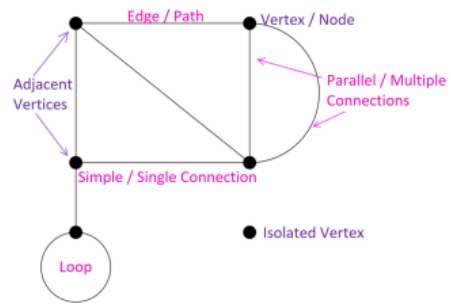
Networks

Vertex (plural Vertices)
An object, represented with a dot.

Edge
A connection between two vertices represented with a line or an arrow.

Loop
An edge that connect from an vertex to itself.

Graph
A collection of vertices that are connected (or not) to each other using edges in some specific way.

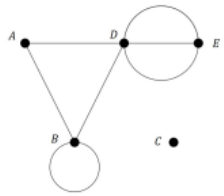


The Degree of a Vertex
The number of edges connected to a vertex.

A loop counts as two edges for the degree.

Vertices are classified as even or odd if their degree is an even number or odd number.

Example



Degrees

$\text{Deg}(A) = 2$
$\text{Deg}(B) = 4$
$\text{Deg}(C) = 0$
$\text{Deg}(D) = 5$
$\text{Deg}(E) = 3$

Handshaking Lemma
Every edge is counted by the degree of two vertices.
Sum of vertex degrees = 2 × number of edges.

A loop counts as one edge in the number of edges.

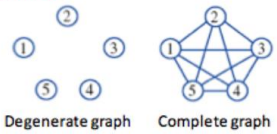
Direction on Graphs

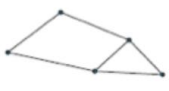
Undirected Graph
A graph where the edge between two vertices acts in both directions.

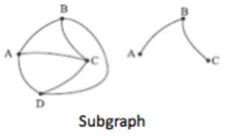
Directed Graph / Digraph
A graph where specific direction is indicated for every edge.
Some vertices may not be reachable from other vertices.

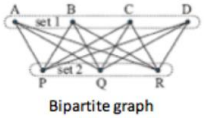
Undirected Graphs

Types of Networks







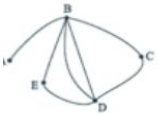


Graph Type	Number of Edges with n vertices
Complete	$\frac{n(n-1)}{2}$
Connected	$n-1$

Adjacency Matrix Representation

Matrix Representation

Networks can be represented using adjacency matrices. The numbers on the leading diagonal represent loops, and all undirected graphs are symmetrical about the leading diagonal.

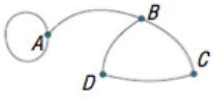


From \ To	A	B	C	D	E
A	0	1	0	0	0
B	1	0	1	2	1
C	0	1	0	1	0
D	0	2	1	0	1
E	0	1	0	1	0

Loops

A loop in an undirected network adds **two** to the degree of a vertex, and adds **one** to the leading diagonal of a matrix. For example:

Node *A* has degree 3.



	A	B	C	D
A	1	1	0	0
B	1	0	1	1
C	0	1	0	1
D	0	1	1	0

The adjacency matrix A of a graph is an $n \times n$ matrix in which, for example, the entry in row C and column F is the number of edges joining vertices C and F.
A loop is a single edge connecting a vertex to itself.
Loops are counted as one edge.

Planar Graphs & Euler's Formula

Planar Graphs

Planar Graph: A graph that can be drawn in such a way that no two edges meet (or have common points), except at the vertices where they are both incident, is called a planar graph.

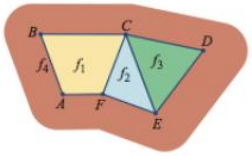
- Some graphs can be redrawn to be planar, others not.
- Euler's formula is used to confirm whether graphs are planar or not.
- All simple graphs with **four** or **fewer vertices** are planar.

Euler's Formula

Euler's Formula:

$$v - e + f = 2$$

V = Vertices	E = Edges	F = Faces



Consider the connected planar graph opposite. It has 4 faces, 6 vertices and 8 edges.

$v - e + f = 2$

$6 - 8 + 4 = 2$

$\therefore$ *Euler's formula confirms that this graph is a planar graph*

Vertices	Edges	Faces	Prove
$v = e - f + 2$	$e = v + f - 2$	$f = e - v + 2$	$v + f - e = 2$

Euler's Formula/Degree of a Face

$e + 6 \leq 3v$

Example VCAA 2016 Exam 1 Sample Question 6 / VCAA 2014 Exam 1 Question 7

Consider the following four graphs. How many of the four graphs above are planar?




Graph 1: $v = 5,\ e = 8$ — $8 + 6 \leq 3 \times 5$ — $14 \leq 15$ ✔

Graph 2: $v = 5,\ e = 5$ — $5 + 6 \leq 3 \times 5$ — $11 \leq 15$ ✔

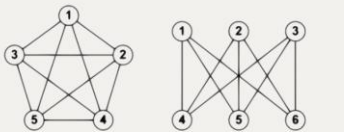
Graph 3: $v = 5,\ e = 7$ — $7 + 6 \leq 3 \times 5$ — $13 \leq 15$ ✔

Graph 4: $v = 5,\ e = 9$ — $9 + 6 \leq 3 \times 5$ — $15 \leq 15$ ✔

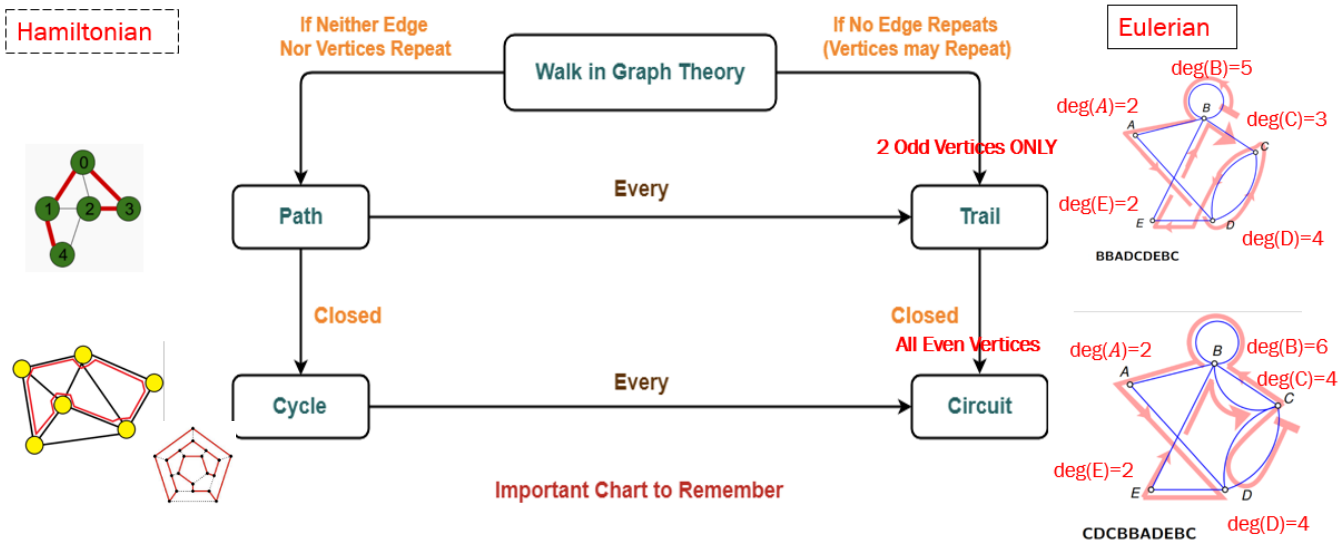
Therefore, all 4 graphs are planar.

Kuratowski's Theorem

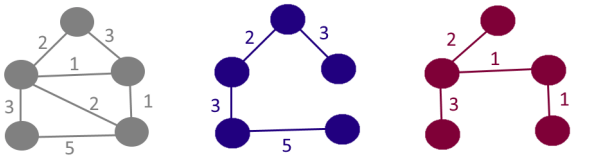
A graph is planar if and only if it does NOT contain a subgraph homeomorphic to K_5 or $K_{3,3}$



Euler & Hamilton



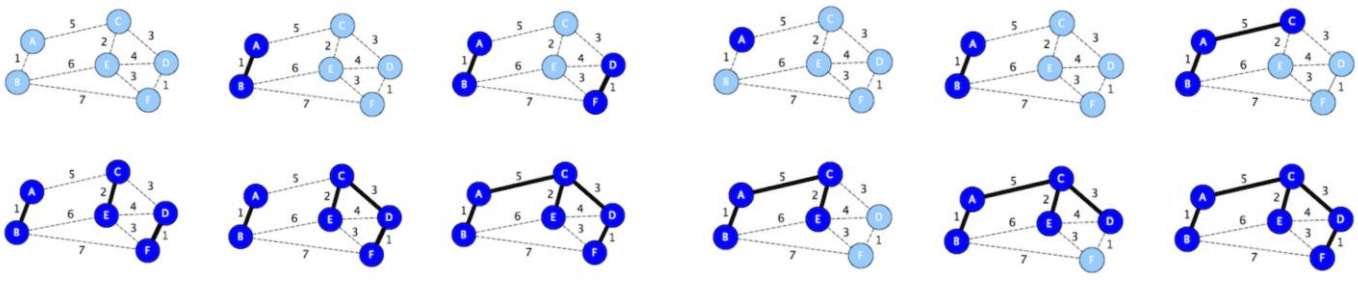
Weighted Graphs



Graph

Spanning Tree
Cost = 13

Minimum Spanning Tree, Cost = 7



Kruskal's Algorithm

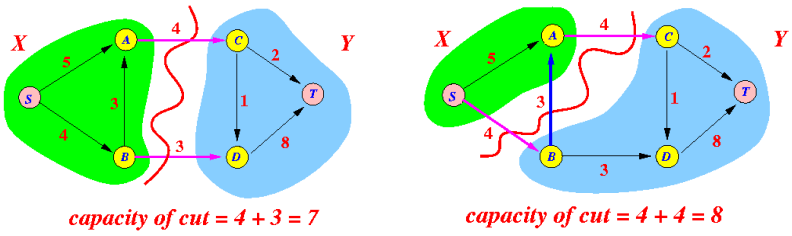
Prim's Algorithm

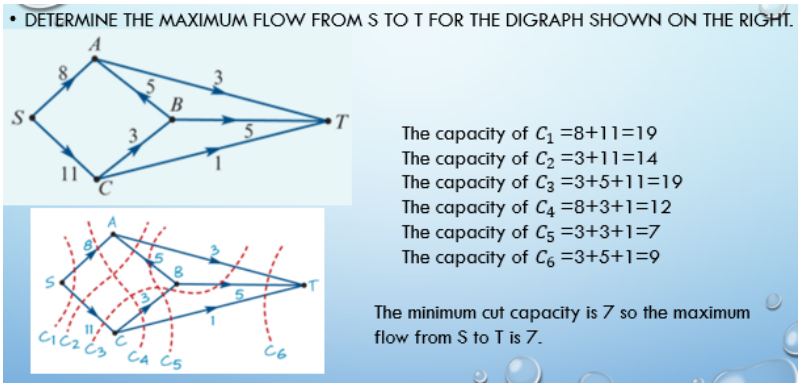
Directed Graphs

Network Flows

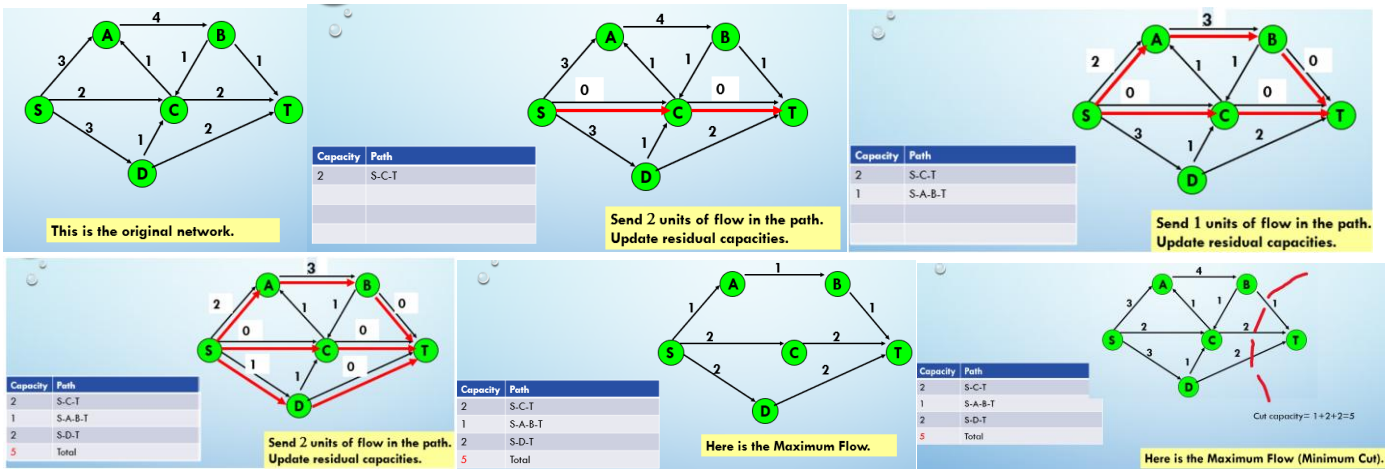
Capacity

• THE MINIMUM CUT CAPACITY = THE MAXIMUM FLOW





Ford Fulkerson Algorithm



Shortest Path Problem

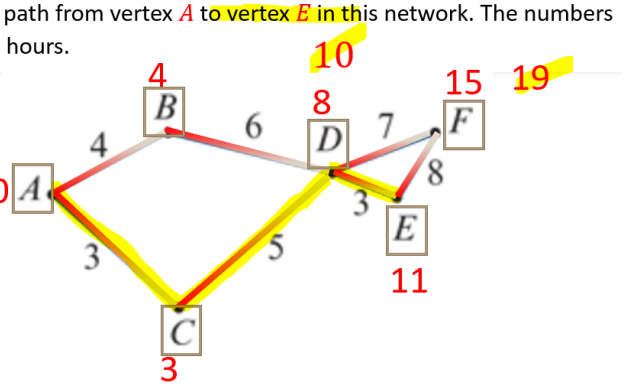
	B	C	D	E	F
A	4	3	X	X	X
C	4	3	8	X	X
B	4	3	8	X	X
D	4	3	8	11	15
E	4	3	8	11	15
F	4	3	8	11	15

