Dynamic Programming

Algorithms

Definition

- A quite generic strategy that reduces the solution of a problem to the solution of similar subproblems
  - Divide and conquer:
    - Division leads to a recursion subject to the Master Theorem
    - Generate two or more subproblems
  - General dynamic programming:
    - In general, no division but a reduction of problem size
    - Leads often to super-polynomial algorithms

Usage

- Dynamic programming is very generic
    - Often, does not lead to poly-time algorithms
    - Used often when problems need to be solved even though it is known that a good scalable algorithm is unavailable
    - I.e. an NP-complete problem

Example 1
Forming sums

- Determine the number of ways we can write a number *n* as a sum of ones and twos (not using commutativity)
  - Example:

$$4 = 1 + 1 + 1 + 1$$

$$4 = 2 + 1 + 1$$

$$4 = 1 + 2 + 1$$

$$4 = 1 + 1 + 2$$

$$4 = 2 + 2$$

  - Five possibilities

Example 1
Forming sums

- Idea:
    - Sum ends with either a +1 or a +2
    - The part before sums to *n*-1 or *n*-2 respectively

Example 1
Forming sums

- Idea: The ways to write *n* are given by writing *n*-2 and *n*-1
- Number of ways for *n*: S_n
- Recursion formula:

$$S_n = S_{n-1} + S_{n-2}$$

$$S_0 = 0$$

$$S_1 = 0$$

- Fibonacci numbers!

Example 1
Forming sums

- Extend to sums with 1, 2, 3:
    - Your turn

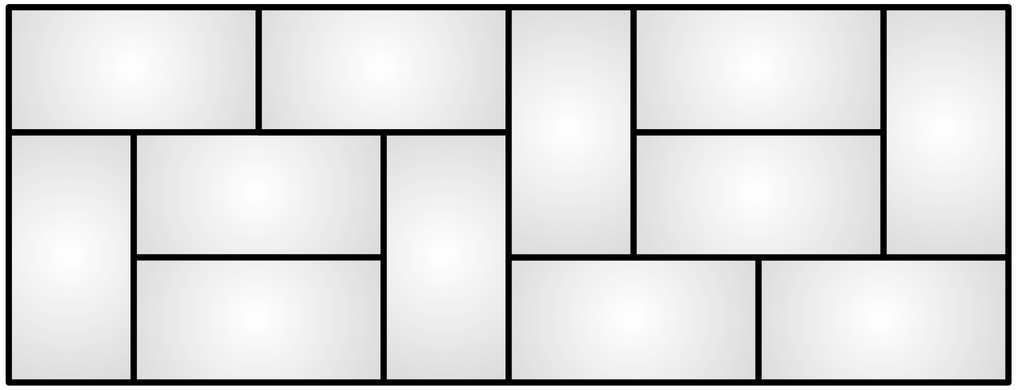
Example 1
Forming sums

- Solution

$$\begin{aligned}
D_0 &= 0 \\
D_1 &= 1 \\
D_2 &= 2 \\
D_n &= D_{n-1} + D_{n-2} + D_{n-3}
\end{aligned}$$

Example 2
Dominos

- Count the number of ways in which a $3 \times n$ field can be filled with domino stones of size 2×1

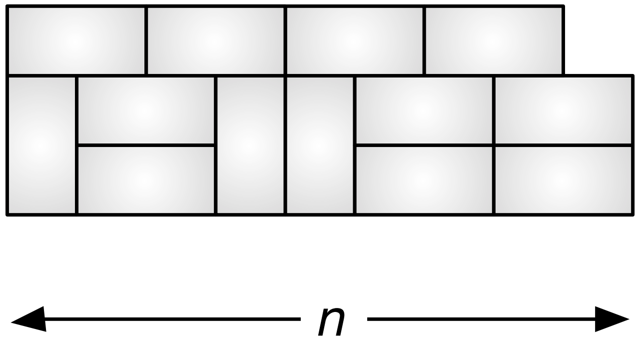


Example 2
Dominos

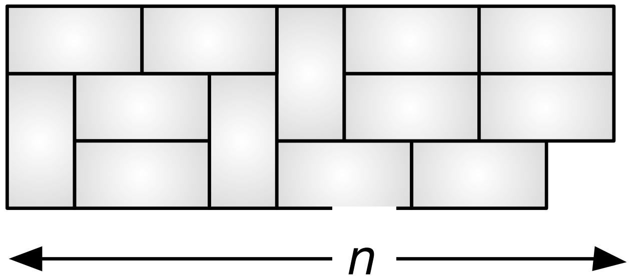
- Can we reduce the problem to simpler ones?
  - T_n number of tessellations for an $3 \times n$ area
  - There is a problem for the reduction

Example 2
Dominos

- Need to introduce two more shapes



A_n

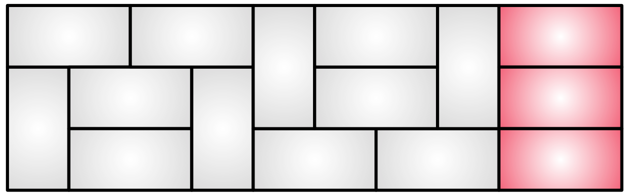


B_n

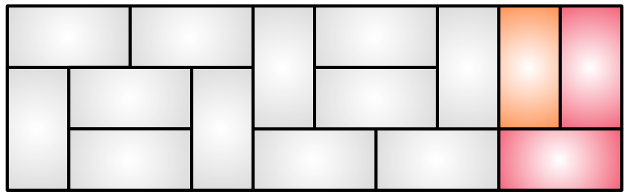
Example 2
Dominos

- Need recursions for all three

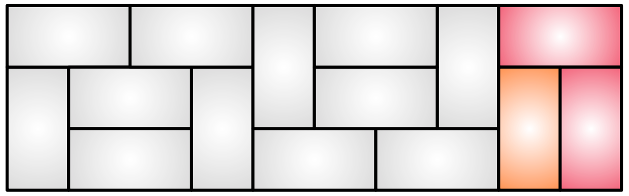
T



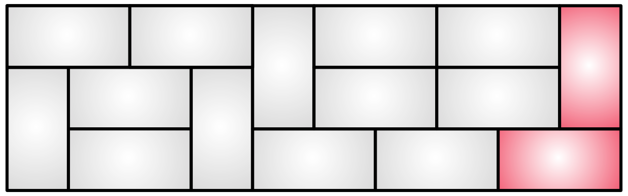
B and T



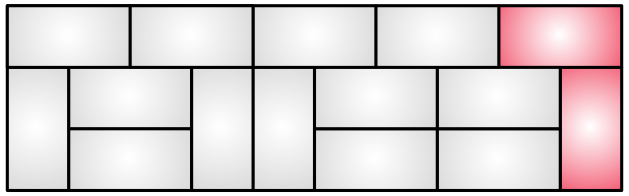
A and T



B



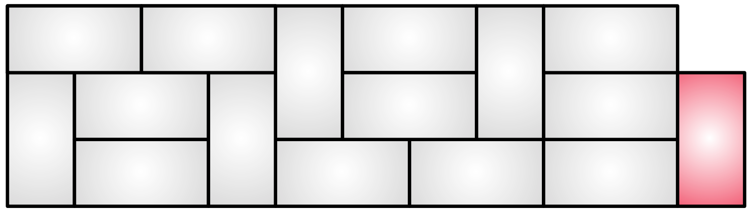
A



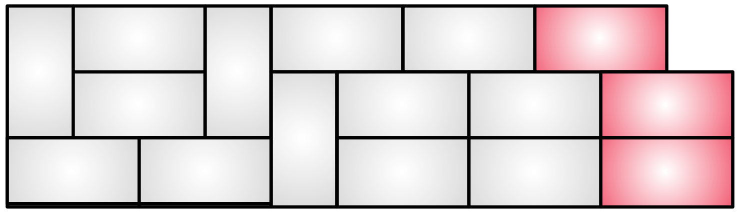
$$T_n = A_{n-1} + B_{n-1} + T_{n-2}$$

Example 2
Dominos

- To generate a type A



$$A_n = A_{n-2} + T_{n-1}$$



Example 2
Dominos

- Need to give base cases:
  - $T_2 = 3, T_1 = 0$
  - $A_1 = 1$
  - $B_1 = 1$

Dynamic Programming

- Three steps:
    - Define sub-problems
    - Set-up a recursion
    - Determine base cases

Knapsack Problem

- Continuous knapsack problem
  - Select items from set $X = \{A_1, A_2, \ldots, A_n\}$
  - Each item has a weight w_i
  - Each item has a value v_i
  - Maximize $\sum_{i \in M} s_i v_i$ subject to $\sum_{i \in M} s_i w_i \leq C$
    - with $s_i \in [0,1]$

Knapsack Problem

- Continuous knapsack problem
  - Greedy algorithm solves the continuous knapsack algorithm:
    - Order items by ratios of value over weight
      - Select items in order of this ratio
        - As long as remaining under capacity
      - Last item might be fractional

Knapsack Problem

- Example

Item	Value	Weight	Ratio
A	9	5	1.80
B	7	4	1.75
C	6	4	1.5
D	3	2	1.5
E	2	2	1
F	1	1	1

- Total capacity is 6

Knapsack Problem

- Example

Item	Value	Weight	Ratio
A	9	5	1.80
B	7	4	1.75
C	6	4	1.5
D	3	2	1.5
E	2	2	1
F	1	1	1

- $s_A = 1,\ s_B = 0.25,\ s_C = s_D = s_E = s_F = 0$
- Total capacity is 6, total value is 10.75

Knapsack Problem

- 0-1 knapsack
  - Can select only an entire item, but not a fraction
- Greedy method is no longer best

Knapsack Problem

- Example

Item	Value	Weight	Ratio
A	9	5	1.80
B	7	4	1.75
C	6	4	1.5
D	3	2	1.5
E	2	2	1
F	1	1	1

- Total capacity is 6

Knapsack Problem

- Example

Item	Value	Weight	Ratio
A	9	5	1.80
B	7	4	1.75
C	6	4	1.5
D	3	2	1.5
E	2	2	1

- Greedy solution: $s_A = 1,\ s_B = s_C = s_D = s_E = s_F = 0$
- Total weight is 5 and total gain is 9

Knapsack Problem

- Example

Item	Value	Weight	Ratio
A	9	5	1.80
B	7	4	1.75
C	6	4	1.5
D	3	2	1.5
E	2	2	1
F	1	1	1

- Better solution: $s_B = 1,\ s_D = 1, s_A = s_C = s_E = s_F = 0$
- Total weight is 6 and total value is 10

Knapsack Problem

- Solving knapsack problems with dynamic programming
    - Sub-problems?
    - Recursion?
    - Base Case?

Knapsack Problem

- Sub-problems
  - Optimal solution needs to be composed of solutions for subproblem
  - Use less items, use fewer capacities

Knapsack Problem

- Order all items in any order
  - Optimal solution:
    - Two alternatives:
      - Last item is included
      - Last item is not included

Knapsack Problem

- Order all items in any order
    - Optimal solution:
        - Two alternatives:
            - Last item is included
                - Before inclusion of the last item
                    - Solved knapsack for all but last item with total capacity minus weight of last item
            - Last item is not included
                - Before non-inclusion of the last item
                    - Solved knapsack for all but last item with total capacity

Knapsack Problem

- Generate Table
  - Columns: set of items is

    $\{A_0\},\ \{A_0, A_1\},\ \{A_0, A_1, A_2\}, \{A_0, A_1, A_2, A_3\}, \ldots$
  - Rows: Capacity below problem capacity

Knapsack Problem

- Example

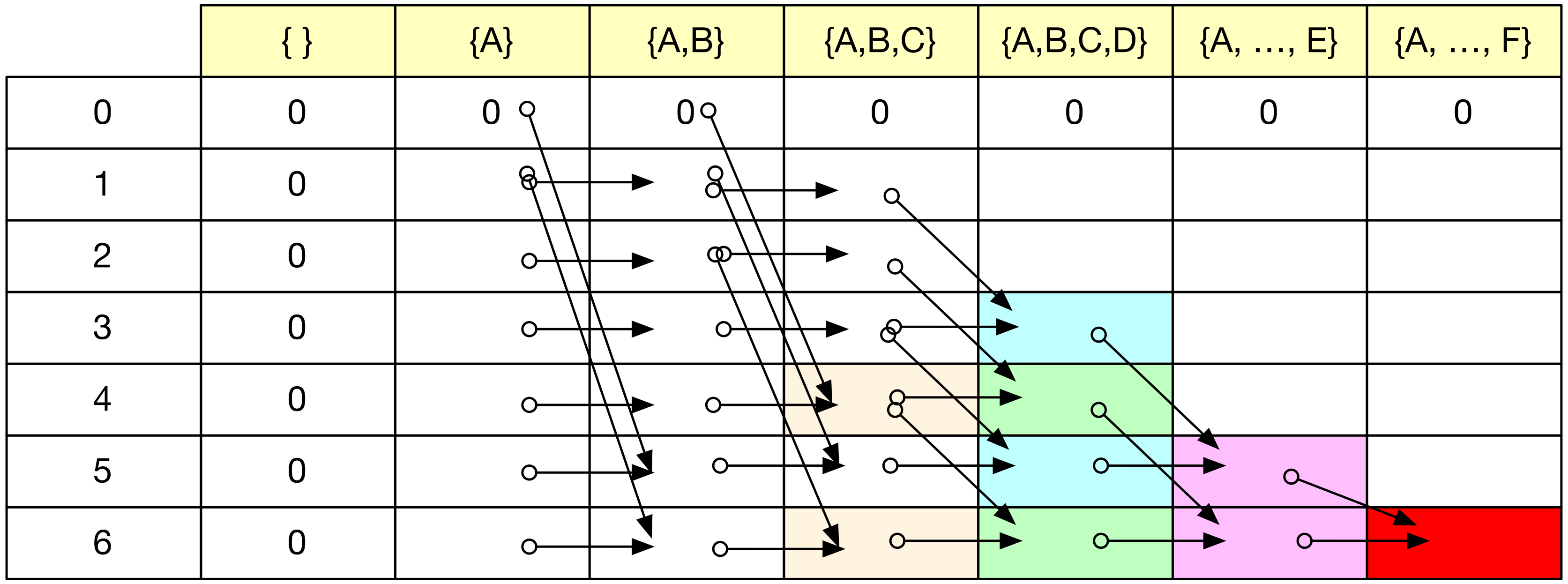
Item	Value	Weight	Ratio
A	9	5	1.80
B	7	4	1.75
C	6	4	1.5
D	3	2	1.5
E	2	2	1
F	1	1	1

- Total capacity is 6

Knapsack Problem

- Example

Item	Value	Weight	Ratio
A	9	5	1.80
B	7	4	1.75
C	6	4	1.5
D	3	2	1.5
E	2	2	1
F	1	1	1



	{ }	{A}	{A,B}	{A,B,C}	{A,B,C,D}	{A, …, E}	{A, …, F}
0	0	0	0	0	0	0	0
1	0						
2	0						
3	0						
4	0						
5	0						
6	0						

- Cell in column {*A*, … , *X*} and row *r* is the gain of selecting from {*A*, … , *X*} and maximum capacity *r*

Knapsack Problem

- Element in row *r* and columns *X_i*

$$g_{r,X_i} = \begin{cases} g_{r,X_{i-1}} & \textbf{if } X_i \textbf{ is not selected} \\ g_{r-w_i,X_{i-1}} + v_i & \textbf{if } X_i \textbf{ is selected} \end{cases}$$

$$= \max\left(g_{r,X_{i-1}}, g_{r-w_i,X_{i-1}} + v_i\right)$$

Knapsack Problem

- Base cases:
  - No items to select: gain is zero
  - Capacity is zero: gain is zero

Knapsack Problem

- Work **forward** adding column after column
  - Item A has weight 5 and value 9

	{ }	{A}	{A,B}	{A,B,C}	{A,B,C,D}	{A, …, E}	{A, …, F}
0	0	0	0	0	0	0	0
1	0	0					
2	0	0					
3	0	0					
4	0	0					
5	0	9					
6	0	9					

Knapsack Problem

- Item B has weight 4 and value 7

	{ }	{A}	{A,B}	{A,B,C}	{A,B,C,D}	{A, …, E}	{A, …, F}
0	0	0	0	0	0	0	0
1	0	0	0				
2	0	0	0				
3	0	0	0				
4	0	0	7				
5	0	9	max(7,9)				
6	0	9	max(7,9)				

Knapsack Problem

- Item C has value 6 and weight 4

	{ }	{A}	{A,B}	{A,B,C}	{A,B,C,D}	{A, …, E}	{A, …, F}
0	0	0	0	0	0	0	0
1	0	0	0	0			
2	0	0	0	0			
3	0	0	0	0			
4	0	0	7	max(6,7)			
5	0	9	9	max(6, 9)			
6	0	9	9	max(6, 9)			

Knapsack Problem

- Item *D* has weight 2 and value 3

	{ }	{A}	{A,B}	{A,B,C}	{A,B,C,D}	{A, …, E}	{A, …, F}
0	0	0	0	0	0	0	0
1	0	0	0	0	0		
2	0	0	0	0	3		
3	0	0	0	0	3		
4	0	0	7	7	max(7,3)		
5	0	9	9	9	max(9,3)		
6	0	9	9	9	max(9,10)		

Knapsack Problem

- Item *E* has weight 2 and value 2

	{ }	{A}	{A,B}	{A,B,C}	{A,B,C,D}	{A, …, E}	{A, …, F}
0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	
2	0	0	0	0	3	max(3,2)	
3	0	0	0	0	3	max(3,2)	
4	0	0	7	7	7	max(7,5)	
5	0	9	9	9	9	max(9,5)	
6	0	9	9	9	10	max(10,9)	

Knapsack Problem

- Item F has weight 1 and value 1

	{ }	{A}	{A,B}	{A,B,C}	{A,B,C,D}	{A, …, E}	{A, …, F}
0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	max(1,0)
2	0	0	0	0	3	3	max(3,1)
3	0	0	0	0	3	3	max(3,4)
4	0	0	7	7	7	7	max(7,4)
5	0	9	9	9	9	9	max(9,8)
6	0	9	9	9	10	10	max(10,10)

Knapsack Problem

- Final table tells us the realizable total value

	{ }	{A}	{A,B}	{A,B,C}	{A,B,C,D}	{A, …, E}	{A, …, F}
0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	1
2	0	0	0	0	3	3	3
3	0	0	0	0	3	3	4
4	0	0	7	7	7	7	7
5	0	9	9	9	9	9	9
6	0	9	9	9	10	10	10

  - but not how to obtain it

Knapsack Problem

- Can either annotate table entry with how we got them

	{ }	{A}	{A,B}	{A,B,C}	{A,B,C,D}	{A, …, E}	{A, …, F}
0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	1
2	0	0	0	0	3	3	3
3	0	0	0	0	3	3	4
4	0	0	7	7	7	7	7
5	0	9	9	9	9	9	9
6	0	9	9	9	10	10	10

- or can backtrack

Knapsack Problem

- Backtracking

	{ }	{A}	{A,B}	{A,B,C}	{A,B,C,D}	{A, …, E}	{A, …, F}
0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	1
2	0	0	0	0	3	3	3
3	0	0	0	0	3	3	4
4	0	0	7	7	7	7	7
5	0	9	9	9	9	9	9
6	0	9	9	9	10	10	10=9+1

- Last entry is either with or without including F

Knapsack Problem

- Backtracking

	{ }	{A}	{A,B}	{A,B,C}	{A,B,C,D}	{A, …, E}	{A, …, F}
0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	1
2	0	0	0	0	3	3	3
3	0	0	0	0	3	3	4
4	0	0	7	7	7	7	7
5	0	9	9	9	9	9	9
6	0	9	9	9	10	10	10=9+1

  - Let’s say we include it

Knapsack Problem

- Backtracking
  - Then the 10 was realized as 9+1 with the previous column and row - weight of item F =1

	{ }	{A}	{A,B}	{A,B,C}	{A,B,C,D}	{A, …, E}	{A, …, F}
0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	1
2	0	0	0	0	3	3	3
3	0	0	0	0	3	3	4
4	0	0	7	7	7	7	7
5	0	9	9	9	9	9	9
6	0	9	9	9	10	10	10=9+1

Knapsack Problem

- Backtracking
  - No such choice with the other ones until we get to *A*

	{ }	{A}	{A,B}	{A,B,C}	{A,B,C,D}	{A, …, E}	{A, …, F}
0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	1
2	0	0	0	0	3	3	3
3	0	0	0	0	3	3	4
4	0	0	7	7	7	7	7
5	0	9	9	9	9	9	9
6	0	9	9	9	10	10	10=9+1

Knapsack Problem

- Backtracking

	{ }	{A}	{A,B}	{A,B,C}	{A,B,C,D}	{A, …, E}	{A, …, F}
0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	1
2	0	0	0	0	3	3	3
3	0	0	0	0	3	3	4
4	0	0	7	7	7	7	7
5	0	9	9	9	9	9	9
6	0	9	9	9	10	10	10=9+1

- Included A and F for a total value of 10 and a total weight of 6

Knapsack Problem

- Backtracking alternative in the first step:

	{ }	{A}	{A,B}	{A,B,C}	{A,B,C,D}	{A, …, E}	{A, …, F}
0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	1
2	0	0	0	0	3	3	3
3	0	0	0	0	3	3	4
4	0	0	7	7	7	7	7
5	0	9	9	9	9	9	9
6	0	9	9	9	10	10	10

