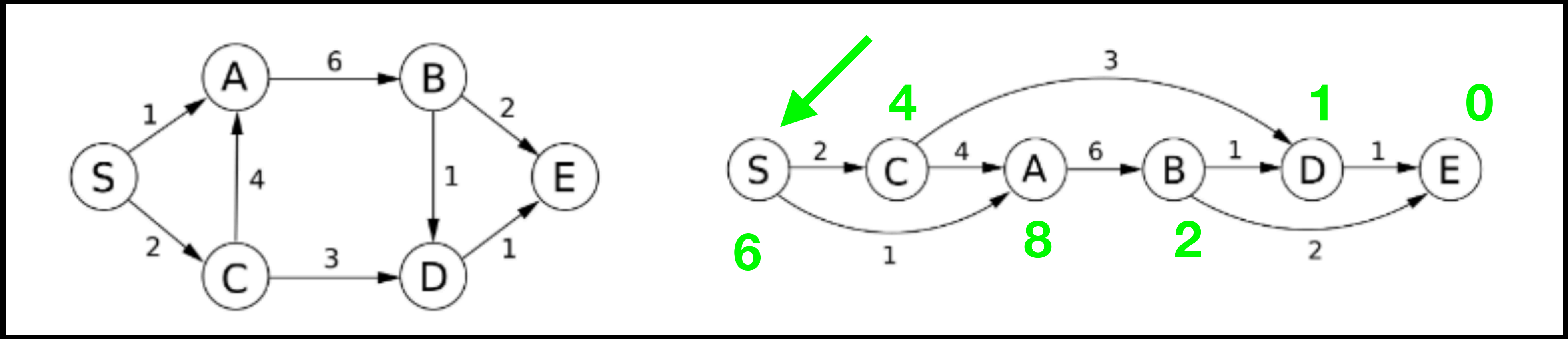
**Shortest Path**

| | S | C | A | B | D | E |
|---|---|---|---|---|---|---|
| **Sp** | | 4 | 8 | 2 | 1 | 0 |

**min(3+1, 4+8)**

# Shortest Path Problem

`sol(S) = min{sol(A) + d(S,A), sol(C) + d(S,C)}`




**Adjacency Matrix**

| | S | C | A | B | D | E |
|---|---|---|---|---|---|---|
| **S** | 0 | 2 | 1 | -1 | -1 | -1 |
| **C** | -1 | 0 | 4 | -1 | 3 | -1 |
| **A** | -1 | -1 | 0 | 6 | -1 | -1 |
| **B** | -1 | -1 | -1 | 0 | 1 | 2 |
| **D** | -1 | -1 | -1 | -1 | 0 | 1 |
| **E** | -1 | -1 | -1 | -1 | -1 | 0 |

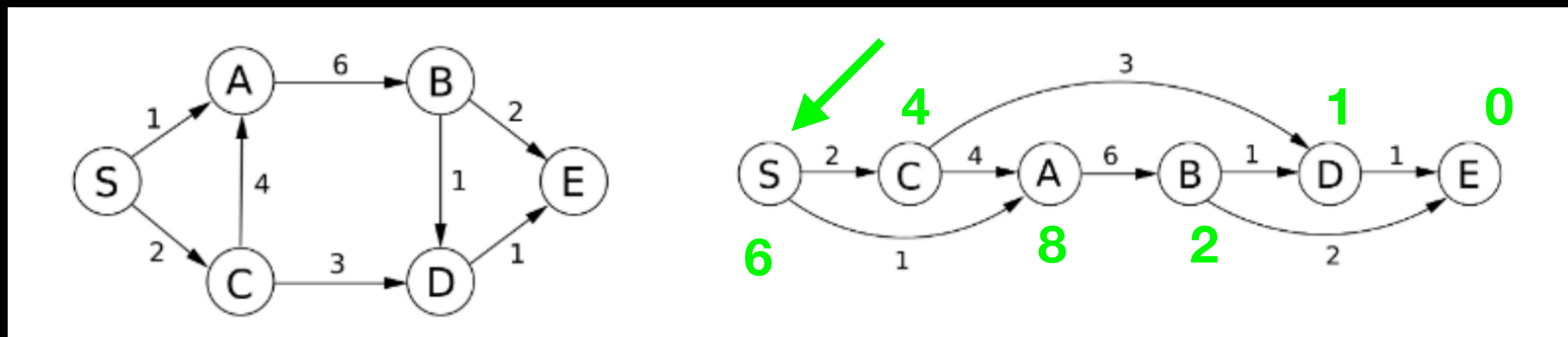
**Shortest Path**

| | S | C | A | B | D | E |
|---|---|---|---|---|---|---|
| **Sp** | | 4 | 8 | 2 | 1 | 0 |

**min(1+8,**

# Shortest Path Problem

```
sol(S) = min{sol(A) + d(S,A), sol(C) + d(S,C)}
```




**Adjacency Matrix**

| | S | C | A | B | D | E |
|---|---|---|---|---|---|---|
| **S** | 0 | 2 | 1 | -1 | -1 | -1 |
| **C** | -1 | 0 | 4 | -1 | 3 | -1 |
| **A** | -1 | -1 | 0 | 6 | -1 | -1 |
| **B** | -1 | -1 | -1 | 0 | 1 | 2 |
| **D** | -1 | -1 | -1 | -1 | 0 | 1 |
| **E** | -1 | -1 | -1 | -1 | -1 | 0 |

