Chapter 09: Directed Graphs

Table of Contents

[Types of Digraphs 5](#_Toc138883226)

[Simple Graphs 5](#_Toc138883227)

[Asymmetric Graphs 5](#_Toc138883228)

[Symmetric Graphs 6](#_Toc138883229)

[Complete Graphs 6](#_Toc138883230)

[Balanced Graphs 7](#_Toc138883231)

[Regular Graph 7](#_Toc138883232)

[Binary Relations 7](#_Toc138883233)

[Reflexive Relation 8](#_Toc138883234)

[Symmetric Relation 8](#_Toc138883235)

[Transitive Relation 9](#_Toc138883236)

[Equivalence Relation 9](#_Toc138883237)

[Directed Paths and Connectedness 10](#_Toc138883238)

[Paths, Walks and Circuits 10](#_Toc138883239)

[Connectedness 11](#_Toc138883240)

[Condensation 12](#_Toc138883241)

[Accessibility 13](#_Toc138883242)

[Euler Digraphs 13](#_Toc138883243)

[Trees with Directed Edges 14](#_Toc138883244)

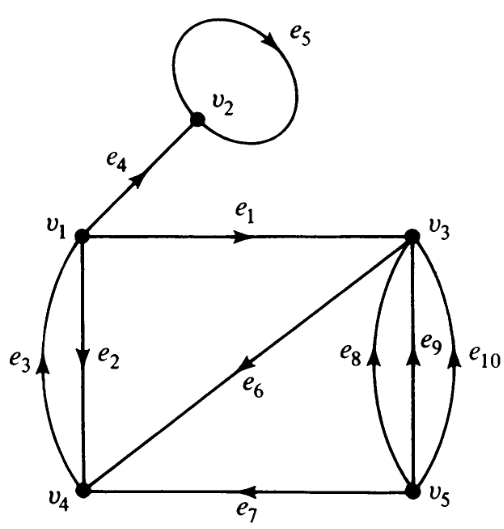
[Polish Notation 15](#_Toc138883245)

[Spanning Arborescence 17](#_Toc138883246)

[Fundamental Circuits in Digraphs 20](#_Toc138883247)

[Paired Comparisons and Tournaments 21](#_Toc138883248)

A **directed graph** (or **digraph**) consists of a set of vertices, , a set of edges, , and a mapping that maps each edge onto an **ordered pair** . The order of the pair is important for directed graphs (unlike their undirected counterparts), because the edge is directed from to .



With reference to the graph above, we will now be looking at a few properties of directed graphs.

Where undirected graphs had the concept of **incidence**, directed graphs have **originating** and **converging** points for edges. An edge is **originating** at a vertex if it is **incident out** of that vertex. The vertex is called the **initial vertex**. An edge is **converging** at a vertex if it is **incident into** a vertex. The vertex is called the **terminal vertex**.

**Self-loops** occur when the incident and terminal vertices are the same vertex.

Undirected graphs had **degrees**, but directed ones subdivide this idea into two parts: **in-degrees** and **out-degrees**. The in-degree of a vertex, , is the number of edges originating at that vertex. The out-degree of a vertex, , is the number of edges converging at that vertex.

Each edge contributes to 1 in-degree and 1 out-degree, meaning .

For an **isolated vertex**, .

For a **pendant vertex**, either or (but never both).

The concept of **parallelism** is also slightly modified for directed graphs. For two edges to be parallel, not only do they have to have the **same vertices** on either end, but they must also be facing the **same direction**. This means that their **originating vertex** and **converging vertex** must be the same. In the graph above, , and are parallel, but and are not.