- Don’t include F, E, D

Knapsack Problem

- Backtracking

	{ }	{A}	{A,B}	{A,B,C}	{A,B,C,D}	{A, …, E}	{A, …, F}
0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	1
2	0	0	0	0	3	3	3
3	0	0	0	0	3	3	4
4	0	0	7	7	7	7	7
5	0	9	9	9	9	9	9
6	0	9	9	9	10	10	10

  - must have included item D

Knapsack Problem

- Backtracking

	{ }	{A}	{A,B}	{A,B,C}	{A,B,C,D}	{A, …, E}	{A, …, F}
0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	1
2	0	0	0	0	3	3	3
3	0	0	0	0	3	3	4
4	0	0	7	7	7	7	7
5	0	9	9	9	9	9	9
6	0	9	9	9	10	10	10

- D has value 3 and weight 2

Knapsack Problem

- Backtracking

	{ }	{A}	{A,B}	{A,B,C}	{A,B,C,D}	{A, …, E}	{A, …, F}
0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	1
2	0	0	0	0	3	3	3
3	0	0	0	0	3	3	4
4	0	0	7	7	7	7	7
5	0	9	9	9	9	9	9
6	0	9	9	9	10	10	10

- Do not include C

Knapsack Problem

- Backtracking

	{ }	{A}	{A,B}	{A,B,C}	{A,B,C,D}	{A, …, E}	{A, …, F}
0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	1
2	0	0	0	0	3	3	3
3	0	0	0	0	3	3	4
4	0	0	7	7	7	7	7
5	0	9	9	9	9	9	9
6	0	9	9	9	10	10	10

- But include B

Knapsack Problem

- Backtracking

	{ }	{A}	{A,B}	{A,B,C}	{A,B,C,D}	{A, …, E}	{A, …, F}
0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	1
2	0	0	0	0	3	3	3
3	0	0	0	0	3	3	4
4	0	0	7	7	7	7	7
5	0	9	9	9	9	9	9
6	0	9	9	9	10	10	10

  - and therefore not A

- Alternative solution: Select B and D for the same total value and capacity

Knapsack Problem

- Your turn: Extend to total capacity 10

Item	Value	Weight
A	9	5
B	7	4
C	6	4
D	3	2
E	2	2
F	1	1

Cap	A	B	C	D	E	F
1	0	0	0	0	0	1
2	0	0	0	3	3	3
3	0	0	0	3	3	4
4	0	7	7	7	7	7
5	9	9	9	9	9	9
6	9	9	9	10	10	10
7	9	9	9	12	12	12
8	9	9	13	13	13	13
9	9	16	16	16	16	16
10	9	16	16	16	16	17

Knapsack Problem

Cap	A	B	C	D	E	F
1	0	0	0	0	0	1
2	0	0	0	3	3	3
3	0	0	0	3	3	4
4	0	7	7	7	7	7
5	9	9	9	9	9	9
6	9	9	9	10	10	10
7	9	9	9	12	12	12
8	9	9	13	13	13	13
9	9	16	16	16	16	16
10	9	16	16	16	16	17

`Include F`

Knapsack Problem

Cap	A	B	C	D	E	F
1	0	0	0	0	0	1
2	0	0	0	3	3	3
3	0	0	0	3	3	4
4	0	7	7	7	7	7
5	9	9	9	9	9	9
6	9	9	9	10	10	10
7	9	9	9	12	12	12
8	9	9	13	13	13	13
9	9	16	16	16	16	16
10	9	16	16	16	16	17

```
Include F
Do not
include E
```

Knapsack Problem

Cap	A	B	C	D	E	F
1	0	0	0	0	0	1
2	0	0	0	3	3	3
3	0	0	0	3	3	4
4	0	7	7	7	7	7
5	9	9	9	9	9	9
6	9	9	9	10	10	10
7	9	9	9	12	12	12
8	9	9	13	13	13	13
9	9	16	16	16	16	16
10	9	16	16	16	16	17

```
Include F
Do not
include E
Do not
include D
```

Knapsack Problem

Cap	A	B	C	D	E	F
1	0	0	0	0	0	1
2	0	0	0	3	3	3
3	0	0	0	3	3	4
4	0	7	7	7	7	7
5	9	9	9	9	9	9
6	9	9	9	10	10	10
7	9	9	9	12	12	12
8	9	9	13	13	13	13
9	9	16	16	16	16	16
10	9	16	16	16	16	17

```
Include F
Do not
include E
Do not
include D
Do not
include C
```

Knapsack Problem

Cap	A	B	C	D	E	F
1	0	0	0	0	0	1
2	0	0	0	3	3	3
3	0	0	0	3	3	4
4	0	7	7	7	7	7
5	9	9	9	9	9	9
6	9	9	9	10	10	10
7	9	9	9	12	12	12
8	9	9	13	13	13	13
9	9	16	16	16	16	16
10	9	16	16	16	16	17

```
Include F
Do not
include E
Do not
include D
Do not
include C
Include B
```

Knapsack Problem

Cap	A	B	C	D	E	F
1	0	0	0	0	0	1
2	0	0	0	3	3	3
3	0	0	0	3	3	4
4	0	7	7	7	7	7
5	9	9	9	9	9	9
6	9	9	9	10	10	10
7	9	9	9	12	12	12
8	9	9	13	13	13	13
9	9	16	16	16	16	16
10	9	16	16	16	16	17

```
Include F
Do not include E
Do not include D
Do not include C
Include B
Include A
```

Knapsack Problem

- Multiple Item Selection
  - When considering item *j*, need to look at including ν items $$\nu \in \{0,1,\ldots,\lfloor\frac{i}{w_j}\rfloor\}$$
  - Formula changes
    - $g_{i,j} = \max(\{g_{i-\nu w_j} + \nu v_j \,|\, \nu \in \{0,1,\ldots,\lfloor\frac{i}{w_j}\rfloor\}\})$

Knapsack Problem

- Example: Items with value-weight of (26,5), (20,4), (14,3), (9,2), (4,1)

Knapsack Problem

TW	A	B	C	D	E
0	0	0	0	0	0
1	0	0	0	0	4
2	0	0	0	9	9
3	0	0	14	14	14
4	0	20	20	20	20
5	26	26	26	26	26
6	26	26	28	29	30
7	26	26	34	35	35
8	26	40	40	40	40
9	26	46	46	46	46
10	52	52	52	52	52
11	52	52	54	55	56
12	52	60	60	61	61
13	52	66	66	66	66
14	52	72	72	72	72
15	78	78	78	78	78
16	78	80	80	81	82

`(26,5), (20,4),`

Knapsack Problem

TW	A	B	C	D	E
0	0	0	0	0	0
1	0	0	0	0	4
2	0	0	0	9	9
3	0	0	14	14	14
4	0	20	20	20	20
5	26	26	26	26	26
6	26	26	28	29	30
7	26	26	34	35	35
8	26	40	40	40	40
9	26	46	46	46	46
10	52	52	52	52	52
11	52	52	54	55	56
12	52	60	60	61	61
13	52	66	66	66	66
14	52	72	72	72	72
15	78	78	78	78	78
16	78	80	80	81	82

(26,5), (20,4)

Backtrack to find optim
selection

Knapsack Problem

TW	A	B	C	D	E
0	0	0	0	0	0
1	0	0	0	0	4
2	0	0	0	9	9
3	0	0	14	14	14
4	0	20	20	20	20
5	26	26	26	26	26
6	26	26	28	29	30
7	26	26	34	35	35
8	26	40	40	40	40
9	26	46	46	46	46
10	52	52	52	52	52
11	52	52	54	55	56
12	52	60	60	61	61
13	52	66	66	66	66
14	52	72	72	72	72
15	78	78	78	78	78
16	78	80	80	81	82

(26,5), (20,4)

Optimal solution:

3 items of type A
1 item of type E

Matrix Chain Multiplication

- Given *n* integer matrices of various dimensions

$$A_1, A_2, A_3, \ldots, A_n$$

- Task is multiply the matrices with the least number of multiplications

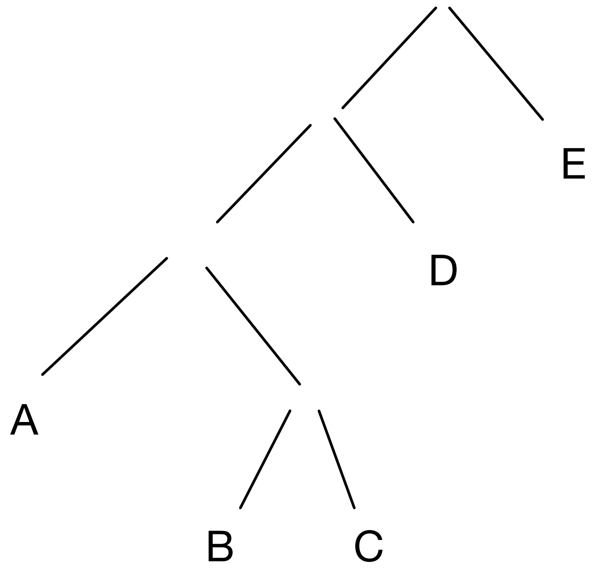
$$A_1 \times A_2 \times A_3 \times \ldots \times A_n$$

  - Can change the order in which we execute the multiplications

Matrix Chain Multiplication

- Different parenthesization have different costs
  - Parenthesization corresponds to different evaluation trees

$((A(BC))D)E$



Matrix Chain Multiplication

- Dynamic programming approach
  - A product of *n* matrices is made up of one of the following
    - Product of 1 with product of *n*-1 matrices
    - Product of 2 with product of *n-2* matrices
    - …
    - Product of *n*-2 with product of 2 matrices
    - Product of *n*-1 with product of 1 matrix

Matrix Chain Multiplication

- Example:
  - A 5×7
  - B 7×2
  - C 2×10
  - D 10×4
  - E 4×5

Matrix Chain Multiplication

- Start with product of two matrices in order
- $AB \qquad 5 \times 7 \times 2 = 70$

  $BC \qquad 7 \times 2 \times 10 = 140$

  $CD \qquad 2 \times 10 \times 4 = 80$

  $DE \qquad 10 \times 4 \times 5 = 200$

$$A : 5 \times 7; \quad B : 7 \times 2; \quad C : 2 \times 10; \quad D : 10 \times 4; \quad E : 4 \times 5$$

Matrix Chain Multiplication

- Then products of three

$A(BC) \qquad 5 \times 7 \times 10 + 140 = 490 \qquad (AB)C \qquad 5 \times 2 \times 10 + 70 = 170$

$B(CD) \qquad 7 \times 2 \times 4 + 80 = 136 \qquad (BC)D \qquad 7 \times 10 \times 4 + 140 = 420$

$C(DE) \qquad 2 \times 10 \times 5 + 200 = 300 \qquad (CD)E \qquad 80 + 2 \times 4 \times 5 = 120$

$A : 5 \times 7; \quad B : 7 \times 2; \quad C : 2 \times 10; \quad D : 10 \times 4; \quad E : 4 \times 5$

Matrix Chain Multiplication

- And products of four $ABCD$

$$(AB)(CD) \qquad 5 \times 2 \times 4 + 70 + 80 = 190$$

$$A(BCD) \qquad 5 \times 7 \times 4 + 136 = 276$$

$$(ABC)D \qquad 170 + 5 \times 10 \times 4 = 370$$

- $BCDE$

$$(BC)(DE) \qquad 7 \times 10 \times 5 + 140 + 200 = 690$$

$$B(CDE) \qquad 7 \times 2 \times 5 + 120 = 190$$

$$(BCD)E \qquad 7 \times 4 \times 5 + 136 = 276$$

$$A : 5 \times 7; \quad B : 7 \times 2; \quad C : 2 \times 10; \quad D : 10 \times 4; \quad E : 4 \times 5$$

Matrix Chain Multiplication

- And finally the complete product

$$A(BCDE): \quad 5\times 7\times 5 + 190 = 175 + 190 = 285$$

$$(AB)(CDE): \quad 5\times 2\times 5 + 70 + 120 = 50 + 190 = 240$$

$$(ABC)(DE): \quad 5\times 10\times 5 + 170 + 200 = 250 + 370 = 620$$

$$(ABCD)E: \quad 5\times 4\times 5 + 190 = 100 + 190 = 290$$

$$A: 5\times 7; \quad B: 7\times 2; \quad C: 2\times 10; \quad D: 10\times 4; \quad E: 4\times 5$$

Matrix Chain Multiplication

- How to best organize the calculation?

A	B	C	D	E
0	0	0	0	0

AB	BC	CD	DE
70	140	80	200

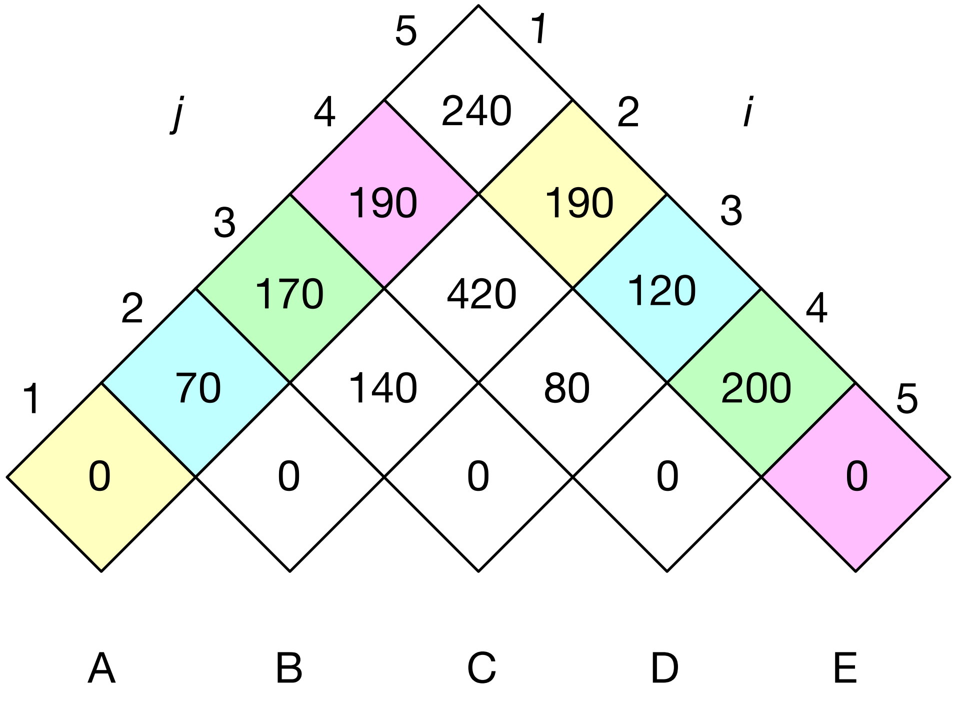
ABC	BCD	CDE
170	136	120

ABCD	BCDE
190	190

ABCDE
240

Matrix Chain Multiplication

- Another way to look at it:



$$240 = \min(190 + \textbf{costs}(A, BCDE), 120 + 70 + \textbf{costs}(AB, CDE), 200 + 170 + \textbf{costs}(ABC, DE), 0 + 190 + \textbf{costs}(ABCD, E))$$

Matrix Chain Multiplication

- Arrange the sizes of the matrices in an array sizes
- Matrix A_i has size `sizes[i-1] x sizes[i]`
- Recursively, define
  - `m[i][j] = 0` if `i==j`
  - `m[i][j] = min([ m[i][k]+m[k+1][j] + sizes[i-1]sizes[k]sizes[j] for k in range(i,j)])`
- To remember our choice for *k*, we mark it in an array
  - `best[i][j] = k`

Matrix Chain Multiplication

- Implementation:
  - We can either fill in the two arrays (m and best)
  - Or we can use memoization with the recursion

Edit Distance

- Levenshtein Distance:
  - Used to find the closest word to a given word for spelling correction
  - Used to define the closeness of two nucleotide sequences

AGGCTATCACCTGACCTCCAGGCCGATGCCC

TAGCTATCACGACCGCGGTCGATTTGCCCGAC

-**AG**G**CTATCAC**CT**GACC**T**C**CA**GG**C**CGA**--**TGCCC**---

T**AG**-**CTATCAC**--**GACC**G**C**--**GG**T**CGA**TT**TGCCC**GAC

Edit Distance

- Edit distance is based on atomic operations
  - Insertion of a character
  - Deletion of a character
  - Substitution of a character
- Assign different weights to each operation
- Edit distance is the minimum cumulative weight of the operations needed to transform one string to another

Edit Distance

- Example
  - algorithm —> altruistic
  - algorithm —> altorithm (substitution)
  - altorithm —>altrithm (deletion)
  - altrithm —> altruithm (insertion)
  - altruithm —> altruishm (substitution)
  - altruithm —> altruistm (substitution)
  - altruistm —> altruistim (insertion)
  - altruistim —> altruistic (substitution

Edit Distance

- There are usually many ways to transform one string to another
    - Often, a series of partial transformation has the same effect as another
    - Hence, we can save on revisiting the same states

Edit Distance

- First attempt:
  - Define the states as consisting of the first *k* letters
    - But this does not take care of deletions and insertions
- Second attempt:
  - Use all sub-strings of the two strings
    - This does not help much because there are too many of them.
    -

Edit Distance

- Third and final attempt:
  - States are given by the prefix of each sequence, but the length can vary
  - Two strings of length *n* and *m*
  - Define *D*(*i, j*) the minimum costs of transforming the first *i* letters of the first into the first *j* letters of the second string

Edit Distance

- Theorem: Any sequence of optimal edits can be ordered by increasing distance from the beginning

Edit Distance

- To calculate $D[i, j]$
  - Use a substitution
  - Use a deletion
  - Use an insertion

Edit Distance

- Initialization:
- From an empty string, insert into the other string

$$\forall i \in \{0,n\} : \; d[i,0] = i$$

$$\forall j \in \{0,m\} : \; d[0,j] = j$$

Edit Distance

- Recursion:
- $$d[i,j] = \begin{cases} d[i,j-1]+1 & \textbf{insertion} \\ d[i-1,j]+1 & \textbf{deletion} \\ d[i-1][j-1] & \textbf{substition if } S_1[i] = S_2[j] \\ d[i-1][j-1]+1 & \textbf{substition if } S_1[i] \neq S_2[j] \end{cases}$$

Edit Distance

- We do not need to make all operations cost 1
  - Substitution cost = 2 corresponds to an insert-delete operation
- Other distances correspond to the ease of making a spelling error or biological facts

Edit Distance

- To find an optimal sequence of edits,
  - we need to note down the way we got the distance
- Or
  - we need to reconstruct it

Edit Distance

- Edit distance of “MANHATTAN” and “MANAHATON”

		M	A	N	H	A	T	T	A	N
	0	1	2	3	4	5	6	7	8	9
M	1	0	1	2	3	4	5	6	7	8
A	2	1	0	1	2	3	4	5	6	7
N	3	2	1	0	1	2	3	4	5	6
A	4	3	2	1	1	1	2	3	4	5
H	5	4	3	2	1	2	2	3	4	5
A	6	5	4	3	2	1	2	3	3	4
T	7	6	5	4	3	2	1	2	3	4
O	8	7	6	5	4	3	2	2	3	4
N	9	8	7	6	5	4	3	3	3	**3**

Edit Distance

- Edit distance of “MANHATTAN” and “MANAHATON”
  - 3 could have come from diagonal, left, top

		M	A	N	H	A	T	T	A	N
	0	1	2	3	4	5	6	7	8	9
M	1	0	1	2	3	4	5	6	7	8
A	2	1	0	1	2	3	4	5	6	7
N	3	2	1	0	1	2	3	4	5	6
A	4	3	2	1	1	1	2	3	4	5
H	5	4	3	2	1	2	2	3	4	5
A	6	5	4	3	2	1	2	3	3	4
T	7	6	5	4	3	2	1	2	3	4
O	8	7	6	5	4	3	2	2	3	4
N	9	8	7	6	5	4	3	3	3	**3**

Edit Distance

- Edit distance of “MANHATTAN” and “MANAHATON”
  - 3 must come from diagonal: copy the last “N”

		M	A	N	H	A	T	T	A	N
	0	1	2	3	4	5	6	7	8	9
M	1	0	1	2	3	4	5	6	7	8
A	2	1	0	1	2	3	4	5	6	7
N	3	2	1	0	1	2	3	4	5	6
A	4	3	2	1	1	1	2	3	4	5
H	5	4	3	2	1	2	2	3	4	5
A	6	5	4	3	2	1	2	3	3	4
T	7	6	5	4	3	2	1	2	3	4
O	8	7	6	5	4	3	2	2	**3**	4
N	9	8	7	6	5	4	3	3	3	**3**

Edit Distance

- Edit distance of “MANHATTA” and “MANAHATO”
  - 3 choices to make

		M	A	N	H	A	T	T	A	N
	0	1	2	3	4	5	6	7	8	9
M	1	0	1	2	3	4	5	6	7	8
A	2	1	0	1	2	3	4	5	6	7
N	3	2	1	0	1	2	3	4	5	6
A	4	3	2	1	1	1	2	3	4	5
H	5	4	3	2	1	2	2	3	4	5
A	6	5	4	3	2	1	2	3	3	4
T	7	6	5	4	3	2	1	$\underline{2}$	$\underline{3}$	4
O	8	7	6	5	4	3	2	$\underline{2}$	**3**	4
N	9	8	7	6	5	4	3	3	3	**3**