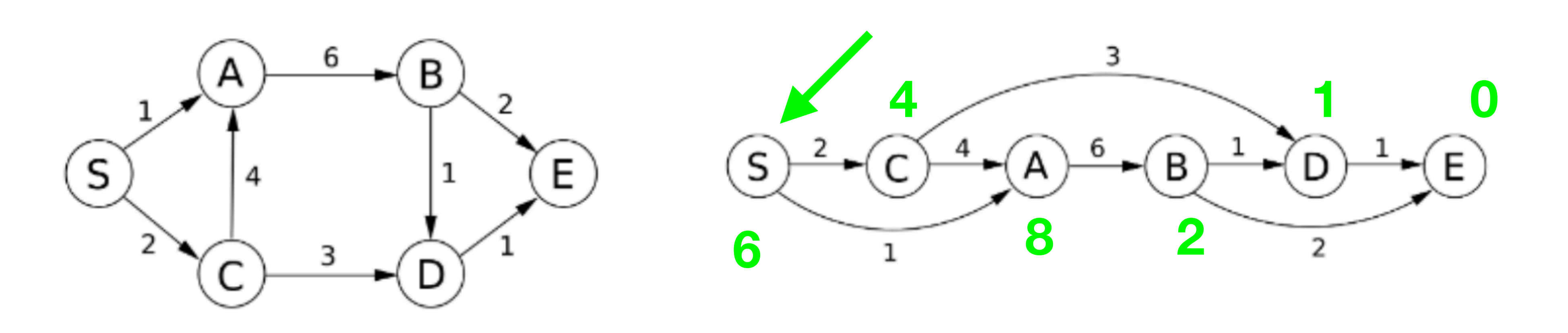
**Shortest Path**

| | S | C | A | B | D | E |
|---|---|---|---|---|---|---|
| **Sp** | | 4 | 8 | 2 | 1 | 0 |

**min(1+8, 2+4)**

# Shortest Path Problem

```
sol(S) = min{sol(A) + d(S,A), sol(C) + d(S,C)}
```




**Adjacency Matrix**

| | S | C | A | B | D | E |
|---|---|---|---|---|---|---|
| **S** | 0 | 2 | 1 | -1 | -1 | -1 |
| **C** | -1 | 0 | 4 | -1 | 3 | -1 |
| **A** | -1 | -1 | 0 | 6 | -1 | -1 |
| **B** | -1 | -1 | -1 | 0 | 1 | 2 |
| **D** | -1 | -1 | -1 | -1 | 0 | 1 |
| **E** | -1 | -1 | -1 | -1 | -1 | 0 |