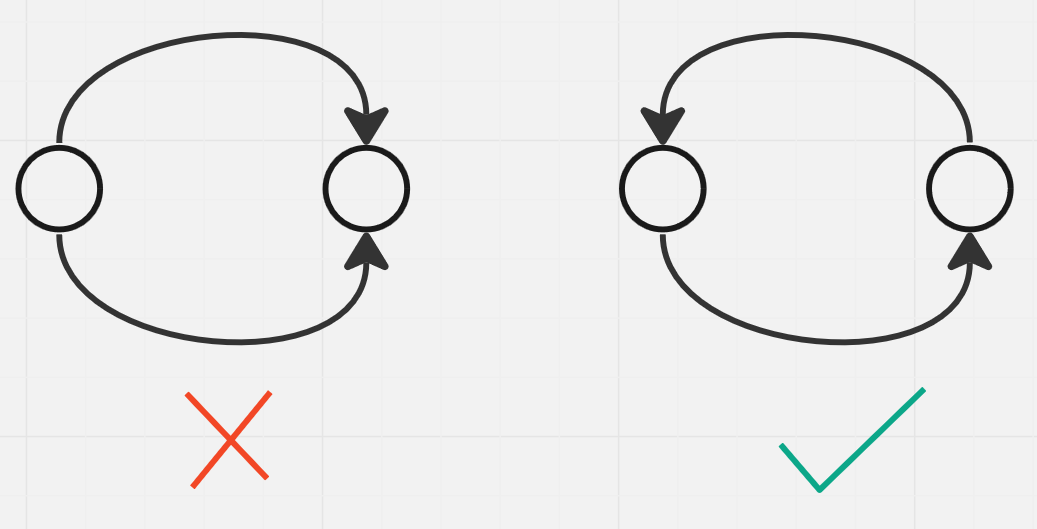
If we remove the directions from a directed graph, we get an **undirected graph**. Specifically, this is called the **corresponding undirected graph**. Similarly, if we add directions to an undirected graph, we get the **corresponding directed graph**. However, note that corresponding directed graph we get depends on how we add directions. As such, each directed graph has **only one corresponding undirected graph**, but each undirected graph can have **multiple corresponding directed graphs**.

For two directed graphs to be **isomorphic**, they must have the same graph theoretical properties. On top of the requirements set for isomorphism by undirected graphs, this means that the **directions of the corresponding edges** must be the same.

## Types of Digraphs

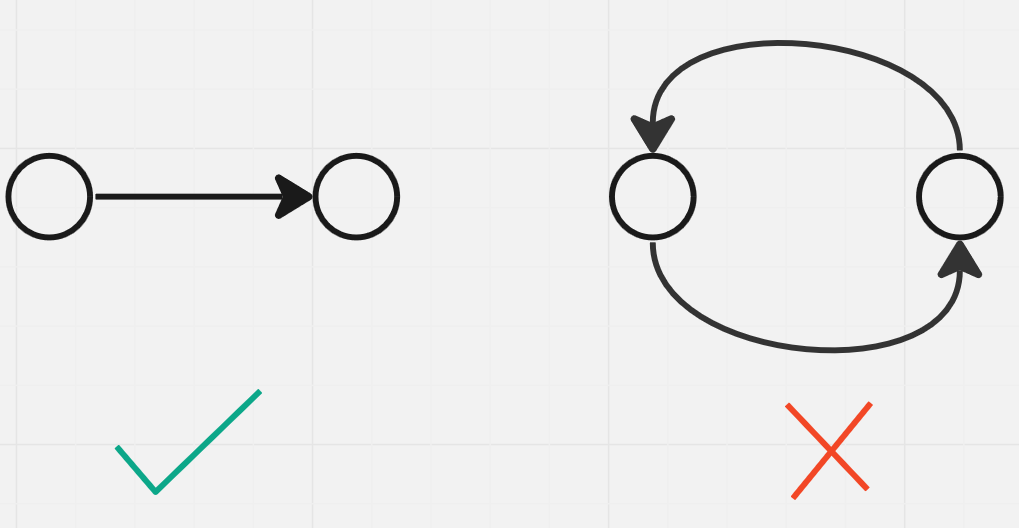
### Simple Graphs

A **simple digraph** has no self-loops or parallel edges.



