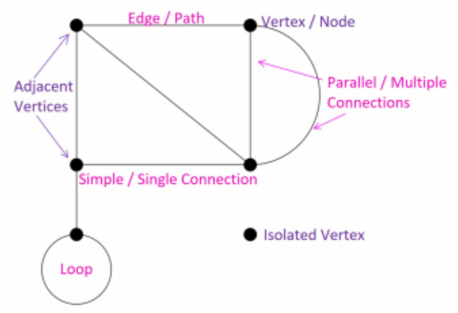
Networks

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.

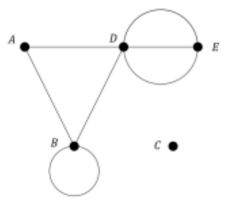


The Degree of a Vertex
The number of edges connected to a vertex.

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



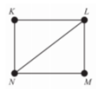
Degrees
$\text{Deg}(A) = 2$
$\text{Deg}(B) = 4$
$\text{Deg}(C) = 0$
$\text{Deg}(D) = 5$
$\text{Deg}(E) = 3$

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

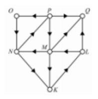
A loop counts as one edge in the number of edges.

Direction on Graphs

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

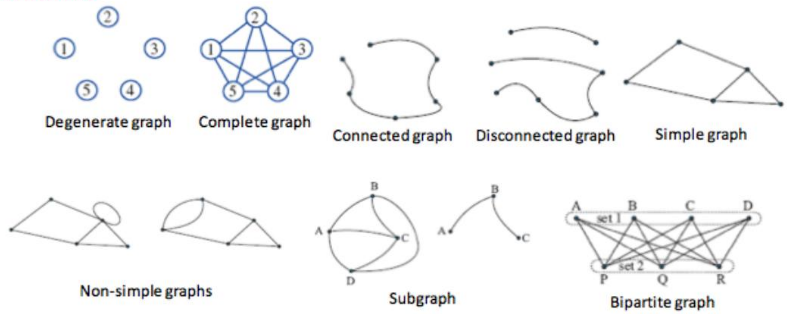


Directed Graph / Digraph
A graph where specific direction is indicated for every edge. Some vertices may not be reachable from other vertices.



Undirected Graphs

Types of Networks



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

Types of graphs

Simple graph – No loops or duplicate edges.
Isolated vertex – A graph has an isolated vertex if there is a vertex that is not connected to another vertex by an edge.
Degenerate graph – Degenerate graphs have all vertices isolated. Therefore, there are no edges in the graph at all.
Connected graph – Each vertex is either directly or indirectly connected to every other vertex.
Bridge – A bridge is an edge that when removed makes the graph unconnected.
Subgraph – Are graphs that are part of larger graphs.
Equivalent (isomorphic) graph – Look different but have the same information
Complete graph – Every vertex has a direct connection to every other vertex.
Bipartite Graph – A bipartite graph is a graph whose set of vertices can be split into two subsets X and Y in such a way that each edge of the graph joins a vertex in X and a vertex in Y.

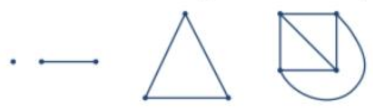
Isomorphic graphs –Two graphs have: ① same numbers of edges and vertices; ② corresponding vertices have the same degree and the edges connect the same vertices.

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.

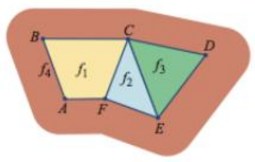


Euler's Formula

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

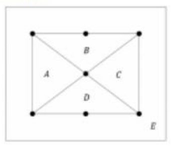
$\boldsymbol{e + 6 \leq 3v}$

Degree of a Face

Degree of a face is the number of
- edges along its boundary
- vertices along its boundary
- faces with which it shares an edge

Every edge is shared by exactly two faces. Therefore,
Sum of face degrees $= 2 \times$ number of edges

Example



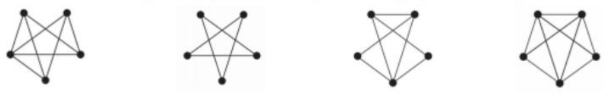
Degrees
$\text{Deg}(A) = 3$
$\text{Deg}(B) = 4$
$\text{Deg}(C) = 3$
$\text{Deg}(D) = 4$
$\text{Deg}(E) = 6$

Since planar graphs are simple, they must have a degree of at least 3, otherwise they would be defined by only two vertices. Therefore

$3 \times$ number of faces $\leq$ sum of face degrees $\qquad 3 \times$ number of faces $\leq 2 \times$ number of edges.

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: $v = 5,\ e = 8$	Graph 2: $v = 5,\ e = 5$	Graph 3: $v = 5,\ e = 7$	Graph 4: $v = 5,\ e = 9$
$8 + 6 \leq 3 \times 5$	$5 + 6 \leq 3 \times 5$	$7 + 6 \leq 3 \times 5$	$9 + 6 \leq 3 \times 5$
$14 \leq 15$ ✓	$11 \leq 15$ ✓	$13 \leq 15$ ✓	$15 \leq 15$ ✓

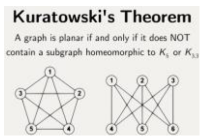
Therefore, all 4 graphs are planar.

Combining Euler's Formula and the Result from the Degree of a Face

Earlier we determined that $3 \times$ number of faces $\leq 2 \times$ number of edges ($3f \leq 2e$).

Multiplying Euler's formula for faces by 3
$f = e - v + 2$
$\Rightarrow 3f = 3e - 3v + 6$

Then using $3f \leq 2e$
$\Rightarrow 3f = 3e - 3v + 6 \leq 2e$
$\Rightarrow e + 6 \leq 3v$



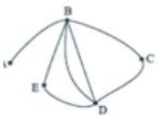
Kuratowski's Theorem

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

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.

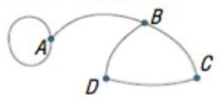


$$\text{From}\quad\begin{array}{c|ccccc} & A & B & C & D & E \\ \hline A & 0 & 1 & 0 & 0 & 0 \\ B & 1 & 0 & 1 & 2 & 1 \\ C & 0 & 1 & 0 & 1 & 0 \\ D & 0 & 2 & 1 & 0 & 1 \\ E & 0 & 1 & 0 & 1 & 0 \end{array}\quad \text{To}$$

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.

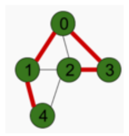


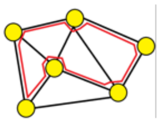
$$\begin{array}{c} \\ A \\ B \\ C \\ D \end{array}\begin{array}{c} \begin{matrix} A & B & C & D \end{matrix} \\ \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix} \end{array}$$

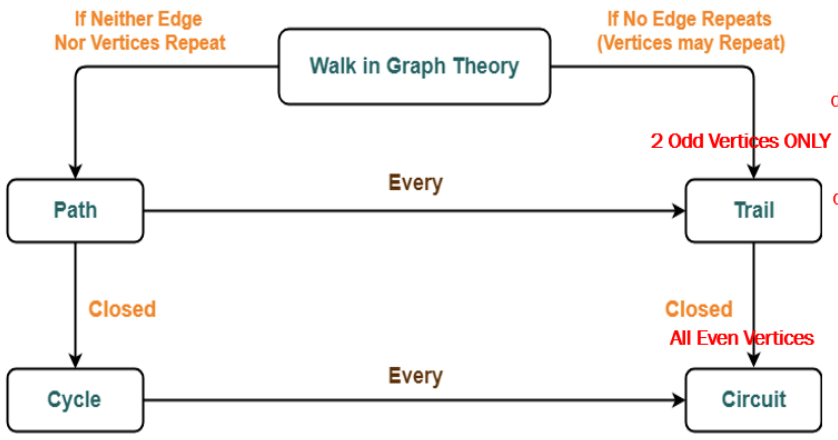
The adjacency matrix A of a graph is an $n \times n$ matrix in which, for example, the entry in row C and column F is the number of edges joining vertices C and F.
A loop is a single edge connecting a vertex to itself.
Loops are counted as one edge.