Edit Distance

- Edit distance of “MANHATT” and “MANAHATO”
  - go up diagonal

		M	A	N	H	A	T	T	A	N
	0	1	2	3	4	5	6	7	8	9
M	1	0	1	2	3	4	5	6	7	8
A	2	1	0	1	2	3	4	5	6	7
N	3	2	1	0	1	2	3	4	5	6
A	4	3	2	1	1	1	2	3	4	5
H	5	4	3	2	1	2	2	3	4	5
A	6	5	4	3	2	1	2	3	3	4
T	7	6	5	4	3	2	**1**	2	3	4
O	8	7	6	5	4	3	2	**2**	**3**	4
N	9	8	7	6	5	4	3	3	3	**3**

Edit Distance

- Edit distance of “MANHATT” and “MANAHAT”
    - go up diagonal, switch “T” and “O”

		M	A	N	H	A	T	T	A	N
	0	1	2	3	4	5	6	7	8	9
M	1	0	1	2	3	4	5	6	7	8
A	2	1	0	1	2	3	4	5	6	7
N	3	2	1	0	1	2	3	4	5	6
A	4	3	2	1	1	1	2	3	4	5
H	5	4	3	2	1	2	2	3	4	5
A	6	5	4	3	2	1	2	3	3	4
T	7	6	5	4	3	2	**1**	2	3	4
O	8	7	6	5	4	3	2	**2**	**3**	4
N	9	8	7	6	5	4	3	3	3	**3**

Edit Distance

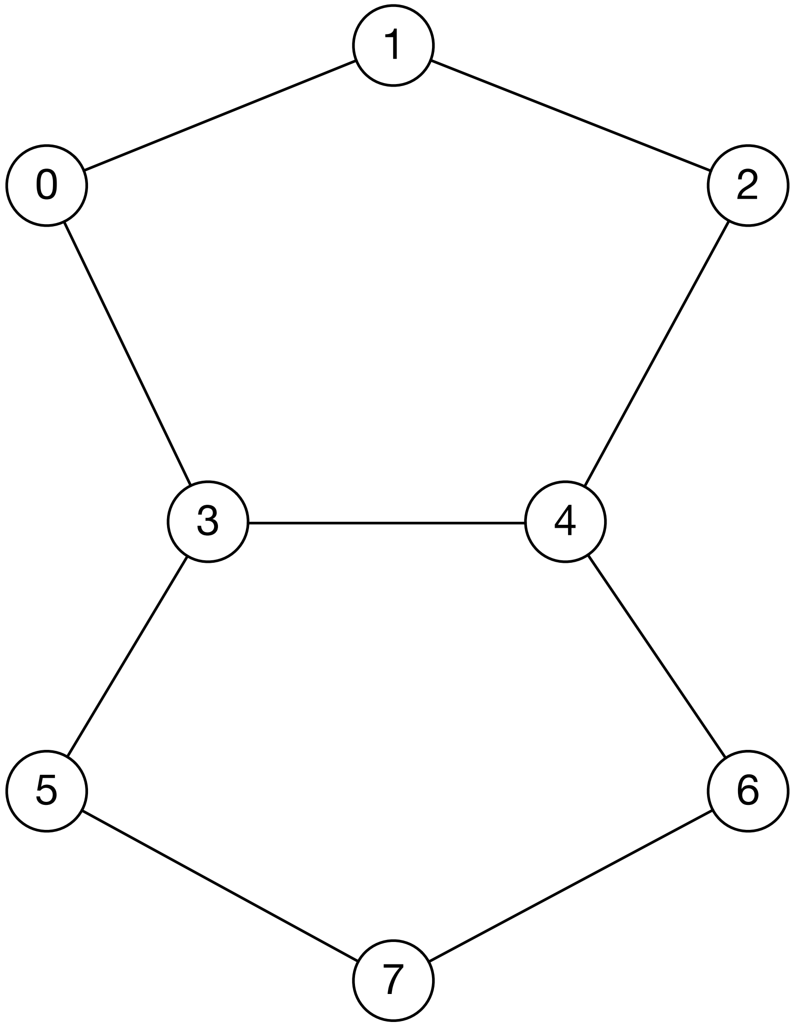
- Follow to the top: “Manhattan” to “Manahaton”
- copy “M”, copy “A”, copy “N”, add “A”, copy “H”, copy “A”, copy “T”, change “T” to “O”, add “A”, copy “N”

		M	A	N	H	A	T	T	A	N
	0	1	2	3	4	5	6	7	8	9
M	1	**0**	1	2	3	4	5	6	7	8
A	2	1	**0**	1	2	3	4	5	6	7
N	3	2	1	**0**	1	2	3	4	5	6
A	4	3	2	**1**	1	1	2	3	4	5
H	5	4	3	2	**1**	2	2	3	4	5
A	6	5	4	3	2	**1**	2	3	3	4
T	7	6	5	4	3	2	**1**	2	3	4
O	8	7	6	5	4	3	2	**2**	**3**	4
N	9	8	7	6	5	4	3	3	3	**3**

Distance in Graphs

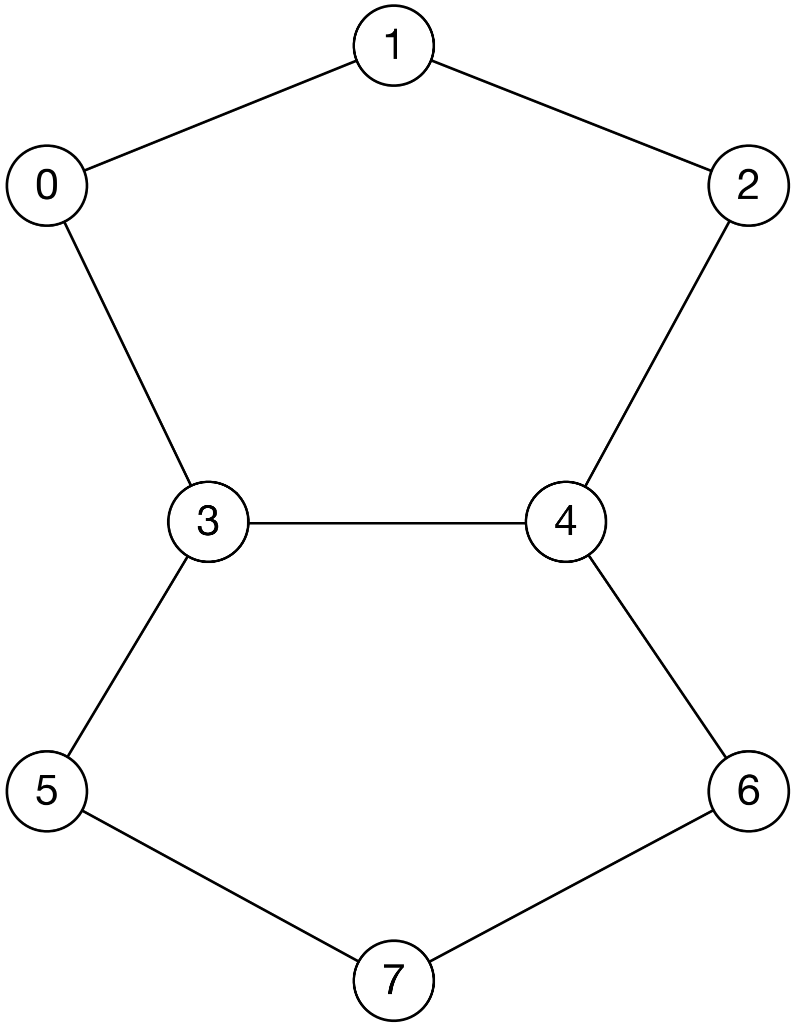
- Graphs form a very important data structure that is becoming more important
  - (e.g. graph databases, social network graphs)
- CS looks at more types of graphs then Mathematics

Representing Graphs



- A mathematical graph consists of vertices and edges
- There is at most one edge between two vertices
- CS looks at multi-edges, edges with weights, edges with direction, etc.

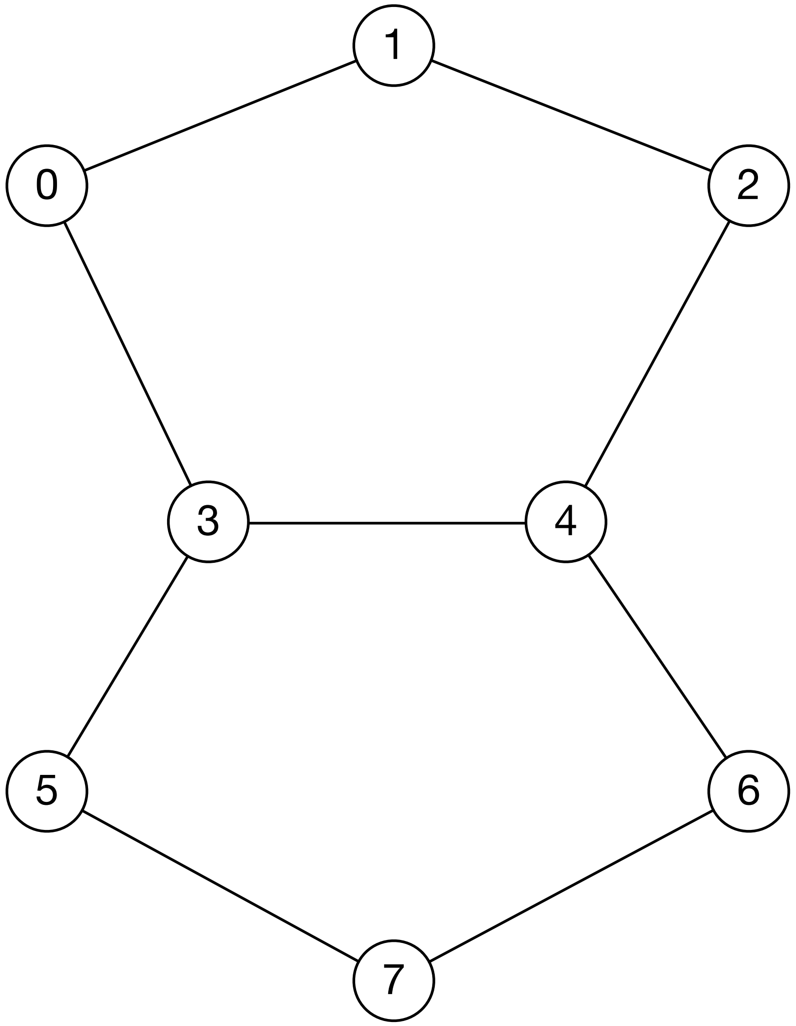
Representing Graph



- Adjacency List:
  - List all the vertices and the vertices that they are connected to.

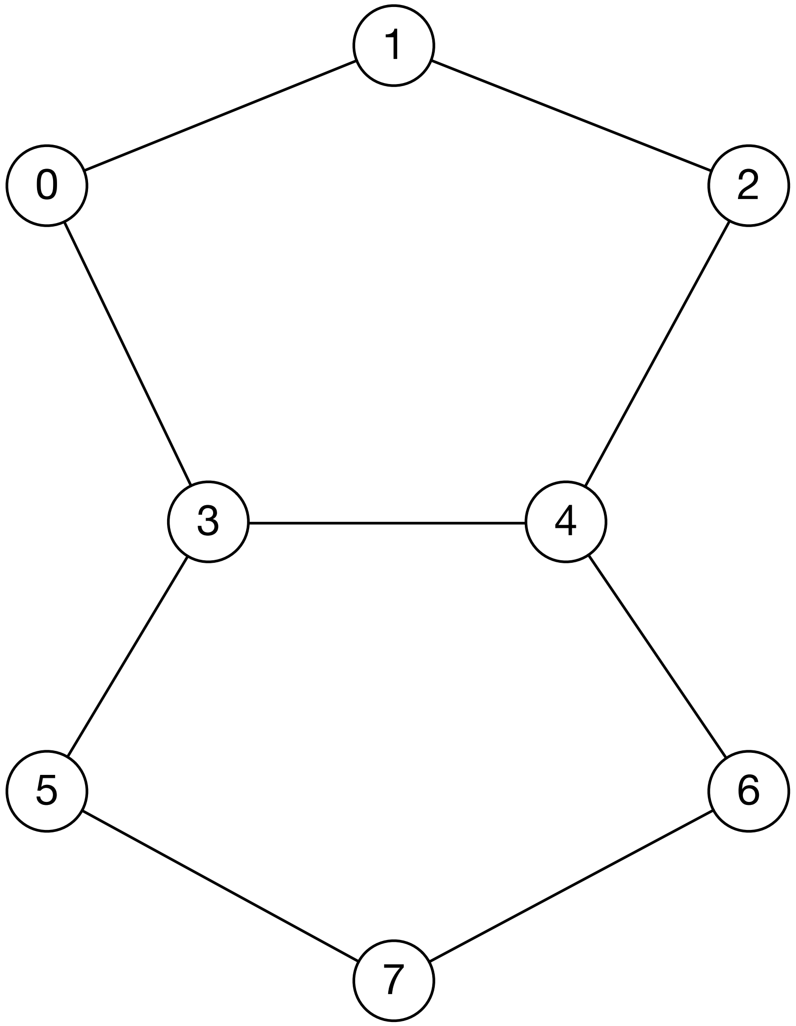
```
{0:[1,3], 1:[0,2],
 2:[1,4], 3:[0,4,5],
 4:[2,3,6], 5:[3,7],
 6:[4,7], 7:[5,6]}
```

Representing Graph



- Adjacency matrix for an undirected graph
  - Row and columns are indexed by the vertices
  - A 1 entry says that the two vertices are connected
  - $$a_{i,j} = \begin{cases} 1 & \textbf{if } (i,j) \in E \\ 0 & \textbf{if}(i,j) \notin E \end{cases}$$

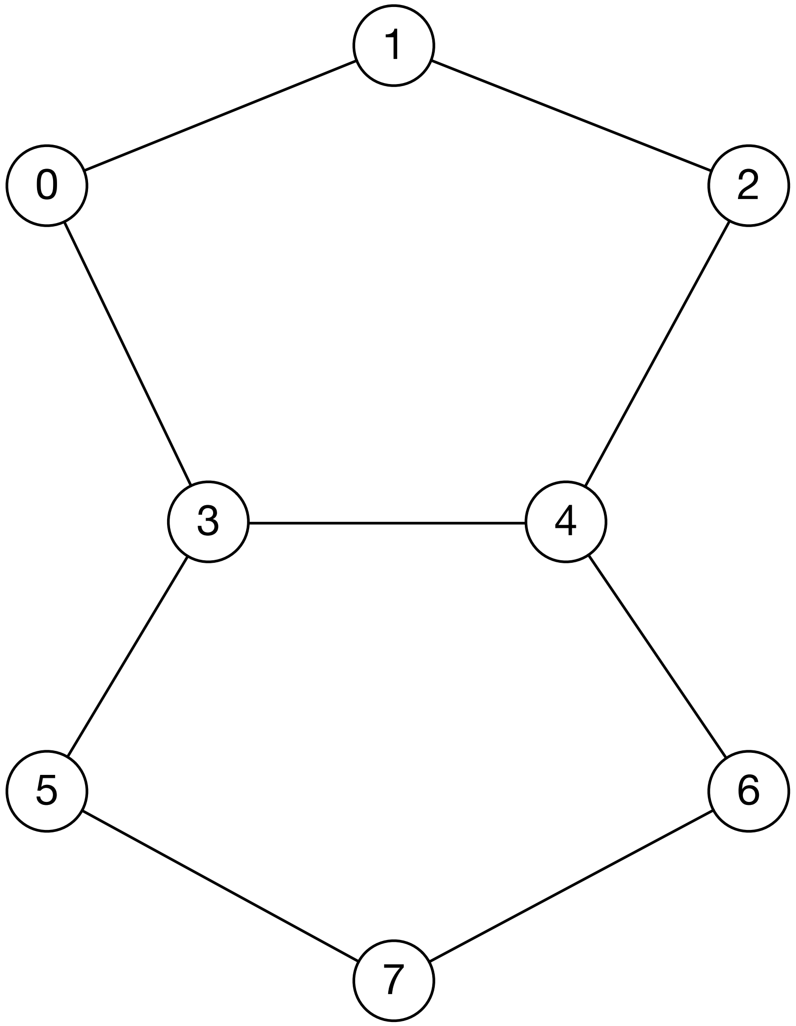
Representing Graph



- Adjacency Matrix

$$\begin{pmatrix}
0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 0
\end{pmatrix}$$

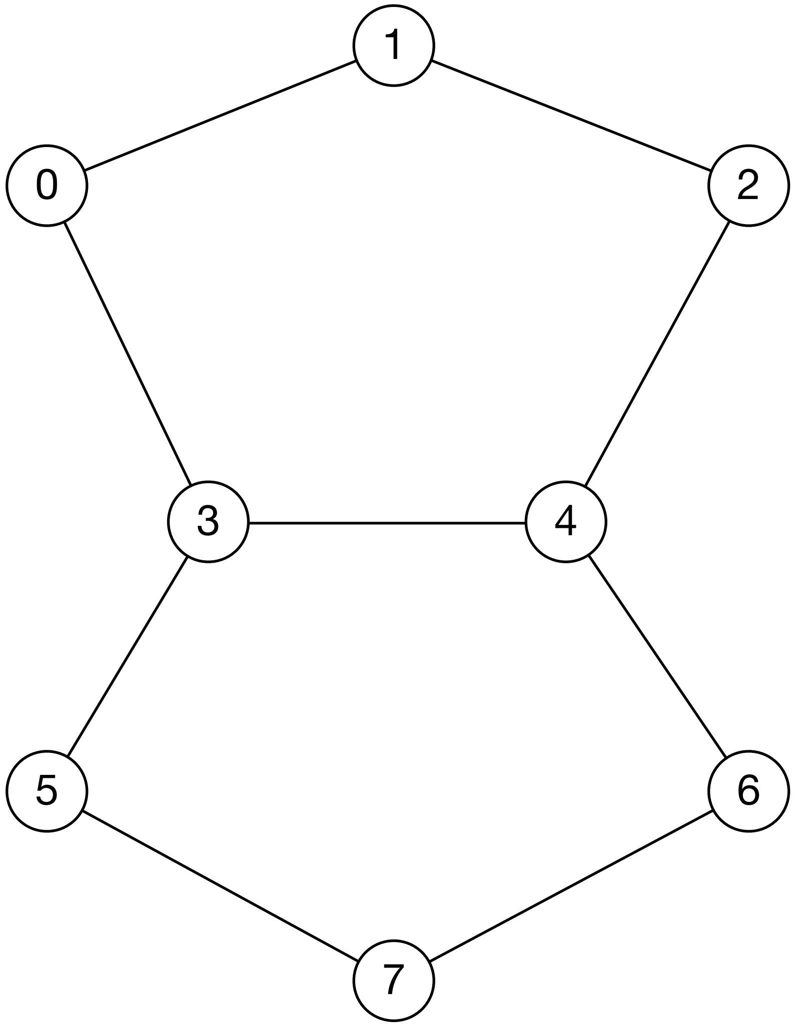
Representing Graph



- List of edges:

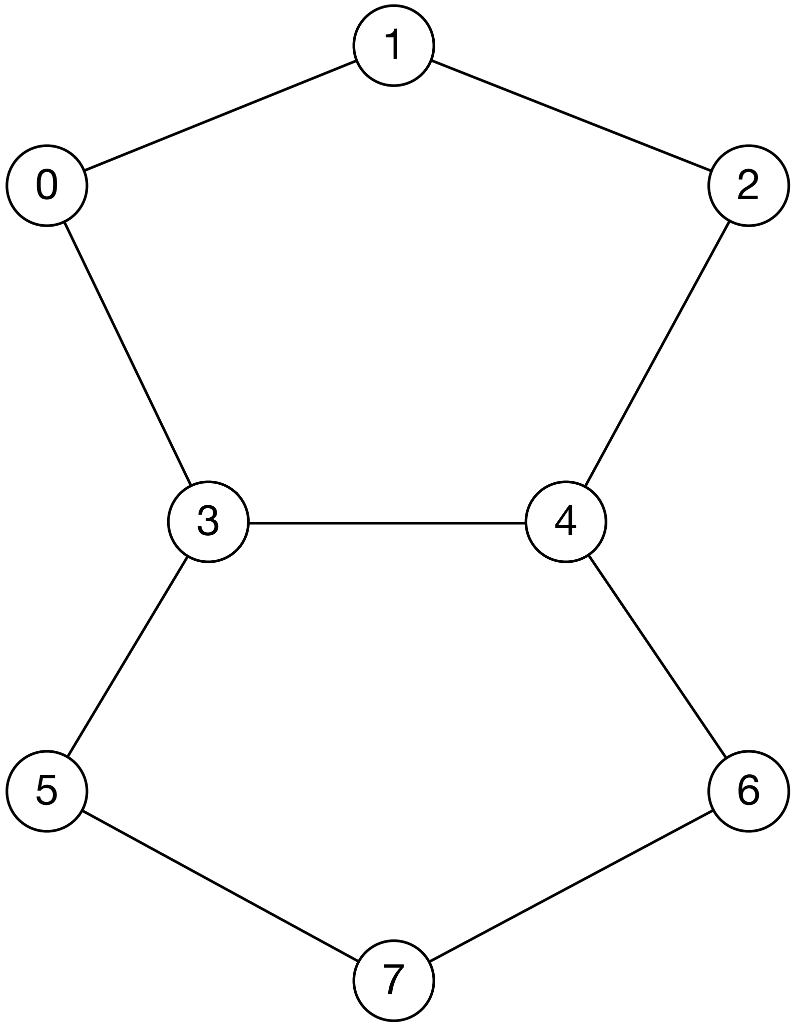
```
[(0,1), (1,2), (2,4), (3,4),
 (0,3), (4,6), (6,7), (3,5),
 (5,7)]
```