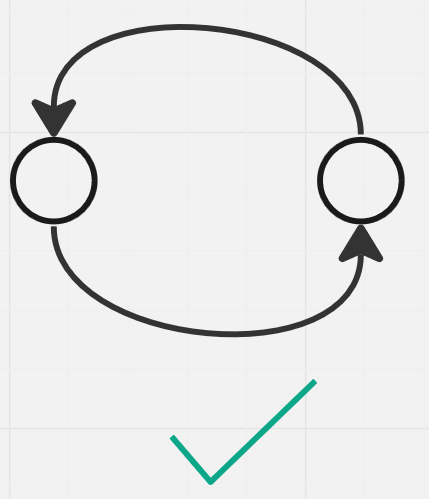
### Asymmetric Graphs

An **asymmetric digraph** has at most 1 directed edge between pairs of vertices. Self-loops are allowed.



### Symmetric Graphs

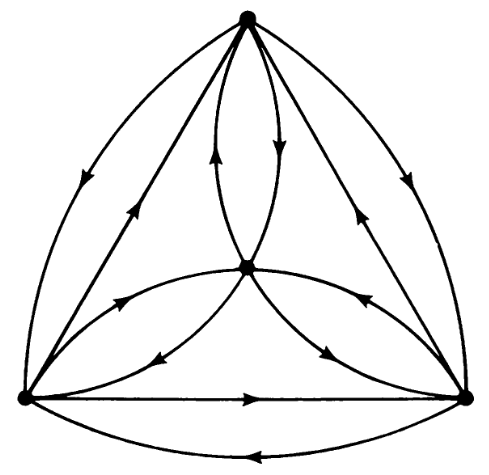
In a **symmetric digraph**, for each edge , there is also an edge . Again, self-loops are allowed.



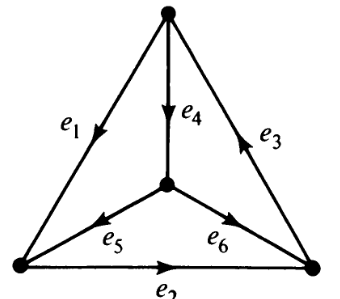
### Complete Graphs

A **complete digraph** can be of two types, symmetric or asymmetric.

A **complete symmetric digraph** is a simple graph with exactly one edge directed from each vertex to every other vertex. Thus, .



A **complete asymmetric digraph** is one for which all pairs of vertices are connected, but the graph is not symmetric. Thus, . Complete asymmetric diagraphs are also called complete tournaments.



### Balanced Graphs

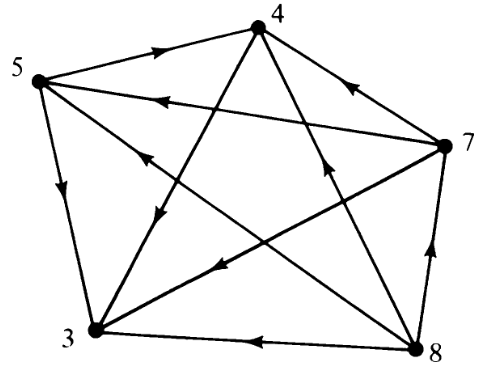
In a **balanced digraph**, for each vertex, .

### Regular Graph

A **regular graph** is a type of balanced graph for which is the same for all vertices. Thus, must also be the same for all vertices (since it is a balanced graph).

## Binary Relations

A **binary relation** () denotes some relationship between two vertices. For example, the relation could be . This relationship can be shown in the form of a graph.



### Reflexive Relation

A **reflexive relation** (), is one where for all vertices. Thus, the graph consists only of **self-loops**. Reflexive relation graphs are also **symmetric**.