Euler & Hamilton

Hamiltonian

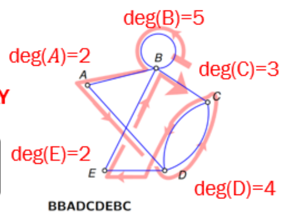




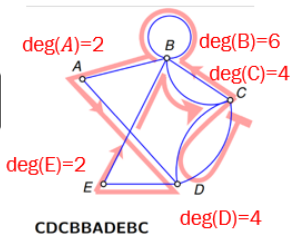


Important Chart to Remember

Eulerian



BBADCDEBC



CDCBBADEBC

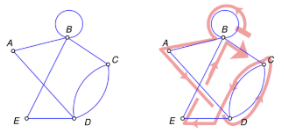
Travelling in graphs

Route – A description of your travels, given by the vertices visited in the order they are visited.
Walk – A walk can be any type of journey within a graph, you can walk wherever you wish.
Trail – A special kind of walk, you **can't repeat** any of the **edges** that you have taken, but you can revisit vertices.
Path – A path is a special kind of trail, with a path you **can't repeat** any **edges or vertices**.

Eulerian trails and circuits

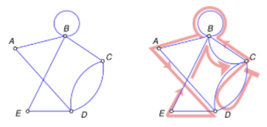
Eulerian trails – Is a trail in which every **edge** is visited **once**. Vertices can be repeated.
A Eulerian trail will only exist if:

- The graph is connected
- The graph has exactly two vertices of an odd degree



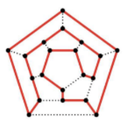
Eulerian circuit – Is a Eulerian trail (travels every **edge once**) that begins and ends from the same vertex.
A Eulerian circuit will only exist if:

- The graph is connected
- All the vertices have an even degree



Hamiltonian paths and cycles

Hamiltonian path – Is a path that visits all of the **vertices** in a graph **only once**.
Hamiltonian cycle – Is a cycle that visits every vertex and begins and ends at the same vertex.



Weighted Graphs

Trees

A **tree** is a connected, simple graph with no circuits.
A **spanning tree** is a sub graph of a connected graph which contains **all the vertices** of the original graph.
The weight of a spanning tree is the combined weight of all its edges, and there are two ways in which the minimum- weight spanning tree can be found:

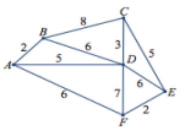
- **Prim's algorithm:** which involves choosing random vertex as a starting graph and constantly building to it by adding the shortest edges which will connect it to another node.

Prim's Algorithm

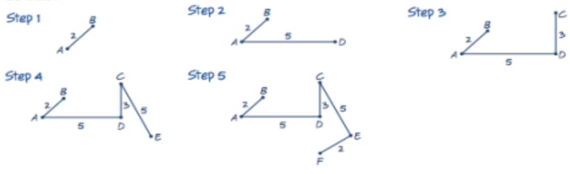
I. Chooseavertexandconnectittoasecondvertexchosensothattheweightoftheedge is as small as possible.
II. In each step thereafter, take the edge with the lowest weight, producing a tree with the edges already selected. (If two edges have the same weight the choice can be arbitrary.)
III. Repeat until all the vertices are connected and then stop.

- A minimum spanning tree = the spanning tree of minimum length (may be minimum distance, minimum time, minimum cost, etc.). There may be more than one minimum spanning tree in a weighted graph.

Apply Prim's algorithm to obtain a minimum spanning tree for the graph shown. Write down its weight, and compare it to the weight of the original graph.



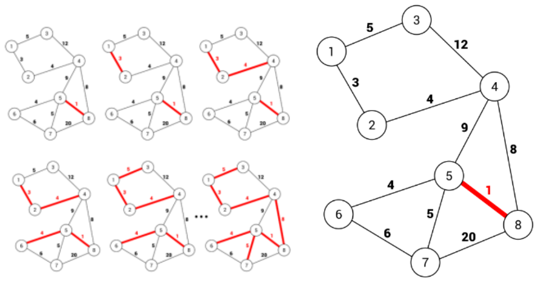
Solution



The total weight is 17. The total weight of the original graph is 50.

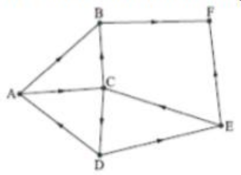
Kruskal's Algorithm

- Choose the edge with the least weight as the starting edge. If there is more than one least-weight edge, any will do.
- Next, from the remaining edges, choose an edge of least weight which does not form a cycle. If there is more than one least-weight edge, any will do.
- Repeat the process until all vertices are connected. The result is a minimum spanning tree.
- Determine the length of the spanning tree by summing the weights of the chosen vertices.



Directed Graphs

In a directed graph, each edge has a direction. Also, each vertex in a network can be reachable or unreachable. This is often abbreviated to **digraph**.



In this network, no node is reachable from F, and A is not reachable from B.

Network Flows

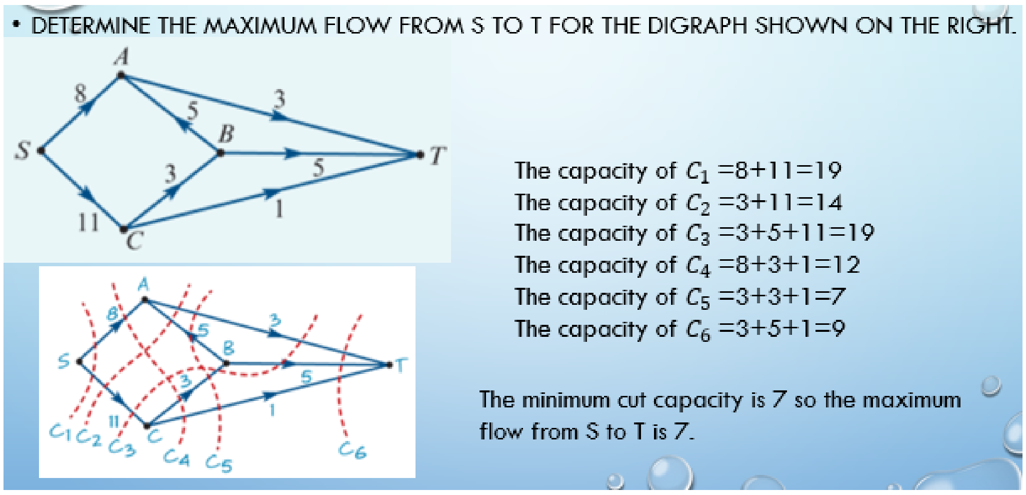
Weighted graphs can be used to model the flow of people, water or traffic. The flow is always from the *source* vertex to the *sink* vertex. The weight of an edge represents its capacity.

- Cuts are used as a way of preventing all flow from source to sink. A valid cut must completely isolate the source from the sink. By adding the weights of the cut edges, the value of a cut can be obtained. The minimum value cut that can be made represents the maximum flow possible through the network.

Capacity

The capacity of an edge is the maximum amount that can flow through it.
The capacity of a cut is the sum of the weights of the edges in the cut.
The capacity of a network is the maximum amount that can flow from the sink to the source.

- THE MINIMUM CUT CAPACITY = THE MAXIMUM FLOW



- DETERMINE THE MAXIMUM FLOW FROM S TO T FOR THE DIGRAPH SHOWN ON THE RIGHT.