A C D E

3 + 5 + 3 = 11 Hours

• Find the shortest path from vertex *A* to vertex *E* in this network. The numbers represent time in hours.

1. From Starting vertex.
2. The shortest edge first.
3. Write distance from the starting vertex.
4. Cover all edges from this vertex.
5. Find the next shortest distance vertex to start over again.
6. Until all vertices and edges covered.



3+5+3=11 Hours

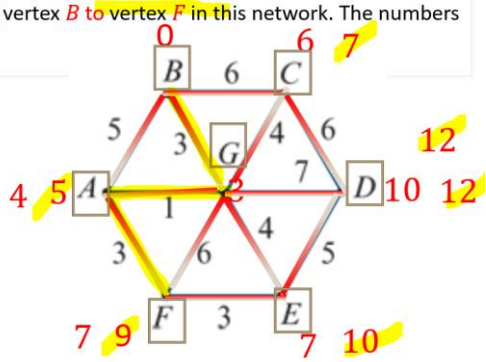
	A	C	D	E	F	G
B	5	6	X	X	X	3
G	4	6	10	7	9	3
A	4	6	10	7	7	3
C	4	6	10	7	7	3
F	4	6	10	7	7	3
E	4	6	10	7	7	3
D	4	6	10	7	7	3

B G A D

3 + 1 + 3 = 7 minutes

• Find the shortest path from vertex *B* to vertex *F* in this network. The numbers represent time in minutes.

1. From Starting vertex.
2. The shortest edge first.
3. Write distance from the starting vertex.
4. Cover all edges from this vertex.
5. Find the next shortest distance vertex to start over again.
6. Until all vertices and edges covered.



3+1+3=7 minutes

Matching & Allocation Problems

Hungarian Algorithm

Example:

Four supermarkets (A, B, C and D) are supplied from four distribution outlets (W, X, Y and Z). The cost in dollars of supplying one vanload of goods is given in the table. This table is called a cost matrix.

	A	*B*	*C*	*D*
W	30	40	50	60
X	70	30	40	70
Y	60	50	60	30
Z	20	80	50	70

The aim is to supply each of the supermarkets at the lowest cost. This can be done by trial and error but that would be time consuming. The Hungarian algorithm gives a method for determining this minimum cost.

Step One

Simplify the cost matrix by **subtracting** the **minimum entry** in each row from each of the elements in that row.

- This process is repeated for columns if there is **no zero entry**.

	A	*B*	*C*	*D*
W	0	10	20	30
X	40	0	10	40
Y	30	20	30	0
Z	0	60	30	50

i.e. 30 is subtracted from all entries in Row 1
30 is subtracted from all entries in Row 2
30 is subtracted from all entries in Row 3
20 is subtracted from all entries in Row 4

Employee	*A*	*B*	*C*	*D*	
Wendy	30	40	50	60	−30
Xenefon	70	30	40	70	−30
Yolanda	60	50	60	30	−30
Zelda	20	80	50	70	−20

Row Minus

	A	*B*	*C*	*D*
W	0	10	10	30
X	40	0	0	40
Y	30	20	20	0
Z	0	60	20	50

Because Row 2 did not contain a zero entry, the process was repeated for column 3.
10 is subtracted from Column 3 to obtain a 0 in Row 2.

Employee	*A*	*B*	*C*	*D*
Wendy	0	10	20	30
Xenefon	40	0	10	40
Yolanda	30	20	30	0
Zelda	0	60	30	50
			−10	

Column Minus

Step Two

Cover the zero elements with the **minimum number of lines**.

- If this minimum number equals the number of rows, then it is possible to obtain a maximum matching using all vertices immediately. Otherwise, continue to step 3.

	A	B	C	D
W	0	10	10	30
X	40	0	0	40
Y	30	20	20	0
Z	0	60	20	50

Employee	A	B	C	D
Wendy	0	10	10	30
Xenefon +10	40	0	0	40 +10
Yolanda	30	20	20	0
Zelda	0	60	20	50

−10 (above) −10 (below)

Intercept Plus & Uncovered Minus

Step Three

	A	B	C	D
W	0	0	0	30
X	50	0	0	50
Y	30	10	10	0
Z	0	50	10	50

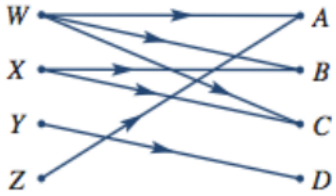
The minimum number of lines is equal to the number of rows, so it is possible to obtain a maximum matching.

4 people = Minimum 4 lines

Step Four

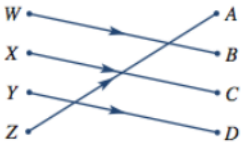
Possible allocations are represented using a **barpitite graph**.

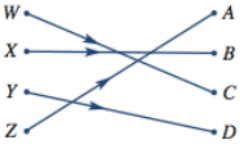
- The edges are chosen through the **zero** entries in the table.



0 in table = A line of Connection

The possibilities with four edges (one task per person) are as follows:





Cost ($)	Cost ($)
W to B = 40	W to C = 50
X to C = 40	X to B = 30
Y to D = 30	Y to D = 30
Z to A = 20	Z to A = 20
Total = 130	**Total = 130**

Critical Path Problems

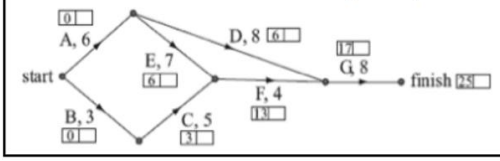
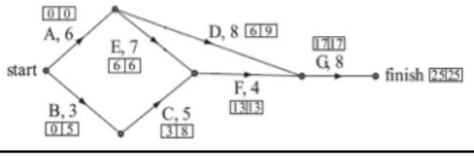
1. Draw a box for each going forward edge/activities. Or |
2. **EST** = Going forward with **LARGEST Number**.
3. **LST** = Going backward with **SMALLEST Number**.
4. **Float time** for each activity = LST — EST @ start of the edge
5. Label / Highlight **Critical Path**
6. Write **minimum completion time**.

1. Boxes or lines at the beginning of each edge including dummy. Or |

2. Forwards scanning with **numbers written on each entry,** take the **LARGEST EST number** when moving forward.

3. Backwards scanning with **smallest LST number going backwards**, take the **SMALLEST LST number** when moving backward.

Earliest and Latest Starting Time and Float Time

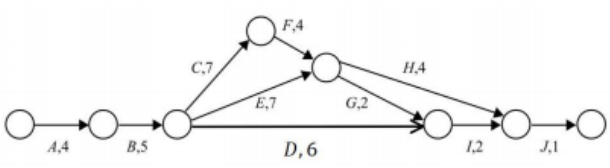
Earliest Starting Time	Latest Starting Time	Float/Slack Time
The earliest starting time refers to the earliest time the activity can commence. The EST for activities without predecessors is zero. The EST for activities should be the **longest** elapsed time	The latest start time is the latest time an activity can be left if the whole project is to be completed on time. Latest event times are established by working backwards through the network.	For **critical activities** the float time is zero For **non-critical activities** the float is worked out using: $Float\ Time = LST - EST$
		**FLOAT TIMES** A: 0 – 0 = 0 F: 13 – 13 = 0 B: 5 – 0 = 5 G: 17 – 17 = 0 C: 8 – 3 = 5 D: 9 – 6 = 3 E: 6 – 6 = 0

Critical Activity/Path

Critical Activity: A critical activity is any task, that if delayed will hold up the earliest project. If LST = EST the activity is said to be critical.
Critical Path: The critical path in a project is the path that has the longest completion time. *In the table, the critical path is: A – E – F – G*

Example Modified VCAA 2001 Question 3

LiteAero Company designs and makes light aircraft for the civil aviation industry. They identify 10 activities required for production of their new model, the MarchFly. A network for this project is shown.



The critical path(s) for this network are $A-B-C-F-H-J$ and $A-B-C-F-G-I-J$
The length (in weeks) of a critical path for this project is 25 weeks.

By using more workers it is possible to speed up some activities. However this will increase costs. Activities which can be reduced in time and the associated increased costs and maximum reduction are shown in Table 3 below. The shortest time in which the aircraft could now be finished is can be found by investigating all the possible paths:

Activity	Cost ($/week)	Max reduction (weeks)
C	6 000	3
D	2 000	2
E	3 000	1
F	4 000	2

Path	Current	$C:-3$	$D:-2$	$E:-1$	$F:-2$	Cost
$A-B-C-F-G-I-J$	25	22	–	–	20	$3\times6000+2\times4000=26000$
$A-B-C-F-H-J$	25	22	–	–	20	$3\times6000+2\times4000=26000$
$A-B-D-I-J$	18	–	16	–	–	$2\times2000=4000$
$A-B-E-G-I-J$	21	–	–	20	–	$1\times3000=3000$
$A-B-E-H-J$	21	–	–	20	–	$1\times3000=3000$

By reducing C, E, F by their maximum amounts, this would reduce the time of this path down to 20 weeks. $A-B-E-G-I-J$ and $A-B-E-H-J$ are now also critical paths. If you don't reduce E then it would take 21 weeks and the critical paths would change. In total this would cost $26\,000+3000=\$29\,000$. Reducing D does not affect the time for the critical path or potential critical paths.

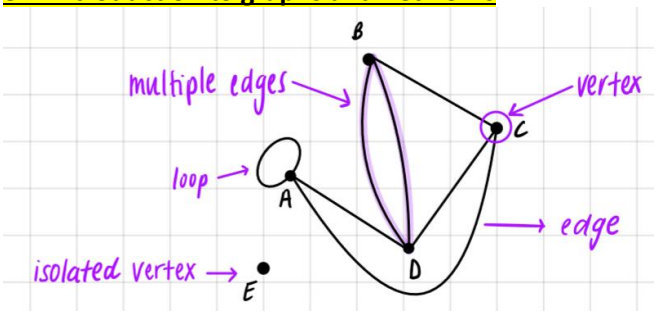
Different Types of Greedy Algorithm

Prim's Minimal Spanning Tree Algorithm
Kruskal's Minimal Spanning Tree algorithm
Dijkstra's Shortest Path Algorithm
Ford-Fulkerson Networks Flows Algorithm
Hungarian Algorithm Matching & Allocation Problems

Mathematical Terminologies

Undirected Graphs		Directed Graphs	
Terminologies	**Algorithm**	**Terminologies**	**Algorithm**
Eulerian trails	Exactly 2 vertices of an odd degree	The Maximum Flow	Ford-Fulkerson Algorithm
Eulerian circuits	All vertices even degree	The Shortest Path	Dijkstra's Algorithm
Hamiltonian paths	Visits all of the vertices in a graph only once	Matching & Allocation Problems	Hungarian Algorithm 8.1
Hamiltonian cycles	Visit All vertices, begin & end @ the same vertex	Critical Path Problems	Forward scanning = Largest Number Backward scanning = Smallest Number Float = LST—EST
Minimal Spanning Tree	Prim's Algorithm, Kruskal's Algorithm		

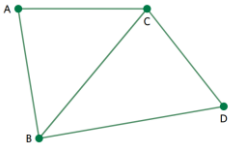
8A: Introduction to graphs and networks



- **Graph:** a diagram used to show connections between groups of things, people of activities.
- **Vertex:** the dots
- **Edge:** line that connects two vertices
- **Isolated vertex:** a vertex that is not connected to an edge
- **Multiple edges:** connects the same vertex using more than one edge
- **Loop:** attach twice to a vertex
- **Degree of a vertex:** the number of edges that attach to a vertex. The degree of vertex D is 4.