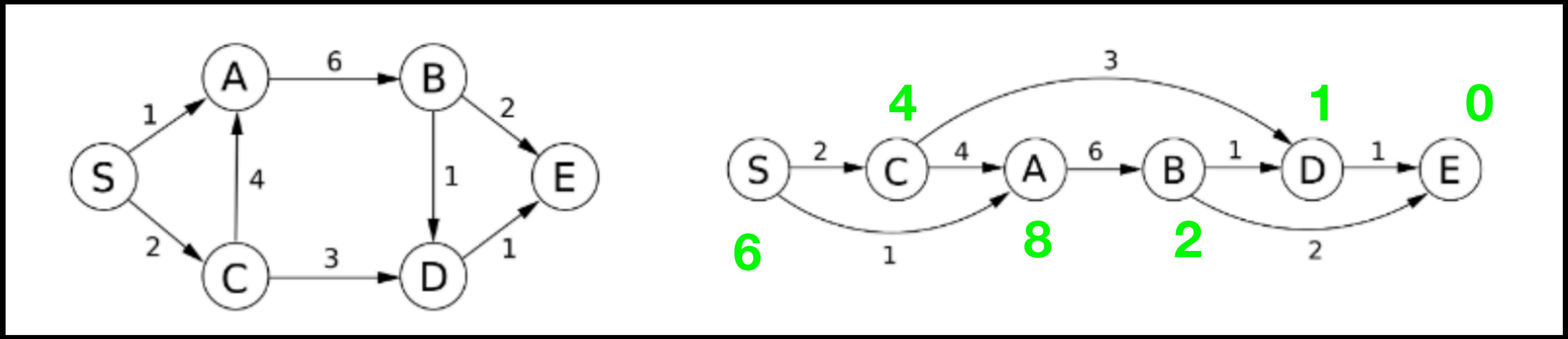
**Shortest Path**

| | S | C | A | B | D | E |
|---|---|---|---|---|---|---|
| **Sp** | 6 | 4 | 8 | 2 | 1 | 0 |

**min(1+8, 2+4)**

# Shortest Path Problem

```
sol(S) = min{sol(A) + d(S,A), sol(C) + d(S,C)}
```




**Adjacency Matrix**

| | S | C | A | B | D | E |
|---|---|---|---|---|---|---|
| **S** | 0 | 2 | 1 | -1 | -1 | -1 |
| **C** | -1 | 0 | 4 | -1 | 3 | -1 |
| **A** | -1 | -1 | 0 | 6 | -1 | -1 |
| **B** | -1 | -1 | -1 | 0 | 1 | 2 |
| **D** | -1 | -1 | -1 | -1 | 0 | 1 |
| **E** | -1 | -1 | -1 | -1 | -1 | 0 |

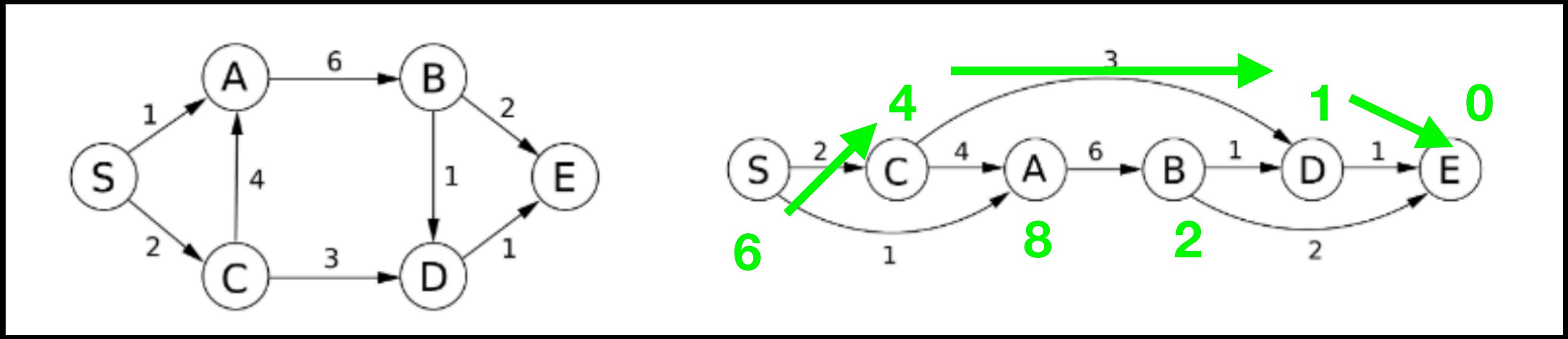
**Shortest Path**

| | S | C | A | B | D | E |
|---|---|---|---|---|---|---|
| **Sp** | 6 | 4 | 8 | 2 | 1 | 0 |

Cost of shortest path

# Shortest Path Problem

```
sol(S) = min{sol(A) + d(S,A), sol(C) + d(S,C)}
```




**Adjacency Matrix**

| | S | C | A | B | D | E |
|---|---|---|---|---|---|---|
| **S** | 0 | 2 | 1 | -1 | -1 | -1 |
| **C** | -1 | 0 | 4 | -1 | 3 | -1 |
| **A** | -1 | -1 | 0 | 6 | -1 | -1 |
| **B** | -1 | -1 | -1 | 0 | 1 | 2 |
| **D** | -1 | -1 | -1 | -1 | 0 | 1 |
| **E** | -1 | -1 | -1 | -1 | -1 | 0 |