The capacity of C_1 =8+11=19
The capacity of C_2 =3+11=14
The capacity of C_3 =3+5+11=19
The capacity of C_4 =8+3+1=12
The capacity of C_5 =3+3+1=7
The capacity of C_6 =3+5+1=9

The minimum cut capacity is 7 so the maximum flow from S to T is 7.

Ford-Fulkerson Algorithm

Ford-Fulkerson Algorithm
An algorithm for finding the maximum flow through a network.

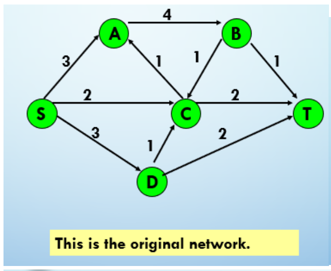
1) Choose any path from the source to the terminal
2) Choose the smallest capacity on that path and write it above each capacity on that path
3) Choose a path from the source to the terminal that is non-full forwards, and non-empty backwards
4) Choose the smallest remaining capacity on that path and write it above each capacity on that path, add it to any that have already got values
5) Repeat steps 3 and 4 until there are no paths that are non-full forwards, and non-empty backwards

For singular source - singular terminal
6) The maximum flow from the source is the capacity of the values leading directly out of the source
7) The maximum flow into the terminal is the capacity of the values leading directly into the terminal.
8) Flow out of the source = flow into the terminal

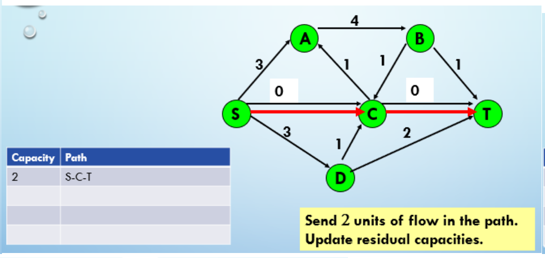
For multiple sources - singular terminal
6) Maximum flow from each source is the capacity of the values leading directly out of each source.
7) The maximum flow into the terminal is the capacity of the values leading directly into the terminal.
8) Total flow out of the sources = flow into the terminal

For multiple sources - multiple terminals
6) Maximum flow from each source is the capacity of the values leading directly out of each source.
7) Maximum flow into each terminal is the capacity of the values leading directly into each terminal.
8) Total flow out of the sources = total flow into the terminals

For singular sources - multiple terminals
6) The maximum flow from the source is the capacity of the values leading directly out of the source
7) Maximum flow into each terminal is the capacity of the values leading directly into each terminal.
8) Flow out of the source = total flow into the terminals

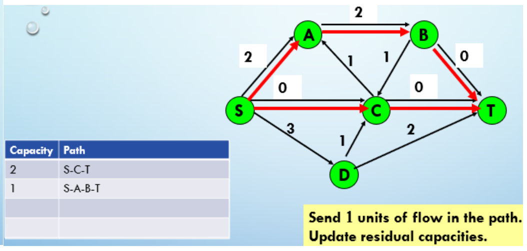


This is the original network.



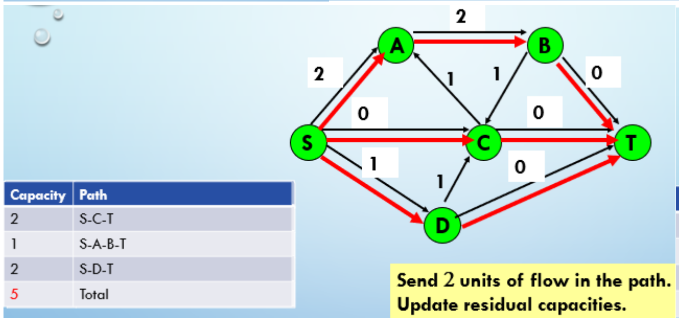
Capacity	Path
2	S-C-T

Send 2 units of flow in the path.
Update residual capacities.



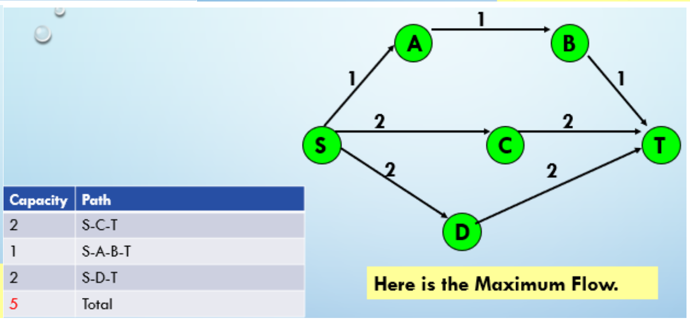
Capacity	Path
2	S-C-T
1	S-A-B-T

Send 1 units of flow in the path.
Update residual capacities.



Capacity	Path
2	S-C-T
1	S-A-B-T
2	S-D-T
5	Total

Send 2 units of flow in the path.
Update residual capacities.



Capacity	Path
2	S-C-T
1	S-A-B-T
2	S-D-T
5	Total

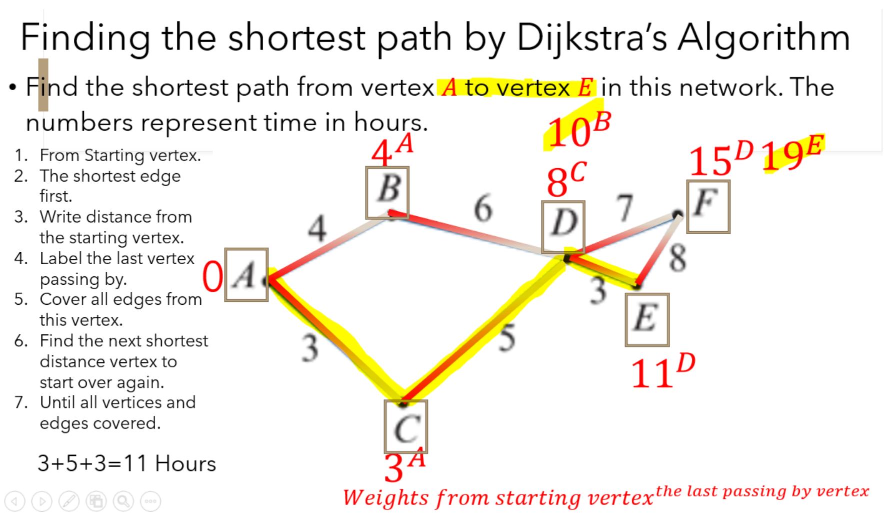
Here is the Maximum Flow.

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



Finding the shortest path by Dijkstra's Algorithm

- 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. Label the last vertex passing by.
5. Cover all edges from this vertex.
6. Find the next shortest distance vertex to start over again.
7. Until all vertices and edges covered.

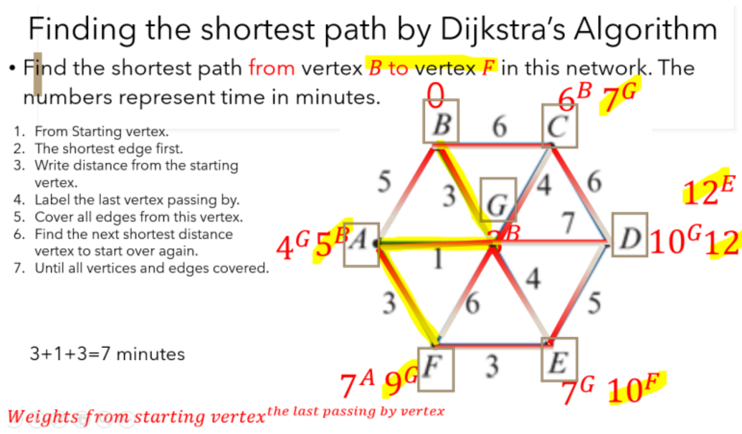
3+5+3=11 Hours

0 A
4^A B
10^B 8^C D
15^D 19^E F
3^A C
11^D E
4
6
7
8
3
3
5
Weights from starting vertex$^{\text{the last passing by vertex}}$

B G A D

3 + 1 + 3 = 7 minutes

	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



Finding the shortest path by Dijkstra's Algorithm

- 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. Label the last vertex passing by.
5. Cover all edges from this vertex.
6. Find the next shortest distance vertex to start over again.
7. 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.

