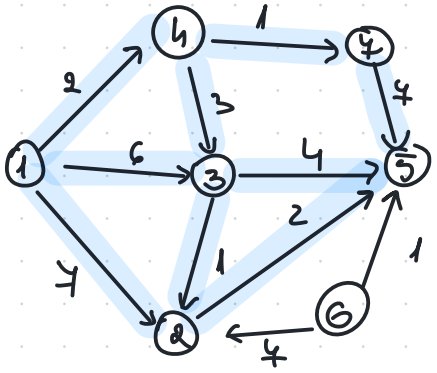
jun '23 (1)

1) minimum cost path from 1 to each vertex with Dijkstra



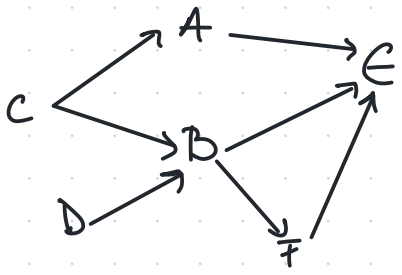
queue

~~(1,0)~~
~~→(4,2)~~
~~→(2,4)~~
~~→(7,3)~~
~~→(3,5)~~
(5,12)
~~→(3,6)~~
(5,9)
~~(2,4)~~
(5,9)

1	2	3	4	5	6	7
0	~~4~~ 6	5	2	~~12~~ 9 ~~9~~ 8	∞	3

2)

Act	Duration	Prerequisites
A	3	C
B	4	CD
C	3	—
D	3	—
E	1	A B F
F	2	B



earliest
(0-3) C
(3-6) A
(3-7) B
(0-3) D
(7-9) F
(9-10) E

latest
(0-3) C
(0-3) D

C: ES = 0
EF = 3

D: ES = 0
EF = 3

A: ES = 3
EF = 6

B: ES = 3
EF = 4

I: ES = 4
EF = 9

E: ES = 9
EF = 10

E: LF = EF
LF = 10
LS = 9

F: LF = 9
LS = 4

A: LF = 9
LS = 6

B: LF = 4
LS = 3

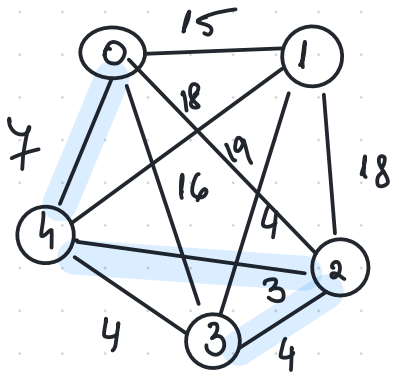
C: LF = 3
LS = 0

D: LF = 3
LS = 0

3) Number of distinct paths of max lenght between 2 vertices

1) topological sort
   - ↳ in degree
   - ↳ proces deg(0)
   - ↳ append to array and decrese
   - ↳ if a neighbour becomes 0 add queu

2) compute in top order



sorted

~~(4, 2)~~

~~(1, 3)~~

(4, 3)

~~(2, 3)~~

~~(0, 4)~~

(3, 0)

(0, 1)

(2, 1)

(4, 1)

(0, 2)

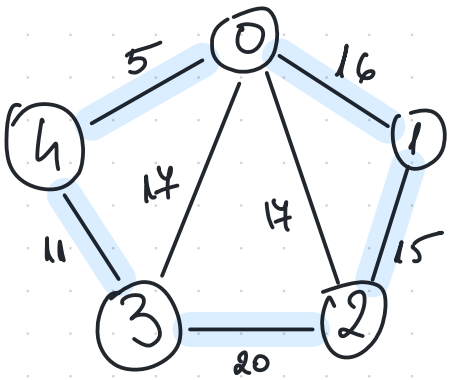
path = [0 -1 -1 -1 -1]

0 4 -1 -1 -1

0 4 2 -1 -1

0 4 2 1 -1

0 4 2 1 3



~~(0, 4)~~

~~(4, 3)~~

(2, 1)

(0, 1)

(2, 0)

(0, 3)