Representing Graph



- More complicated data structures:
  - A dictionary that associates to each edge its two adjacent vertices

Representing Graph



- More complicated data structures:
  - A dictionary that associates to each edge its two adjacent vertices
  - A dictionary that associates to each vertex its edges
- Used in order to make algos run faster

Representing Weighted Graphs

- Weighted graphs have weights on their edges
    - Representing distances in a road network
    - Delay in a computer network
    - Capacity in a computer network
    - …

Representing Weighted Graphs

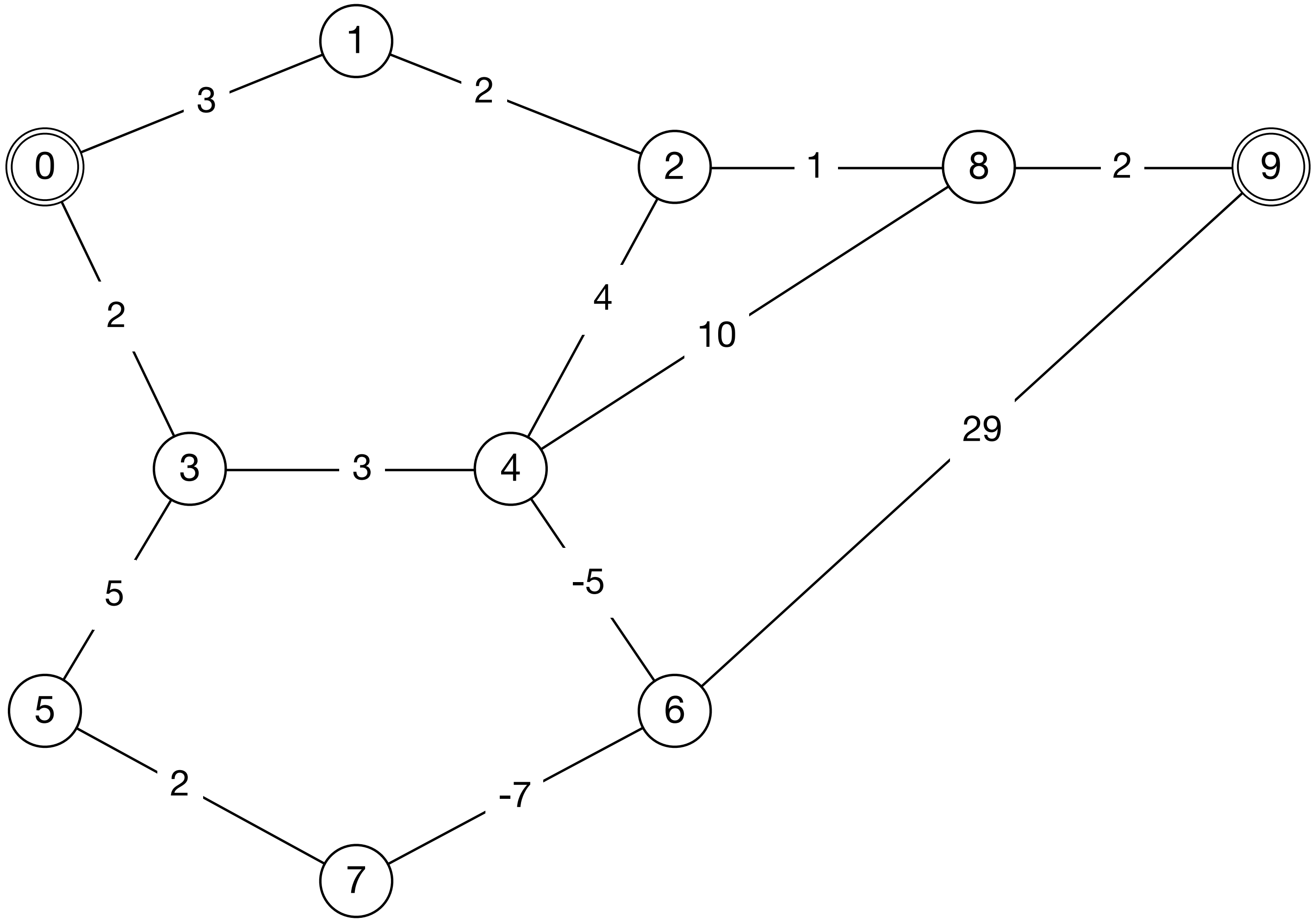
- Can use
  - list of edges adorned with weight
  - adjacency matrix where 0 means no edge and a non-zero number means an edge with a weight of that number
  - adjacency list giving destination edge and weight
  - …

Shortest Path Problems

- Given a graph $G = (V, E)$ and a source $v_0 \in V$
  - Find the shortest paths from v_0 to all other vertices
  - Find the shortest path from v_0 to a specific vertex
  - Find shortest past from all vertices to all other vertices
    - (E.g. finding a routing table in a network)

Shortest Path Problem

- We assume that all weights are positive
    - Otherwise, you might have a cycle where you accumulate a total weight that is negative, so you want to travel it as much as possible, which is infinitely often (or until Mom calls you for dinner)



Shortest Path Problem

- Optimal substructure of a shortest path problem
  - Should be another shortest path problem
    - Needed for dynamic programming structure

Shortest Path Problem

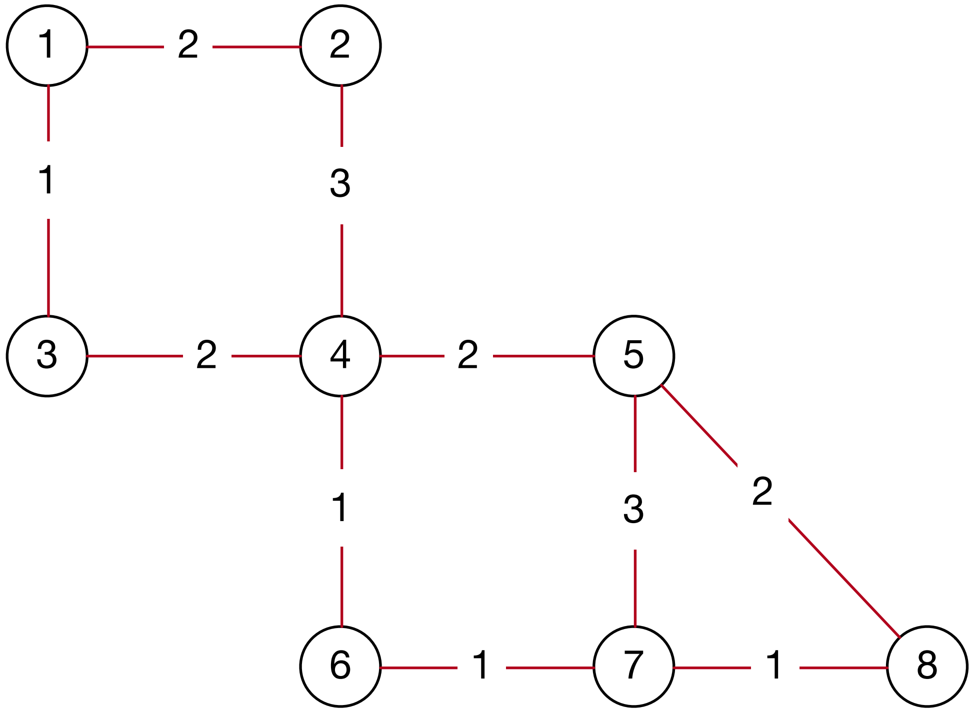
- Theorem: Let $u \rightsquigarrow w_0 \rightsquigarrow w_1 \rightsquigarrow v$ be a shortest path between $u, v \in V$. Then the subpath $w_0 \rightsquigarrow w_1$ is a shortest path between $w_0, w_1 \in V$.

Floyd Warshal Algorithm

- Solves the find all to all shortest path problem
  - Subproblems: Find the shortest path using vertices in a restricted set

Floyd Warshal Algorithm

- Floyd Warshal base case:
  - Only allow direct paths
    - 0 for itself, weight if edge exists, infinity otherwise



$$\begin{pmatrix} 0 & 2 & 1 & \infty & \infty & \infty & \infty & \infty \\ 2 & 0 & \infty & 3 & \infty & \infty & \infty & \infty \\ 1 & \infty & 0 & 2 & \infty & \infty & \infty & \infty \\ \infty & 3 & 2 & 0 & 2 & 1 & \infty & \infty \\ \infty & \infty & \infty & 2 & 0 & \infty & 3 & 2 \\ \infty & \infty & \infty & 1 & \infty & 0 & 1 & \infty \\ \infty & \infty & \infty & \infty & 3 & 1 & 0 & 1 \\ \infty & \infty & \infty & \infty & 2 & \infty & 1 & 0 \end{pmatrix}$$

Floyd Warshal Algorithm

- Recursive steps.
    - Allow intermediate vertices $\{1,2,\ldots,k-1\}$
    - Now allow additionally vertex *k* as intermediate.
    - Shortest path involving 1, …, *k* as intermediates = Shortest path involving 1, …, *k*-1 or a path that goes through *k*

Floyd Warshal Algorithm

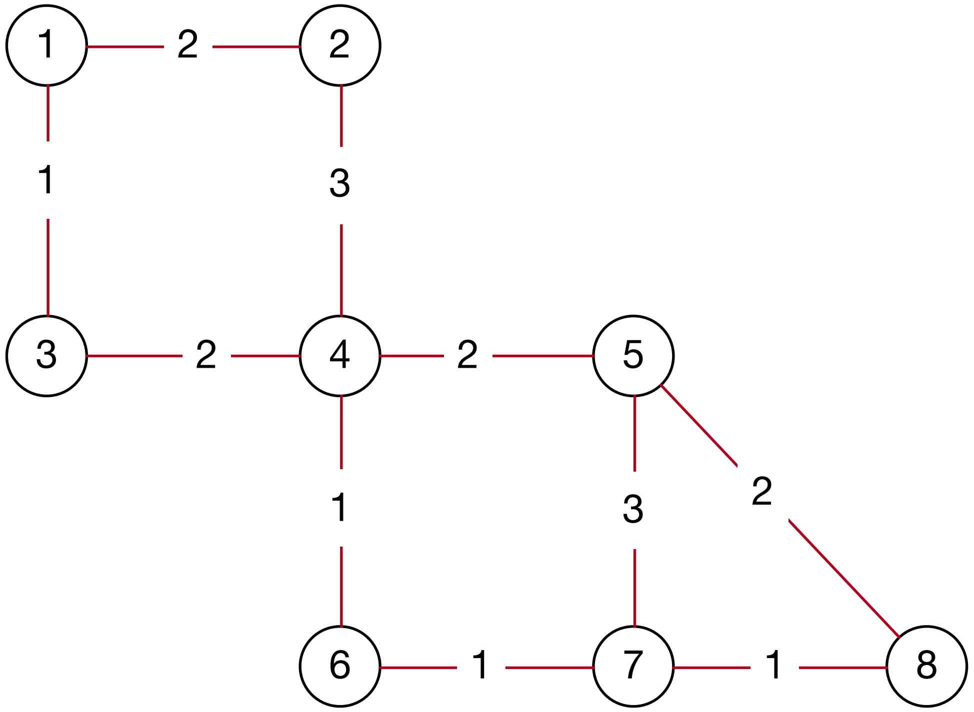
- Recursive steps
  - $D(i, j, l)$ distance between *i* and *j* involving vertices as intermediaries $\{1, 2, \ldots, l\}$

$$D(i, j, k) = \min(D(i, j, k-1), D(i, k, k-1) + D(k, j, k-1))$$

  - Shortest path between *i* and *j* avoids *k*
  - Shortest path between *i* and *j* passes through *k*

Floyd Warshal Algorithm

- Example continued

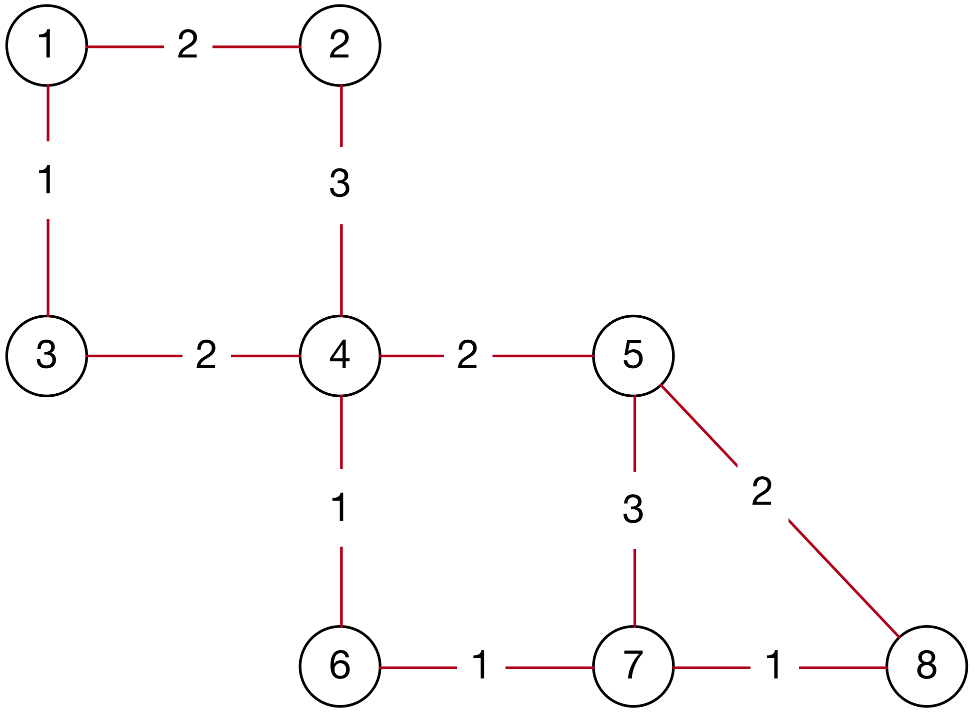


$$D_0 = \begin{pmatrix} 0 & 2 & 1 & \infty & \infty & \infty & \infty & \infty \\ 2 & 0 & \infty & 3 & \infty & \infty & \infty & \infty \\ 1 & \infty & 0 & 2 & \infty & \infty & \infty & \infty \\ \infty & 3 & 2 & 0 & 2 & 1 & \infty & \infty \\ \infty & \infty & \infty & 2 & 0 & \infty & 3 & 2 \\ \infty & \infty & \infty & 1 & \infty & 0 & 1 & \infty \\ \infty & \infty & \infty & \infty & 3 & 1 & 0 & 1 \\ \infty & \infty & \infty & \infty & 2 & \infty & 1 & 0 \end{pmatrix}$$

$$D_1[1,2] = \min(D_0[1,2], D_0[1,1] + D_0[1,2]) = 2$$

Floyd Warshal Algorithm

- Example continued

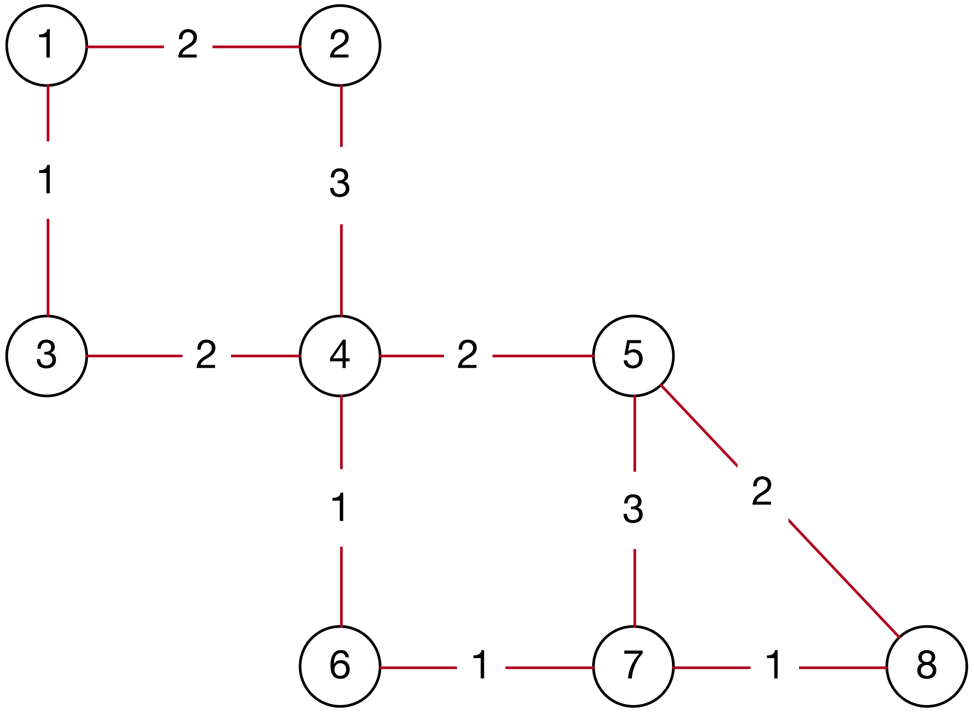


$$D_0 = \begin{pmatrix} 0 & 2 & 1 & \infty & \infty & \infty & \infty & \infty \\ 2 & 0 & \infty & 3 & \infty & \infty & \infty & \infty \\ 1 & \infty & 0 & 2 & \infty & \infty & \infty & \infty \\ \infty & 3 & 2 & 0 & 2 & 1 & \infty & \infty \\ \infty & \infty & \infty & 2 & 0 & \infty & 3 & 2 \\ \infty & \infty & \infty & 1 & \infty & 0 & 1 & \infty \\ \infty & \infty & \infty & \infty & 3 & 1 & 0 & 1 \\ \infty & \infty & \infty & \infty & 2 & \infty & 1 & 0 \end{pmatrix}$$

$$D_1[1,3] = \min(D_0[1,3], D_0[1,1] + D_0[1,3]) = 1$$

Floyd Warshal Algorithm

- Example continued

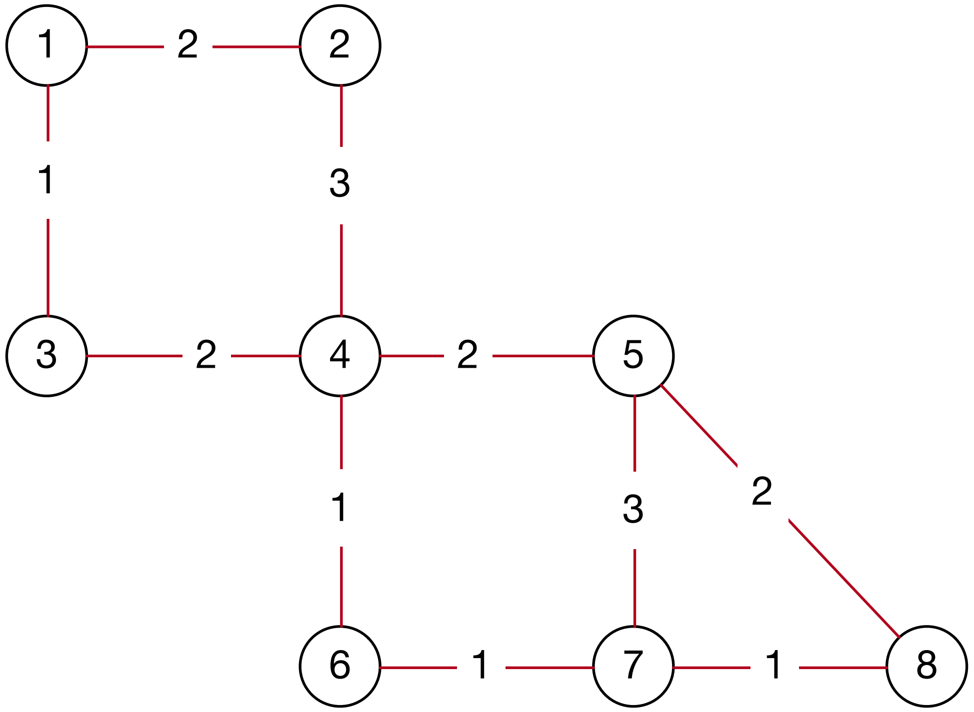


$$D_0 = \begin{pmatrix} 0 & 2 & 1 & \infty & \infty & \infty & \infty & \infty \\ 2 & 0 & \infty & 3 & \infty & \infty & \infty & \infty \\ 1 & \infty & 0 & 2 & \infty & \infty & \infty & \infty \\ \infty & 3 & 2 & 0 & 2 & 1 & \infty & \infty \\ \infty & \infty & \infty & 2 & 0 & \infty & 3 & 2 \\ \infty & \infty & \infty & 1 & \infty & 0 & 1 & \infty \\ \infty & \infty & \infty & \infty & 3 & 1 & 0 & 1 \\ \infty & \infty & \infty & \infty & 2 & \infty & 1 & 0 \end{pmatrix}$$