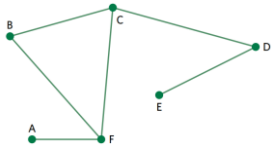
Types of graphs:

Simple graph: no loops or multiple edges

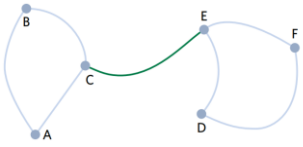


Degenerate graph: all vertices are isolated, there are no edges

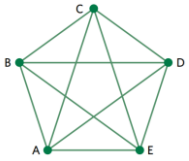
Connected graph: every vertex is connected to every other vertex, either directly or indirectly



Bridge: an edge in a connected graph, that if removed will cause the graph to be **disconnected.** Edge C to E is a bridge

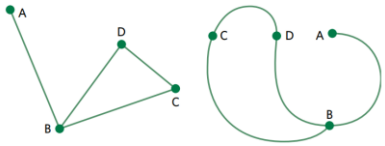


Complete graph: every vertex is connected to every other vertex

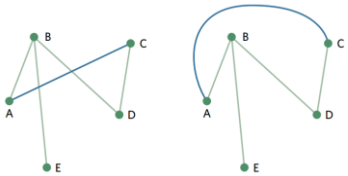


Subgraph: a tree, a part of a larger graph

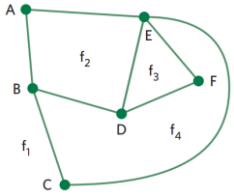
Equivalent/isomorphic graph: graphs that contain the EXACT same information, vertices and the connections between them, they are just drawn differently



Planar graph: a graph that can be re-drawn without any overlapping edges. If it cannot be re-drawn and has overlapping edges, the graph is **non-planar.**



Faces: faces are found in **planar** graphs and are the sections enclosed by edges. There is an infinite face outside of the graph. There is an infinite face outside of the graph.



Euler's rule: there is a relationship between the number of vertices (v), edges (e) and faces (f) in a connected planar graph.

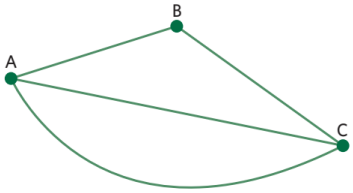
$$v - e + f = 2, \text{where}$$

- v is the number of vertices
- e is the number of edges
- f is the number of faces

8B: Graphs, networks and matrices

Adjacency matrix: a matrix that records the number of connections between vertices. The number of vertices on the graph tells you how many rows and columns you need. A '0' mean no direct connection, a '1' means one connecting edge etc. Ensure you label the matrix with the vertex letters.

$$\begin{array}{c} \begin{matrix} A & B & C \end{matrix} \\ \begin{bmatrix} 0 & 1 & 2 \\ 1 & 0 & 1 \\ 2 & 1 & 0 \end{bmatrix} \end{array} \begin{matrix} A \\ B \\ C \end{matrix}$$



Representing directed graphs: this is a network containing arrows on each edge, signalling one direction. These matrices need to be labelled with 'from' and 'to'.

8C: Travelling

Identifying types of walks: A route is a list of the vertices travelled through, in order, when moving from one vertex to another.
Walk: a continuous sequence of edges that pass through any number of vertices, in any order, starting and finishing at any vertex.
Trail: a walk with no repeated edges. The same vertex can be visited multiple times.
Path: a walk with no repeated edges or vertices.
Circuit: a trail beginning and ending at the same vertex.
Cycle: a path beginning and ending at the same vertex.

Eulerian trails: a walk that includes every edge in a graph exactly once. It must:

- Exactly two vertices of odd degree, the rest are even
- Start and finish at a vertices of odd degree

Eulerian circuit: an Eulerian trail that starts and ends at the same vertex. It must:

- Have all vertices of even degree

Hamiltonian paths: a walk that includes every vertex exactly once, with no repeated edges. Every edge does not need to be included.
Hamiltonian cycle: a Hamiltonian path that starts and ends at the same vertex.

8D: Minium connector problems

Weighted graph: numerical information attached to each edge of a graph. 'Weights' often represent time or distance.

Tree: type of connected graph that has no loops, duplicate edges or cycles. It uses the least number of edges to connect the vertices.

- The number of edges in a tree is always one less than the number of vertices
- e = v – 1
- A tree can be a subgraph and so not all vertices in the larger graph need to be included.

Spanning tree: a tree which connects ALL vertices in the original graph.

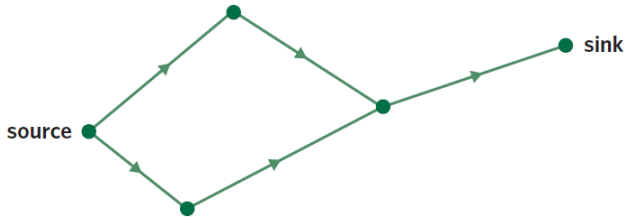
- There are often multiple spanning trees for the one graph

Minimum spanning tree: a spanning tree with the LOWEST TOTAL WEIGHT.

- Prim's algorithm can be used to find the minimum spanning tree.

8E: Flow Problems

Directed graph: network containing arrows on each edge. Networks show directional information between vertices. Weights can represent distance, time or cost.
Flow: flow problems involve the transfer or flow of material from one point (source) to another point (sink), example; water flowing through pipes or traffic flow on roads.



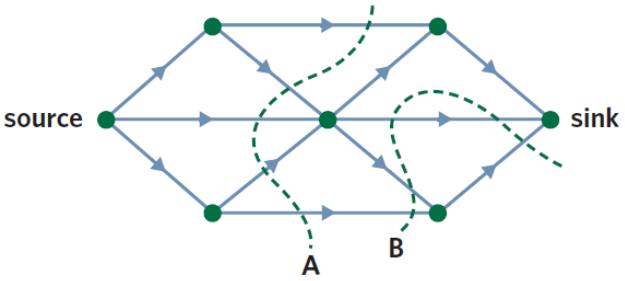
- No matter the situation, things flow in **one direction** only
- Weights of graphs are called capacities

Maximum flow: if edges of different capacities are connected one after, the other, the maximum flow through the edges is equal to the minimum capacity of the individual edges

- Maximum flow = minimum capacity

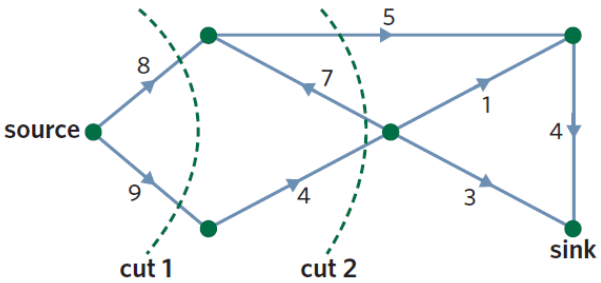
Cuts: a cut divides the network into two parts, completely separating the source from the sink

- It completely stops flow
- Cut A is a successful cut, cut B is not as it doesn't completely separate source from sink



Cut capacity: the sum of all capacities of the edges that the cut passes through, taking into account the direction of flow

- The capacity is only counted if it flows from source to sink



The capacity of cut 1 is $8 + 9 = 17$.

The capacity of cut 2 is $5 + 4 = 9$.

Minimum cut capacity: a cut that has the lowest cut capacity for a network

- Minimum cut capacity = maximum flow

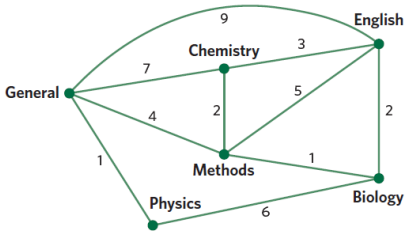
8F: Shortest path problems

Exist so that you can minimise cost, time or distance. Involves finding the shortest path from one vertex to another. You can do this by eye or using Dijkstras algorithm.

Dijkstra's Algorithm

1. Create a table, the starting vertices is in the first row, the rest of the vertices form the column labels
2. Complete the first row
   - Write the distance from the starting vertex to each other vertex in the corresponding column
   - If there is no direct connection, mark with a cross 'X'
   - Find the smallest number in the first row and put a box/square around it (if there are two or more the same, any can be chosen)
   - The column vertex that has the box around it becomes the next row
3. Complete further rows
   - Copy all boxed numbers into the next row
   - Add the boxed number from the row vertex to the column vertex
     - If the value is greater than the value above it, ignore the new number and copy down the smaller one
     - If the value is less than or equal to the value above it, write down the new value
     - If there is no direct connection, mark with a cross 'X'
   - Look for the smallest unboxed number in the row and draw a box around it
   - The column vertex for this new boxed number becomes the next row vertex
4. Repeat step 3 until the destination vertex value has a box around it
5. Backtrack to identify the shortest path and it's length
   - Start at the destination box – this is the length of the shortest path
   - Draw a line up the column to the last number that is the same
   - Look at the row vertex for this number and draw a horizontal line to the column
   - Repeat until you reach the start
   - The horizontal lines indicate the shortest path

Shortest path from General to English:



	P	B	M	C	E
G	1	X	4	7	9
P	1	7	4	7	9
M	1	5	4	6	9
B	1	5	4	6	7
C	1	5	4	6	7

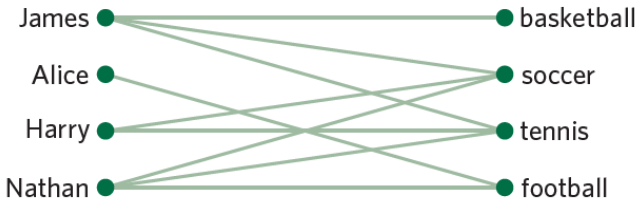
The shortest path is G–M–B–E.

Helpful hints:

- If there is no direct connection and there is a number above, bring it down
- Always bring down the smallest number
- don't forget to add the previous total to your new moves
- if two numbers are the same it doesn't matter which one you bring down

8G: Matching Problems

Bipartite graph: used to show connections between vertices which fall into two separate groups. A vertex cannot be directly connected to another vertex from the same group.



Solving matching problems: the **Hungarian Algorithm** uses cost matrices to find the optimal allocation, the one which gives the minimum cost.

Hungarian Algorithm steps:

1. Locate the lowest value in each row and subtract that from each element in the row
2. Find the minimum number of lines required to cover all of the zeros (if it is the same as the number of subjects then skip to step 7) (it is usually 4 so if you have four lines then skip)
3. Locate the lowest value in each column (it may be zero) and subtract that from each element in the column
4. Find the minimum number of lines requires to cover all of the zeros (skip to step 7 if you have the same number of lines as there are subjects)
5. Locate the smallest value which is not covers by a line. Subtract that from all uncovered elements and add the value to all elements covered by 2 lines
6. Find the minimum number of lines required to cover all the zeros in the table (if it is not the same as the number of subjects repeat step 5)
7. Make the allocation based on the location of the zeros in the matrix
8. Find the minimum time by referring to the initial table costs

8H: Activity Networks and Precedence Tables

Immediate predecessor (IP): is an activity that must be completed before another activity can begin. They are displayed in precedence tables.

Precedence tables: show activities and IP's. The information is used to draw networks.

- Activity networks **no longer label vertices but instead label edges**
- Start/finish vertices get labelled

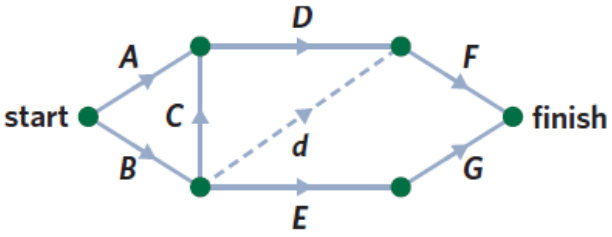
Dummy activities: are required is two activities share **SOME** but **NOT ALL** IP's

- The dummy begins at the end of the shared IP
- The dummy ends at the beginning of the activity that has additional IP's
- Drawn as a dotted line and labelled 'd'

Example:

activity	immediate predecessor(s)
A	–
B	–
C	B
D	A, C
E	B
F	B, D
G	E

- A and B have no IP, therefore they are both come from the start vertex.
- A and C must merge together for D to begin.
- C, E and F all share B, except F also has D as an IP. Therefore, the dummy starts at the end of B (as this is the shared activity) and ends at the beginning of F. This allows for B and D to be brought together so that F can begin.
- F and G are the only activities that aren't an IP, this means that they attach to the finish vertex.