~~(3, 2)~~

path 0 -1 -1 -1 -1

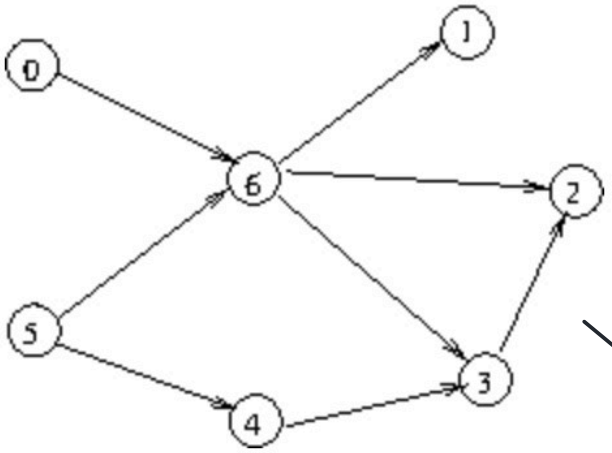
0 4 -1 -1 -1

0 4 3 -1 -1

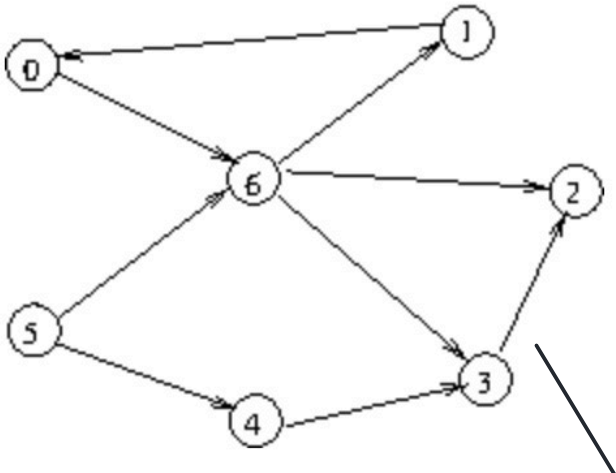
0 4 3 2 -1

0	4	3	2	1

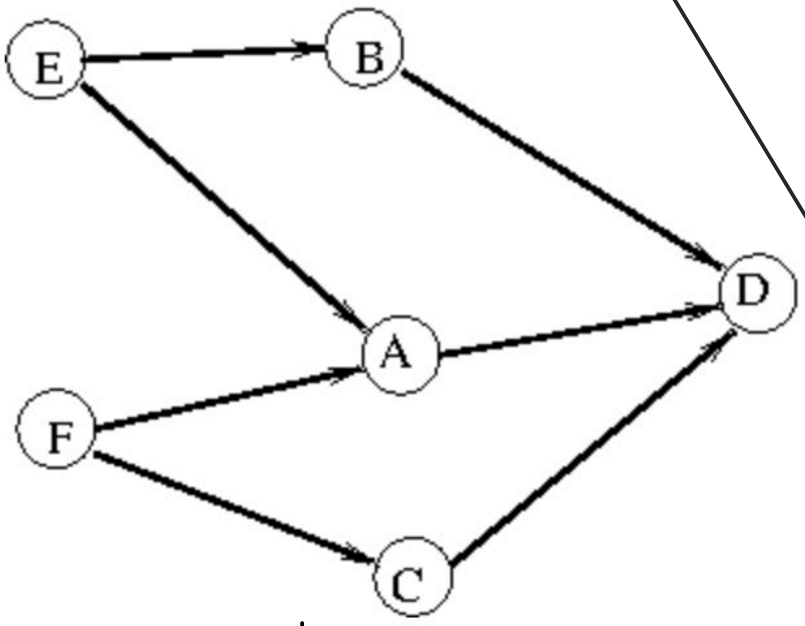
DAG example



Cycle-containing graph



Second DAG example



predecessor counting:

* initialize with indegree for each
* enqueue all vertices with 0
* while not empty
  - ↳ dequeue v and add
  - ↳ for each neighbour reduce the degree by 1
  - ↳ if deg(0) → enqueue it

1)

vertex	0	1	2	3	4	5	6
degree	0	~~1~~ 0	~~2~~ ~~1~~ 0	~~2~~ ~~1~~ 0	~~1~~ 1	0	~~2~~ ~~1~~ 0

⇒ Q: {0, 5}

{0, 5, 4, 6}

{0, 5, 4, 3, 1, 2}

X	Q	out						
		0	1	2	3	4	5	6
	0, 5	0	1	2	2	1	0	2
0	5	–	1	2	2	1	0	1
5	4, 6	–	1	2	2	0	–	0
4	6	–	1	2	1	–	–	0
6	1, 3	–	0	1	0	–	–	–
1	3	–	–	1	0	–	–	–
3	2	–	–	0	–	–	–	–
2		–	–	–	–	–	–	–

X	Q	out						
		0	1	2	3	4	5	6
	5	1	1	2	2	1	0	2
5	4	1	1	2	2	0	–	1
4		1	1	2	1	–	–	1

Stops ⇒ cycle exists

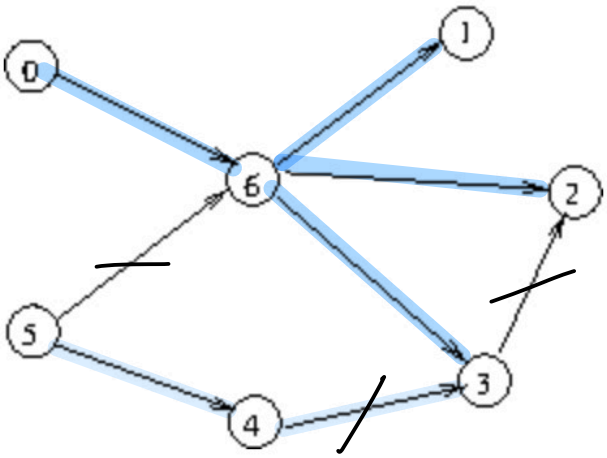
X	q	cut					
		A	B	C	D	E	F
	E,F	2	1	1	3	0	0
E	F, B	1	0	1	3	–	0
F	B, A, C	0	0	0	3	–	–
B	A, C	0	–	0	2	–	–
A	C	–	–	0	1	–	–
C	D	–	–	–	0	–	–
D		–	–	–	–	–	–