$$D_1[1,4] = \min(D_0[1,4], D_0[1,1] + D_0[1,4]) = \infty$$

Floyd Warshal Algorithm

- Example continued

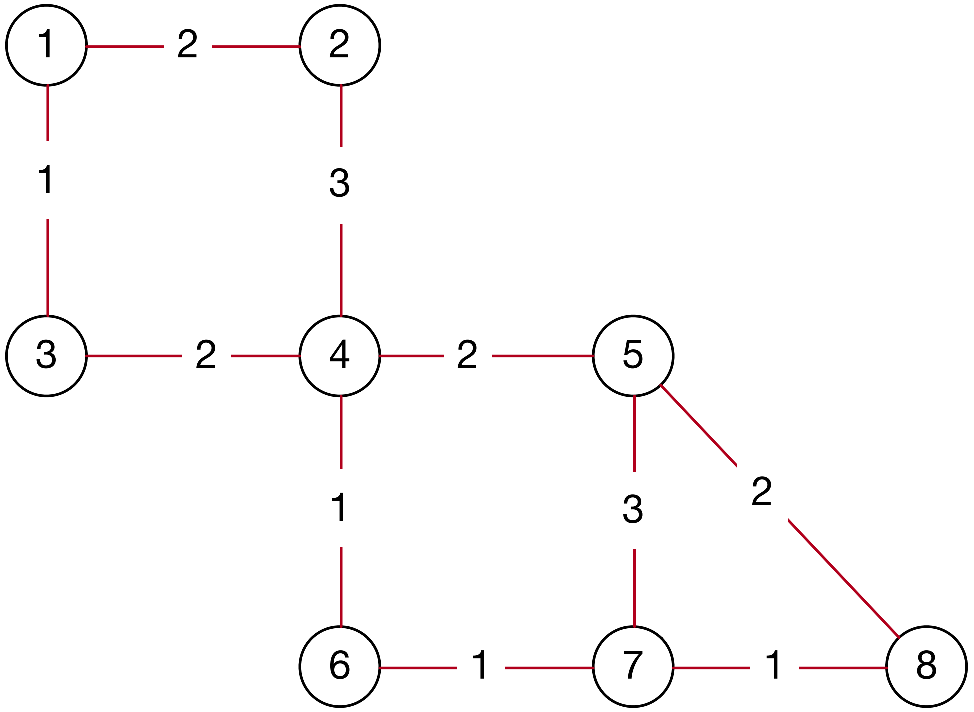


$$D_0 = \begin{pmatrix} 0 & 2 & 1 & \infty & \infty & \infty & \infty & \infty \\ 2 & 0 & \infty & 3 & \infty & \infty & \infty & \infty \\ 1 & \infty & 0 & 2 & \infty & \infty & \infty & \infty \\ \infty & 3 & 2 & 0 & 2 & 1 & \infty & \infty \\ \infty & \infty & \infty & 2 & 0 & \infty & 3 & 2 \\ \infty & \infty & \infty & 1 & \infty & 0 & 1 & \infty \\ \infty & \infty & \infty & \infty & 3 & 1 & 0 & 1 \\ \infty & \infty & \infty & \infty & 2 & \infty & 1 & 0 \end{pmatrix}$$

$$D_1[1,5] = \min(D_0[1,5], D_0[1,1] + D_0[1,5]) = \infty$$

Floyd Warshal Algorithm

- Example continued

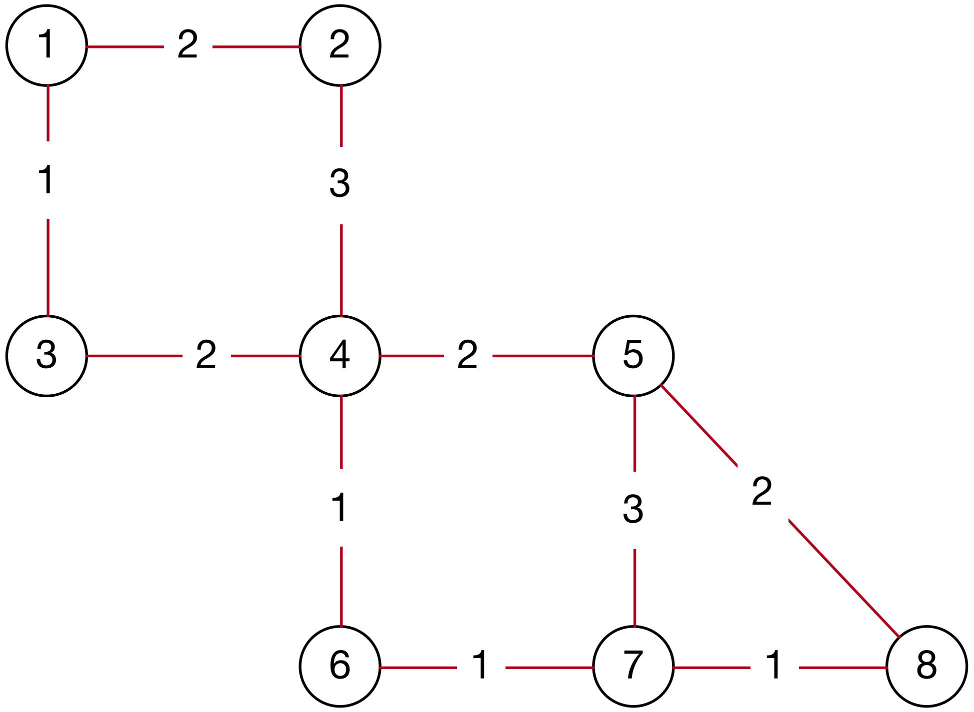


$$D_0 = \begin{pmatrix} 0 & 2 & 1 & \infty & \infty & \infty & \infty & \infty \\ 2 & 0 & \infty & 3 & \infty & \infty & \infty & \infty \\ 1 & \infty & 0 & 2 & \infty & \infty & \infty & \infty \\ \infty & 3 & 2 & 0 & 2 & 1 & \infty & \infty \\ \infty & \infty & \infty & 2 & 0 & \infty & 3 & 2 \\ \infty & \infty & \infty & 1 & \infty & 0 & 1 & \infty \\ \infty & \infty & \infty & \infty & 3 & 1 & 0 & 1 \\ \infty & \infty & \infty & \infty & 2 & \infty & 1 & 0 \end{pmatrix}$$

$$D_1[1,6] = \min(D_0[1,6], D_0[1,1] + D_0[1,6]) = \infty$$

Floyd Warshal Algorithm

- Example continued

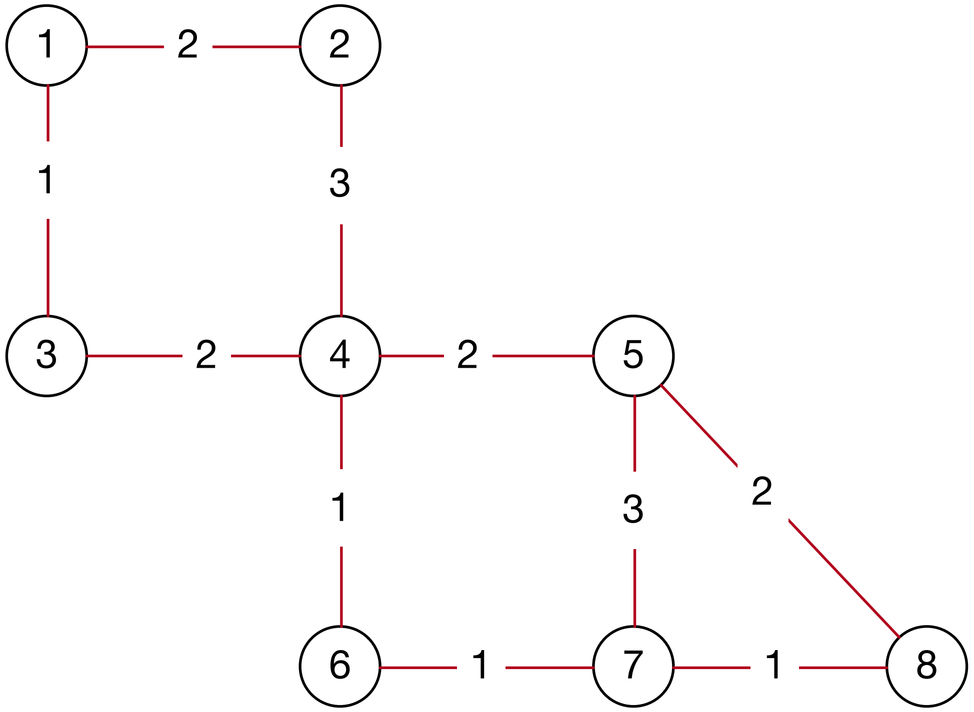


$$D_0 = \begin{pmatrix} 0 & 2 & 1 & \infty & \infty & \infty & \infty & \infty \\ 2 & 0 & \infty & 3 & \infty & \infty & \infty & \infty \\ 1 & \infty & 0 & 2 & \infty & \infty & \infty & \infty \\ \infty & 3 & 2 & 0 & 2 & 1 & \infty & \infty \\ \infty & \infty & \infty & 2 & 0 & \infty & 3 & 2 \\ \infty & \infty & \infty & 1 & \infty & 0 & 1 & \infty \\ \infty & \infty & \infty & \infty & 3 & 1 & 0 & 1 \\ \infty & \infty & \infty & \infty & 2 & \infty & 1 & 0 \end{pmatrix}$$

$$D_1[1,7] = \min(D_0[1,7], D_0[1,1] + D_0[1,7]) = \infty$$

Floyd Warshal Algorithm

- Example continued

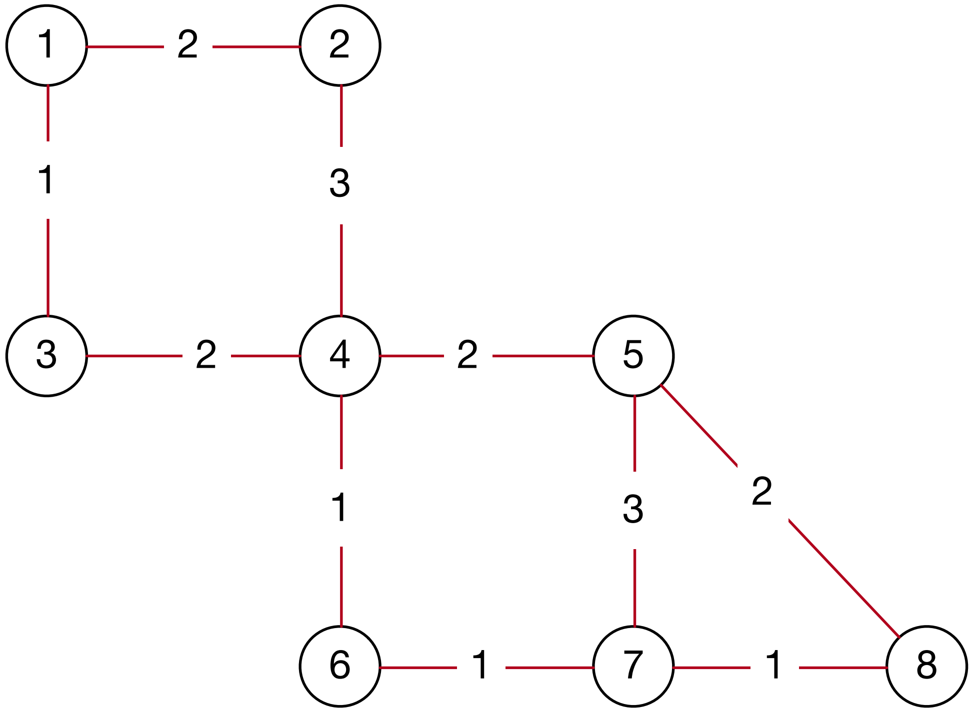


$$D_0 = \begin{pmatrix} 0 & 2 & 1 & \infty & \infty & \infty & \infty & \infty \\ 2 & 0 & \infty & 3 & \infty & \infty & \infty & \infty \\ 1 & \infty & 0 & 2 & \infty & \infty & \infty & \infty \\ \infty & 3 & 2 & 0 & 2 & 1 & \infty & \infty \\ \infty & \infty & \infty & 2 & 0 & \infty & 3 & 2 \\ \infty & \infty & \infty & 1 & \infty & 0 & 1 & \infty \\ \infty & \infty & \infty & \infty & 3 & 1 & 0 & 1 \\ \infty & \infty & \infty & \infty & 2 & \infty & 1 & 0 \end{pmatrix}$$

$$D_1[2,1] = \min(D_0[2,1], D_0[2,1] + D_0[1,1]) = 2$$

Floyd Warshal Algorithm

- Example continued

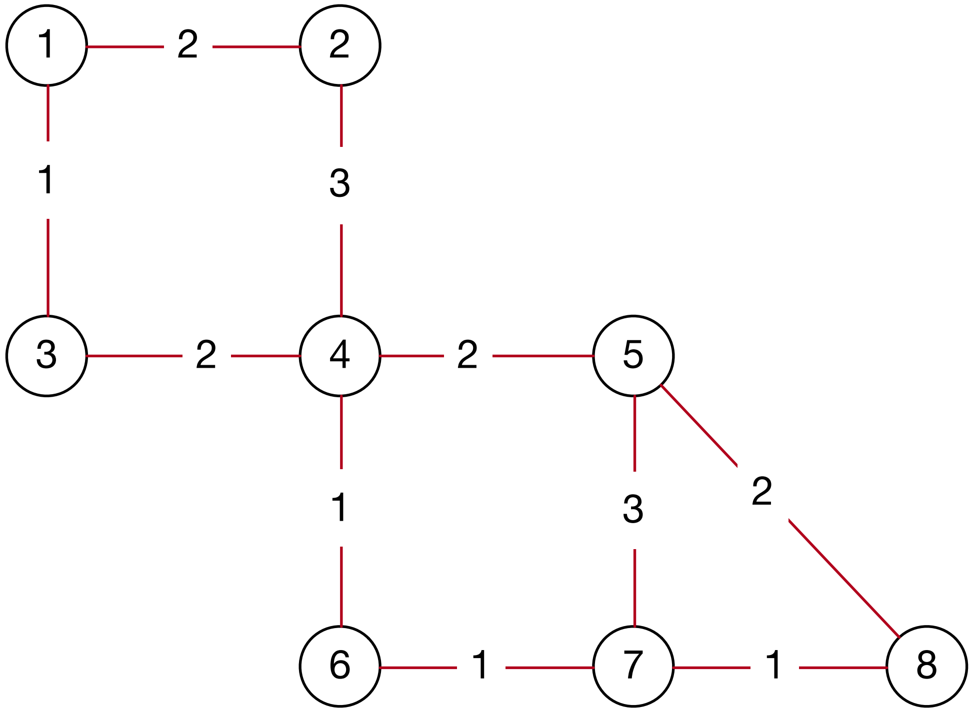


$$D_0 = \begin{pmatrix} 0 & 2 & 1 & \infty & \infty & \infty & \infty & \infty \\ 2 & 0 & \infty & 3 & \infty & \infty & \infty & \infty \\ 1 & \infty & 0 & 2 & \infty & \infty & \infty & \infty \\ \infty & 3 & 2 & 0 & 2 & 1 & \infty & \infty \\ \infty & \infty & \infty & 2 & 0 & \infty & 3 & 2 \\ \infty & \infty & \infty & 1 & \infty & 0 & 1 & \infty \\ \infty & \infty & \infty & \infty & 3 & 1 & 0 & 1 \\ \infty & \infty & \infty & \infty & 2 & \infty & 1 & 0 \end{pmatrix}$$

$$D_1[2,2] = 0$$

Floyd Warshal Algorithm

- Example continued

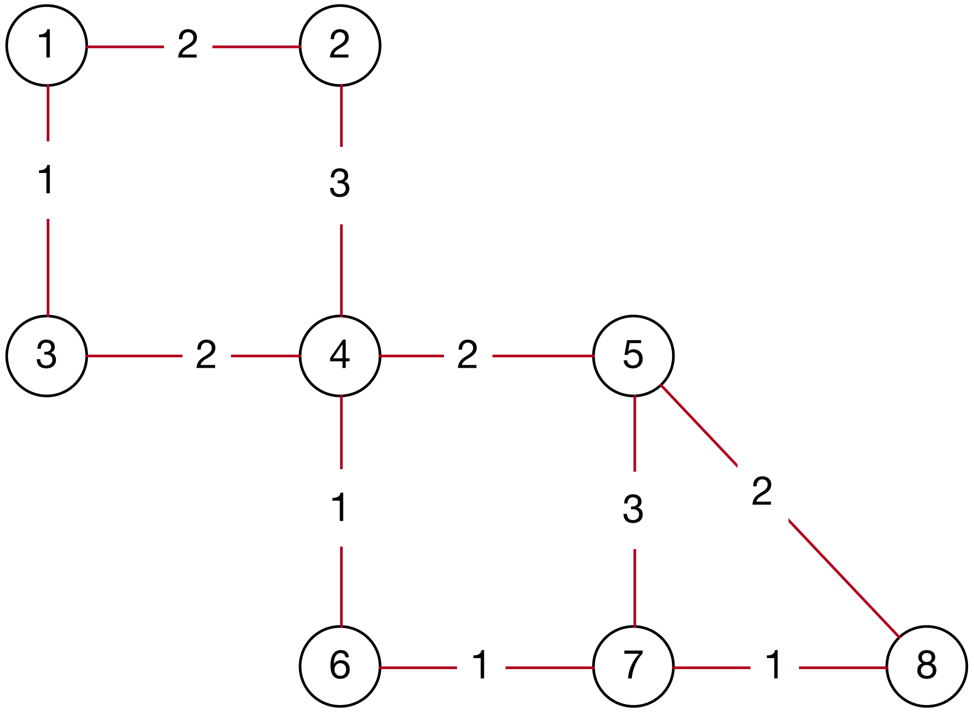


$$D_0 = \begin{pmatrix} 0 & 2 & 1 & \infty & \infty & \infty & \infty & \infty \\ 2 & 0 & \infty & 3 & \infty & \infty & \infty & \infty \\ 1 & \infty & 0 & 2 & \infty & \infty & \infty & \infty \\ \infty & 3 & 2 & 0 & 2 & 1 & \infty & \infty \\ \infty & \infty & \infty & 2 & 0 & \infty & 3 & 2 \\ \infty & \infty & \infty & 1 & \infty & 0 & 1 & \infty \\ \infty & \infty & \infty & \infty & 3 & 1 & 0 & 1 \\ \infty & \infty & \infty & \infty & 2 & \infty & 1 & 0 \end{pmatrix}$$

$$D_1[2,3] = \min(D_0[2,3], D_0[2,1] + D_0[1,3]) = \min(\infty,3) = 3$$

Floyd Warshal Algorithm

- Example continued

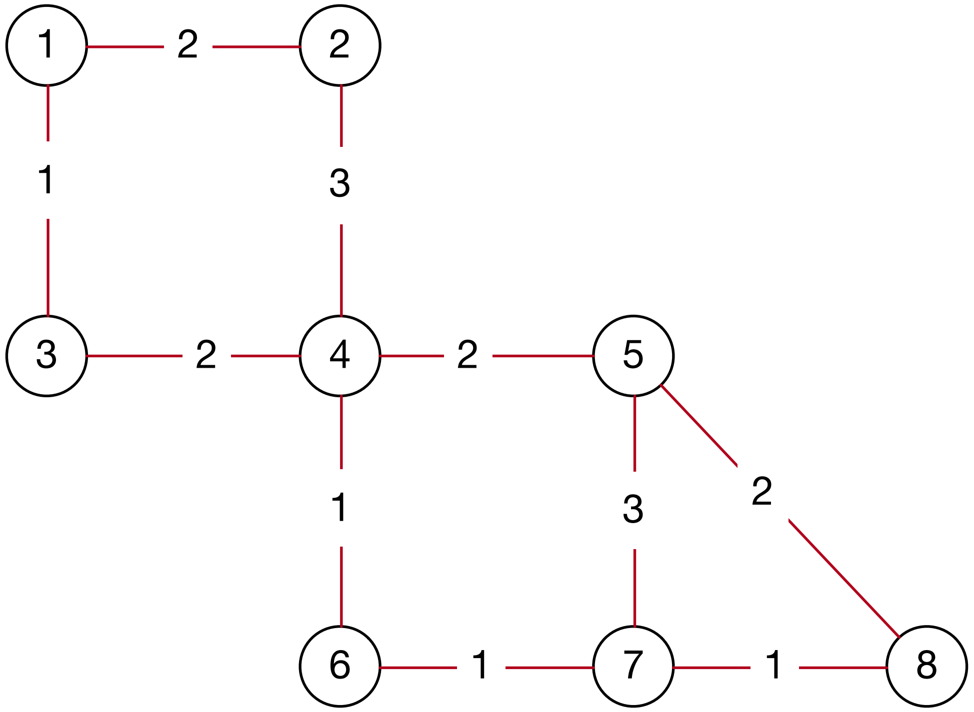


$$D_0 = \begin{pmatrix} 0 & 2 & 1 & \infty & \infty & \infty & \infty & \infty \\ 2 & 0 & \infty & 3 & \infty & \infty & \infty & \infty \\ 1 & \infty & 0 & 2 & \infty & \infty & \infty & \infty \\ \infty & 3 & 2 & 0 & 2 & 1 & \infty & \infty \\ \infty & \infty & \infty & 2 & 0 & \infty & 3 & 2 \\ \infty & \infty & \infty & 1 & \infty & 0 & 1 & \infty \\ \infty & \infty & \infty & \infty & 3 & 1 & 0 & 1 \\ \infty & \infty & \infty & \infty & 2 & \infty & 1 & 0 \end{pmatrix}$$