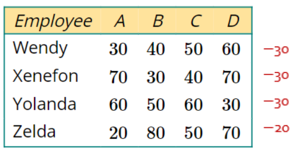
Step One

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



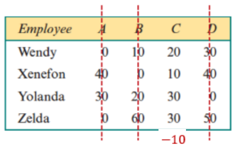
Employee	*A*	*B*	*C*	*D*	
Wendy	30	40	50	60	−30
Xenefon	70	30	40	70	−30
Yolanda	60	50	60	30	−30
Zelda	20	80	50	70	−20

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.

Column Minus



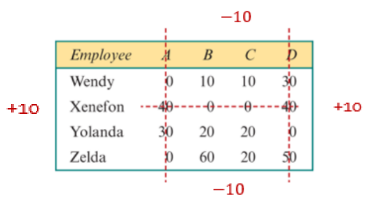
Employee	*A*	*B*	*C*	*D*
Wendy	0	10	20	30
Xenefon	40	0	10	40
Yolanda	30	20	30	0
Zelda	0	60	30	50
			−10	

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



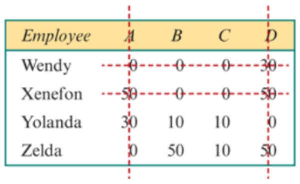
	Employee	*A*	*B*	*C*	*D*	
		−10				
	Wendy	0	10	10	30	
+10	Xenefon	40	0	0	40	+10
	Yolanda	30	20	20	0	
	Zelda	0	60	20	50	
		−10				

Intercept Plus & Uncovered Minus

Step Three

Add the minimum uncovered element to the rows and columns that are covered.
The minimum uncovered element (10) is now subtracted from all entries and **step 2 is repeated**.

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



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

4 people = Minimum 4 lines

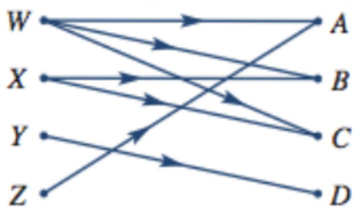
	A	*B*	*C*	*D*
~~*W*~~	~~0~~	~~0~~	~~0~~	~~30~~
~~*X*~~	~~50~~	~~0~~	~~0~~	~~50~~
~~*Y*~~	~~30~~	~~10~~	~~10~~	~~0~~
~~*Z*~~	~~0~~	~~50~~	~~10~~	~~50~~

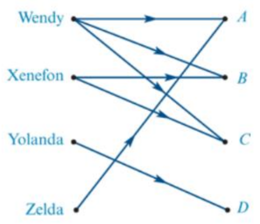
The minimum number of lines is equal to the number of rows, so it is possible to obtain a maximum matching.

Step Four

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

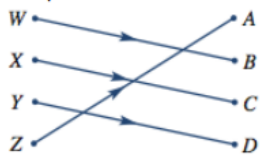
❖ The edges are chosen through the **zero** entries in the table.

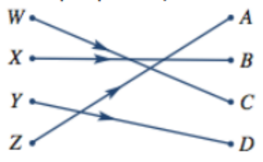




0 in table = A line of Connection

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

EST	LST

1. Draw a box for each going forward edge/activities.
2. EST = Going forward with **Biggest** 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.

Conventions of Critical Path Problems

Developing and manufacturing a product frequently involves many interrelated activities. It is often the case that some of these activities cannot be started until other activities are completed.

Two important facts about critical paths are:

I. The weight of the critical path is the minimal length of time required to complete the project.
II. Increasing the time required for any critical activity will also increase the time necessary to complete the project.

Digraphs can be used to represent such situations with the following conditions applying:

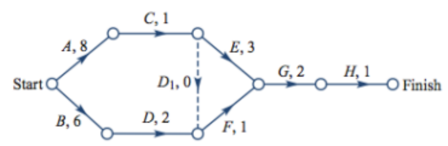
- The edges (or arcs) represent the activities.
- The vertices (or nodes) represent events.
- The start/finish of one or more activities is called an event
- An edge should lead from a
- A vertex (called the finish node) representing the completion of the project should be included in the network.
- An activity should not be represented by more than one edge in the network.
- Two nodes can be connected directly by, at most, one edge.

Precedence Table

A table that details the events that must occur immediately before an event may begin and its duration.

Activities	Immediate predecessors
A	-
B	-
C	A
D	B
E	C
F	C, D
G	E, F
H	G

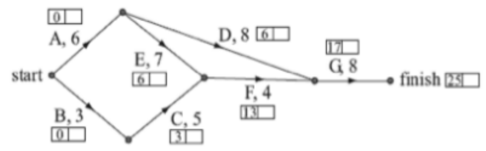
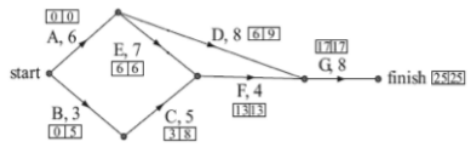
In order to satisfy the final two conventions, it is sometimes necessary to introduce a dummy activity that takes zero time. Following these conventions the weighted digraph can be redrawn.



EST | LST

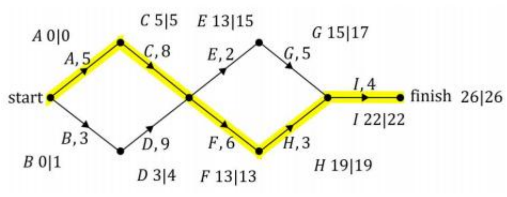
EST	LST

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*



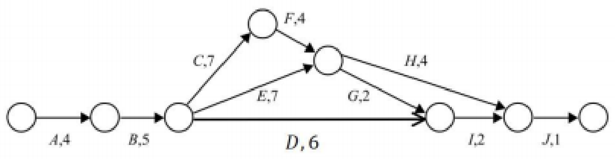
Project Crashing

If a project looks like running overtime, it may be crashed. Crashing involves spending extra money to reduce the time taken by certain activities in order to avoid costs of completing the project late. The following steps illustrate the crashing process:

I. Write down all of the paths from the start node to the finish node, and determine the length of each.
II. Calculate the cost per day of crashing each activity (note that some questions may already have the cost per day, and watch for reductions that must be made in full)
III. Reduce the cheapest (cost per day) activity **on the critical path** by one day.
IV. Calculate the new lengths of all of the paths.
V. Repeat steps three and four, each time using the new longest path after reductions. Stop when the budget is reached or the longest path cannot be reduced. The new critical path is the longest path.