stack

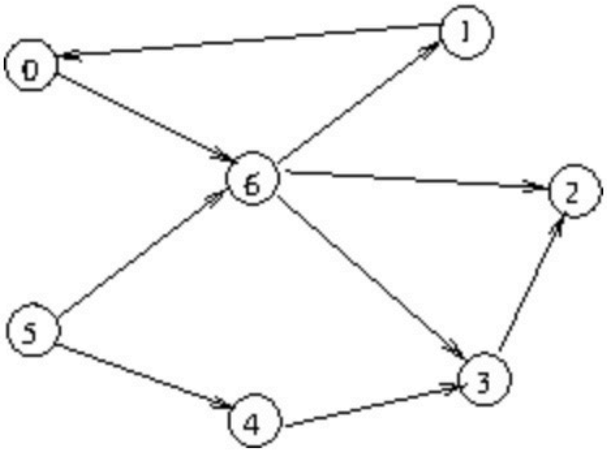
5
4
0
6
3
2
1

visited : 0 6 1 2 3 5 4

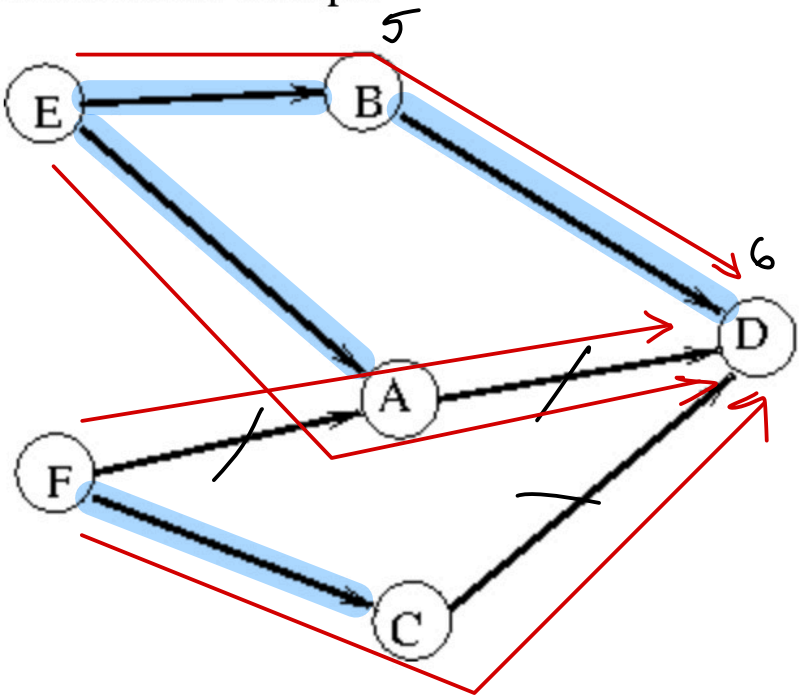
DAG example



Cycle-containing graph



Second DAG example



stack

F
C
E
A
B
D

visited :
E B D A F C

⇒ sort F C E A B D

Scheduling problem

Act.	Prerequisites	Duration	Earliest	Latest
A	E, F	1	4-5	5-6
B	E	2	4-6	4-6
C	F	3	3-6	3-6
D	A, B, C	5	6-11	6-11
E	-	4	0-4	
F	-	3	0-3	

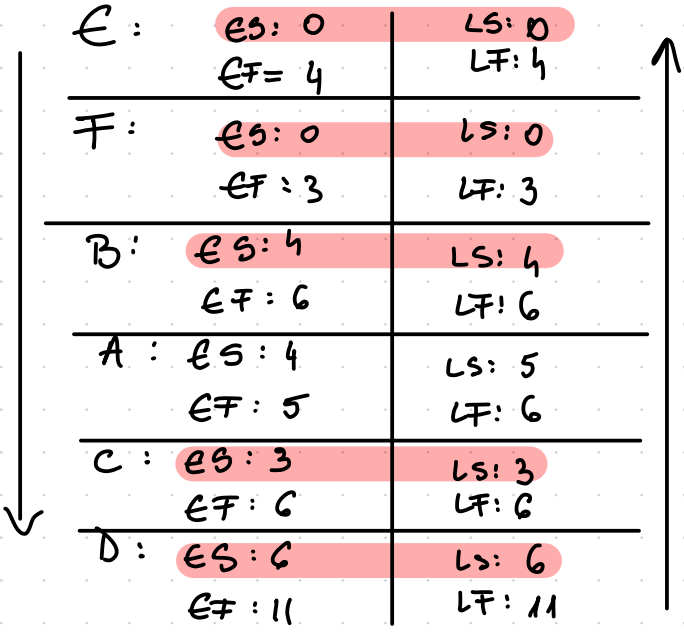
critical path is when
ES = LS

multiple critical paths

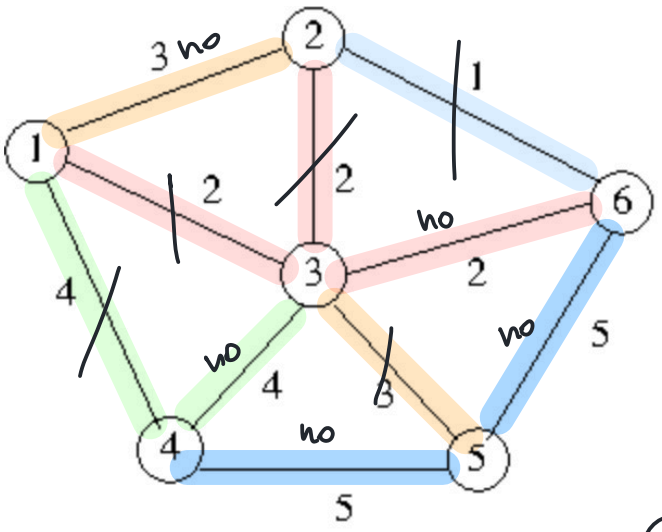
1) Topological sort

E F B A C D

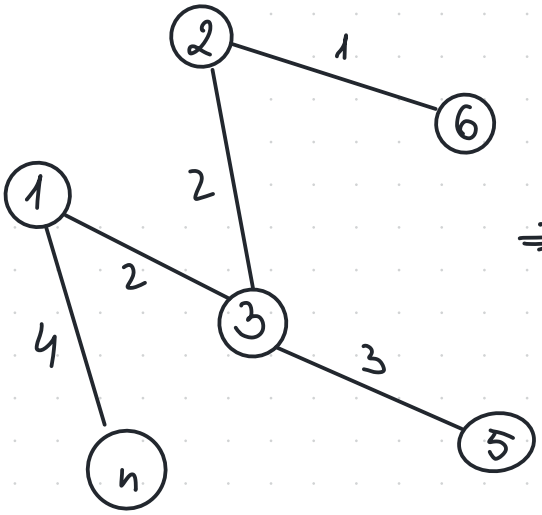
2) Earliest start & finish



Construct the Minimum Spanning Tree for the following graph



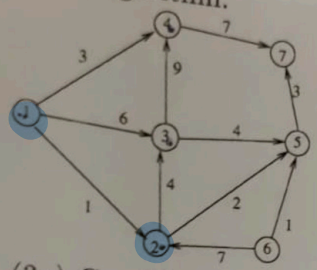
Kruskal: 1) sort edges by cost
2) pick the cheapest edge & check for cycle
3) repeat until there are V-1 edge ⇒ tree



⇒ MST cost = 12

Exam to Graph algorithms
jun 2023, subject no. 2

1. (3p) In the digraph below, find the minimum cost paths from vertex 1 to each of the vertices, using the Dijkstra algorithm.



1: 0
2: 1→2 (1)
3: 1→2→3 (5)
4: 1→4 (3)
5: 1→2→5 (3)
6: ∞
7: 1→2→5→7 (6)

2. (3p) Considering the following activities, determine the earliest and the latest schedulings, and the critical activities (show the step-by-step computations for the topological sorting, then for the starting and ending times):

Act.	Duration	Prerequisites
A	2	E, F
B	1	E
C	3	F
D	5	A, B, C
E	4	—
F	3	—

3. (3p) Write a polynomial time algorithm that, given a directed acyclic graph with costs and two vertices, finds the number of distinct paths of minimum cost between the two vertices.