$$D_1[2,4] = \min(D_0[2,4], D_0[2,1] + D_0[1,4]) = 3$$

Floyd Warshal Algorithm

- Example continued

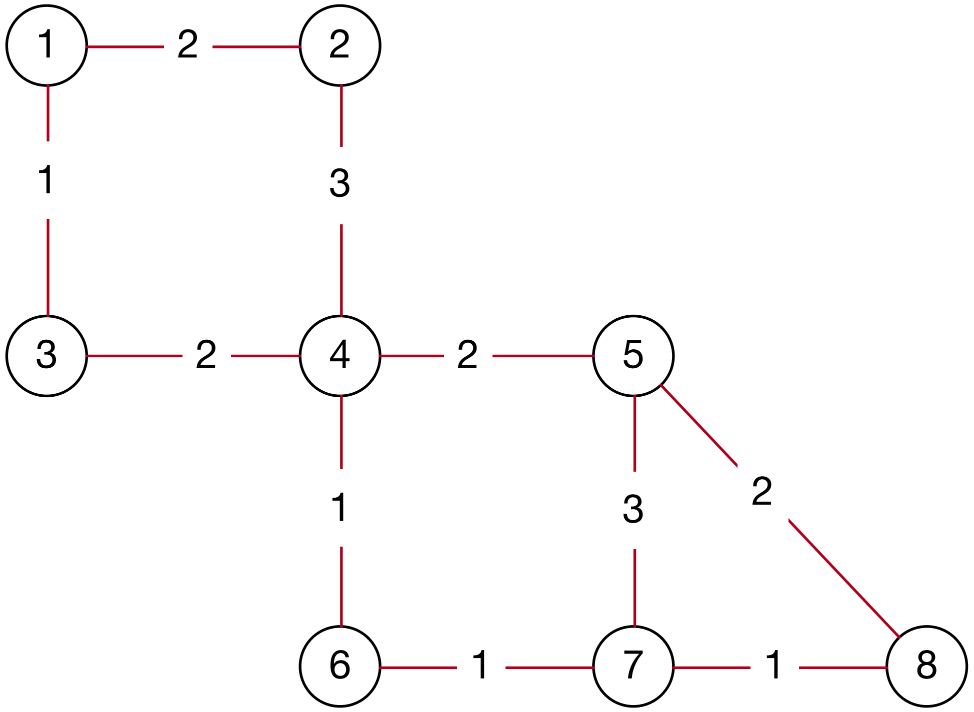


$$D_0 = \begin{pmatrix} 0 & 2 & 1 & \infty & \infty & \infty & \infty & \infty \\ 2 & 0 & \infty & 3 & \infty & \infty & \infty & \infty \\ 1 & \infty & 0 & 2 & \infty & \infty & \infty & \infty \\ \infty & 3 & 2 & 0 & 2 & 1 & \infty & \infty \\ \infty & \infty & \infty & 2 & 0 & \infty & 3 & 2 \\ \infty & \infty & \infty & 1 & \infty & 0 & 1 & \infty \\ \infty & \infty & \infty & \infty & 3 & 1 & 0 & 1 \\ \infty & \infty & \infty & \infty & 2 & \infty & 1 & 0 \end{pmatrix}$$

$$D_1[2,5] = \min(D_0[2,5], D_0[2,1] + D_0[1,5]) = \infty$$

Floyd Warshal Algorithm

- Example continued

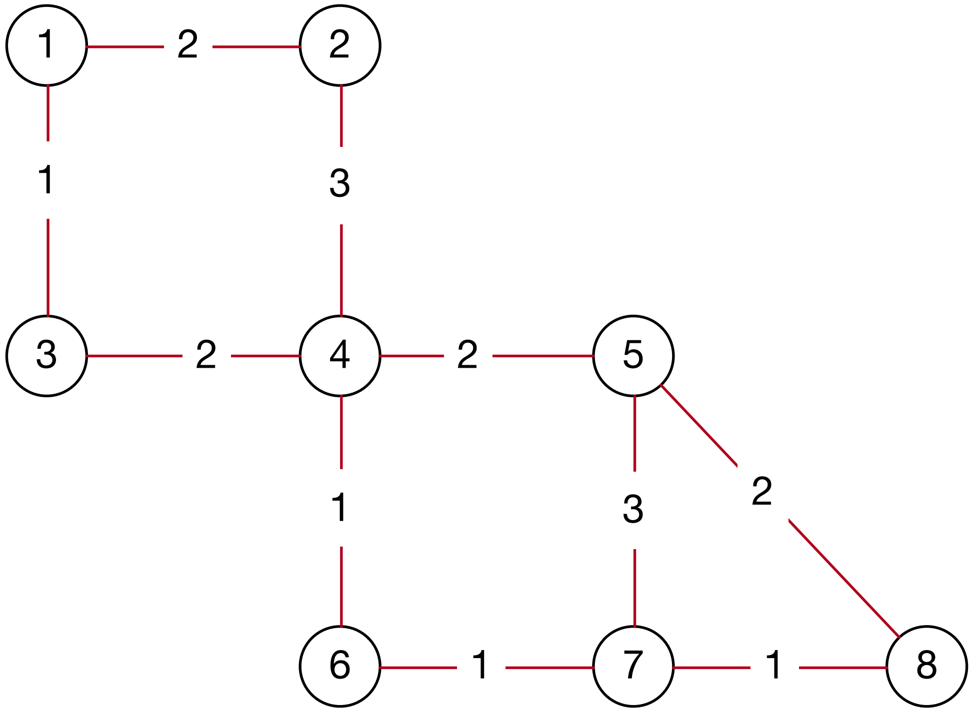


$$D_0 = \begin{pmatrix} 0 & 2 & 1 & \infty & \infty & \infty & \infty & \infty \\ 2 & 0 & \infty & 3 & \infty & \infty & \infty & \infty \\ 1 & \infty & 0 & 2 & \infty & \infty & \infty & \infty \\ \infty & 3 & 2 & 0 & 2 & 1 & \infty & \infty \\ \infty & \infty & \infty & 2 & 0 & \infty & 3 & 2 \\ \infty & \infty & \infty & 1 & \infty & 0 & 1 & \infty \\ \infty & \infty & \infty & \infty & 3 & 1 & 0 & 1 \\ \infty & \infty & \infty & \infty & 2 & \infty & 1 & 0 \end{pmatrix}$$

$$D_1[2,5] = \min(D_0[2,5], D_0[2,1] + D_0[1,5]) = \infty$$

Floyd Warshal Algorithm

- Example continued

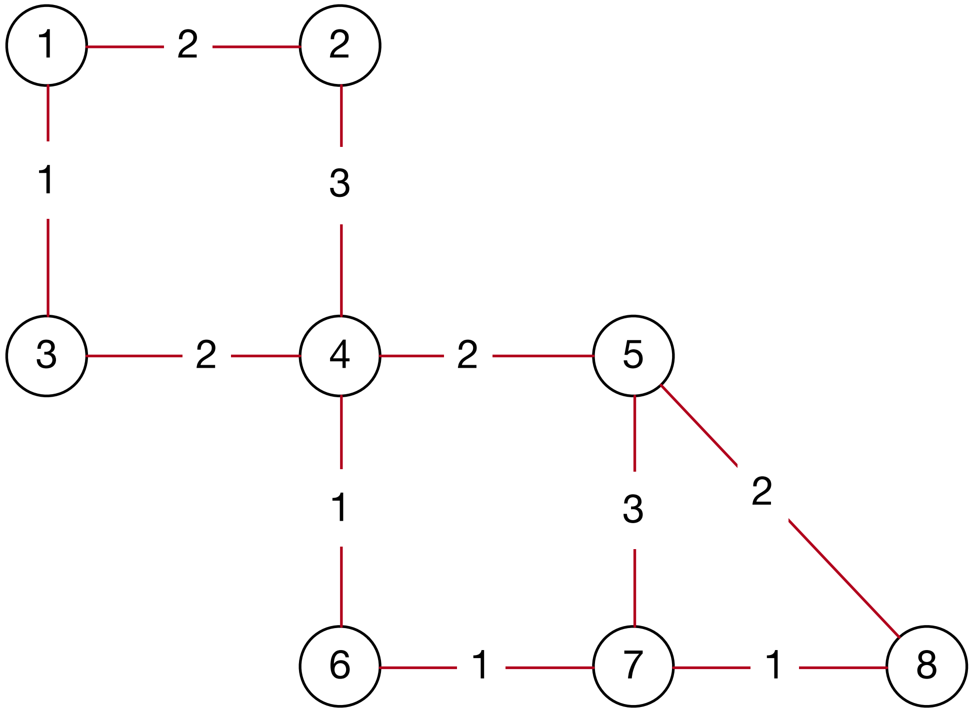


$$D_0 = \begin{pmatrix} 0 & 2 & 1 & \infty & \infty & \infty & \infty & \infty \\ 2 & 0 & \infty & 3 & \infty & \infty & \infty & \infty \\ 1 & \infty & 0 & 2 & \infty & \infty & \infty & \infty \\ \infty & 3 & 2 & 0 & 2 & 1 & \infty & \infty \\ \infty & \infty & \infty & 2 & 0 & \infty & 3 & 2 \\ \infty & \infty & \infty & 1 & \infty & 0 & 1 & \infty \\ \infty & \infty & \infty & \infty & 3 & 1 & 0 & 1 \\ \infty & \infty & \infty & \infty & 2 & \infty & 1 & 0 \end{pmatrix}$$

$$D_1[2,6] = \min(D_0[2,6], D_0[2,1] + D_0[1,6]) = \infty$$

Floyd Warshal Algorithm

- Example continued

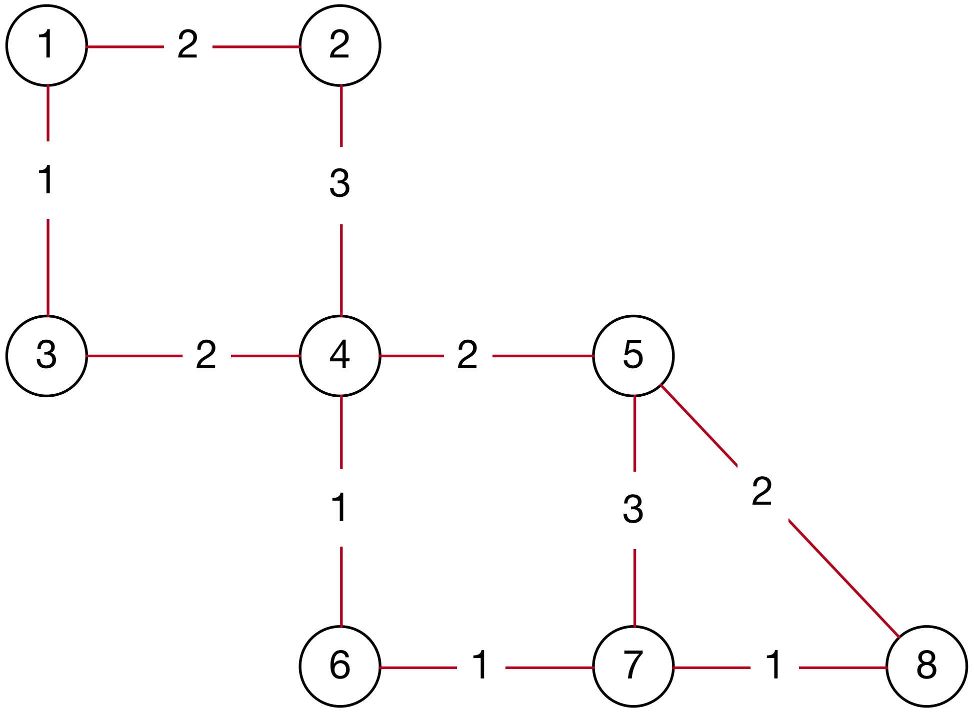


$$D_0 = \begin{pmatrix} 0 & 2 & 1 & \infty & \infty & \infty & \infty & \infty \\ 2 & 0 & \infty & 3 & \infty & \infty & \infty & \infty \\ 1 & \infty & 0 & 2 & \infty & \infty & \infty & \infty \\ \infty & 3 & 2 & 0 & 2 & 1 & \infty & \infty \\ \infty & \infty & \infty & 2 & 0 & \infty & 3 & 2 \\ \infty & \infty & \infty & 1 & \infty & 0 & 1 & \infty \\ \infty & \infty & \infty & \infty & 3 & 1 & 0 & 1 \\ \infty & \infty & \infty & \infty & 2 & \infty & 1 & 0 \end{pmatrix}$$

$$D_1[2,7] = \min(D_0[2,7], D_0[2,1] + D_0[1,7]) = \infty$$

Floyd Warshal Algorithm

- Example continued

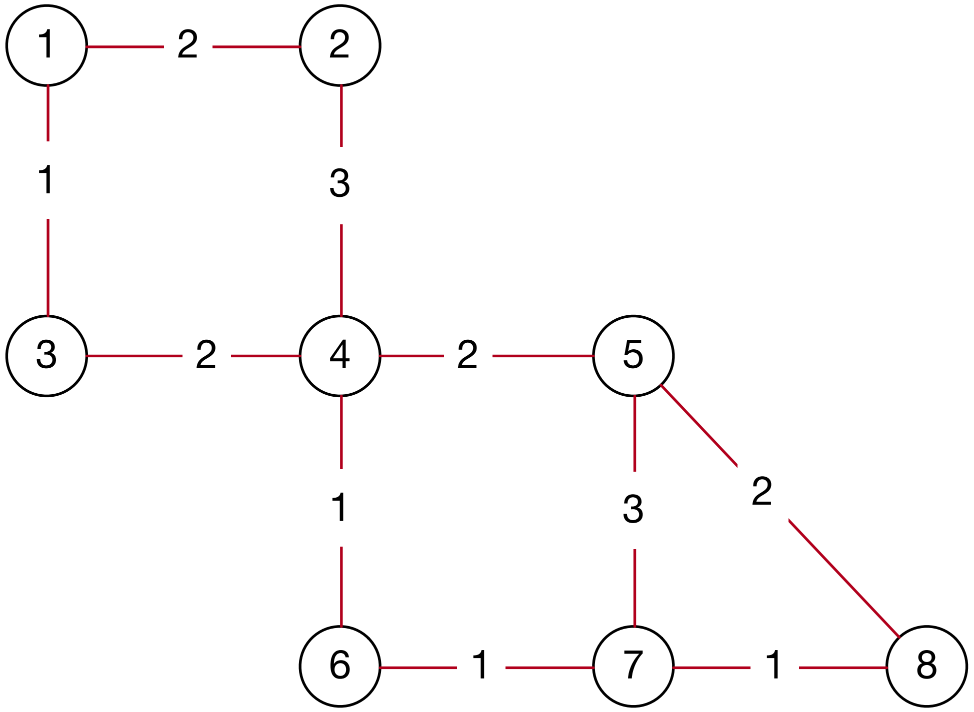


$$D_0 = \begin{pmatrix} 0 & 2 & 1 & \infty & \infty & \infty & \infty & \infty \\ 2 & 0 & \infty & 3 & \infty & \infty & \infty & \infty \\ 1 & \infty & 0 & 2 & \infty & \infty & \infty & \infty \\ \infty & 3 & 2 & 0 & 2 & 1 & \infty & \infty \\ \infty & \infty & \infty & 2 & 0 & \infty & 3 & 2 \\ \infty & \infty & \infty & 1 & \infty & 0 & 1 & \infty \\ \infty & \infty & \infty & \infty & 3 & 1 & 0 & 1 \\ \infty & \infty & \infty & \infty & 2 & \infty & 1 & 0 \end{pmatrix}$$

$$D_1[2,8] = \min(D_0[2,8], D_0[2,1] + D_0[1,8]) = \infty$$

Floyd Warshal Algorithm

- Example continued

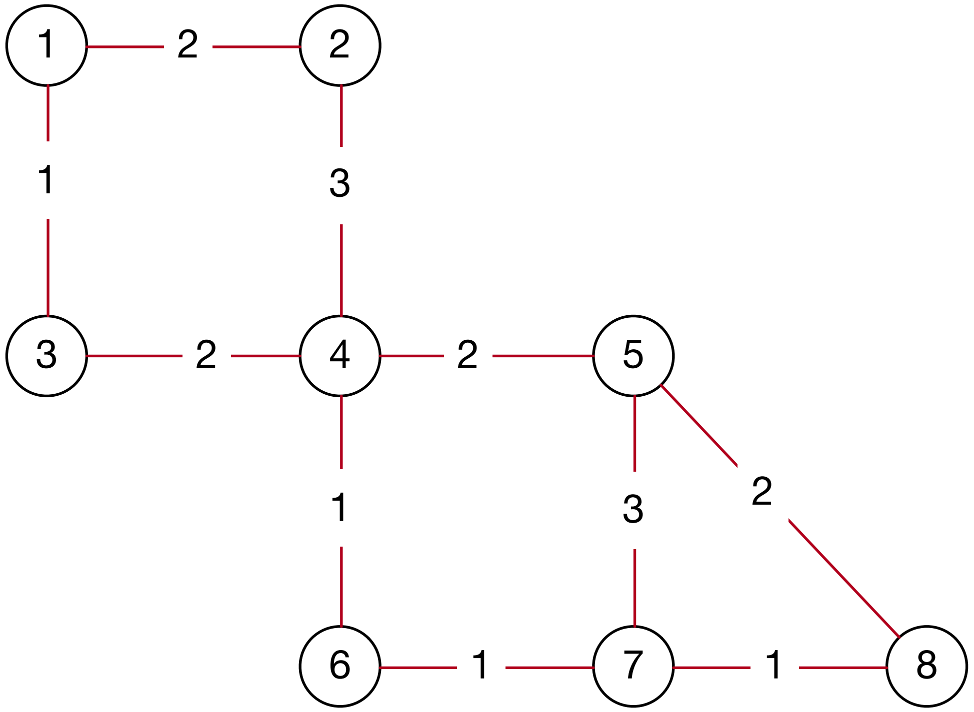


$$D_0 = \begin{pmatrix} 0 & 2 & 1 & \infty & \infty & \infty & \infty & \infty \\ 2 & 0 & \infty & 3 & \infty & \infty & \infty & \infty \\ 1 & \infty & 0 & 2 & \infty & \infty & \infty & \infty \\ \infty & 3 & 2 & 0 & 2 & 1 & \infty & \infty \\ \infty & \infty & \infty & 2 & 0 & \infty & 3 & 2 \\ \infty & \infty & \infty & 1 & \infty & 0 & 1 & \infty \\ \infty & \infty & \infty & \infty & 3 & 1 & 0 & 1 \\ \infty & \infty & \infty & \infty & 2 & \infty & 1 & 0 \end{pmatrix}$$

$$D_1[3,1] = \min(D_0[3,1], D_0[3,1] + D_0[1,1]) = 1$$

Floyd Warshal Algorithm

- Example continued

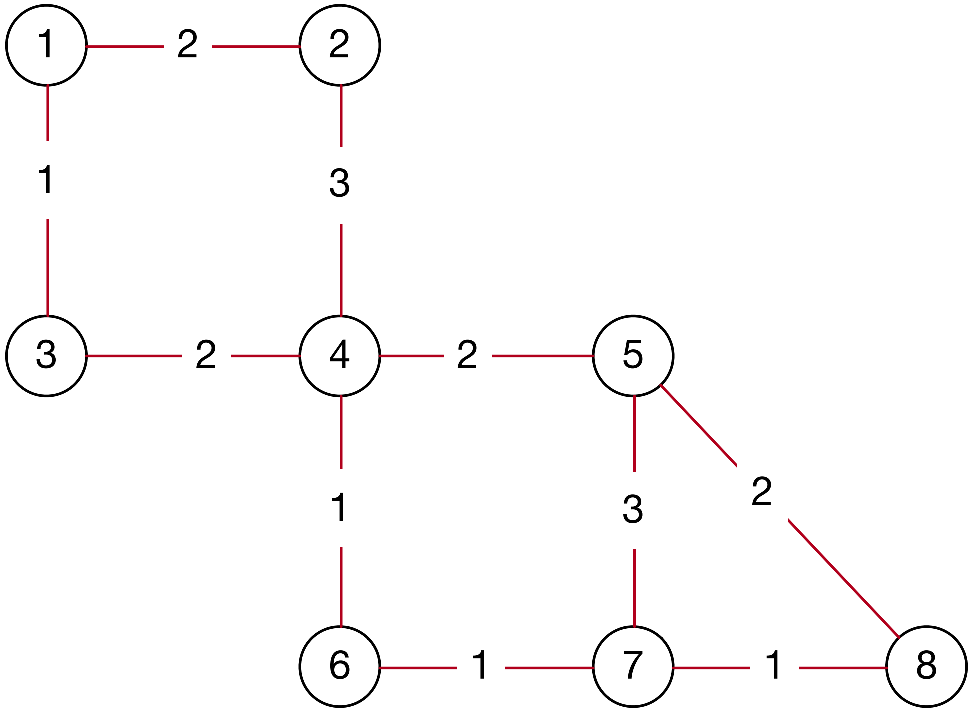


$$D_0 = \begin{pmatrix} 0 & 2 & 1 & \infty & \infty & \infty & \infty & \infty \\ 2 & 0 & \infty & 3 & \infty & \infty & \infty & \infty \\ 1 & \infty & 0 & 2 & \infty & \infty & \infty & \infty \\ \infty & 3 & 2 & 0 & 2 & 1 & \infty & \infty \\ \infty & \infty & \infty & 2 & 0 & \infty & 3 & 2 \\ \infty & \infty & \infty & 1 & \infty & 0 & 1 & \infty \\ \infty & \infty & \infty & \infty & 3 & 1 & 0 & 1 \\ \infty & \infty & \infty & \infty & 2 & \infty & 1 & 0 \end{pmatrix}$$