8I: Critical Path Planning:

Scheduling: times that are associated with activities/edges, looks into the minimum overall time to complete a project.

FORWARD SCANNING for EARLIEST START TIME (EST):

- To minimise the total completion time for a project, each activity should begin at its earliest start time (EST)
- Done via process of forward scanning

Steps:

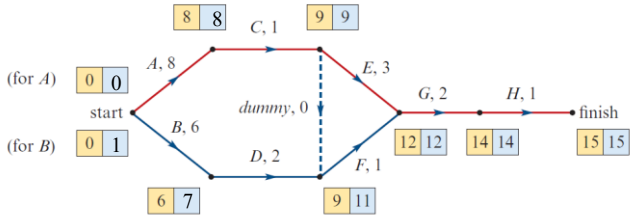
1. Draw a box at each activity of the network, as well as the finish vertex
2. The activities that connect to the start vertex have a 0 in the EST box
3. Start filling in the EST box for all activities and the finish vertex
   i. If an activity has one immediate predecessor, its EST can be found by adding the EST and duration of the immediate predecessor
   ii. If an activity has two or more predecessors, the EST will be the **LARGEST** value
4. The EST for the finish vertex is the minimum possible completion time for the project

BACKWARD SCANNING for LATEST START TIME (LST):

- Some activities may be able to start later than the earliest possible start time and not impact the total minimum completion time for the project.
- Done via backward scanning

Steps:

1. Complete forward scanning (all left hand boxes should be full)
2. Fill in the LST for the finish as the minimum possible completion time (this is the same number as the EST)
3. Continue filling in the LST for all other activities
   i. If an activity has only one activity following it, its LST can be found by subtracting the duration of the activity from the LST of the activity following it
   ii. If an activity has two or more activities following it, its LST will be the **SMALLEST** value
4. The LST for one of the starting activities should be zero.



Critical Path = A-C-E-G-H

Float time: flexibility around the start time of an activity. It is the maximum amount of time an activity can be delayed without impacting the minimum completion time.

- Float time = LST – EST
- LST = latest start time
- EST = earlier start time

Critical path planning: an activity is on the 'critical path' if it has a float time of zero. A delay in these activities will increase completion time. It is possible for there to be more than one critical path.

8J: Crashing

- Used to reduce the completion time of an activity network/project
- Crashing an activity **on** the critical path will immediately impact the minimum completion time
- Crashing off the critical path is sometimes still necessary to alter the minimum completion time
- Crashing can cause the critical path/s to change
- An extra cost may be applied when shortening the completion time of a new project

8.1 Using "gm_net" Hungarian Algorithm

$$cost\_matrix := \begin{bmatrix} 5. & 5. & 9. & 7. & 4. \\ 8. & 7. & 5. & 7. & 4. \\ 5. & 6. & 7. & 9. & 4. \\ 8. & 7. & 5. & 8. & 4. \\ 7. & 4. & 9. & 5. & 3. \end{bmatrix}$$

allocate(cost_matrix)

▸Allocation 1. :

"Worker"	"Task"	"Duration"
1.	2.	5.
2.	3.	5.
3.	1.	5.
4.	5.	4.
5.	4.	5.

▸Allocation 2. :

"Worker"	"Task"	"Duration"
1.	2.	5.
2.	5.	4.
3.	1.	5.
4.	3.	5.
5.	4.	5.

▸Minimum cost: 24.

$$cost\_matrix := \begin{bmatrix} 12. & 8. & 16. & 9. \\ 10. & 7. & 15. & 10. \\ 11. & 10. & 18. & 12. \\ 10. & 14. & 16. & 11. \end{bmatrix}$$

hungarian(cost_matrix)

▸Given cost matrix:

"Task"	1.	2.	3.	4.
"Worker 1."	12.	8.	16.	9.
"Worker 2."	10.	7.	15.	10.
"Worker 3."	11.	10.	18.	12.
"Worker 4."	10.	14.	16.	11.

▸Step 1: Row reduction

"Task"	1.	2.	3.	4.	"Min"
"Worker 1."	4.	0.	8.	1.	8.
"Worker 2."	3.	0.	8.	3.	7.
"Worker 3."	1.	0.	8.	2.	10.
"Worker 4."	0.	4.	6.	1.	10.

▸Step 2: Column reduction

"Task"	1.	2.	3.	4.
"Worker 1."	4.	0.	2.	0.
"Worker 2."	3.	0.	2.	2.
"Worker 3."	1.	0.	2.	1.
"Worker 4."	0.	4.	0.	0.
"Min"	0.	0.	6.	1.

▸Step 3. : Update by the min value of 1.

"Task"	1.	2.	3.	4.
"Worker 1."	3.	0.	1.	0.
"Worker 2."	2.	0.	1.	2.
"Worker 3."	0.	0.	1.	1.
"Worker 4."	0.	5.	0.	1.

▸Step 4. : Number of lines equals number of workers, ready for allocation

▸Optimal allocations

"Task"	1.	2.	3.	4.
"Worker 1."	0.	0.	0.	1.
"Worker 2."	0.	1.	0.	0.
"Worker 3."	1.	0.	0.	0.
"Worker 4."	0.	0.	1.	0.

▸Step 5. : Calculate minimum cost

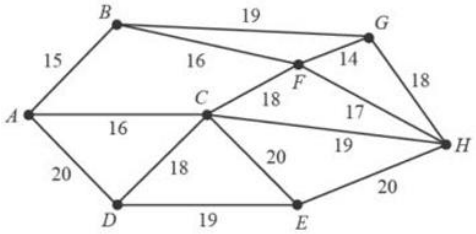
9.+7.+11.+16. =43.

▸Summary:

"Worker"	"Task"	"Duration"
1.	4.	9.
2.	2.	7.
3.	1.	11.
4.	3.	16.

Min Cost = 43.

8.2 Using "gm_net" Prim's Algorithm



graph:={ab15c16d20,bg19f16,ca16f18h19e20d18,da20c18e19,ed19c20h20,fb16g14c18h17,gh19f14h18,hg18f17c19e20}

prim(graph)

▸Working:

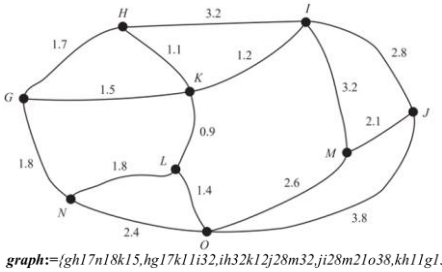
"Edge"	"Length"
"AB"	15.
"BF"	16.
"FG"	14.
"AC"	16.
"FH"	17.
"CD"	18.
"DE"	19.

▸Solution:

115.

Done

8.3 Using "gm_net" Dijkstra's Algorithm



graph:={gh17n18k15,hg17k11i32,ih32k12j28m32,ji28m21o38,kh11g15i12l9,ln18k9o14,mj21i32o26,ng18l18o24,on24l14m26j38}

dijkstra(graph,g,m)

▸Dijkstra:

▸Working:

"Iter"	"Visited"	"G"	"H"	"I"	"J"	"K"	"L"	"M"	"N"	"O"
0.	"G"	0.	∞	∞	∞	∞	∞	∞	∞	∞
1.	"K"	0.	17.	∞	∞	15.	∞	∞	18.	∞
2.	"H"	0.	17.	27.	∞	15.	24.	∞	18.	∞
3.	"N"	0.	17.	27.	∞	15.	24.	∞	18.	∞
4.	"L"	0.	17.	27.	∞	15.	24.	∞	18.	42.
5.	"I"	0.	17.	27.	∞	15.	24.	∞	18.	38.
6.	"O"	0.	17.	27.	55.	15.	24.	59.	18.	38.
7.	"J"	0.	17.	27.	55.	15.	24.	59.	18.	38.
8.	"M"	0.	17.	27.	55.	15.	24.	59.	18.	38.

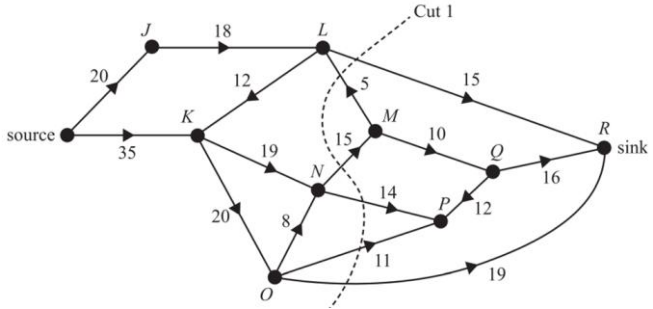
▸Result:

Minimum Distance: 59.

Optimal Path: GKIM

Done

8.4 Using "gm_net" Maximum Flow



graph:={sj20k35,jl18,kn19o20,lk12r15,mq10l5,nm15p14,on8p11r19,p,qr16p12,r0}

flow(graph,s,r)

▸Maximum Flow:

44.

▸Minimum Cut:

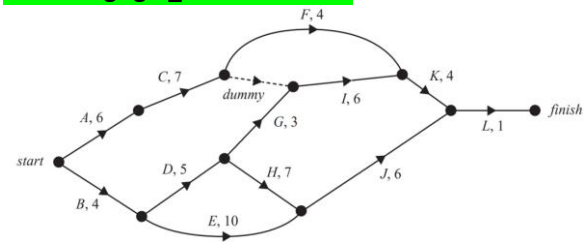
"Edge"	"Cap"	"New Flow"	"ΔFlow"
"LR"	23.	52.	8.
"MQ"	15.	49.	5.
"OR"	20.	45.	1.

▸Reversible Edges:

"Edge"	"New Flow"	"ΔFlow"
"QP"	50.	6.

Done

8.5 Using "gm_net" Float Time



graph*:=*{a6,b4,c7a,d5b,e10b,f4c,g3d,h7d,i6cg,j6he,k4fi,l1jk}

float_time(graph)

▸Float Time:

"Activity"	"A"	"B"	"C"	"D"	"E"	"F"	"G"	"H"	"I"	"J"	"K"	"L"	"Finish"
"EST"	0.	0.	6.	4.	4.	13.	9.	9.	13.	16.	19.	23.	24.
"LST"	0.	1.	6.	5.	7.	15.	10.	10.	13.	17.	19.	23.	24.
"Float"	0.	1.	0.	1.	3.	2.	1.	1.	0.	1.	0.	0.	0.

▸Non−Critical Acitivities:

{"B","D","E","F","G","H","J"}

Total: 7.

▸Critical Activities:

{"A","C","I","K","L"}

Total: 5.

▸Critical Path(s):

ACIKL

▸Critical Time:

24.

Done

8.6 Using "gm_net" Dummy Activity

Activity	Immediate predecessor(s)
A	–
B	–
C	–
D	*A*
E	*B, F*
F	*C*
G	*C*
H	*D, E*
I	*F*
J	*E, G, I*
K	*H, J*

graph*:=*{a,b,c,da,ebf,fc,gc,hde,"if",jegi,khj}

dummy(graph)

▸Dummy Activities:

"End"	"Start"
"E"	"H"
"E"	"J"
"F"	"E"
"Total"	3.

Warning: This program assumes multiple distinct activities can start and end at the same nodes.

Done

8.7 Using "gm_net" Project Crashing

crashing(ELG, Tasks, Reduction(s), Budget, Deadline)

Where,

ELG represents an edge labelled graph

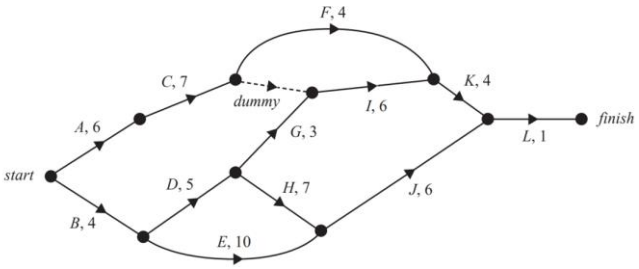
Tasks represents either a matrix or a list containing the activity labels

Reduction(s) represents either a number or a list used to specify the maximum reduction of each activity

Budget represents the available budget

Deadline represents either the target completion time (if positive), the maximum reduction in completion (if negative), or the maximum possible reduction (if a blank string)

Warning: Please enter the starting activities into the list in **alphabetical order** to ensure accuracy.