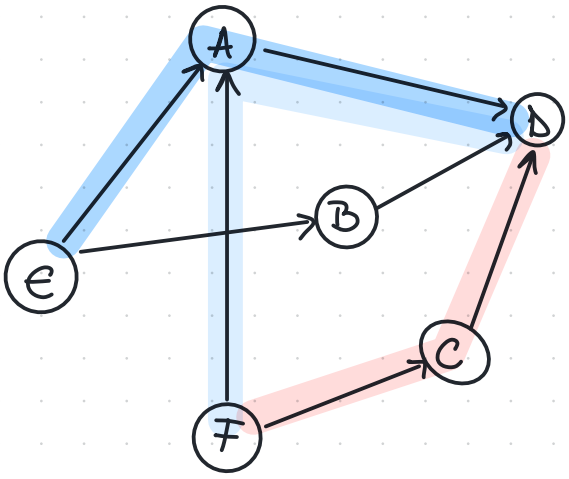
v	1	2	3	4	5	6	7
d	0	1	5	3	3	∞	6
prev	1	1	2	1	2		5

priority queue:

(vertex, dist)

~~(1, 0)~~
~~(2, 1)~~
~~(4, 3)~~
neighbours of 1 in order of dist
(3, 6) no
~~(3, 5)~~
~~(5, 3)~~
(7, 10)
(7, 6)
(5, 6)

dist[3] = min(6, 4+1) = 5
dist[5] = 3
dist[7] = 10
dist[7] = min(10, 3+3) = 6



X	Q	inbound					
		A	B	C	D	E	F
	E,F	2	1	1	3	0	0
E	F,B	1	0	1	3	—	0
F	B,A	0	0	0	3	—	—
B	A,C	0	—	0	2	—	—
A	C	—	—	0	1	—	—
C	D	—	—	—	0	—	—
D							

⇒ Topological sort E F B A C D

E: ES: 0 LS: 0
EF: 4 LF: 4

F: ES: 0 LS: 0
EF: 3 LF: 3

B: ES: 4 LS: 5
EF: 5 LF: 6

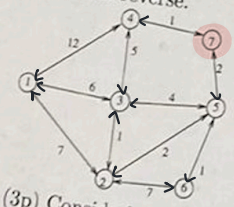
A: ES: 4 LS: 4
EF: 6 LF: 6

C: ES: 3 LS: 3
EF: 6 LF: 6

D: ES: 6 LS: 6
EF: 11 LF: 11

Exam to Graph algorithms
jun 2023, subject no. 3

1. (3p) In the digraph below, find the minimum cost paths from all vertices to vertex 7, using the Dijkstra algorithm in reverse.



2. (3p) Considering the following activities, determine the earliest and the latest schedulings, and the critical activities (show the step-by-step computations for the topological sorting, then for the starting and ending times):

Act.	Duration	Prerequisites
A	5	—
B	4	A, D
C	1	B, F
D	3	E
E	2	—
F	5	E

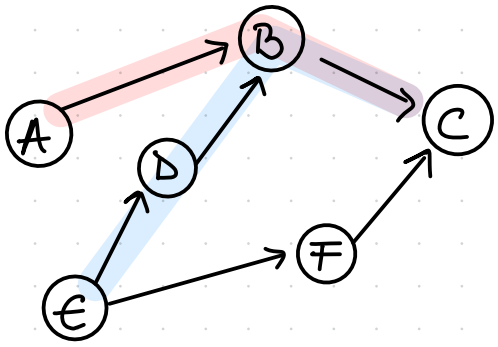
3. (3p) Write a polynomial time algorithm that, given a directed acyclic graph with costs and two vertices, finds the number of distinct paths of minimum cost between the two vertices.

1)

v	1	2	3	4	5	6	7
d	11	4	5	1	2	3	0
prev	2	5	2	4	4	5	

priority queue: ~~(4,1)~~ ~~(5,2)~~ (1,13) ~~(3,6)~~ ~~(3,6)~~ ~~(2,4)~~ ~~(6,3)~~ ~~(6,11)~~ ~~(3,5)~~ ~~(1,11)~~ (1,11)

2)



Scheduling problem

Act.	Prerequisites	Duration	Earliest	Latest
A	—	5	0-5	0-5
B	A, D	4	5-9	5-9
C	B, F	1	9-10	9-10
D	E	3	2-5	2-5
E	—	2	0-2	0-2
F	E	5	2-7	4-9

A: ES: 0, EF: 5 | LS: 0, LF: 5

E: ES: 0, EF: 2 | LS: 0, LF: 2

D: ES: 2, EF: 5 | LS: 2, LF: 5

B: ES: 5, EF: 9 | LS: 5, LF: 9

F: ES: 2, EF: 4 | LS: 4, LF: 9

C: ES: 9, EF: 10 | LS: 9, LF: 10

F	3	F
		D

3. (3p) Write a polynomial time algorithm that, given a directed acyclic graph and two vertices, finds the number of distinct paths of maximum length between the two vertices.

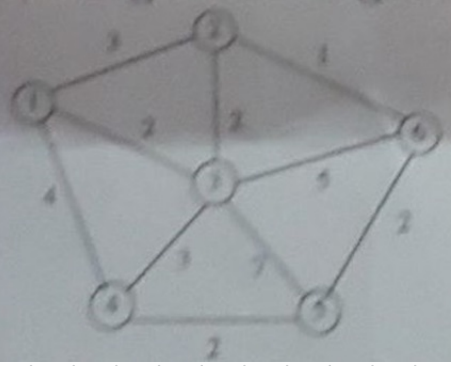
3. (3p) Write a polynomial time algorithm that, given a directed graph and a pair of vertices s and t, finds the number of distinct paths of minimum length from s to t.

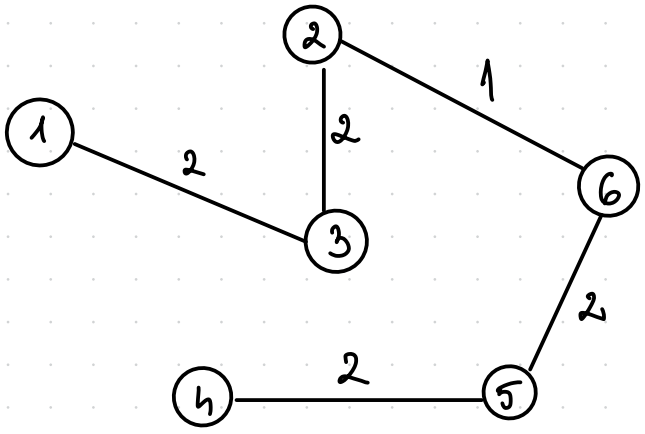
3. (3p) Write a polynomial time algorithm that, given a directed graph with costs and a vertex s in that graph, finds the shortest cycle of strictly negative total cost that starts and ends in s, if one exists. Note: if there are several cycles of negative cost, the algorithm shall return one that minimises the length.