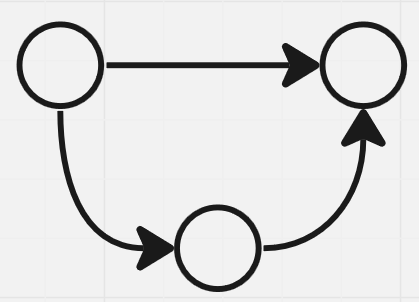
### Symmetric Relation

A **symmetric relation** (if then ) is represented by a **symmetric graph**. Symmetric relation graphs can be **irreflexive** (e.g. is the spouse of ) or **reflexive** (e.g. ).



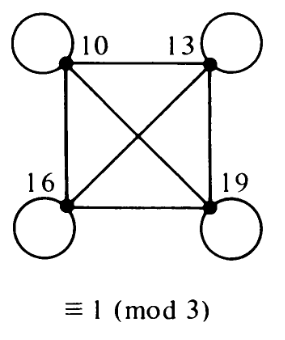
### Transitive Relation

A **transitive relation** says that if and , then . The relation ‘is greater than’ for example, is a transitive relation. The graph below is transitive, irreflexive and asymmetric.



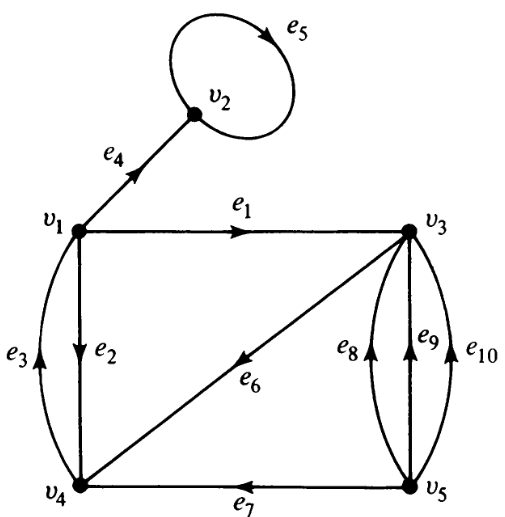
### Equivalence Relation

A **equivalence relation** is reflexive, symmetric and transitive at the same time. This is seen in cases such as ‘is equal to’ and ‘is congruent to’. For example, the graph for is congruent.



## Directed Paths and Connectedness

### Paths, Walks and Circuits



For the graph above, forms a **directed path**. It meets the definition that the vertices preceding and following each edge are the endpoints of the edge and there are no repeated vertices. This is also a path in the corresponding undirected graph.

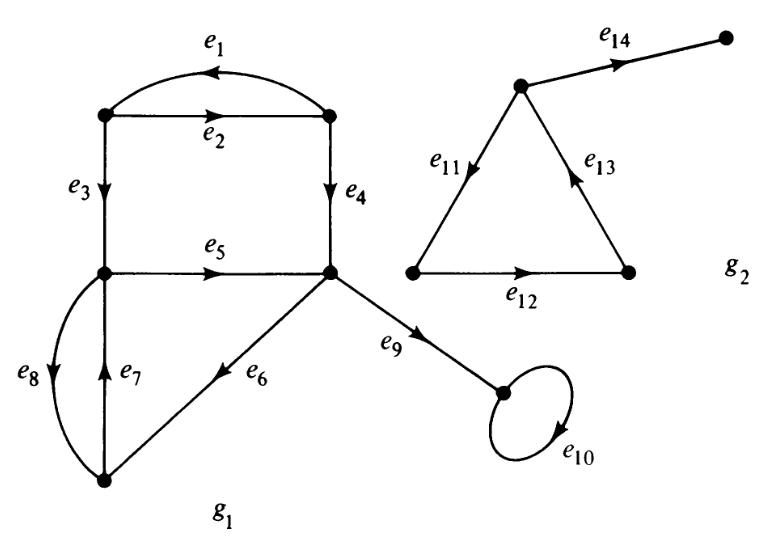
However, is not a directed path even though it is a path in the corresponding undirected graph. A path which exists in the corresponding undirected graph but is not a directed path is called a **semi-path**.