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

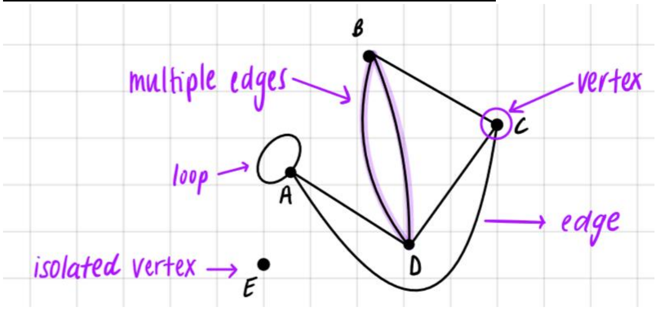
Prim's Minimal Spanning Tree Algorithm
Kruskal's Minimal Spanning Tree algorithm
Dijkstra's Shortest Path Algorithm
Ford-Fulkerson Networks Flows Algorithm
Hungarian Algorithm

Mathematical Terminologies

Undirected Graphs		Directed Graphs	
Terminologies	**Algorithm**	**Terminologies**	**Algorithm**
Eulerian trails	Exactly 2 vertices of an odd degree	The Maximum Flow 8.1	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 8.2
Hamiltonian cycles	Visit All vertices, begin & end @ the same vertex	Critical Path Problems	Forward scanning = Biggest Number Backward scanning = Smallest Number Float = LST—EST
Minimal Spanning Tree	Prim's Algorithm, Kruskal's Algorithm		

Networks Notes

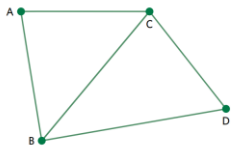
8A: Introduction to graphs and networks



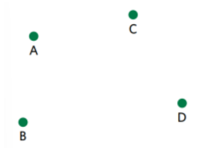
- **Graph:** a diagram used to show connections between groups of things, people of activities.
- **Vertex:** the dots
- **Edge:** line that connects two vertices
- **Isolated vertex:** a vertex that is not connected to an edge
- **Multiple edges:** connects the same vertex using more than one edge
-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

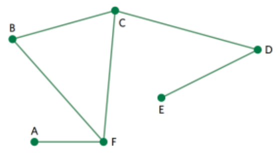
Simple graph: no loops or multiple edges



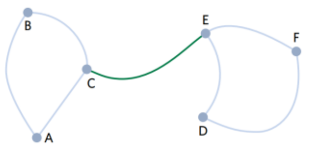
Degenerate graph: all vertices are isolated, there are no edges



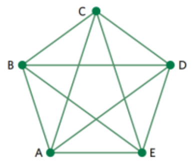
Connected graph: every vertex is connected to every other vertex, either directly or indirectly



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

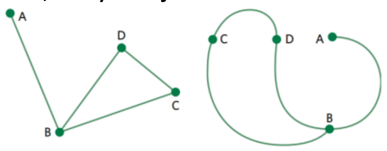


Complete graph: every vertex is connected to every other vertex

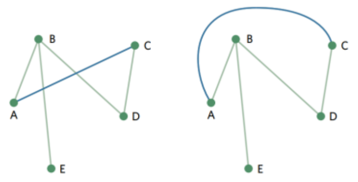


Subgraph: a tree, a part of a larger graph

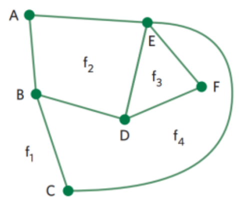
Equivalent/isomorphic graph: graphs that contain the EXACT same information, vertices and the connections between them, they are just drawn differently



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.

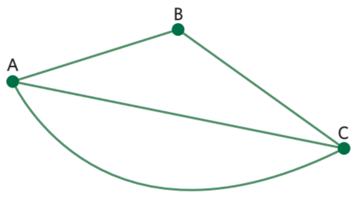
$$v - e + f = 2, \text{ where}$$

- 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.

$$\begin{array}{c} \begin{array}{ccc} A & B & C \end{array} \\ \left[\begin{array}{ccc} 0 & 1 & 2 \\ 1 & 0 & 1 \\ 2 & 1 & 0 \end{array}\right] \end{array} \begin{array}{c} \\ A \\ B \\ C \end{array}$$



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'.

Networks Notes

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.

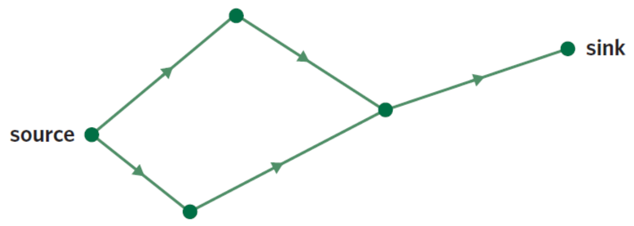
- There are often multiple spanning trees for the one graph

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.



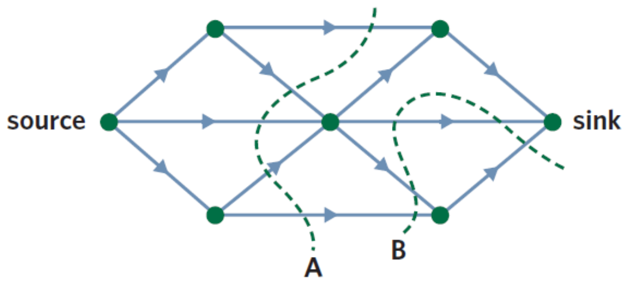
- No matter the situation, things flow in **one direction** only
- Weights of graphs are called capacities

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

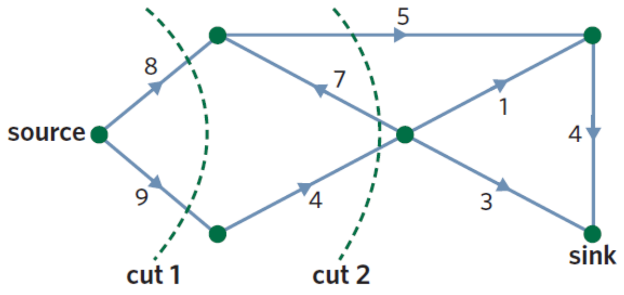
Cuts: a cut divides the network into two parts, completely separating the source from the sink

- 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

Networks Notes

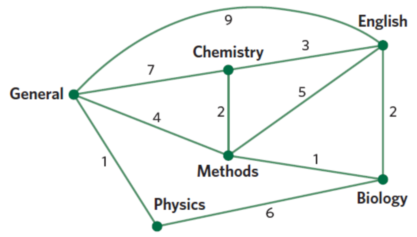
8F: Shortest path problems

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	$\boxed{1}$	X	4	7	9
P	$\boxed{1}$	7	$\boxed{4}$	7	9
M	$\boxed{1}$	$\boxed{5}$	$\boxed{4}$	6	9
B	$\boxed{1}$	$\boxed{5}$	$\boxed{4}$	$\boxed{6}$	$\boxed{7}$
C	$\boxed{1}$	$\boxed{5}$	$\boxed{4}$	$\boxed{6}$	$\boxed{7}$

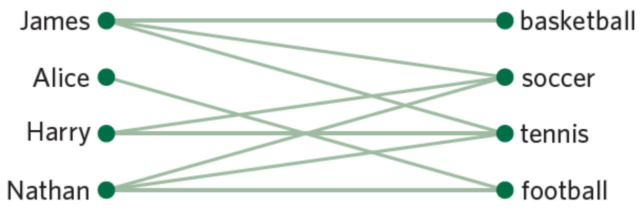
The shortest path is G–M–B–E.

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.

Networks Notes

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

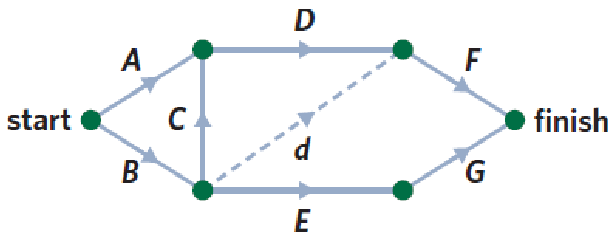
Dummy activities: are required is two activities share **SOME** but **NOT ALL** IP's

- 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.