3. (3p) Design an algorithm that, given a directed graph, finds a cycle of minimum length, if one exists.

3. (3p) Design an algorithm that, given a directed graph, finds a cycle of minimum length through a given vertex, if such a cycle exists.

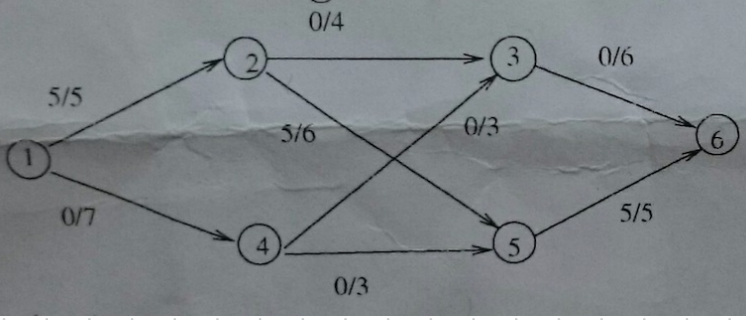
2. (3p) In the following graph, find the minimum spanning tree, using Prim's algorithm:



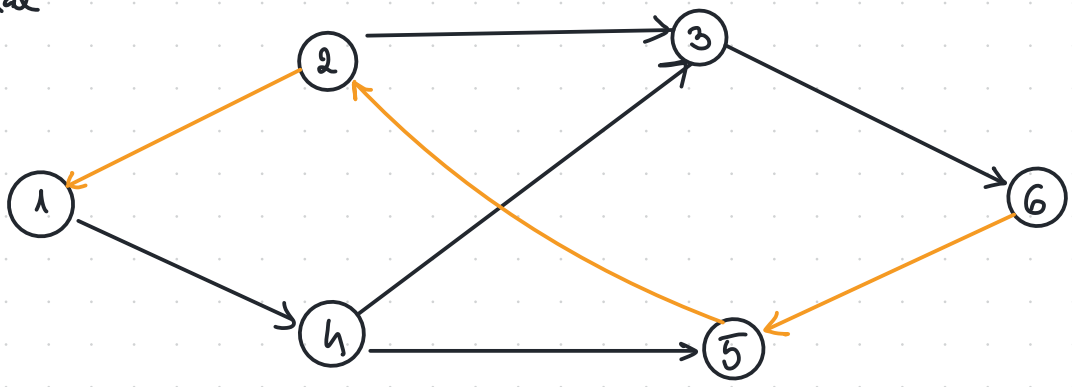


* keeps adding the lowest connected edge, without forming cycles

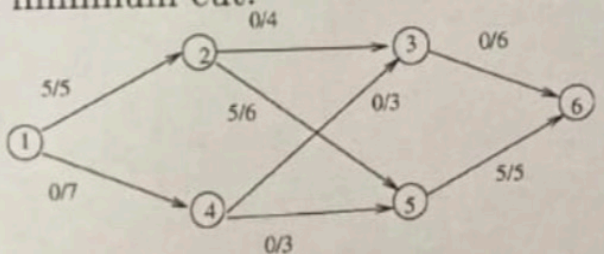
2. (3p) In the following graph, find the maximum flow from vertex 1 to 6. Start from the given flow and maximize it using Ford-Fulkerson algorithm. Also show the minimum cut.



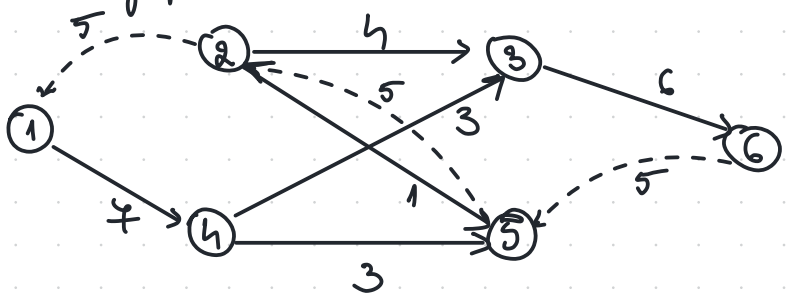
residual



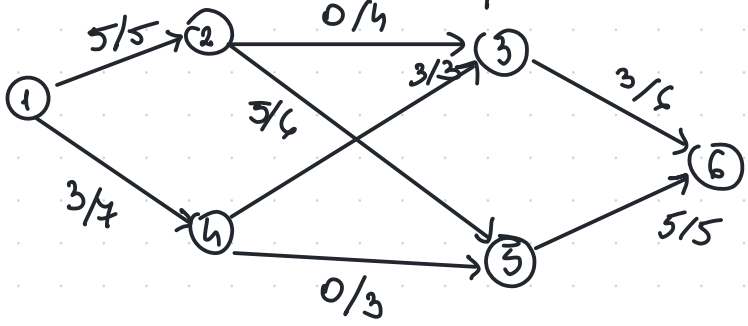
2. (3p) In the following graph, find the maximum flow from vertex 1 to 6. Start from the given flow and maximize it using Ford-Fulkerson algorithm. Also show the minimum cut.