The maximum decrease in time of any activity is two days.

Activity	*A*	*B*	*F*	*H*	*I*	*K*
Daily cost ($)	1500	2000	2500	1000	1500	3000

The managers of the project have a maximum budget of $15 000 to reduce the time for several activities to produce the maximum reduction in the project's overall completion time.

graph*:=*{a6,b4,c7a,d5b,e10b,f4c,g3d,h7d,i6cg,j6he,k4fi,l1jk}

$$crashing\left(graph,\begin{bmatrix} a & 1500. \\ b & 2000. \\ f & 2500. \\ h & 1000. \\ i & 1500. \\ k & 3000. \end{bmatrix},2,15000,"\square"\right)$$

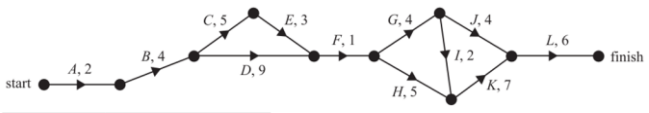
▸Working:

"Path"	"Start"	"↓2A"	"↓2B"	"↓2H"	"↓2I"	"↓1K"
"BDGIKL"	23	23	21	21	19	18
"ACIKL"	24	22	22	22	20	19
"ACFKL"	22	20	20	20	20	19
"BEJL"	21	21	19	19	19	19
"BDHJL"	23	23	21	19	19	19
"□"	"□"	"□"	"□"	"□"	"□"	"□"
"Crit Time"	24	23	22	22	20	19
"Min Cost"	0	3000	7000	9000	12000	15000

▸Solution:

"Cost"	"Time"	*a*	*b*	*f*	*h*	*i*	*k*
15000.	19.	2.	2.	0.	2.	2.	1.

Done



Activity	Weekly cost ($)
C	3000
D	2000
G	2500
H	1000
K	4000

The completion time for each of these five activities can be reduced by a maximum of two weeks. Fencedale High School requires the overall completion time for the renovation project to be reduced by four weeks at minimum cost.

graph*:=*{a2,b4a,c5b,d9b,e3c,f1ed,g4f,h5f,i2g,j4g,k7ih,l6jk}

$$crashing\left(graph,\begin{bmatrix} c & 3000 \\ d & 2000 \\ g & 2500 \\ h & 1000 \\ k & 4000 \end{bmatrix},2,\infty,\text{-}4\right)$$

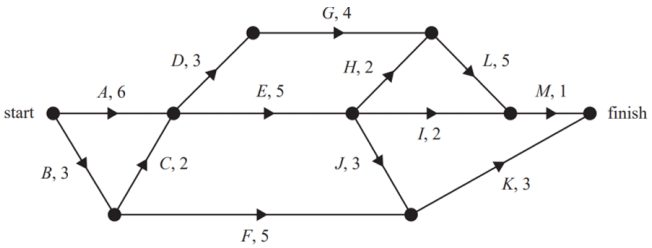
▸Working:

"Path"	"Start"	"↓1D"	"↓2G"	"↓1H"	"↓1K"
"ABDFHKL"	34	33	33	32	31
"ABCEFHKL"	33	33	33	32	31
"ABDFGIKL"	35	34	32	32	31
"ABCEFGIKL"	34	34	32	32	31
"ABDFGJL"	30	29	27	27	27
"ABCEFGJL"	29	29	27	27	27
"□"	"□"	"□"	"□"	"□"	"□"
"Crit Time"	35	34	33	32	31
"Min Cost"	0	2000	7000	8000	12000

▸Solution:

"Cost"	"Time"	c	d	g	h	k
12000.	31.	0.	1.	2.	1.	1.

Done



Activity	Weekly cost ($)
A	140000
E	100000
F	100000
L	120000
K	80000

The completion time for each of these five activities can be reduced by a maximum of two weeks.
The overall completion time for the roadworks can be reduced to 16 weeks.

graph:={a6,b3,c2b,d3ac,e5ac,f5b,g4d,h2e,i2e,j3e,k3fj,l5gh,m1li}

$$crashing\left(graph,\begin{bmatrix} a & 140 \\ e & 100 \\ f & 100 \\ l & 120 \\ k & 80 \end{bmatrix},2,\infty,16\right)$$

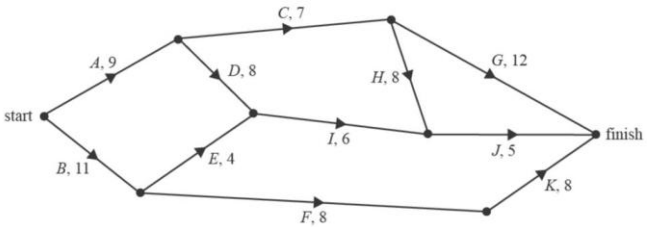
▸Working:

"Path"	"Start"	"↓1A"	"↓2L"
"BCEIM"	13	13	13
"AEIM"	14	13	13
"BCEHLM"	18	18	16
"AEHLM"	19	18	16
"BCDGLM"	18	18	16
"ADGLM"	19	18	16
"BCEJK"	16	16	16
"AEJK"	17	16	16
"BFK"	11	11	11
"□"	"□"	"□"	"□"
"Crit Time"	19	18	16
"Min Cost"	0	140	380

▸Solution:

"Cost"	"Time"	a	e	f	l	k
380.	16.	1.	0.	0.	2.	0.

Done



A reduction in completion time of an activity will incur an additional cost of $10000 per week.

Activities can be reduced by a maximum of two weeks.

The minimum number of weeks an activity can be reduced to is seven weeks.

What is the minimum amount the owners of the supermarket will have to pay to reduce the completion time of the project as much as possible?

graph:={a9,b11,c7a,d8a,e4b,f8b,g12cde,h8cde,i6de,j5hi,k8f}

$$crashing\left(graph,\{a,b,d,f,h,g,k,10000\},\{2,2,1,1,1,2,1\},\infty,\text{"□"}\right)$$

▸Working:

"Path"	"Start"	"↓2A"	"↓1B"	"↓1D"	"↓1H"
"BFK"	27	27	26	26	26
"BEIJ"	26	26	25	25	25
"ADIJ"	28	26	26	25	25
"BEHJ"	28	28	27	27	26
"ADHJ"	30	28	28	27	26
"ACHJ"	29	27	27	27	26
"BEG"	27	27	26	26	26
"ADG"	29	27	27	26	26
"ACG"	28	26	26	26	26
"□"	"□"	"□"	"□"	"□"	"□"
"Crit Time"	30	28	28	27	26
"Min Cost"	0	20000	30000	40000	50000

▸Solution:

"Cost"	"Time"	a	b	d	f	h	g	k
50000.	26.	2.	1.	1.	0.	1.	0.	0.

Done

8A: Review of Graphs and Networks

graph: a diagram used to show connections between groups of things, people or activities

vertex: the dots

edges: lines that connect vertices

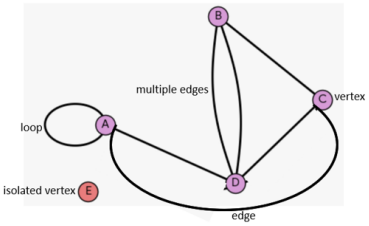
isolated vertex: a vertex that is not connected to an edge

multiple edges: connect the same vertices with more than one edge

loops: attach twice to a vertex

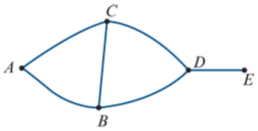
degree of a vertex: is the number of edges that attach to that vertex. A loop counts as 2 degrees.

example: degree (A) = 4, degree (B) = 3, degree (C) = 3, degree (D) = 4

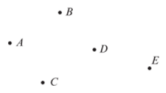


types of graphs:

simple graphs: no loops or multiple edges

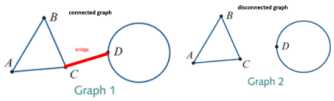


degenerate graph: all vertices are isolated; there are no edges



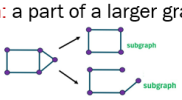
connected graph: every vertex is connected to every other vertex, either directly or indirectly

bridge: an edge in a connected graph, that if removed will cause the graph to be disconnected

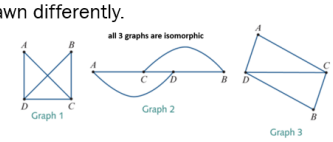


complete graph: every vertex is connected to every other vertex

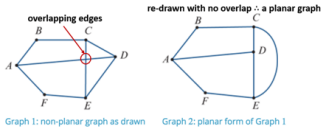
subgraph: a part of a larger graph



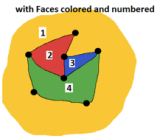
equivalent/isomorphic graphs: graphs that contain the exact same information, vertices and the connections between them, they are just drawn differently.



planar graph: a graph that can be re-drawn without any overlapping edges. If it cannot be re-drawn and has overlapping edges, the graph is non-planar



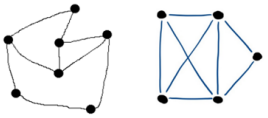
faces: faces are found in planar graphs and are the sections enclosed by edges. There is an infinite face outside of the graph.



Euler's rule: there is a relationship between the number of vertices (v), edges (e) and faces (f) in a connected planar graph.

rule: # vertices — # edges + # faces =2

V — E + F=2



$v = 7, e = 9, f = 4$

$7 - 9 + 4 = 2$ ✓

∴ this graph is planar

$v = 5, e = 8, f = 6$

$5 - 8 + 6 = 3$

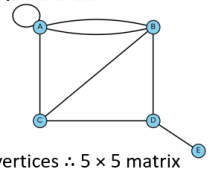
∴ this graph is nonplanar

8B: Using Matrices to represent graphs adjacency matrices

- a matrix that records the number of connections between vertices
- the number of vertices on the graph tells you how many rows and columns you need
- a **'0'** means no direct connection, a **'1'** means one connecting edge
- ensure you label the matrix with the vertex letters

examples:

1. construct an adjacency matrix for the following graph



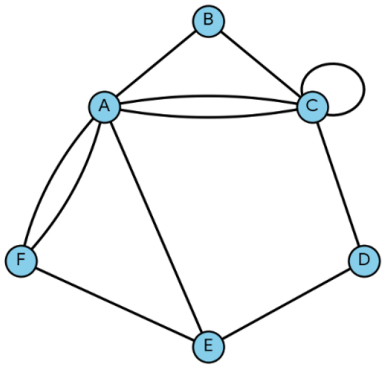
5 vertices ∴ 5 × 5 matrix

	A	B	C	D	E
A	1	2	1	0	0
B	2	0	1	1	0
C	1	1	0	1	0
D	0	1	1	0	1
E	0	0	0	1	0

* note: symmetrical matrix

2. create the graph given the adjacency matrix

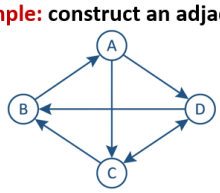
	A	B	C	D	E	F
A	0	1	2	0	1	2
B	1	0	1	0	0	0
C	2	1	1	1	0	0
D	0	0	1	0	1	0
E	1	0	0	1	0	1
F	2	0	0	0	1	0



representing directed graphs:

- network containing arrows on each edge
- one way direction

example: construct an adjacency matrix



from

	A	B	C	D
A	0	1	0	0
B	0	0	1	1
C	1	0	0	1
D	1	0	1	0

to

*** note: no symmetrical matrix**

also note: some questions may be labelled with 'from' and 'to' differently

to

	A	B
A		
B		

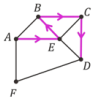
from

8C: Walks, Trails, Paths, Circuits and Cycles

route: a list of vertices travelled through, in order

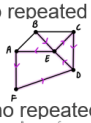
walk: a walk is a sequence of edges linking successive vertices

Example:
A, E, B, C, D



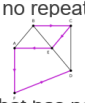
trail: a trail is a walk with no repeated edges

Example:
B, E, C, D, E, A, F, D



path: a path is a walk with no repeated vertices

Example:
B, C, E, A, F, D



circuit: a circuit is a walk that has no repeated edges and starts and ends at the same vertex

Example:
B, C, E, A, F, D, E, B

cycle: a cycle is a walk that has no repeated vertices and starts and ends at the same vertex

Example:
B, A, F, D, E, B

Eulerian Trails and Circuits

Eulerian Trail:

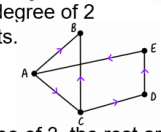
- follows **EVERY** edge
- no repeat edges
- has exactly **2** vertices of odd degree, the rest are even
- will begin at one of the odd vertices and end at the other

Eulerian Circuit:

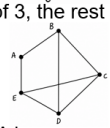
- follows every edge, starts and ends at the same vertex
- no repeat edges
- vertices are **ALL** an even degree
- it can begin at any vertex

examples:

1. ADBCA
All vertices have an even degree of 2
∴ an **Eulerian Circuit** exists.



2. CDEACBA
A and C have an odd degree of 3, the rest are even
∴ an **Eulerian Trail** exists.



3. BCDE are all odd degrees, A is even
∴ neither an Eulerian Trail or Circuit exists.

Hamiltonian Paths and Cycles

Hamiltonian Path:

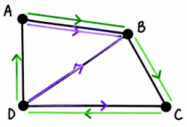
- visits EVERY vertex
- no repeat vertices
- does not need to go through every edge

Hamiltonian Cycle:

- visits EVERY vertex, starts and ends at same vertex
- no repeat vertices
- does not need to go through every edge

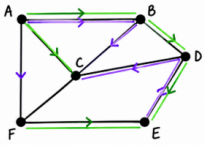
Examples:

1.