These definitions can also be repeated for **walks** and **circuits**, meaning we have directed walks and circuits and semi walks and circuits.

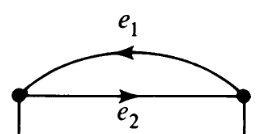
### Connectedness

If there is a **directed path** between all pairs of vertices in a digraph, the graph is said to be **strongly connected**. If there is a **semi-path** between all pairs of vertices (meaning the digraph is not connected by the corresponding undirected graph is connected), the graph is said to be **weakly connected**.

Since the connectedness of a digraph has two types, the **components** of a digraph can also be of two types. A maximally connected subgraph of a graph is called a **component** regardless of whether it is strongly or weakly connected. If it is strongly connected however, it is called a **fragment** or a strongly connected fragment.



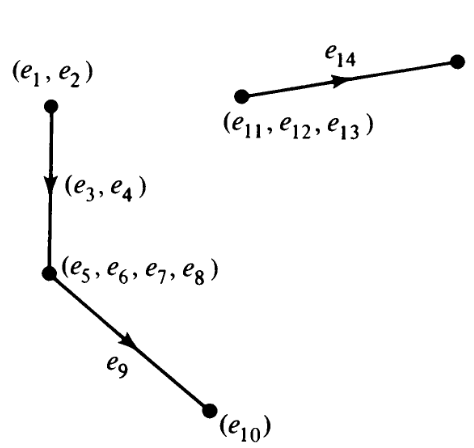
In the graph above, the top-most loop (consisting of and ) forms a **fragment**.



has two fragments. This is because the vertices in the triangle are all reachable from each other (thus forming one fragment) while the top-right-most vertex is a pendant (thus forming another fragment).

### Condensation

If we denote each fragment of a diagraph as a vertex, we can create a new graph. This is called the **condensation** of the graph, . For the graph shown above, the condensation is shown below.



Notice that the edges that are **inside** a fragment become part of the vertex denoting the fragment while the edges between two fragments connect the corresponding vertices.

A question may arise regarding the **ambiguity of the directions** of the edges connecting two fragments. For the case above, we see that and are both edges connecting two fragments (lets say and ). Both of these edges are facing the same direction, but what if they were facing opposite directions? Which direction would we use then?

It is important to recognize that such a case can never exist. If there is one edge going from to and another going from to , then and are part of the **same fragment**. A condensation can **never have a cycle**. This also means that the condensation of a **strongly connected** graph always results in a single vertex.

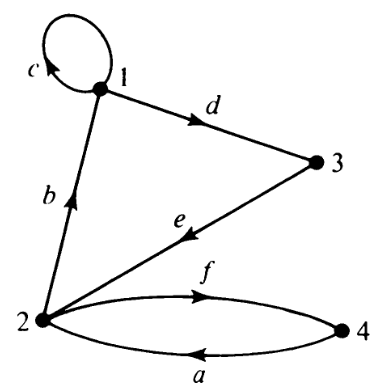
### Accessibility

A vertex in a digraph is said to be **accessible** from a vertex if there is a **directed path** from to .

In a strongly connected graph, all vertices are accessible from one another.

## Euler Digraphs

**Euler graphs** can exist in digraphs as well. If a digraph has a closed walk that uses all its edges exactly once, it is an **Euler digraph**. For example, the graph below is an Euler diagraph.



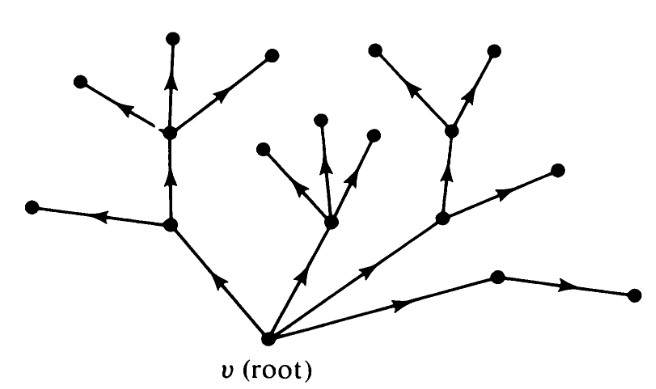
An Euler digraph must be **strongly connected**, with the exception of **isolated vertices**.