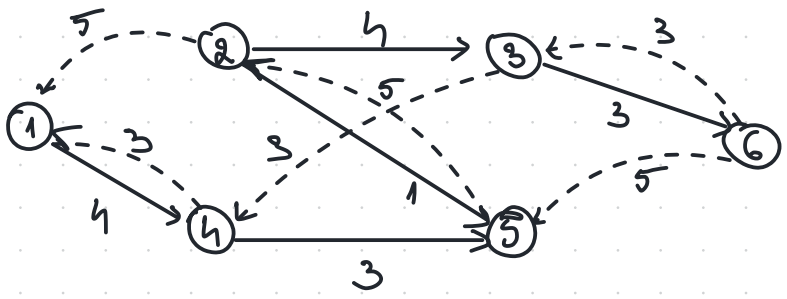


residual graph

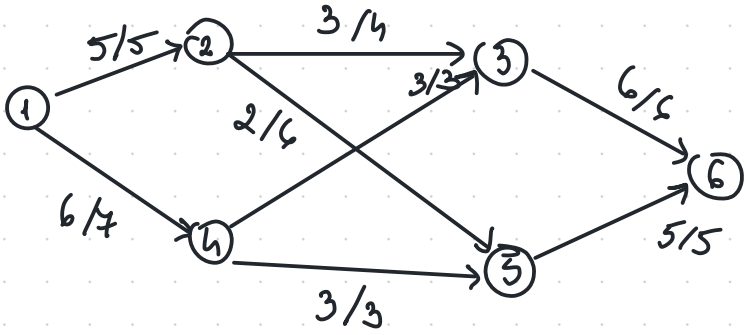


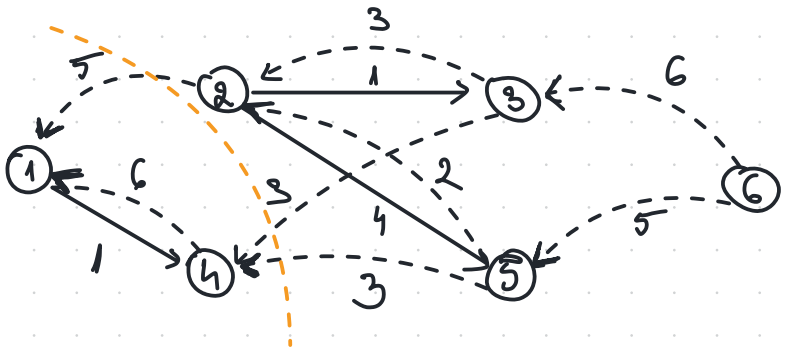
1 – 4 – 3 – 6 cap 3



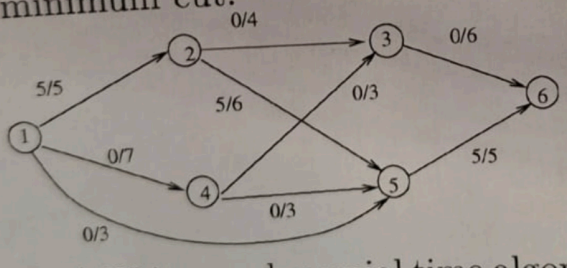


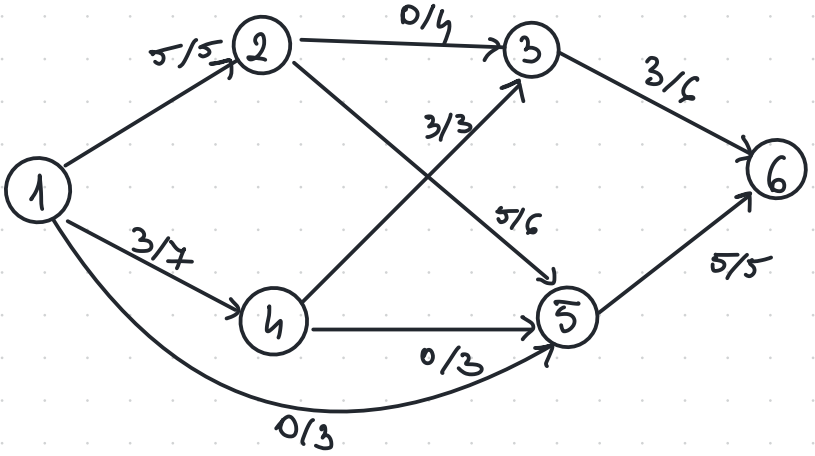
1 – 4 – 5 – 2 – 3 – 6 cap 3

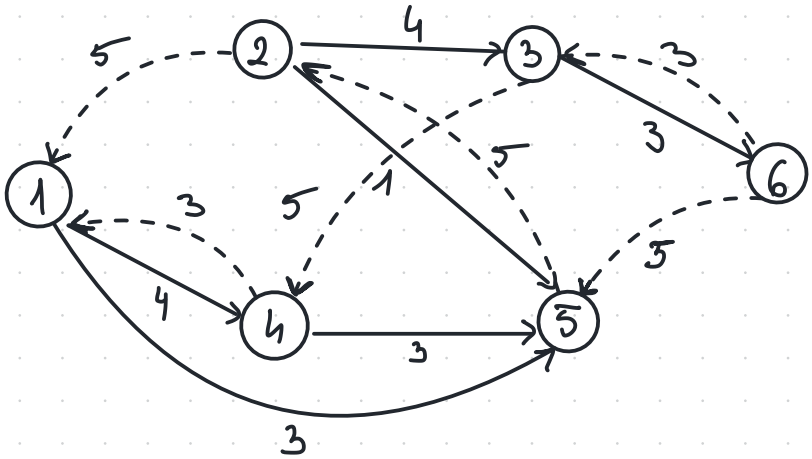




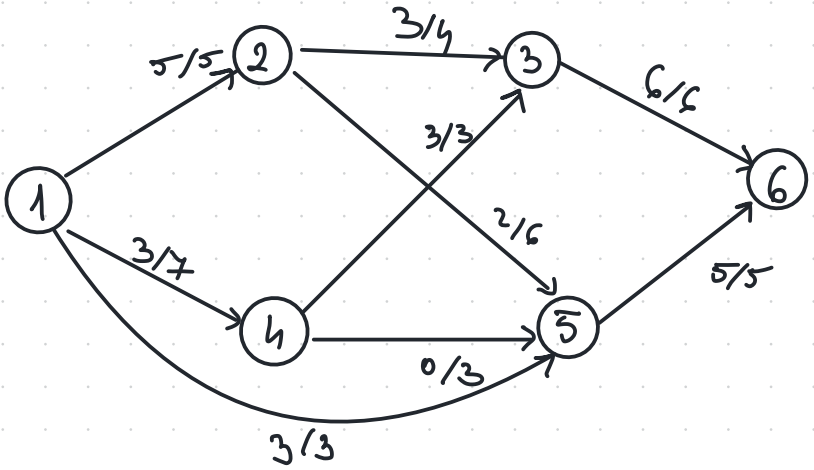
2. (3p) In the following graph, find the maximum flow from vertex 1 to 6. Start from the given flow and maximize it using Ford-Fulkerson algorithm. Also show the minimum cut.