Networks Notes

<u>8I: Critical Path Planning:</u>

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:

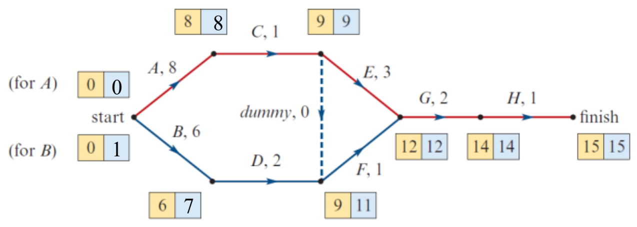
1. Draw a box at each activity of the network, as well as the finish vertex
2. The activities that connect to the start vertex have a 0 in the EST box
3. Start filling in the EST box for all activities and the finish vertex
   i. If an activity has one immediate predecessor, its EST can be found by adding 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.

<u>8J: Crashing</u>

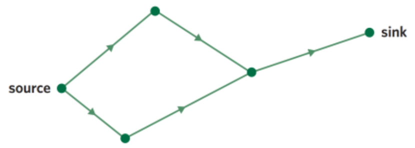
- Used to reduce the completion time of an activity network/project
- Crashing an activity **on** the critical path will immediately impact the minimum completion time
- Crashing off the critical path is sometimes still necessary to alter the minimum completion time
- Crashing can cause the critical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

8E Directed graph:

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

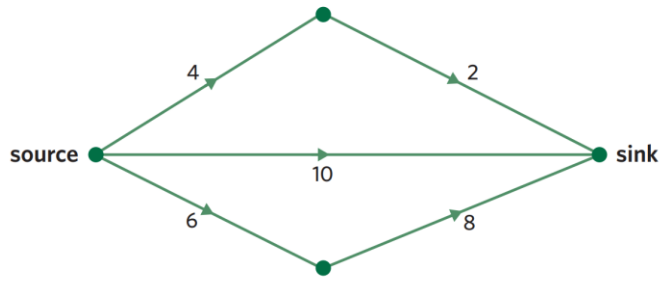
Flow:

- 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



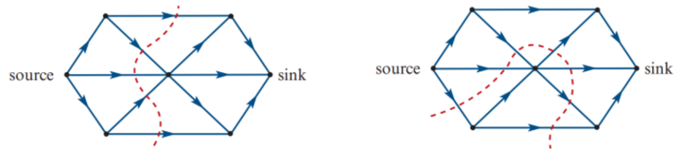
Maximum flow:

- 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



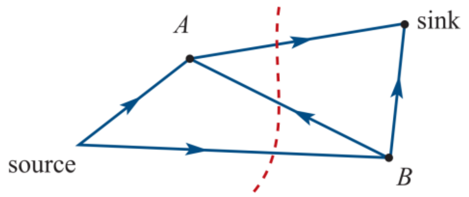
Cuts:

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



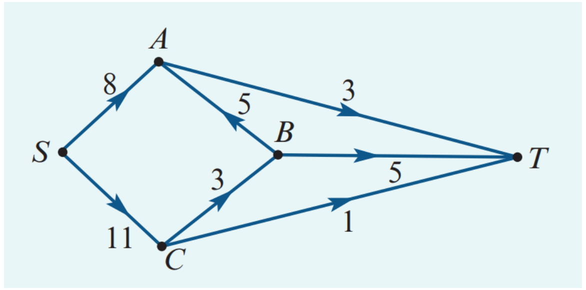
Cut capacity:

- 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)



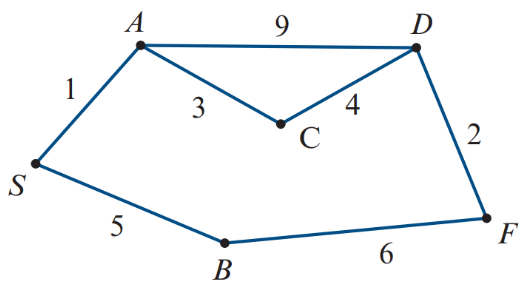
8F Dijkstra’s Algorithm

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

OR

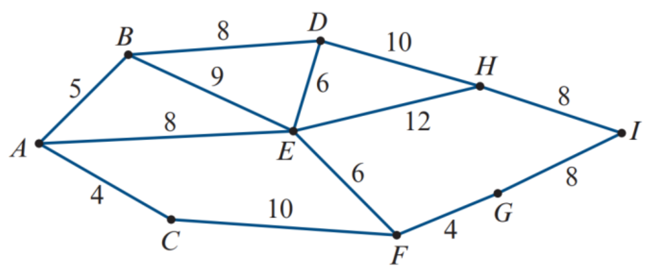
1. Rest vertices on row 1
2. Starting vertex with distance on row 2, x for no connection vertex.
3. Frame smallest number & copy to whole column
4. Smallest number vertex with accumulating distance on row 3
5. Frame next smallest number & copy to whole column
6. Next smallest number vertex with accumulating 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

Hungarian Algorithm Task Match Working Steps

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

Person / Task	A	B	C	D	E
1					
2					
3					
4					
5					

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

Person / Task	A	B	C	D	E
1					
2					
3					
4					
5					

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

≥ More than or equal

Person / Task	A	B	C	D	E
1					
2					
3					
4					
5					

Enough lines to **step 6**

Not enough lines to **step 4**

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

Person / Task	A	B	C	D	E
1					
2					
3					
4					
5					

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

≥ More than or equal

Person / Task	A	B	C	D	E
1					
2					
3					
4					
5					

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

1	A
2	B
3	C
4	D
5	E

7. Task Allocation (possible 2 solutions)

1	A
2	B
3	C
4	D
5	E

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

8I BACKWARD SCANNING FOR LATEST START TIMES (LST):

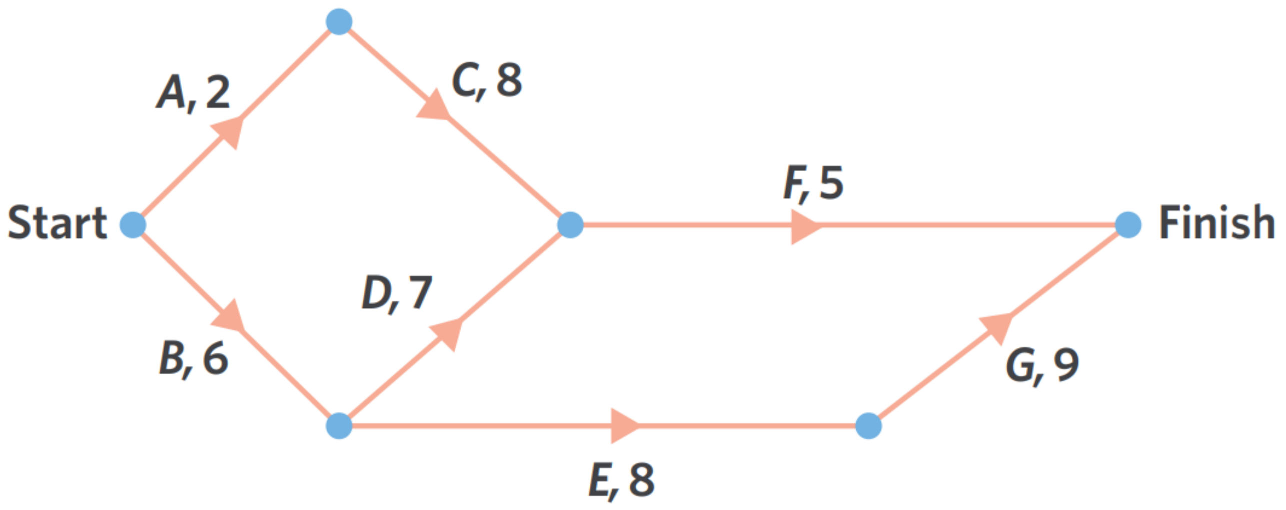
- Some activities may be able to start later than the earliest possible start time and not impact the total minimum completion time for the project.
- Done via backward scanning
- Steps:
    1. Complete forward scanning (all left hand boxes should be full)
    2. Fill in the LST for the finish as the 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.