Theorem 9.1: A digraph is an Euler diagraph if an only if is connected and is balanced, i.e., for every vertex in .

## Trees with Directed Edges

A **tree** is a connected graph with no circuits (directed or otherwise), vertices and edges.

An **arborescence** is a diagraph with no circuits (directed or otherwise) with exactly one vertex with an in-degree of 0.



Theorem 9.2: An arborescence is a tree in which every vertex other than the root has an in-degree of exactly 1.

Proof: Since there are edges, we know that there must be in-degrees. If the root has an in-degree of 0, the in-degrees must be spread across the vertices. If any vertex has more than 1 in-degree, then some other vertex must have a 0 in-degree, which cannot be. Thus, each vertex has to have exactly 1 in-degree.

As can be seen, an arborescence is a tree that is directed out of the root. Thus, it is also sometimes called an **out tree**. This is opposed to an **in tree**, which would have the directions reversed.

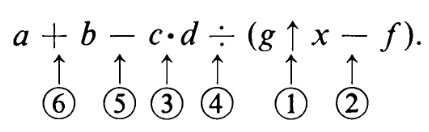
Theorem 9.3: In an arborescence, there is a directed path from the root to every other vertex.

Proof: Since we know that every vertex has an in-degree of 1, that edge must come from some other vertex. The path can only start from the root (since it has no in degree) and can only end at one of the pendent vertices (since otherwise, a vertex would have an in-degree of more than 1).

### Polish Notation

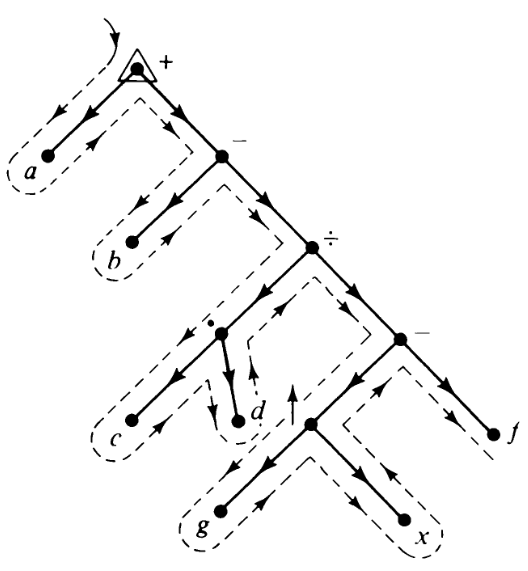
Consider the following equation:

If we write this equation down from left to right in one line it becomes . A compiler however, would have to process the operations in this order:



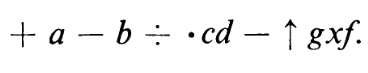
Jumping around to different parts of the equation creates additional complexity, so a different notation was created to solve equations in the order in which a compiler must process the operations.

We will not be studying the details of how the alternative notation is created, but will instead be going over an example. The process uses a **arborescence**, such as the one shown below:



Here, the **pendant vertices** are the operands and the **non-pendant vertices** are the operators. This is an **ordered tree**, meaning unlike normally trees, the order of the vertices matter. Using an isomorph of this tree will give incorrect results.

To create the notation, we must traverse the tree using **in-order traversal** and a stack. The stack is as follows:



This is called the **Polish notation**.