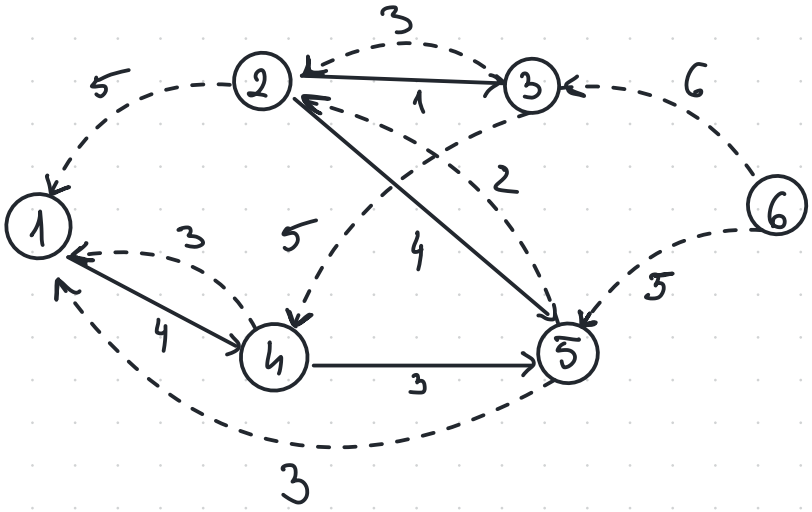




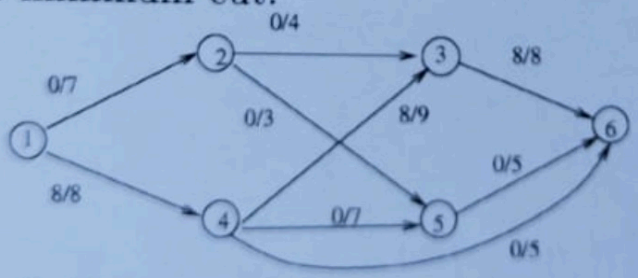


1 – 5 – 2 – 3 – 6 cap 3

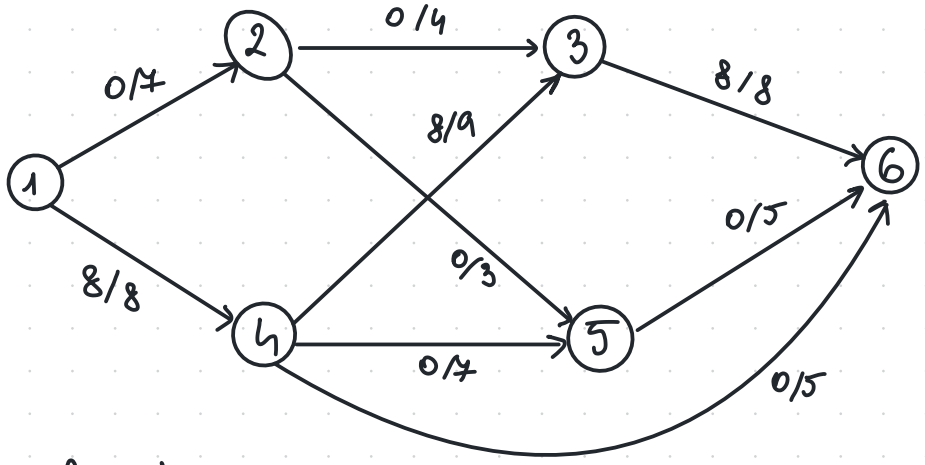

1
2
3
4
5
6
5/5
3/4
6/6
3/3
2/6
3/7
5/5
0/3
3/3


1
2
3
4
5
6
3
5
1
6
2
3
5
4
4
5
3
3

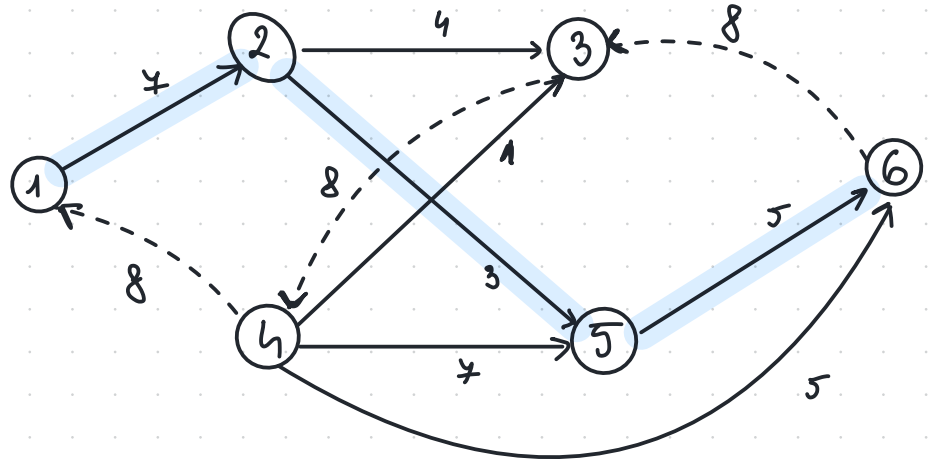
2. (3p) In the following graph, find the maximum flow from vertex 1 to 6. Start from the given flow and maximize it using Ford-Fulkerson algorithm. Also show the minimum cut.