$$D_1[3,2] = \min(D_0[3,2], D_0[3,1] + D_0[1,2]) = 3$$

Floyd Warshal Algorithm

- Example continued

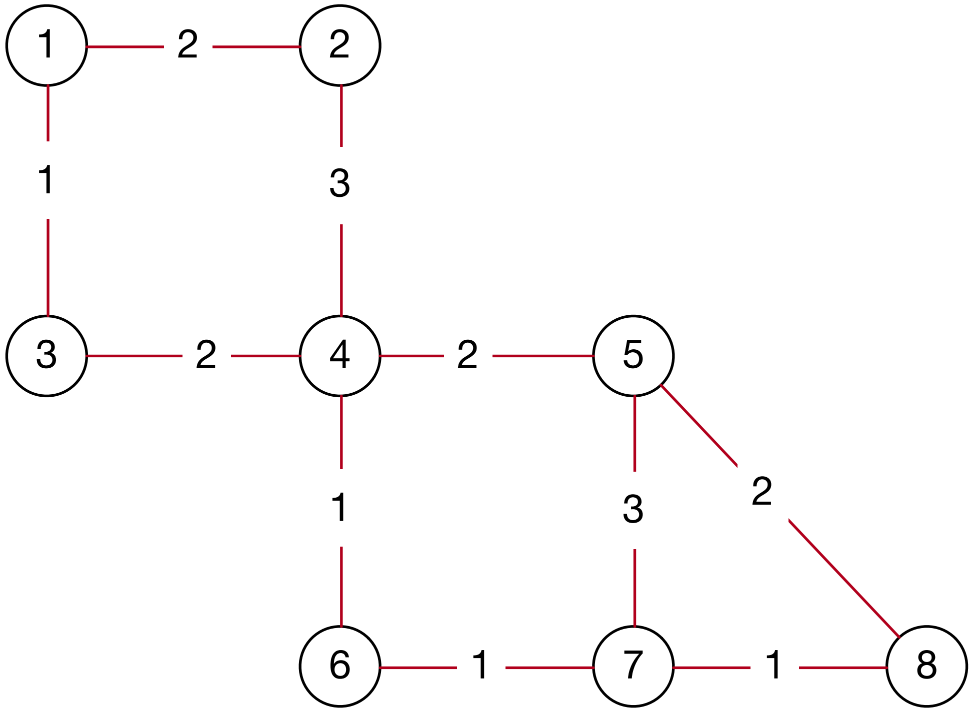


$$D_0 = \begin{pmatrix} 0 & 2 & 1 & \infty & \infty & \infty & \infty & \infty \\ 2 & 0 & \infty & 3 & \infty & \infty & \infty & \infty \\ 1 & \infty & 0 & 2 & \infty & \infty & \infty & \infty \\ \infty & 3 & 2 & 0 & 2 & 1 & \infty & \infty \\ \infty & \infty & \infty & 2 & 0 & \infty & 3 & 2 \\ \infty & \infty & \infty & 1 & \infty & 0 & 1 & \infty \\ \infty & \infty & \infty & \infty & 3 & 1 & 0 & 1 \\ \infty & \infty & \infty & \infty & 2 & \infty & 1 & 0 \end{pmatrix}$$

$$D_1[3,3] = 0$$

Floyd Warshal Algorithm

- Example continued

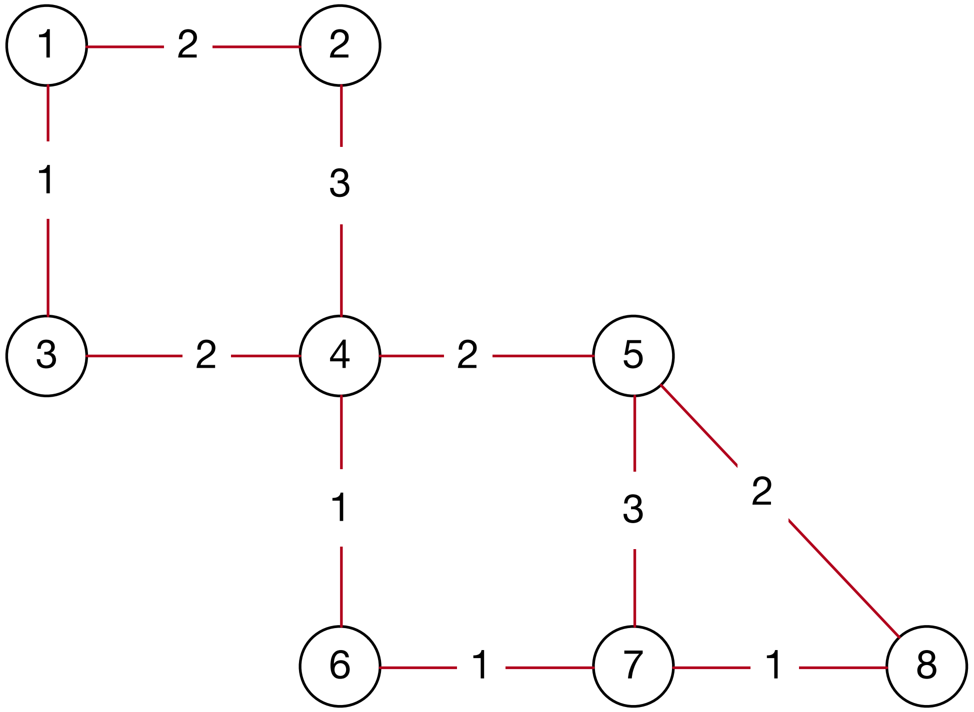


$$D_0 = \begin{pmatrix} 0 & 2 & 1 & \infty & \infty & \infty & \infty & \infty \\ 2 & 0 & \infty & 3 & \infty & \infty & \infty & \infty \\ 1 & \infty & 0 & 2 & \infty & \infty & \infty & \infty \\ \infty & 3 & 2 & 0 & 2 & 1 & \infty & \infty \\ \infty & \infty & \infty & 2 & 0 & \infty & 3 & 2 \\ \infty & \infty & \infty & 1 & \infty & 0 & 1 & \infty \\ \infty & \infty & \infty & \infty & 3 & 1 & 0 & 1 \\ \infty & \infty & \infty & \infty & 2 & \infty & 1 & 0 \end{pmatrix}$$

$$D_1[3,4] = \min(D_0[3,4], D_0[3,1] + D_0[1,4]) = 2$$

Floyd Warshal Algorithm

- Example continued

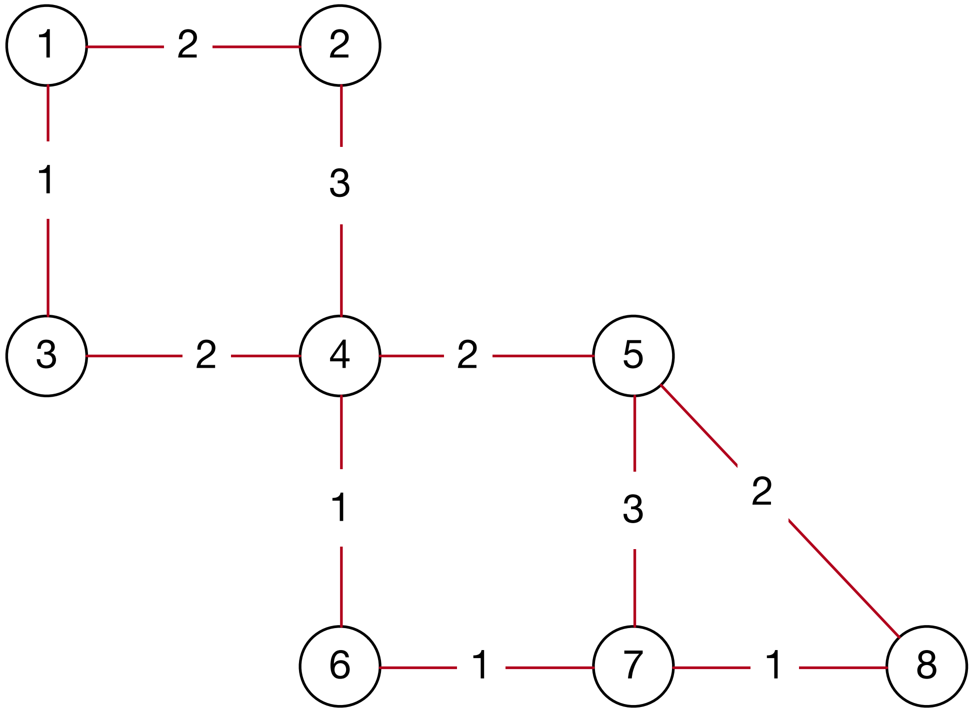


$$D_0 = \begin{pmatrix} 0 & 2 & 1 & \infty & \infty & \infty & \infty & \infty \\ 2 & 0 & \infty & 3 & \infty & \infty & \infty & \infty \\ 1 & \infty & 0 & 2 & \infty & \infty & \infty & \infty \\ \infty & 3 & 2 & 0 & 2 & 1 & \infty & \infty \\ \infty & \infty & \infty & 2 & 0 & \infty & 3 & 2 \\ \infty & \infty & \infty & 1 & \infty & 0 & 1 & \infty \\ \infty & \infty & \infty & \infty & 3 & 1 & 0 & 1 \\ \infty & \infty & \infty & \infty & 2 & \infty & 1 & 0 \end{pmatrix}$$

$$D_1[3,5] = \min(D_0[3,5], D_0[3,1] + D_0[1,5]) = \infty$$

Floyd Warshal Algorithm

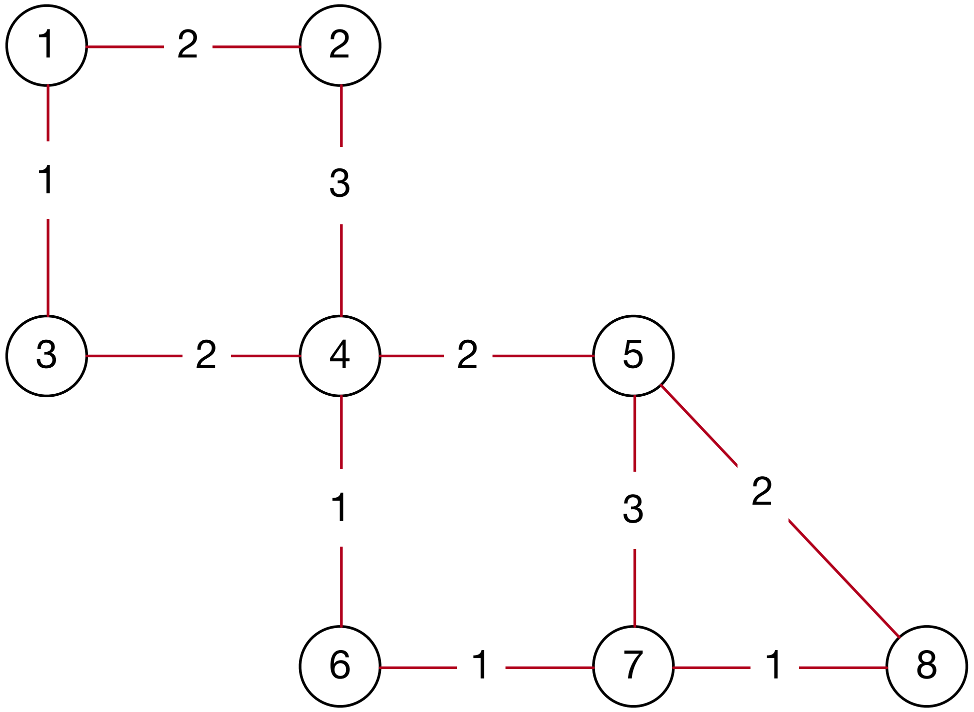
- This finishes the round with 1 as only intermediate vertex



$$D_1 = \begin{pmatrix} 0 & 2 & 1 & \infty & \infty & \infty & \infty & \infty \\ 2 & 0 & 3 & 3 & \infty & \infty & \infty & \infty \\ 1 & 3 & 0 & 2 & \infty & \infty & \infty & \infty \\ \infty & 3 & 2 & 0 & 2 & 1 & \infty & \infty \\ \infty & \infty & \infty & 2 & 0 & \infty & 3 & 2 \\ \infty & \infty & \infty & 1 & \infty & 0 & 1 & \infty \\ \infty & \infty & \infty & \infty & 3 & 1 & 0 & 1 \\ \infty & \infty & \infty & \infty & 2 & \infty & 1 & 0 \end{pmatrix}$$

Floyd Warshal Algorithm

- Next round with 1 and 2 as intermediate vertices

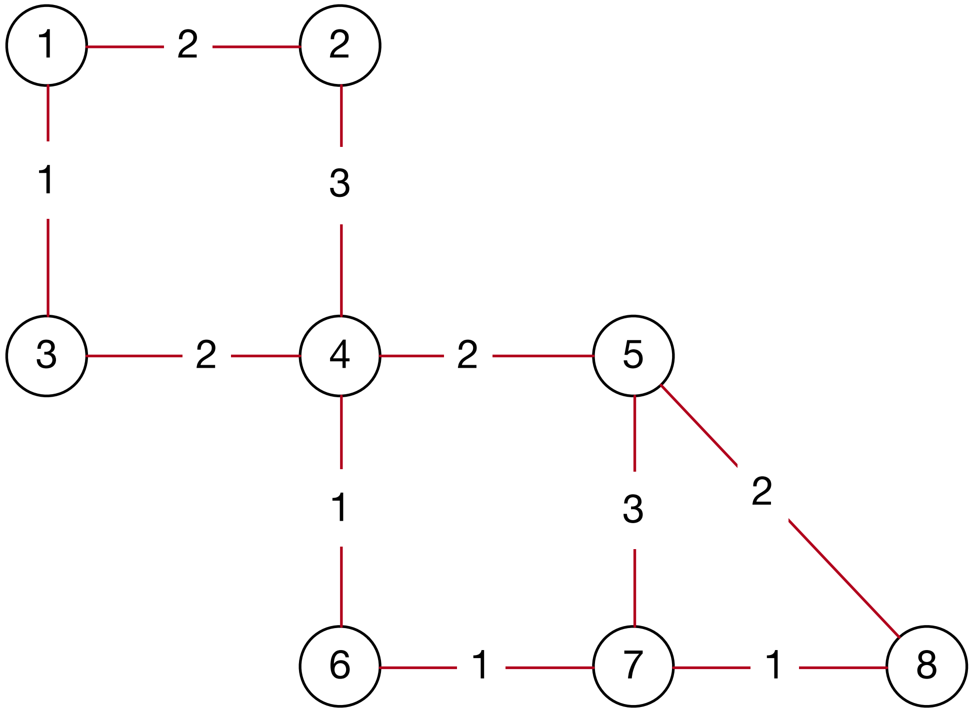


$$D_1 = \begin{pmatrix} 0 & 2 & 1 & \infty & \infty & \infty & \infty & \infty \\ 2 & 0 & 3 & 3 & \infty & \infty & \infty & \infty \\ 1 & 3 & 0 & 2 & \infty & \infty & \infty & \infty \\ \infty & 3 & 2 & 0 & 2 & 1 & \infty & \infty \\ \infty & \infty & \infty & 2 & 0 & \infty & 3 & 2 \\ \infty & \infty & \infty & 1 & \infty & 0 & 1 & \infty \\ \infty & \infty & \infty & \infty & 3 & 1 & 0 & 1 \\ \infty & \infty & \infty & \infty & 2 & \infty & 1 & 0 \end{pmatrix}$$

$$D_2[1,2] = \min(D_1[1,2], D_1[1,2] + D_1[2,2]) = 2$$

Floyd Warshal Algorithm

- Next round with 1 and 2 as intermediate vertices

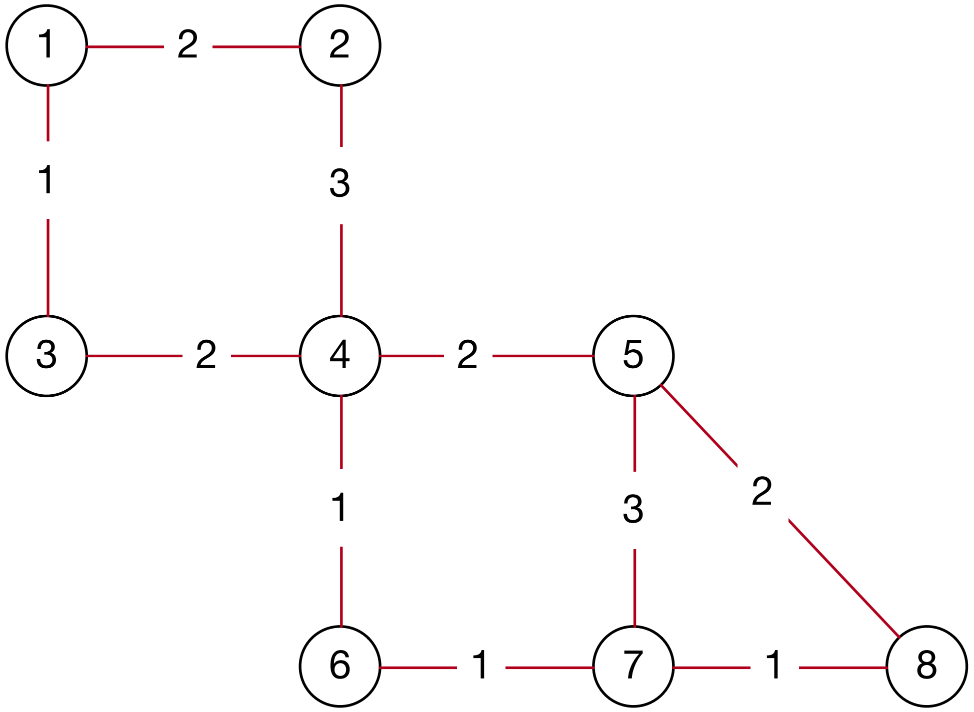


$$D_1 = \begin{pmatrix} 0 & 2 & 1 & \infty & \infty & \infty & \infty & \infty \\ 2 & 0 & 3 & 3 & \infty & \infty & \infty & \infty \\ 1 & 3 & 0 & 2 & \infty & \infty & \infty & \infty \\ \infty & 3 & 2 & 0 & 2 & 1 & \infty & \infty \\ \infty & \infty & \infty & 2 & 0 & \infty & 3 & 2 \\ \infty & \infty & \infty & 1 & \infty & 0 & 1 & \infty \\ \infty & \infty & \infty & \infty & 3 & 1 & 0 & 1 \\ \infty & \infty & \infty & \infty & 2 & \infty & 1 & 0 \end{pmatrix}$$

$$D_2[1,3] = \min(D_1[1,3], D_1[1,2] + D_1[2,3]) = 3$$

Floyd Warshal Algorithm

- Next round with 1 and 2 as intermediate vertices

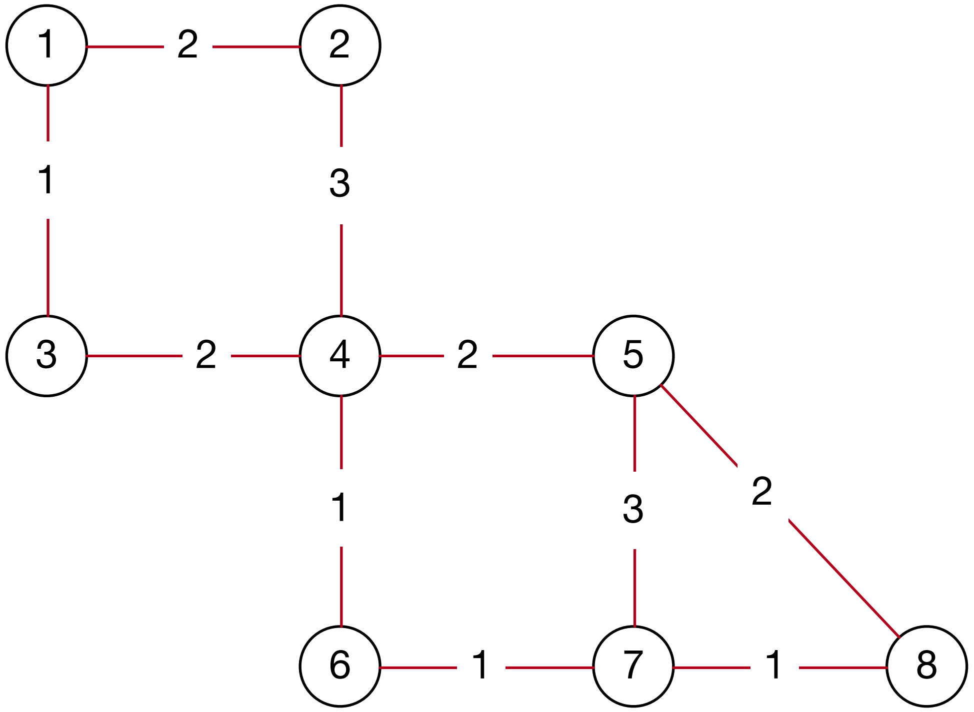


$$D_1 = \begin{pmatrix} 0 & 2 & 1 & \infty & \infty & \infty & \infty & \infty \\ 2 & 0 & 3 & 3 & \infty & \infty & \infty & \infty \\ 1 & 3 & 0 & 2 & \infty & \infty & \infty & \infty \\ \infty & 3 & 2 & 0 & 2 & 1 & \infty & \infty \\ \infty & \infty & \infty & 2 & 0 & \infty & 3 & 2 \\ \infty & \infty & \infty & 1 & \infty & 0 & 1 & \infty \\ \infty & \infty & \infty & \infty & 3 & 1 & 0 & 1 \\ \infty & \infty & \infty & \infty & 2 & \infty & 1 & 0 \end{pmatrix}$$