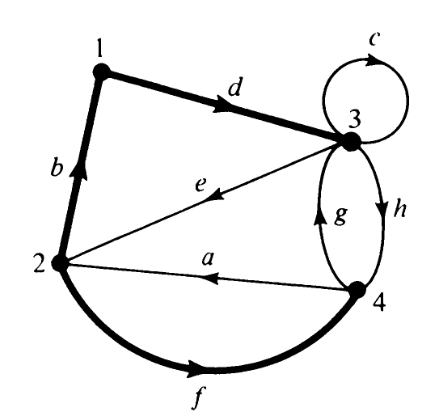
The actually calculate the results, we use another stack in which we push in the operands being popped from the first stack. When we encounter an operator, we pop out two operands from the second stack, calculate the results and push in the results to the second stack.

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

### Spanning Arborescence

A **spanning tree** is a subgraph of a graph which forms a tree and connects all of the vertices of the graph.

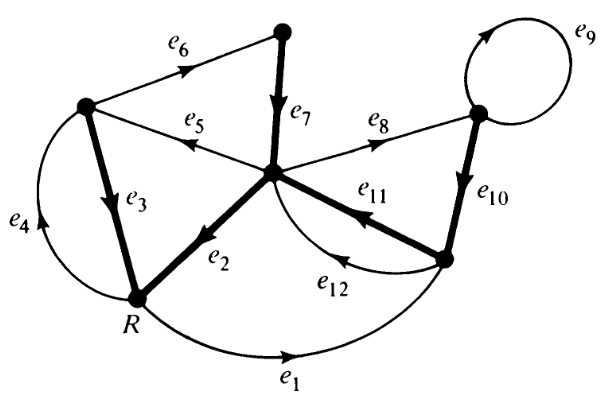
A **spanning arborescence** is a subgraph of a graph which forms an arborescence and connects all of the vertices of the graph.



Theorem 9.4: In a connected, balanced digraph with vertices and edges, let be an Euler line that starts and ends at . Among the edges of , there must be edges that enter each of the vertices other than . There can be multiple edges that enter each of the vertices, but we are only considering the ones which we used the first time. The sub-digraph of these edges along with the vertices forms a spanning arborescence root at .

For the graph above, we can have a spanning tree . Here, the sub-digraph forms a spanning arborescence.

Proof: In the sub-digraph, the root vertex has an in-degree of and every other vertex has an in-degree of since the sub-digraph has only one edge going to each of the other vertices. We also know that the sub-digraph is connected and contains exactly edges. Therefore, it must be a spanning arborescence rooted at .



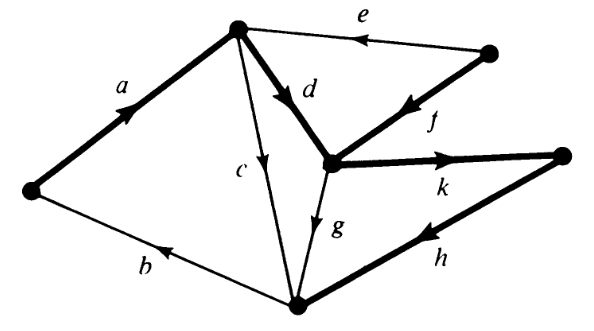
Theorem 9.5: Let be an Euler digraph and be a spanning in-tree in , rooted at a vertex . Let be an edge in incident out of the vertex . Then, a directed walk is a directed Euler line if it is constructed as follows:

1. No edge is included in more than once.
2. In exiting a vertex, the one edge belonging to is not used until all other outgoing edges have been traversed.
3. The walk is terminated only when a vertex is reached from which there is no edge left on which to exit.

Proof: The walk must terminate at because all vertices must have been entered as often as they have been left ( is balanced). Suppose there is an edge, , that is not included in . Let be the terminal vertex of . Since is balanced, must also be the initial vertex of some edge that is also not included in . This in turn means that terminates at some vertex which also has an outgoing edge not included in . Following this, we will eventually arrive at and find an outgoing edge not included in . This contradicts rule 3. Thus, since the graph is balanced, starting at is guaranteed to give us an Euler line if we follow the provided rules.

The Euler line we will get using the above process depends entirely on the in-tree we setup. If we setup a different in-tree, we will get a different Euler line. Specifically, if there are outgoing edges at some vertex , one of which is part of (and must always be selected last), there are combinations. Considering all the vertices, there are thus Different Euler lines.