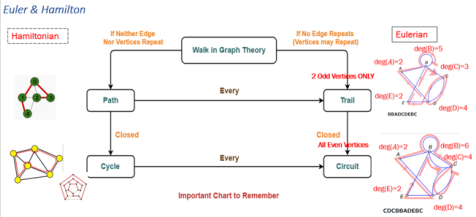
Hamiltonian Path: A B D C
Hamiltonian Cycle: A B C D A

2.



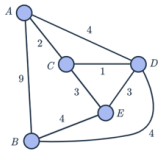
Hamiltonian Path: E D C B A F
Hamiltonian Cycle: A C F E D B A

Euler & Hamilton



8D: Minimum Connector Problems.

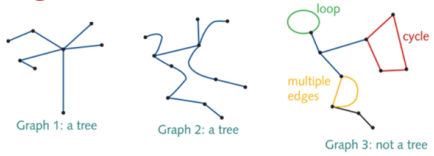
weighted graph: numerical information attached to each edge. 'weights' often represent time or 'distance'.



tree: connected graph

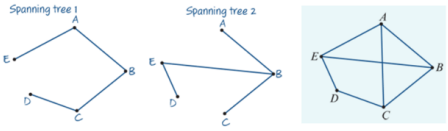
- minimum amount of edges
- no duplicate edges, loops or cycles

 # edges = # vertices − 1



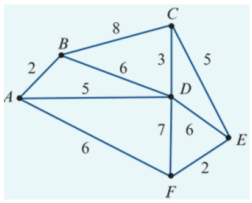
spanning tree:

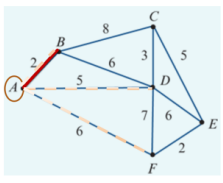
- a tree that connects **all** vertices
- multiple

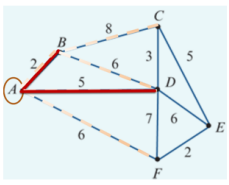


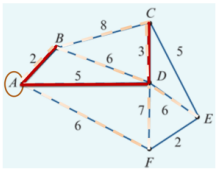
minimum spanning tree: (Prim's algorithm)

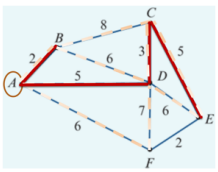
- a spanning tree that considers the weights
- connects all vertices with the **LOWEST** total weight when edges are added.

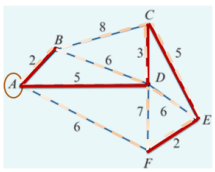




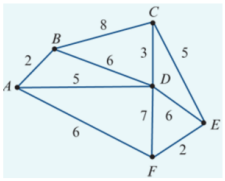








minimum spanning tree: (Kruskal's algorithm)
Pick the smallest edge.



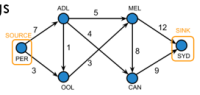
8E: Using networks to model flow

directed graph:

- network containing arrows on each edge
- networks show directional information between vertices
- weights can represent distance, time or cost

flow:

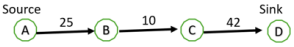
- flow problems involve the transfer or flow of material from one point (**source**) to another point (**sink**)
- eg: water flowing through pipes or traffic flow on roads
- no matter the situation, things flow in **one direction** only
- weights on graphs are called **capacities**



maximum flow:

- if edges of different capacities are connected one after the other, the maximum flow through the edges is equal to the minimum capacity of the individual edges
- maximum flow = minimum capacity

Example 1: what is the maximum flow?



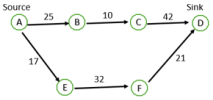
The above network displays **3 connected roads** and the capacities for each road.

* **25** cars can travel along **A** →**B**, however only **10** along **B** →**C**, this would slow traffic reducing flow.

* although **C** → **D** can hold **42** cars, there are only **10** travelling through **B** → **C** at a time

∴ max flow = min capacity = 10 cars

Example 2.



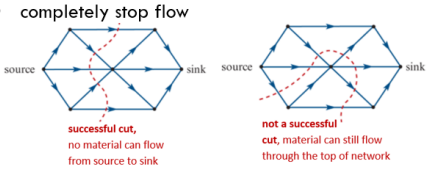
max flow through ABCD = 10

max flow through AEFD = 17

∴ max flow = 10 + 17 = 27 cars

cuts:

- a cut divides the network into two parts, completely separating the source from the sink
- completely stop flow



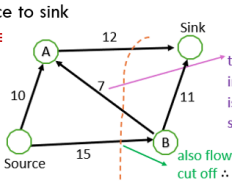
successful cut, no material can flow from source to sink

not a successful cut, material can still flow through the top of network

cut capacity:

- the sum of all capacities of the edges that the cut passes through, taking into account the direction of flow
- the capacity is only counted if it flows from source to sink

example



this capacity is not included as the arrow is flowing from sink to source.

also flow to B has already been cut off ∴ it is not included in the cut capacity

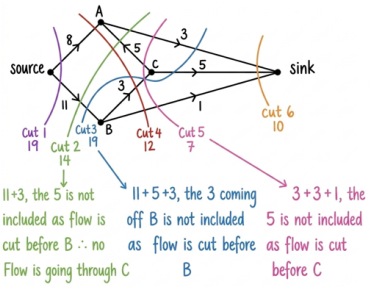
cut capacity = 12 + 15 = 27

minimum cut capacity:

- a cut that has the lowest cut capacity. for a network

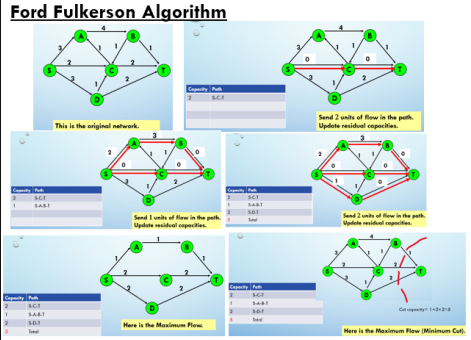
minimum cut capacity = maximum flow

Example: determine the max flow through the network



cut 5 is the minimum cut, 7 is the maximum flow

Ford Fulkerson Algorithm



8F: Dijkstra's Algorithm and Shortest Path Problems

shortest path problems:

- exists so that you can minimise cost, time or distance
- involves finding the shortest path from one vertex to another
- you can do this by eye or using Dijkstra's Algorithm

Dijkstra's Algorithm:

steps:

1. create a table, the **starting vertex** is in the **first row**, the **rest** of the **vertices** form the **column labels**

2. complete the first row

- write the **distance from the starting vertex** to each other vertex in the corresponding column
- if there is no direct connection, mark a cross 'x'
- find the **smallest number** in the first row and put a **box/square** around it (if there are two or more the same, any can be chosen)
- the **column vertex** that has the **box around** it becomes the **next row**

3. complete further rows

- copy all boxed numbers into the next row
- add the boxed number from the row vertex to the column vertex
  - ❖ if the **value is greater** than the value above it, **ignore** the new one and copy down the smaller number
  - ❖ if the **value is less** than or equal to the value above it, **write down** the new value
  - ❖ if there is no direct connection, mark a cross 'x'
- look for the **smallest unboxed number** in the row and draw a **box** around it
- the **column vertex** for this **new boxed number** becomes the **next row vertex**
- repeat until the destination vertex value has a box around it

4. backtrack to identify the shortest path and its length

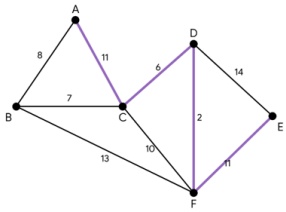
- start at the **destination box** - this is the length of the shortest path
- draw a **line up** the column to the **last number** that is the **same**
- look at the **row vertex** for this number and **draw a horizontal line** to the column
- repeat until you reach the start
- the **horizontal lines** indicate **the shortest path**

helpful hints:

- if there is no direct connection and there is a number above, bring it down
- always bring down the smallest number
- don't forget to add the previous total to your new moves
- if two numbers are the same it doesn't matter which one you bring down

example:

1. find the shortest path from A → E

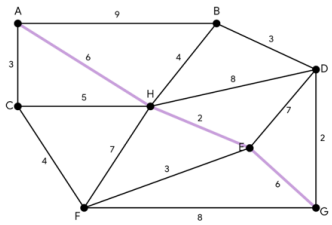


	B	C	D	E	F
A	8	11	x	x	x
B	8	11	x	x	21
C	8	11	17	x	21
D	8	11	17	31	19
F	8	11	17	30	19

30 = shortest distance

shortest path = A C D F E

2. find the shortest path from A to G

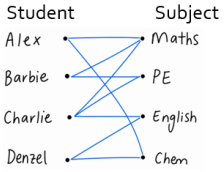


	B	C	D	E	F	G	H
A	9	3	x	x	x	x	6
C	9	3	x	x	7	x	6
H	9	3	14	8	7	x	6
F	9	3	14	8	7	15	6
E	9	3	14	8	7	14	6
B	9	3	12	8	7	14	6
D	9	3	12	8	7	14	6

8G Matching Problems

bipartite graph: Shows connections between two groups

- there is no connection b/w vertices of the same group.



The Hungarian algorithm

Steps:

1. Locate the lowest value in each **row** and **subtract** that from each element in the row
2. Find the minimum number of **lines** required to **cover** all of the **zeros** (if it is the same as the number of subjects then skip to step 7) (it is usually 4 so if you have four lines then skip)
3. Locate the lowest value in each **column** (it may be zero) and **subtract** that from each element in the column
4. Find the minimum number of **lines** requires to **cover** all of the **zeros** (skip to step 7 if you have the same number of lines as there are subjects)
5. Locate the smallest value which is not covers by a line. **Subtract** that from all **uncovered** elements and **add** the value to all elements covered by 2 lines (line **intersection**)
6. Find the minimum number of **lines** required to **cover** all the **zeros** in the table (if it is not the same as the number of subjects repeat 5)
7. Make the allocation based on the location of the zeros in the matrix (**Bipartite**)
8. Find the **minimum time** by referring to the initial table costs

Row Subtraction

	Employee	A	B	C	D
−30	Wendy	30	40	50	60
−30	Xenefon	70	30	40	70
−30	Yolanda	60	50	60	30
−20	Zelda	20	80	50	70

Employee	A	B	C	D
Wendy	0	10	20	30
Xenefon	40	0	10	40
Yolanda	30	20	30	0
Zelda	0	60	30	50

- Column C does not have a zero.
- 10 has been subtracted from every value in column C.

Column Subtraction

Employee	A	B	C	D
Wendy	0	10	20	30
Xenefon	40	0	10	40
Yolanda	30	20	30	0
Zelda	0	60	30	50

−10

Employee	A	B	C	D
Wendy	0	10	10	30
Xenefon	40	0	0	40
Yolanda	30	20	20	0
Zelda	0	60	20	50

- The zeros can be covered with three lines. This is less than the number of allocations to be made (4).
- Continue to step 5a.

Cover all zero with minimum lines

Employee	A	B	C	D
Wendy	0	10	10	30
Xenefon	40	0	0	40
Yolanda	30	20	20	0
Zelda	0	60	20	50

Not enough lines, Intersection addition & Uncovered Subtraction

- The smallest uncovered element is 10.
- 10 has been added to Xenefon–A and Xenefon–D because these values are covered by two lines.
- 10 has been subtracted from all the uncovered values.