$$D_2[1,4] = \min(D_1[1,4], D_1[1,2] + D_1[2,4]) = \min(\infty, 2+3) = 5$$

Floyd Warshal Algorithm

- Next round with 1 and 2 as intermediate vertices

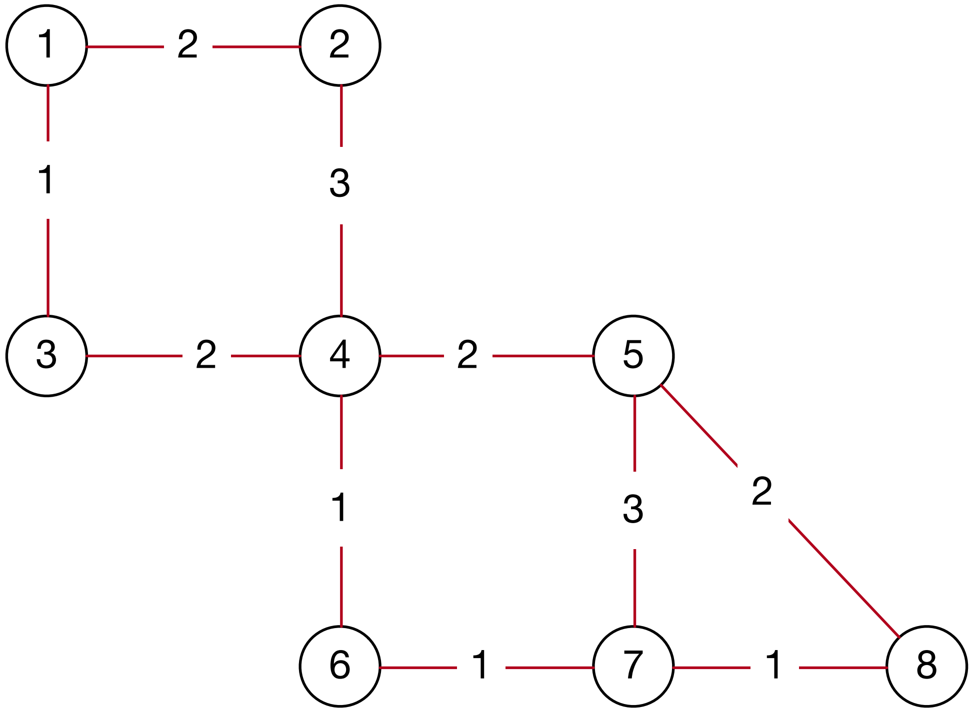


$$D_1 = \begin{pmatrix} 0 & 2 & 1 & \infty & \infty & \infty & \infty & \infty \\ 2 & 0 & 3 & 3 & \infty & \infty & \infty & \infty \\ 1 & 3 & 0 & 2 & \infty & \infty & \infty & \infty \\ \infty & 3 & 2 & 0 & 2 & 1 & \infty & \infty \\ \infty & \infty & \infty & 2 & 0 & \infty & 3 & 2 \\ \infty & \infty & \infty & 1 & \infty & 0 & 1 & \infty \\ \infty & \infty & \infty & \infty & 3 & 1 & 0 & 1 \\ \infty & \infty & \infty & \infty & 2 & \infty & 1 & 0 \end{pmatrix}$$

$$D_2[1,5] = \min(D_1[1,5], D_1[1,2] + D_1[2,5]) = \infty$$

Floyd Warshal Algorithm

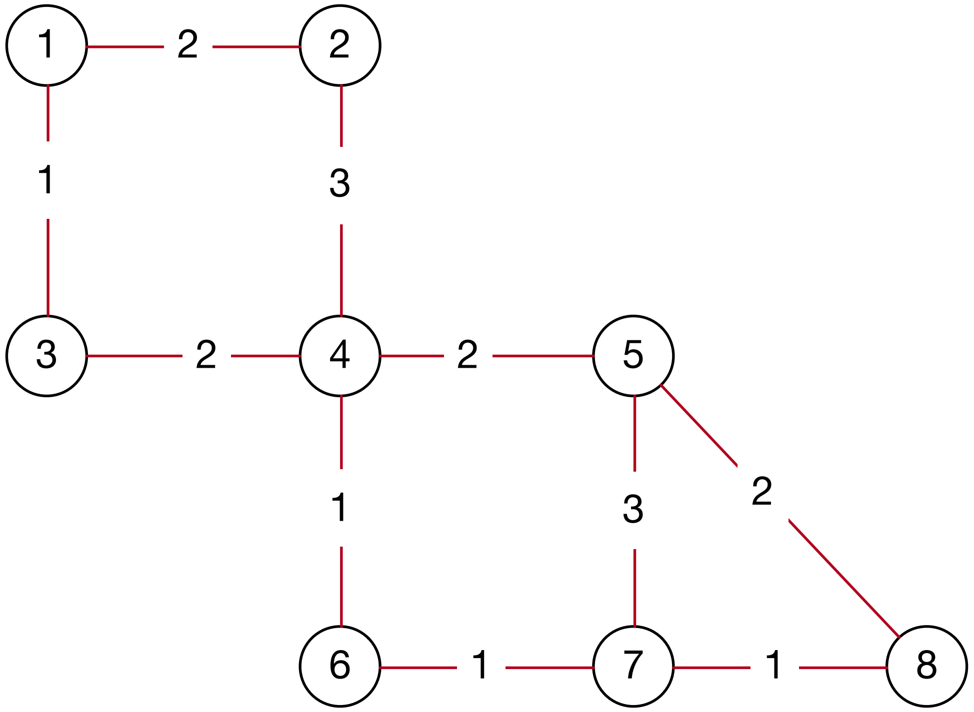
- After 1 and 2



$$D_2 = \begin{pmatrix} 0 & 2 & 1 & 5 & \infty & \infty & \infty & \infty \\ 2 & 0 & 3 & 3 & \infty & \infty & \infty & \infty \\ 1 & 3 & 0 & 2 & \infty & \infty & \infty & \infty \\ 5 & 3 & 2 & 0 & 2 & 1 & \infty & \infty \\ \infty & \infty & \infty & 2 & 0 & \infty & 3 & 2 \\ \infty & \infty & \infty & 1 & \infty & 0 & 1 & \infty \\ \infty & \infty & \infty & \infty & 3 & 1 & 0 & 1 \\ \infty & \infty & \infty & \infty & 2 & \infty & 1 & 0 \end{pmatrix}$$

Floyd Warshal Algorithm

- Final version



$$D_8 = \begin{pmatrix} 0 & 2 & 1 & 3 & 5 & 4 & 5 & 6 \\ 2 & 0 & 3 & 3 & 5 & 4 & 5 & 6 \\ 1 & 3 & 0 & 2 & 4 & 3 & 4 & 5 \\ 3 & 3 & 2 & 0 & 2 & 1 & 3 & 3 \\ 5 & 5 & 4 & 2 & 0 & 3 & 3 & 2 \\ 4 & 4 & 3 & 1 & 3 & 0 & 1 & 2 \\ 5 & 5 & 4 & 2 & 3 & 1 & 0 & 1 \\ 6 & 6 & 5 & 3 & 2 & 2 & 1 & 0 \end{pmatrix}$$

- But what about routing?

Floyd Warshal Algorithm

- When we update the distance matrix, we should also update a matrix with path information
  - Not necessary to give the complete path with this distance
  - Can either specify predecessor or — as in network routing tables — the next node

Floyd Warshal Algorithm

- Algorithm maintains a NEXT table
    - We use to sentinels:
        - inf: if there is currently no path from source to destination
        - -1: if we are already there, e.g. in the main diagonal

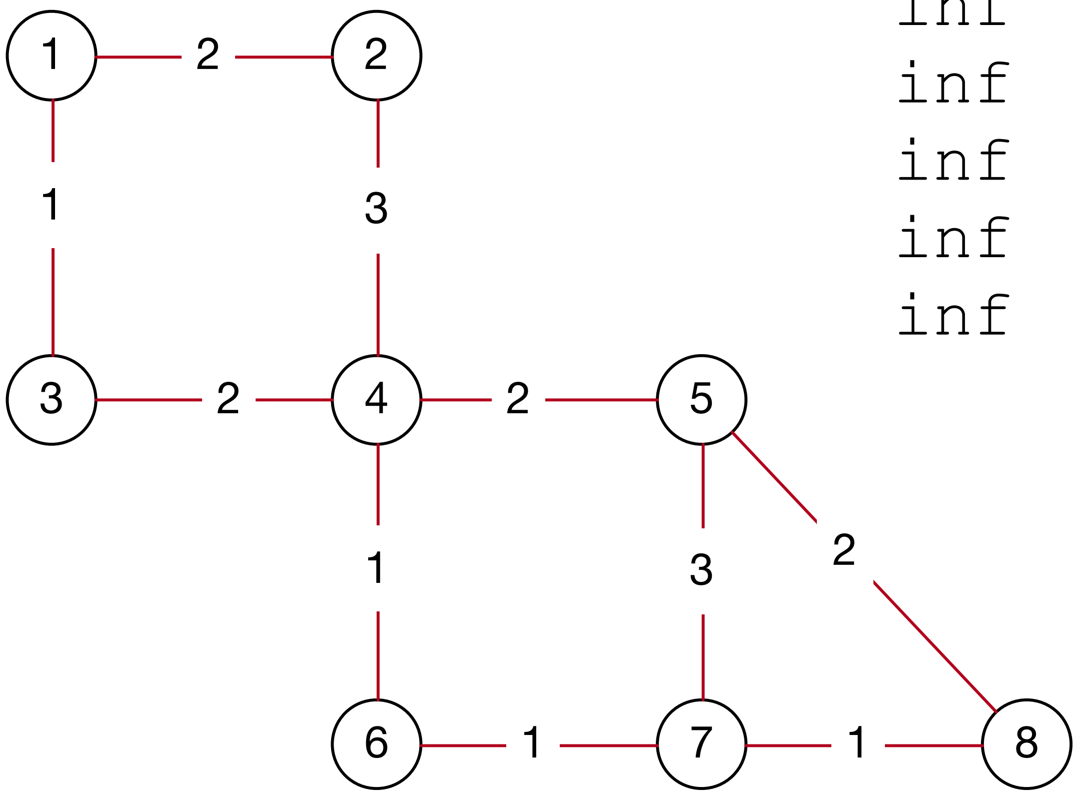
Floyd Warshal Algorithm

- Initialize the NEXT table by:
  - inf — if there is no edge between vertices
  - -1 — if the two vertices are the same
  - weight of edge — if there is an edge

Floyd Warshal Algorithm

- Example:

```
Next
  -1    1    2  inf  inf  inf  inf  inf
   0   -1  inf    3  inf  inf  inf  inf
   0  inf   -1    3  inf  inf  inf  inf
 inf    1    2   -1    4    5  inf  inf
 inf  inf  inf    3   -1  inf    6    7
 inf  inf  inf    3  inf   -1    6  inf
 inf  inf  inf  inf    4    5   -1    7
 inf  inf  inf  inf    4  inf    6   -1
```



Floyd Warshal Algorithm

- Change the update condition for each FW round:

```
for imv in range(self.nr_nodes):
            print(imv)
            for source in range(self.nr_nodes):
                for dest in range(self.nr_nodes):
                    if source == dest:
                        continue
                    if self.dis[source][dest] >
self.dis[source][imv]+self.dis[imv][dest]:
                        self.dis[source][dest] =
self.dis[source][imv]+self.dis[imv][dest]
                        self.next[source][dest] =
self.next[source][ imv]
```

Floyd Warshal Algorithm

- Change the update:
  - If going through the new intermediate vector is faster:
    - Use the next value for the intermediate vector

Floyd Warshal Algorithm

- Correctness:
  - Loop invariant:
    - After processing a FW-round with intermediate vertex *k*:
      - The distances are the distances of the best path involving intermediate vertices in $\{1, 2, \ldots, k\}$.

Floyd Warshal Algorithm

- Proof of loop invariant:
  - By induction hypothesis, $D_{k-1}[i, j], D_{k-1}[i, k], D_{k-1}[k, j]$ reflect the length of the shortest paths with intermediaries $\in \{1,2,\ldots, k-1\}$
  - A shortest path between *i* and *j* with intermediaries in 1 … *k* passes through *k* once or not at all
    - because shortest paths do not contain cycles
  - A case distinction now shows the truth of the loop invariant

Floyd Warshal Algorithm

- Similarly: Proof that the NEXT entry is correct

Floyd Warshal Algorithm

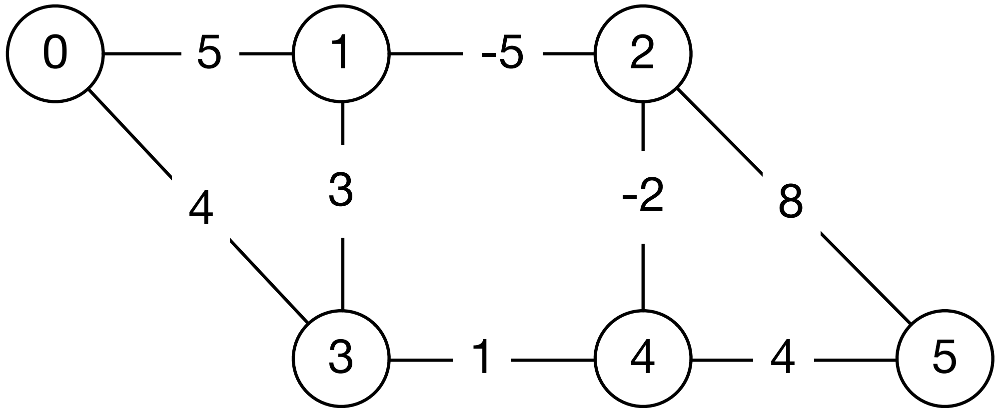
- Notice:
  - We never used the fact that edges are bi-directional
  - Algorithm also works for for directed graphs

Floyd Warshal Algorithm

- We never used the fact that all weights have to be positive
- If there is a negative weight cycle, we detect it because the distance between *i* and *i* (for any vertex *i* on the cycle) becomes negative after processing all the nodes.

Floyd Warshal Algorithm

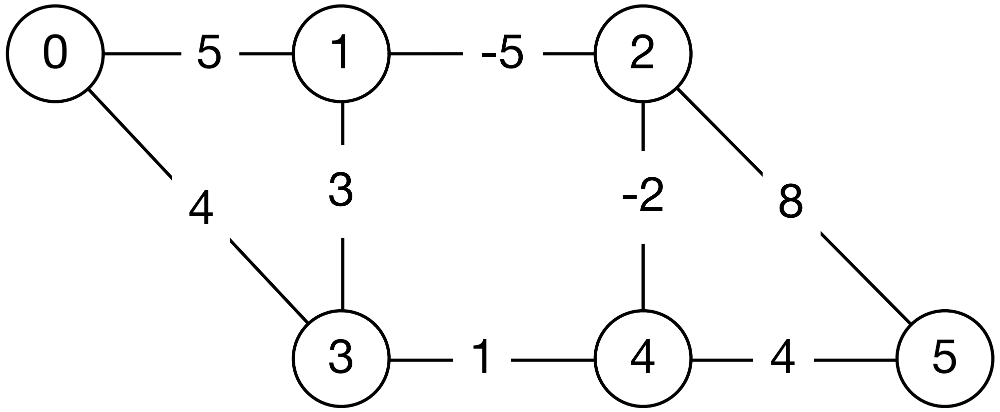
- Example:



```
Distance
   0     5   inf     4   inf   inf
   5     0    -5     3   inf   inf
 inf    -5     0   inf    -2     8
   4     3   inf     0     1   inf
 inf   inf    -2     1     0     4
 inf   inf     8   inf     4     0
Next
  -1     1   inf     3   inf   inf
   0    -1     2     3   inf   inf
 inf     1    -1   inf     4     5
   0     1   inf    -1     4   inf
 inf   inf     2     3    -1     5
 inf   inf     2   inf     4    -1
```

Floyd Warshal Algorithm

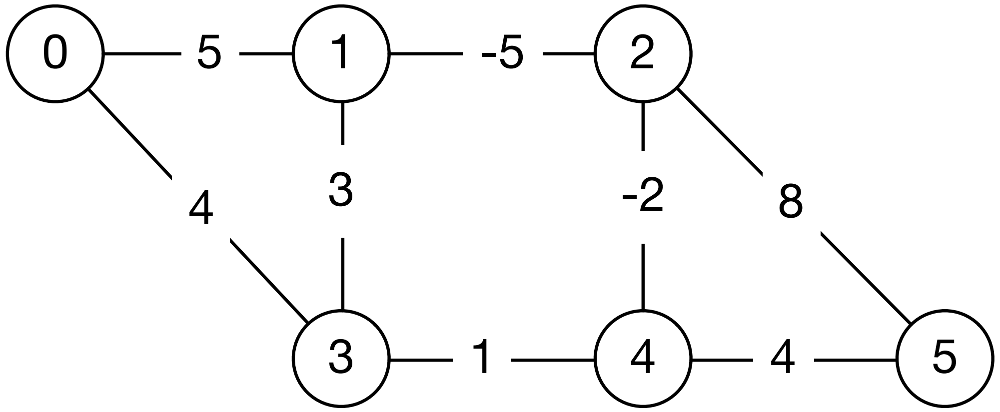
- Example:



```
0
Distance
    0     5   inf     4   inf   inf
    5     0    -5     3   inf   inf
  inf    -5     0   inf    -2     8
    4     3   inf     0     1   inf
  inf   inf    -2     1     0     4
  inf   inf     8   inf     4     0
Next
   -1     1   inf     3   inf   inf
    0    -1     2     3   inf   inf
  inf     1    -1   inf     4     5
    0     1   inf    -1     4   inf
  inf   inf     2     3    -1     5
  inf   inf     2   inf     4    -1
```

Floyd Warshal Algorithm

- Example:

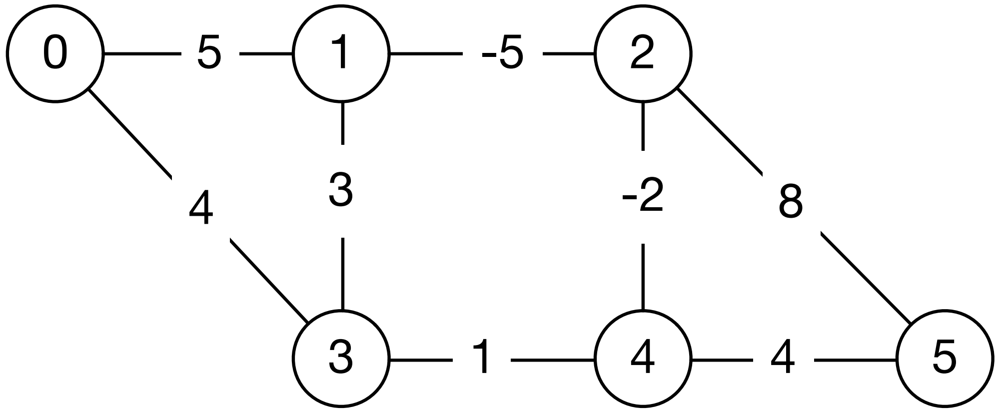


```
1
Distance
   0    5    0    4  inf  inf
   5    0   -5    3  inf  inf
   0   -5  -10   -2   -2    8
   4    3   -2    0    1  inf
 inf  inf   -2    1    0    4
 inf  inf    8  inf    4    0
Next
  -1    1    1    3  inf  inf
   0   -1    2    3  inf  inf
   1    1    1    1    4    5
   0    1    1   -1    4  inf
 inf  inf    2    3   -1    5
 inf  inf    2  inf    4   -1
```

- We are detecting the first cycle: 2-1-2 with a cost of -10

Floyd Warshal Algorithm

- Example:



```
5
Distance
-1236 -1241 -1246 -1268 -1400 -2182
-1241 -1246 -1251 -1273 -1405 -2187
-1246 -1251 -1256 -1278 -1410 -2192
-1268 -1273 -1278 -1300 -1432 -2214
-1400 -1405 -1410 -1432 -1564 -2346
-2182 -2187 -2192 -2214 -2346 -3128
Next
    1     1     1     1     1     1
    2     2     2     2     2     2
    1     1     1     1     1     1
    1     1     1     1     1     1
    2     2     2     2     2     2
    2     2     2     2     2     2
```

Floyd Warshal Algorithm

- Why are the final distance numbers not negative infinity:
    - Because we assumed that each intermediary vertex on a shortest path is only visited **once**
    - To fully exploit a cycle, we want to exploit it more than once

Floyd Warshal Algorithm

- Complexity:
  - With n vertices:
    - n rounds
    - Each round updates $2n^2$ matrix elements
    - Time Complexity is $\Theta(n^3)$
  - Space complexity is $2n^2$ in order to store the matrices