## Fundamental Circuits in Digraphs



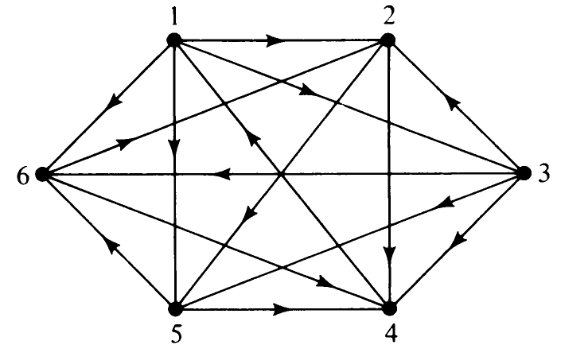
Just like undirected graphs, adding a **chord** to the spanning tree of a digraph forms a **fundamental circuit**. This circuit could be a directed circuit or a semi-circuit. For the graph above, adding the chord forms a directed fundamental circuit while adding the chord forms a fundamental semi-circuit.

We can also form **fundamental cut-sets**. The concept for this remains unchanged for directed graphs. For example, is a fundamental cut-set in both the digraph above and the corresponding undirected graph.

The concept of **ring-sums** can also be applied to digraphs. As with undirected graphs, the ring-sum of two circuits (directed or not) is either a third circuit or the union of edge-disjoint circuits. An important property of a set of fundamental circuits is that we can obtain any circuit in the graph using the ring-sum operation on these fundamental circuits. Unfortunately, the ring-sum of two directed circuits is not guaranteed to produce a third directed circuit, As such, there are no specialties to study here.

## Paired Comparisons and Tournaments

There are several cases where we need to rank several objects even though there is no straightforward metric to measure them against. For example, how would we rank dog foods? There is no way to ask dogs about their preference. As a solution to cases such as these, it has been proposed that we compare the objects two at a time (such as by giving a dog two brands of dog food and seeing which brand they eat first). This process is called the **method of paired comparisons**. There are paired comparisons, which forms a graph called a **preference graph**.



Even if we manage to create a preference graph such as the one shown above, ranking the object remains a difficult task. For example, the graph shows that is preferred over , over but over . From a practical standpoint, this makes no sense. This sort of situation is frequently faced in round-robin tournaments (which is exactly why asymmetric digraphs are called tournaments).

A straightforward way to rank the players in a tournament would be to use a scoring system, the **score** being the number of games they won. This is equal to the **out-degree** of the corresponding vertex. For the graph above, the and score the highest, , and come after that and comes last. This solves the problem to a certain extent but leaves ties at times. Additionally, if the tournament is incomplete (all the players have not participated in the same number of games), this system becomes meaningless.

Another method is to use a **Hamiltonian path**, so that each player has defeated their successor. For the graph above, one such ranking is .

Theorem 9.14: Every complete tournament has a Hamiltonian path.

Ranking using Hamiltonian paths still has its drawbacks. Firstly, there could be discrepancies between the Hamiltonian path and the scores of the players. Secondly, a tournament could have multiple Hamiltonian paths. The tournament we are considering does in fact have two.

A third mechanisms is called **ranking by minimum violations**. A violation is defined as an edge directed from to if precedes in the ranking. For example, if we use the ranking , there are two violations, the edge to and the edge to . This mechanism results in the fewest possible upsets in a tournament. It can be shown that ranking by minimum violations includes the ranking according to score as well as the Hamiltonian ranking. Additionally, it is meaningful for incomplete tournaments as well.

Although ranking by minimum violations is the best ranking mechanism, it is also the most computationally expensive. There are possible orders for vertices and finding the one with the minimum violations is difficult.

The minimum number of violations among all rankings forms the smallest set of edges whose removal from the digraph will eliminate all **directed circuits**, making the digraph acyclic.