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 |

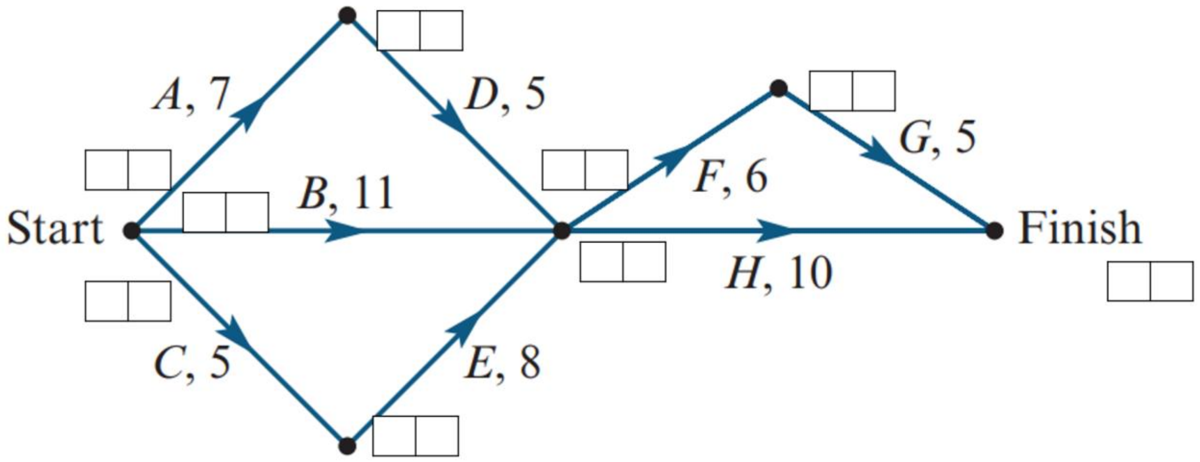
EST	LST

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

3. Backwards scanning with **smallest LST** number going backwards. $\leftleftarrows$

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 $\rightrightarrows$ 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.

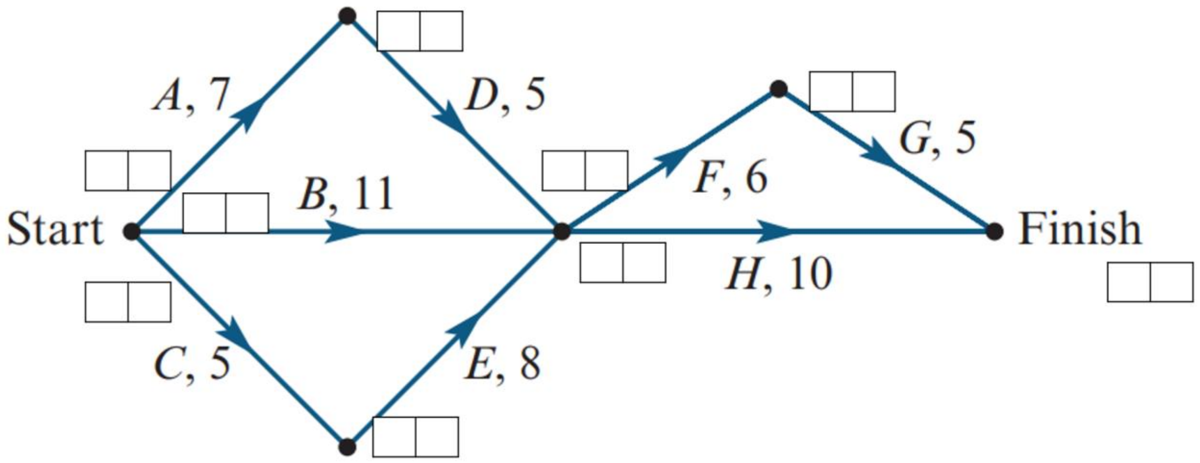
a What is the new minimum completion time, in days, of the project?

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.

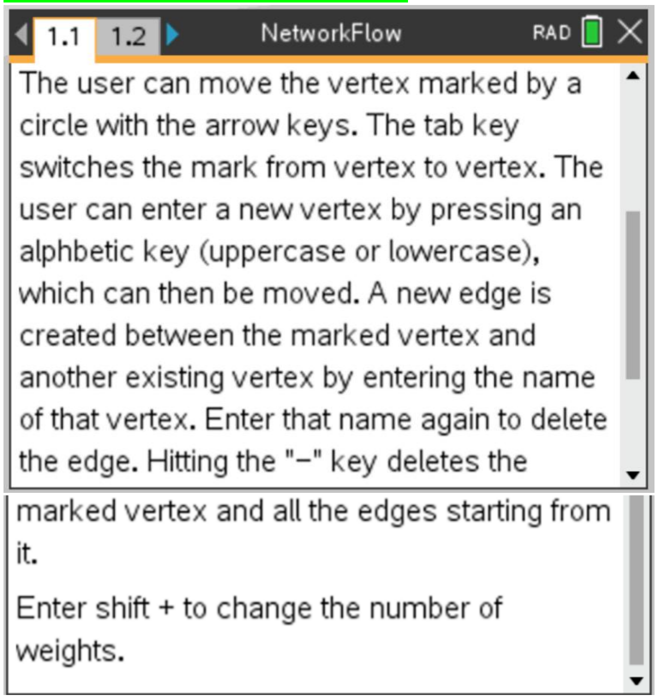
Activity	Daily cost($)
D	170
E	350
H	200

Days reduced	Cost /activity

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



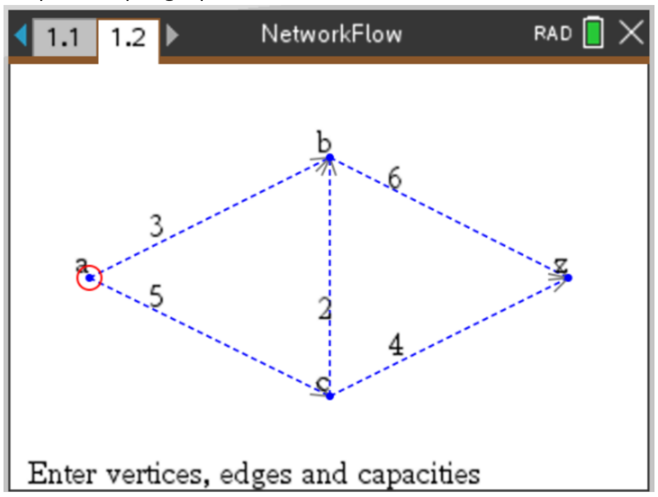
8.1 Using “NetworkFlow” Template



The user can move the vertex marked by a circle with the arrow keys. The tab key switches the mark from vertex to vertex. The user can enter a new vertex by pressing an alphbetic key (uppercase or lowercase), which can then be moved. A new edge is created between the marked vertex and another existing vertex by entering the name of that vertex. Enter that name again to delete the edge. Hitting the "−" key deletes the marked vertex and all the edges starting from it.

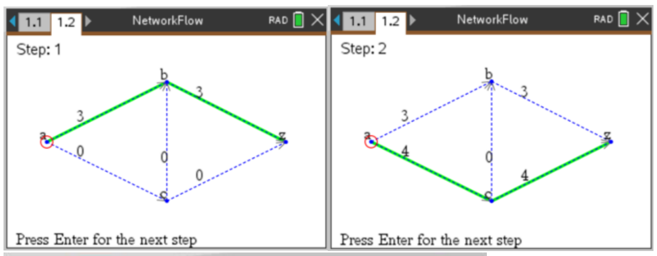
Enter shift + to change the number of weights.