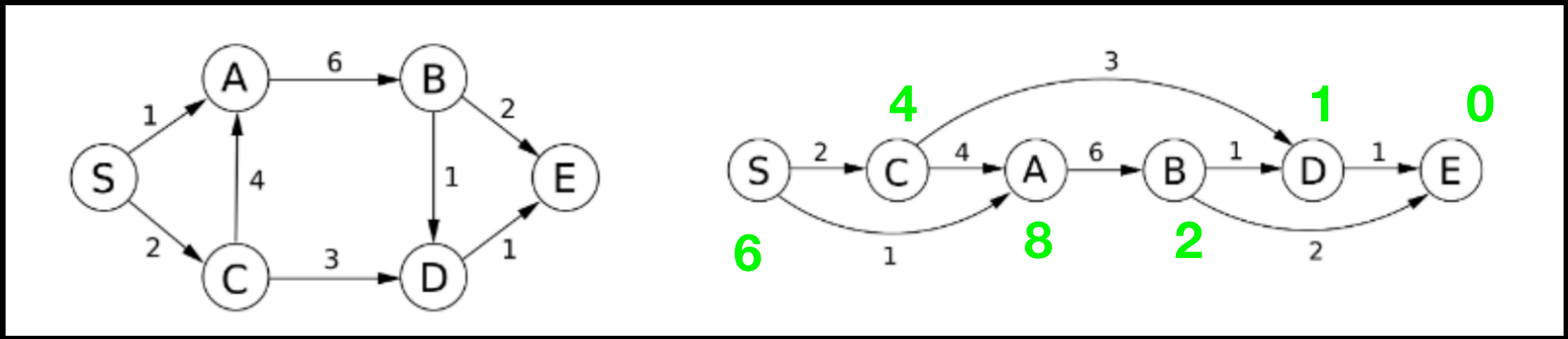
**Shortest Path**

| | S | C | A | B | D | E |
|---|---|---|---|---|---|---|
| **Sp** | 6 | 4 | 8 | 2 | 1 | 0 |

**Using adjacency matrix AND shortest path table, follow cheapest neighbor+edge (not cheapest edge)**

# Shortest Path Problem

```
sol(S) = min{sol(A) + d(S,A), sol(C) + d(S,C)}
```




**Adjacency Matrix**

| | S | C | A | B | D | E |
|---|---|---|---|---|---|---|
| **S** | 0 | 2 | 1 | -1 | -1 | -1 |
| **C** | -1 | 0 | 4 | -1 | 3 | -1 |
| **A** | -1 | -1 | 0 | 6 | -1 | -1 |
| **B** | -1 | -1 | -1 | 0 | 1 | 2 |
| **D** | -1 | -1 | -1 | -1 | 0 | 1 |
| **E** | -1 | -1 | -1 | -1 | -1 | 0 |

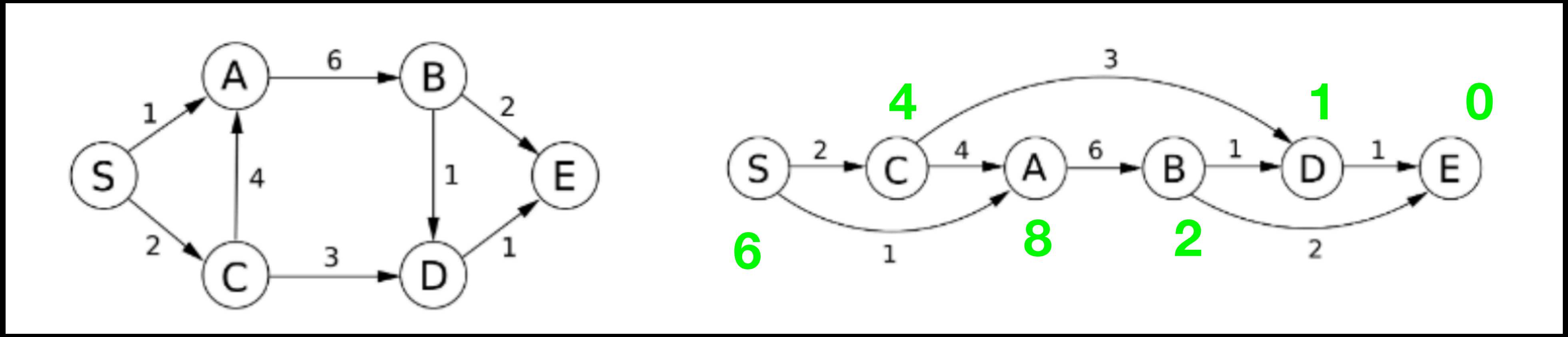
**Shortest Path**

| | S | C | A | B | D | E |
|---|---|---|---|---|---|---|
| **Sp** | 6 | 4 | 8 | 2 | 1 | 0 |

**O(?)**

# Shortest Path Problem

```
sol(S) = min{sol(A) + d(S,A), sol(C) + d(S,C)}
```




**Adjacency Matrix**

| | S | C | A | B | D | E |
|---|---|---|---|---|---|---|
| **S** | 0 | 2 | 1 | -1 | -1 | -1 |
| **C** | -1 | 0 | 4 | -1 | 3 | -1 |
| **A** | -1 | -1 | 0 | 6 | -1 | -1 |
| **B** | -1 | -1 | -1 | 0 | 1 | 2 |
| **D** | -1 | -1 | -1 | -1 | 0 | 1 |
| **E** | -1 | -1 | -1 | -1 | -1 | 0 |

**Shortest Path**

| | S | C | A | B | D | E |
|---|---|---|---|---|---|---|
| **Sp** | 6 | 4 | 8 | 2 | 1 | 0 |

$O(n^2)$