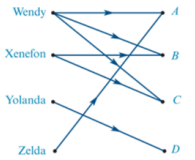
Employee	A	B	C	D
Wendy	0	10	10	30
Xenefon	+10 40	0	0	+10 40
Yolanda	30	20	20	0
Zelda	0	60	20	50

Employee	A	B	C	D
Wendy	0	0	0	30
Xenefon	30	0	0	50
Yolanda	30	10	10	0
Zelda	0	50	10	50

o is the connection line for bipartite graph

- In the bipartite graph:
- Wendy will be connected to A, B and C
- Xenefon will be connected to B and C
- Yolanda will be connected to D
- Zelda will be connected to A.

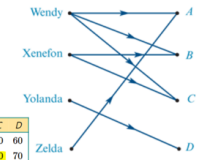
Employee	A	B	C	D
Wendy	0	0	0	30
Xenefon	50	0	0	50
Yolanda	30	10	10	0
Zelda	0	50	10	50



- Zelda must operate machine A (20 minutes).
- Yolanda must operate machine D (30 minutes).
- Wendy can operate either machine B (40 minutes) or C (50 minutes).
- Xenefon can operate either machine B (30 minutes) or C (40 minutes).
- The minimum time taken to finish the work=20+30+50+30=130 minutes.

Start with one connection line

Employee	A	B	C	D
Wendy	30	40	50	60
Xenefon	70	30	40	70
Yolanda	60	50	60	30
Zelda	20	80	50	70



8H Precedence Tables + Networks.

immediate predecessor: (IP)

- an activity that must be completed before another activity can begin
- they are displayed in precedence tables.

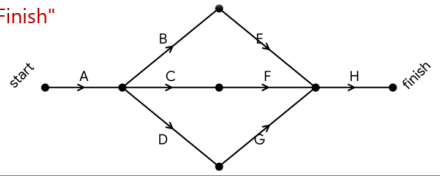
precedence tables:

- shows activities and IP's
- info is used to draw networks
- activity networks no longer label vertices but instead label edges
- start/finish vertices get labelled.

Example 1: draw an activity network from the precedence table

activity	IP	
A	—	it comes off start
B	A	all come off A.
C	A	
D	A	
E	B	
F	C	
G	D	
H	E, F, G	MERGE TOGETHER

H is not an IP ∴ attaches to "Finish"



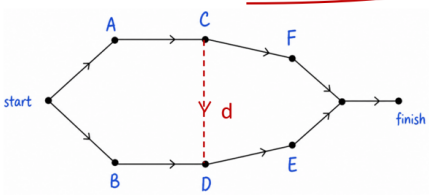
dummy activities:

- required if two activities share SOME but NOT ALL IP's
- begins @ the shared IP
- ends @ the start of the activity that has additional IP's
- drawn as a dotted line, and labelled 'd'.

Example 2:

activity	IP	
A	—	comes off start
B	—	
C	A	
D	A	
E	C, D	share C but not D ∴ they share some but not all IPs!
F	C	
G	E, F	MERGE TOGETHER

C was shared

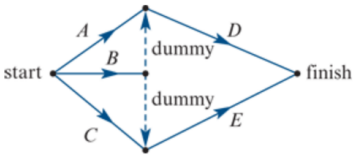


ends @ start of E

example 3:

Activity	Immediate predecessors
A	–
B	–
C	–
D	A, B
E	B, C

End of **B** → start of **D** & **E**

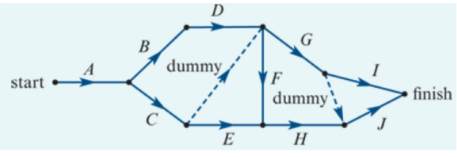


example 4:

Activity	Immediate predecessors
A	—
B	A
C	A
D	B
E	C
F	C, D
G	D, C
H	F, E
I	G
J	H, G

End of **C** → start of **F** & **G**

End of **G** → start of **J**

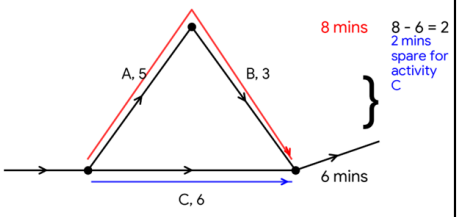


8I: critical path planning

scheduling: times associated with activities/edges, look into minimum overall time to complete a project.

Float time: flexibility around the start time of an activity. It is the max amount of time an activity can be delayed without impacting the min completion time.

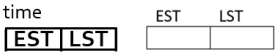
e.g.



Float time = **LST – EST**

LST = latest start time

EST = earliest start time



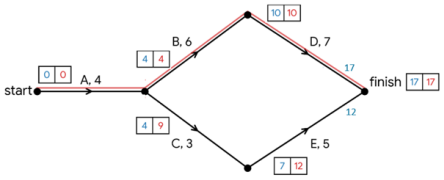
1. Boxes or lines at the beginning of each edge including dummy.

2. Forwards scanning with **numbers written on each entry,** take the **LARGEST EST number** when moving forward.

3. Backwards scanning with **smallest LST number going backwards,** take the **SMALLEST LST number** when moving backward.

FORWARD SCANNING for EST

- activities that connect to the start have a zero in the EST box
- move thru network filling EST box
- if activity has 1 IP, add the previous duration to arrive at the EST
- if 2 or more IPs, the EST is the **LARGEST** value



A B D is critical path

Minimum completion time is 17

B E G is critical path

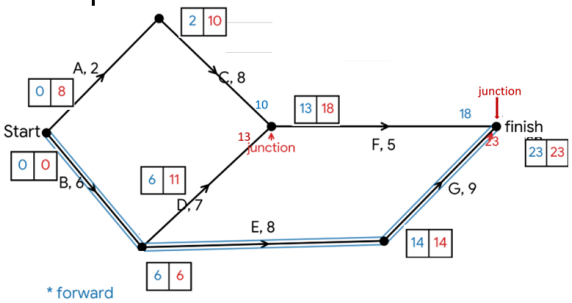
Minimum completion time is 23

BACKWARD SCANNING for LST

- complete EST
- the finish box has the same LST as EST
- fill LST boxes (from right to left)
- subtract duration of activity from LST of activity following it
- if 2 or more IPs activities filling it, its LST is the **SMALLEST** value
- the LST for one of the starting activities should be zero

CRITICAL PATH PLANNING

- an activity is in the critical path if it has a float time of zero
- a delay in these activities will increase completion time
- possible to have more than one

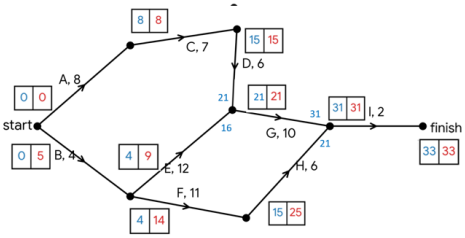


8J: Crashing

- used to reduce the completion time of activity network/project.
- crashing an activity on the critical path alters the min completion time.
- crashing off the critical path can still be done.
- crashing can cause the critical path to change.
- ❖ an extra cost may be applied when shortening the completion time of a network/project.

example:

The network displays a landscaping project. The activities are times for each task in days.

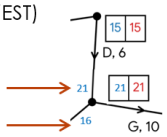


a. determine the critical path:

A C D G I

b. what is the max amount of time activity **D** can be crashed to keep the same critical path.

5 days so the first meeting point of D and E is 16 (EST)



21-16=5

c. A, B, C, D, E can be reduced by $150 a day for up to 2 days.

What is the minimum cost of reducing the network as much as possible?

List all possible routes and starting to reduce from critical path activities.

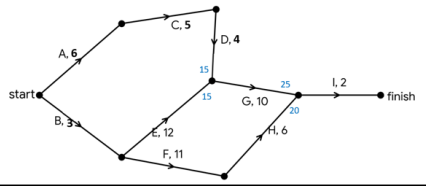
Routes	Current	A ↓ × 2	C ↓ × 2	D ↓ × 2	B ↓ × 1	B ↓ × 1	E ↓ × 2
A-C-D-G-I	33	31	29	27	27	critical	Path
B-E-G-I	28	28	28	28	27	critical	path
B-F-H-I	23	23	23	23	22		

33 → 27 completion time

7 × 150 = $1050

A–C–D = 21 → 15

B–E = 16 → 15

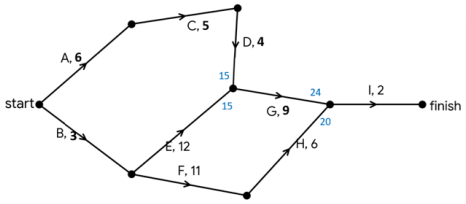


d. Further reductions are made to. either A($140) by 1 day, G ($175) by 1 day or I ($225) by 1 day. Which activity should be reduced?

Routes	New Current	A ↓ × 1 $140	G ↓ × 1 $175	I ↓ × 1 $225
A-C-D-G-I	27	26 ✗	26	
B-E-G-I	27	27	26	
B-F-H-I	22	22	22	

A- there is no point in crashing as B → E =15 and would keep completion time the same.

CRASH G over I as it is cheaper!!



27 → 26 completion time

1 × 175 = $175

A–C–D–G– I = critical path

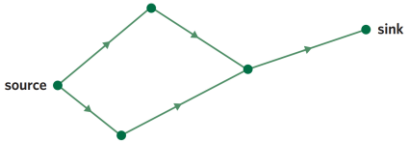
B–E –G– I = another critical path

8E Directed graph:

- Networks contain arrows on each edge
- Networks show directional information between vertices

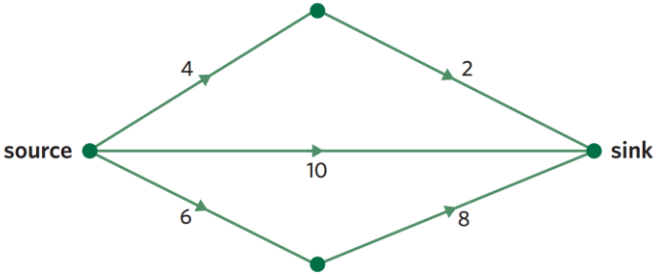
Flow:

- Flow problems involve the transfer or flow of material from one point (source) to another (sink)
- Eg: water flowing through pipes or traffic flow on roads
- No matter the situation, things flow in <u>one direction</u> only
- Weights on graphs are called capacities



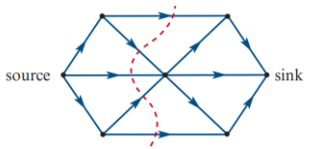
Maximum flow:

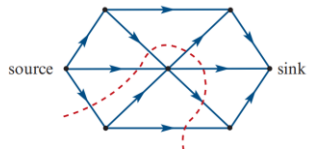
- If edges of different capacities are connected one after the other, the maximum flow through the edges is equal to the minimum capacity of the individual edge
- Maximum flow = minimum capacity



Cuts:

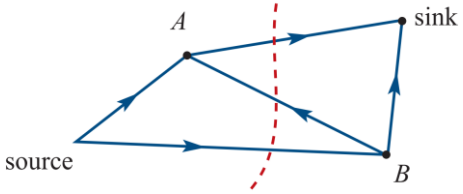
- A cut divides the network into two parts, completely separating the source from the sink
- Completely stops flow!