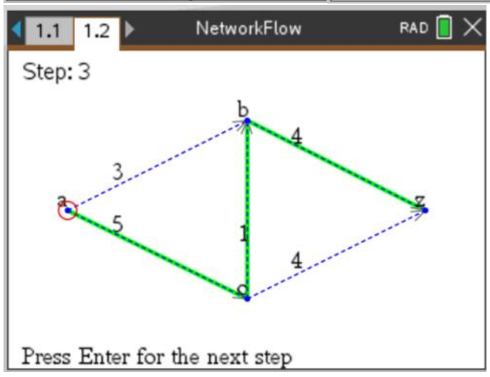
Step 1 Sample graph: **shift letter** for new vertex



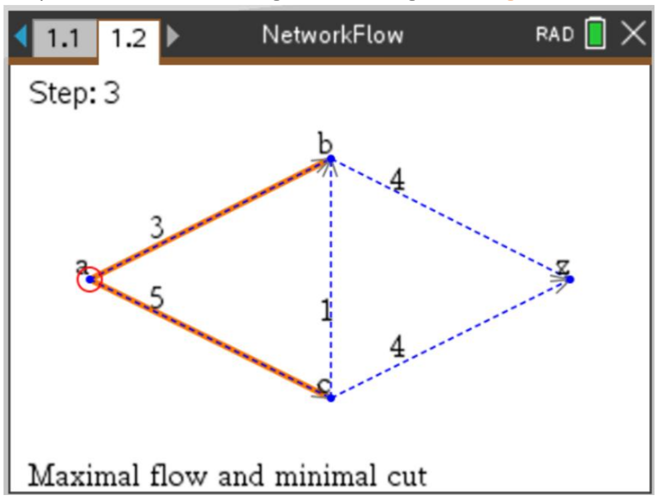
Enter vertices, edges and capacities

Step 2 Press **Enter** to add Max Flow=3+4+1=8 in Green





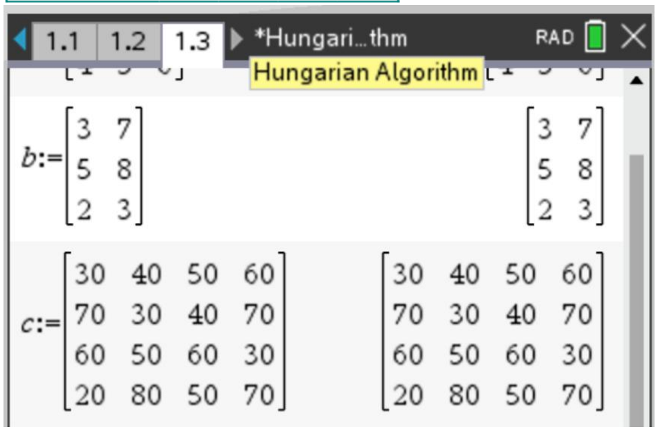
Step 3 Min Cut is showing as following in Orange



8.2 Using “Hungarian Algorithms” Template

Step 1: Define Matrix in file 1.3

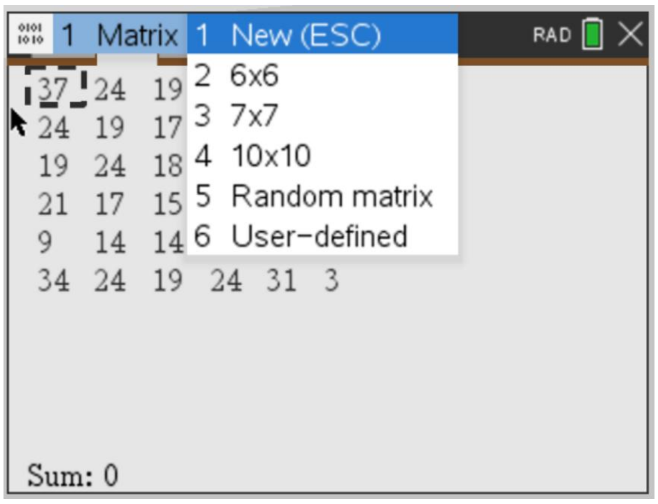
Employee	*A*	*B*	*C*	*D*
Wendy	30	40	50	60
Xenefon	70	30	40	70
Yolanda	60	50	60	30
Zelda	20	80	50	70



$$b:=\begin{bmatrix}3&7\\5&8\\2&3\end{bmatrix}\qquad\begin{bmatrix}3&7\\5&8\\2&3\end{bmatrix}$$

$$c:=\begin{bmatrix}30&40&50&60\\70&30&40&70\\60&50&60&30\\20&80&50&70\end{bmatrix}\qquad\begin{bmatrix}30&40&50&60\\70&30&40&70\\60&50&60&30\\20&80&50&70\end{bmatrix}$$

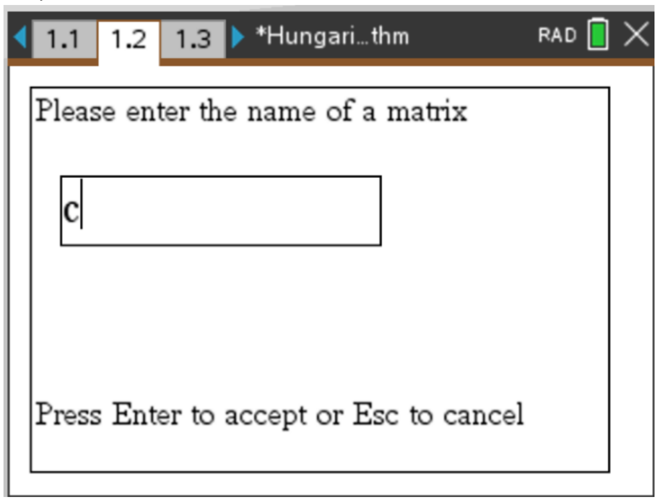
Step 2: Go back to file 1.2 and press **MENU→ Matrix→ User-defined**



```
37 24 19
24 19 17
19 24 18
21 17 15
9  14 14
34 24 19 24 31 3
```

Sum: 0

Step 3: Enter the defined matrix name here

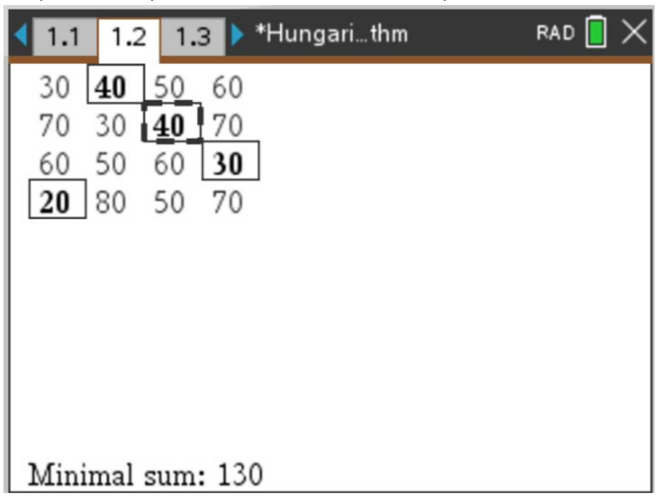


Please enter the name of a matrix

c

Press Enter to accept or Esc to cancel

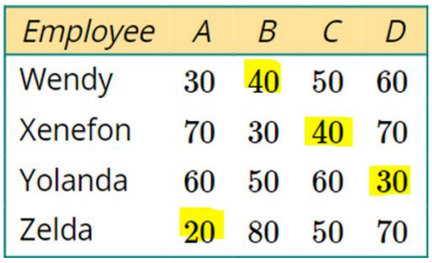
Step 4: Once you see all matrix number, press **ENTER**



```
30  40  50  60
70  30  40  70
60  50  60  30
20  80  50  70
```

Minimal sum: 130

Step 5: Compare the result with manual calculation



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

- The minimum time taken to finish the work=20+30+50+30=130 minutes.