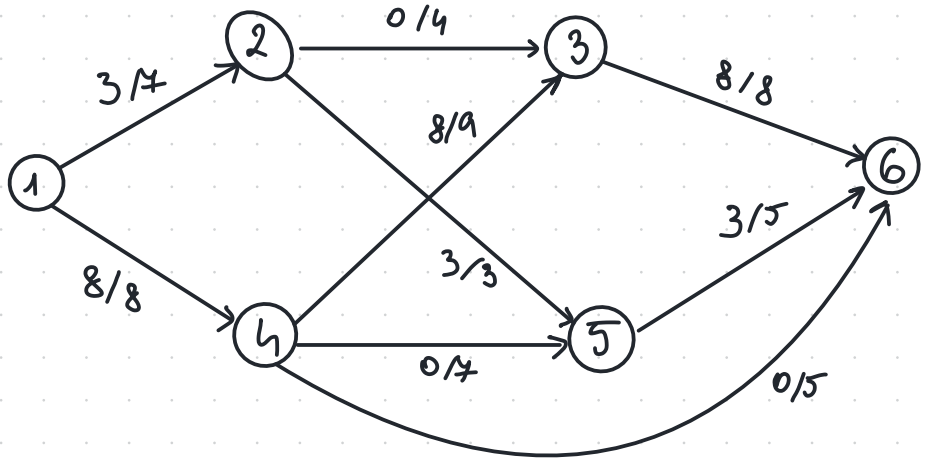
3 (3p) Write a polynomial time algorithm that ...



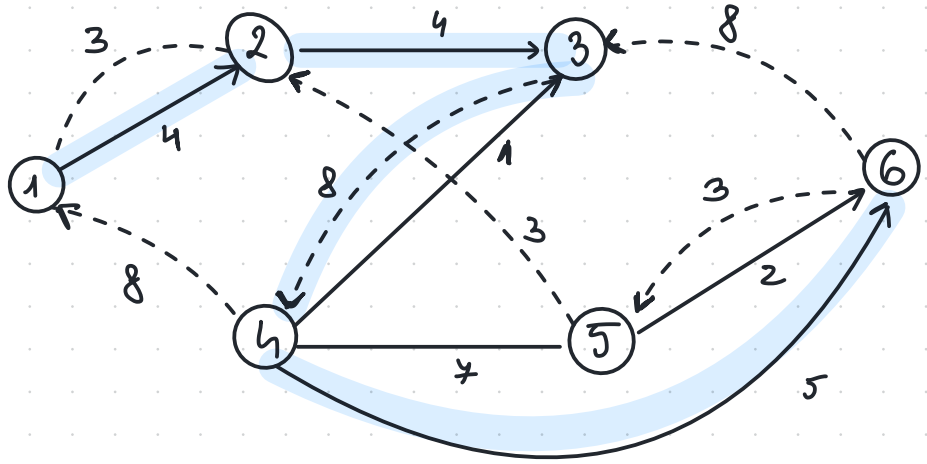
residual graph



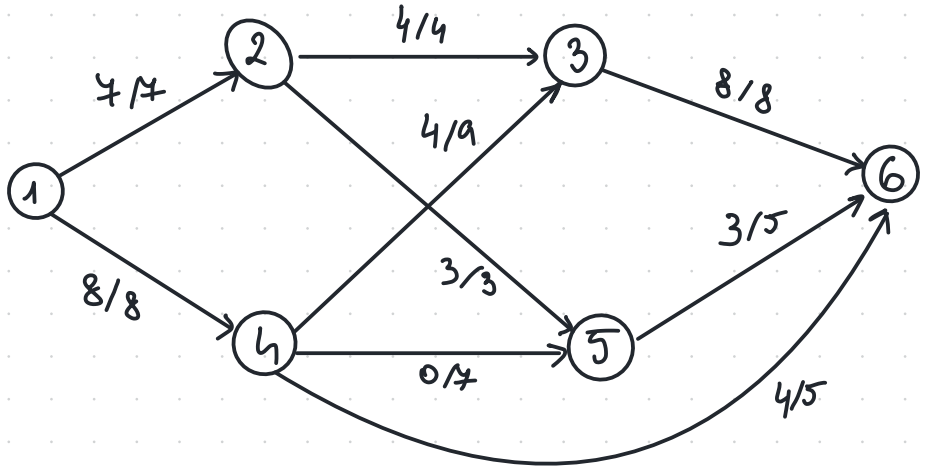
choose 1 − 2 − 5 − 6 capacity 3



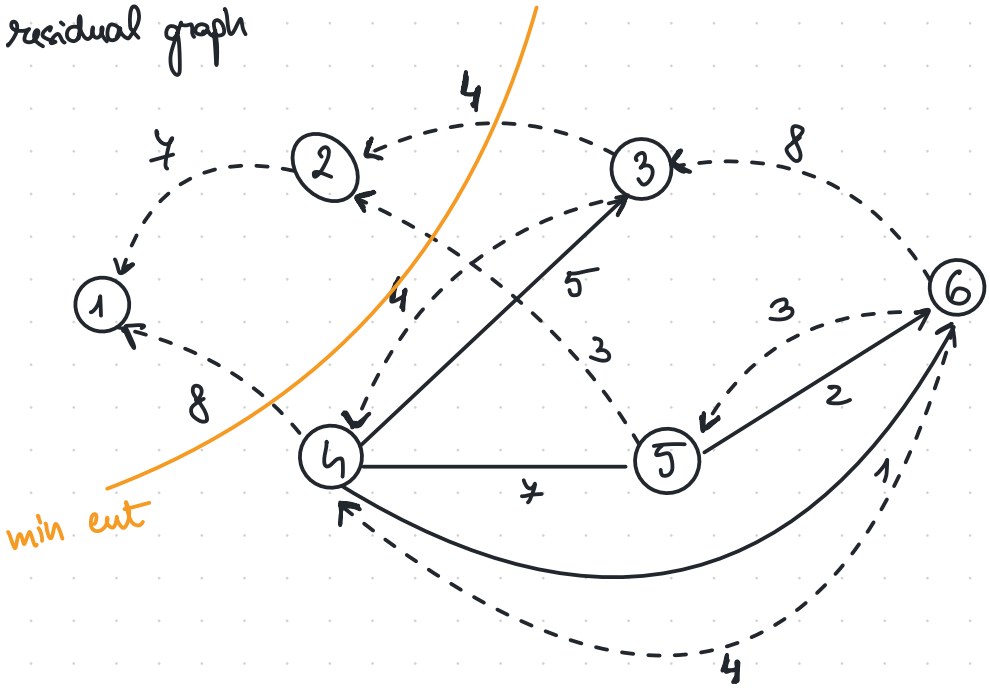
residual graph



choose 1 – 2 – 3 – 4 – 6 cap 4



residual graph



flow : 8 + 3 + 4 = 15