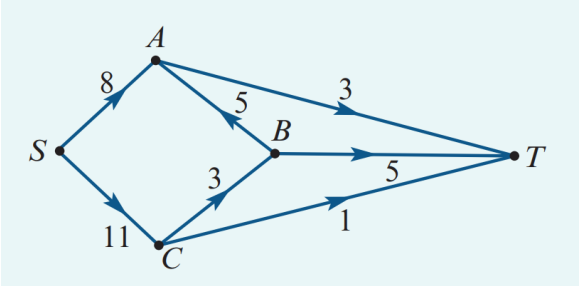
Cut capacity:

- The sum of all the capacities of the edges that the cut passes through, considering the direction of flow
- The capacity is only counted it is flows from source to sink or isn't cut off earlier



Minimum cut capacity:

- A cut that has the lowest cut capacity for a network
- Minimum cut capacity = maximum flow
- Example: determine the maximum flow from source (S) to sink (T)



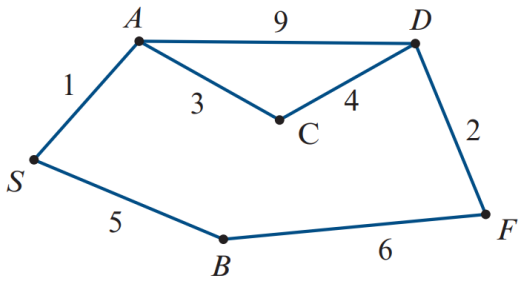
8F Dijkstra's Algorithm

6. Create a table, the starting vertices is in the first row, the rest of the vertices form the column labels
7. Complete the first row
    - Write the distance from the starting vertex to each other vertex in the corresponding column
    - If there is no direct connection, mark with a cross 'X'
    - Find the smallest number in the first row and put a box/square around it (if there are two or more the same, any can be chosen)
    - The column vertex that has the box around it becomes the next row
8. Complete further rows
    - Copy all boxed numbers into the next row
    - Add the boxed number from the row vertex to the column vertex
        - If the value is greater than the value above it, ignore the new number and copy down the smaller one
        - If the value is less than or equal to the value above it, write down the new value
        - If there is no direct connection, mark with a cross 'X'
    - Look for the smallest unboxed number in the row and draw a box around it
    - The column vertex for this new boxed number becomes the next row vertex
9. Repeat step 3 until the destination vertex value has a box around it
10. Backtrack to identify the shortest path and it's length
    - Start at the destination box – this is the length of the shortest path
    - Draw a line up the column to the last number that is the same
    - Look at the row vertex for this number and draw a horizontal line to the column
    - Repeat until you reach the start
    - The horizontal lines indicate the shortest path

OR

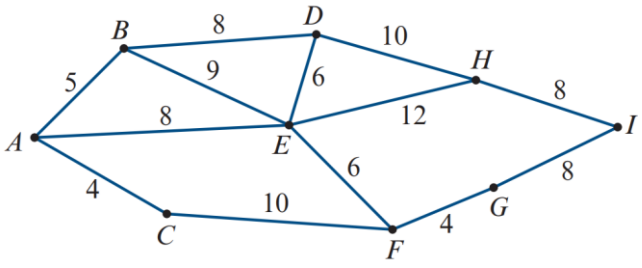
1. Rest vertices on row 1
2. Starting vertex with distance on row 2, x for no connection vertex.
3. Frame smallest number & copy to whole column
4. Smallest number vertex with accumulating distance on row 3
5. Frame next smallest number & copy to whole column
6. Next smallest number vertex with accumulating distance on row 4
7. Repeat step 5 & 6 until table complete
8. Circle all vertices' intersections
9. Trace same column number from ending vertex
10. Locate next circled vertex on the row
11. Repeat step 9 & 10 until trace back to starting vertex
12. Write down all passing by vertices from tracing above

Examples:



Shortest path from S to F

S					



Shortest path from A to I

A								

Helpful hints:

- If there is no direct connection and there is a number above, bring it down
- Always bring down the smallest number
- don't forget to add the previous total to your new moves
- if two numbers are the same it doesn't matter which one you bring down

8G The table shows four employees, Wendy, Xenefon, Yolanda and Zelda. The machines in a factory are represented by the letters A, B, C, D. The numbers in the tables are the times in minutes it takes each employee to finish a task on each machine. Which machine should each employee operate to ensure the minimum time is taken?

Employee	*A*	*B*	*C*	*D*
Wendy	30	40	50	60
Xenefon	70	30	40	70
Yolanda	60	50	60	30
Zelda	20	80	50	70

Steps:

1. Locate the lowest value in each **row** and **subtract** that from each element in the row
2. Find the minimum number of **lines** required to **cover** all of the **zeros** (if it is the same as the number of subjects then skip to step 7) (it is usually 4 so if you have four lines then skip)
3. Locate the lowest value in each **column** (it may be zero) and **subtract** that from each element in the column
4. Find the minimum number of **lines** requires to **cover** all of the **zeros** (skip to step 7 if you have the same number of lines as there are subjects)
5. Locate the smallest value which is not covers by a line. **Subtract** that from all **uncovered** elements and **add** the value to all elements covered by 2 lines (line **intersection**)
6. Find the minimum number of **lines** required to **cover** all the **zeros** in the table (if it is not the same as the number of subjects repeat 5)
7. Make the allocation based on the location of the zeros in the matrix (**Bipartite**)
8. Find the **minimum time** by referring to the initial table costs

	A	B	C	D
Wendy				
Xenefon				
Yolanda				
Zelda				

	A	B	C	D
Wendy				
Xenefon				
Yolanda				
Zelda				

	A	B	C	D
Wendy				
Xenefon				
Yolanda				
Zelda				

	A	B	C	D
Wendy				
Xenefon				
Yolanda				
Zelda				

	A	B	C	D
Wendy				
Xenefon				
Yolanda				
Zelda				

8G Hungarian Algorithm Task Match Working Steps

1. **Row deduction (Min. No. of each row to deduct)**

Tasks / Person	Task 1	Task 2	Task 3	Task 4	Task 5
Person 1					
Person 2					
Person 3					
Person 4					
Person 5					

2. **Column deduction (Min. No. of each non 0 column)**

Tasks / Person	Task 1	Task 2	Task 3	Task 4	Task 5
Person 1					
Person 2					
Person 3					
Person 4					
Person 5					

3. Line fitting to cover Max. 0 possible ≥ No. of tasks

≥ More than or equal

Tasks / Person	Task 1	Task 2	Task 3	Task 4	Task 5
Person 1					
Person 2					
Person 3					
Person 4					
Person 5					

↓ Enough lines to **step 6** ↓ Not enough lines to **step 4**

4. Intersection addition uncovered deduction (use uncovered Min. No)

Tasks / Person	Task 1	Task 2	Task 3	Task 4	Task 5
Person 1					
Person 2					
Person 3					
Person 4					
Person 5					

5. Line fitting to cover Max. 0 possible ≥ No. of tasks

≥ More **than or equal**

Tasks / Person	Task 1	Task 2	Task 3	Task 4	Task 5
Person 1					
Person 2					
Person 3					
Person 4					
Person 5					

6. Bipartite task graph matching (0=line match)

Person 1	Task 1
Person 2	Task 2
Person 3	Task 3
Person 4	Task 4
Person 5	Task 5

7. Task Allocation (possible 2 solutions)

Person 1	Person 1
Person 2	Person 2
Person 3	Person 3
Person 4	Person 4
Person 5	Person 5

8H: Precedence Tables and Networks

Immediate Predecessor (IP):

- An activity that must be completed before
- another activity can begin
- They are displayed in a precedence table

Precedence tables:

- Shows activities and Ips
- Info is used to draw networks
- Activity networks no longer label vertices but instead label edges
- Start/finish vertices get labelled

Example- draw an activity network from the precedence table: **Locate Start & Finish**

Activity	IP
A	--
B	A
C	A
D	A
E	B
F	C
G	D
H	E F G

Dummy activities:

- Required if two activities share SOME but NOT ALL IP's
- Begins at the shared IP and ends at the start of the activity that has additional IPS
- Drawn as a **dotted line** and labelled '**d**'

Examples- draw an activity network from the precedence table: **Locate Start, Finish & Dummy**

Activity	IP
A	--
B	--
C	A
D	B
E	C **D**
F	**C**
G	E F

More Practice: **Locate Start, Finish & Dummy1 Dummy 2**

Activity	IP
A	--
B	A
C	A
D	B
E	**C**
F	C **D**
G	C **D**
H	E F
I	**G**
J	G **H**

8I FORWARD SCANNING for EARLIEST START TIMES (EST):

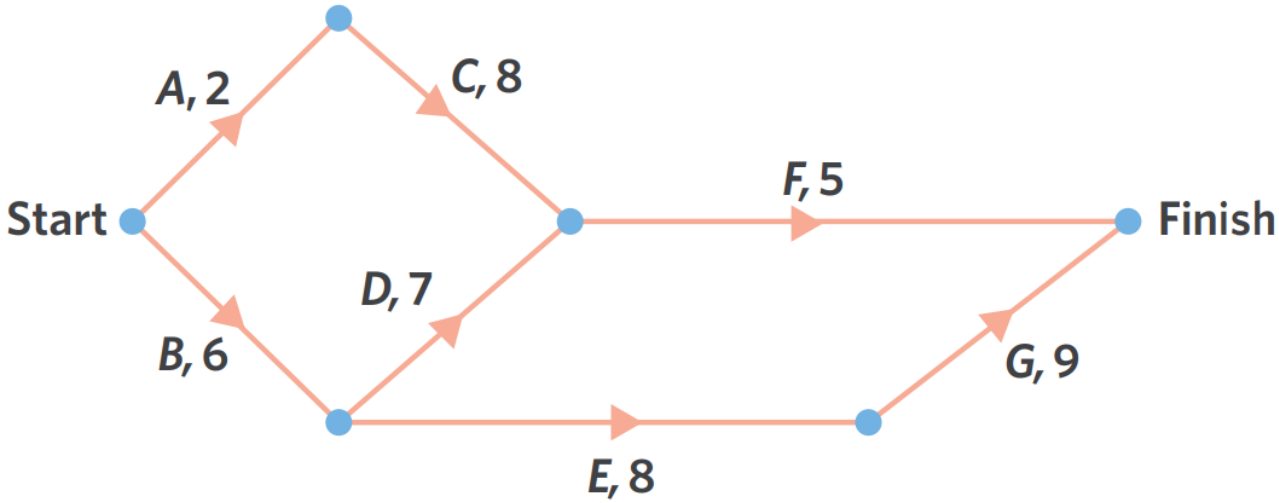
- To minimise the total completion time for a project, each activity should begin at its earliest start time (EST)
- Done via process of forward scanning
- Steps:
    1. Draw a box at each activity of the network, as well as the finish vertex
    2. The activities that connect to the start vertex have a 0 in the EST box
    3. Start filling in the EST box for all activities and the finish vertex
        i. If an activity has one immediate predecessor, its EST can be found by adding the EST and duration of the immediate predecessor
        ii. If an activity has two or more predecessors, the EST will be the **LARGEST** value
    4. The EST for the finish vertex is the minimum possible completion time for the project

1. Boxes or lines at the beginning of each edge including dummy. Or

EST	LST

2. Forwards scanning with **numbers written on each entry,** **biggest EST** number going forward.

3. Backwards scanning with **smallest LST** number going backwards.



8J

1. Boxes or lines at the beginning of each edge including dummy. Or

EST LST

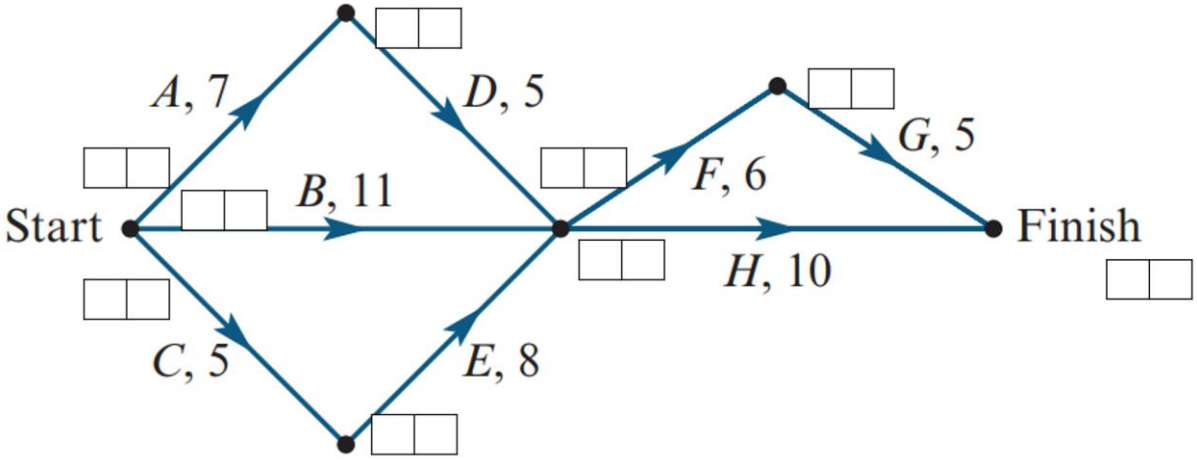
2. Forwards scanning with **numbers written on each entry**, **biggest EST** number going forward.

3. Backwards scanning with **smallest LST** number going backwards.

Crashing Steps:

1. Reducing all possible activities.
2. Froward and backward scanning to find critical path.
3. Any activities on critical path must crash.
4. Checking multiple entry numbers not to reduce some activities to lower the cost.
5. write reducing amount and cost on the table.

The directed network below shows the sequence of 8 activities that are needed to complete a project. The time, in days, that it takes to complete each activity is also shown.



The minimum completion time for the project is 24 days. It is possible to reduce the completion time for activities D, E and H. The completion time for each of these three activities can be reduced by a maximum of two days.

a What is the new minimum completion time, in days, of the project?

The reduction in completion time for each of these three activities will incur an additional cost. The table opposite shows the three activities that can have their completion times reduced and the associated daily cost, in dollars.

Activity	Daily cost($)
D	170
E	350
H	200

Days reduced	Cost /activity

b What is the minimum cost that will achieve the greatest reduction in time taken to complete the project?