Week 4

Vertex Degree and Counting*

If I give you a list of non negative integers representing degree of vertices (say {5,3,1,1} or {2,2}), How do you find out if it is a valid sequence?

4.1 Graphic Sequence

A sequence of numbers can be called as graphic sequence if you can construct a graph using the sequence as degree sequence.

For example {5,3,3,3,2,2} is a valid graphic sequence. The picture representation of {5,3,3,3,2,2} looks like as shown below

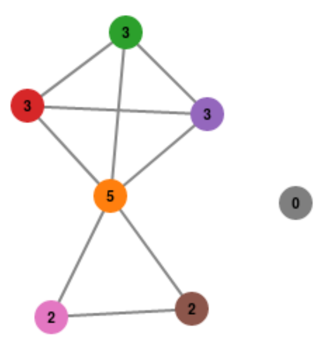


Figure 4.1: Degree Sequence = {5,3,3,3,2,2}

Lets say we have a list of numbers {5,3,3,3,2} we need atleast 5 more vertices to satisfy the graphic sequence, but here we have only 4 so it is not a graphic sequence

*Lecturer: Dr. Anand Mishra. Scribe: Deepak Arjariya.

Note :- A sequence containing Only Zeroes is always Graphic

Note :- A graph has a unique Degree Sequence, but a degree sequence can have multiple corresponding graphs

---

4.2 Havel - Hakimi Algorithm

Havel-Hakimi Algorithm is one such algorithm to determine whether a given sequence is graphic and represent a valid degree sequence.

Theorem 4.1. *Havel - Hakimi Theorem*

Havel - Hakimi Theorem says that

The non-increasing sequence ($a_1, a_2, a_3, a_4, ...$) is graphic if and only if the sequence ($a_2 - 1, a_3 - 1, a_4 - 1, ..., a_{a1+1} - 1, a_{a1+2}, a_{a1+3}, ...$) is also graphic.

Since the first sequence is non-increasing, which means a_1 is the highest degree in the sequence. We took the highest degree vertex and connected it to the next highest degree vertices.
The second sequence is built by removing the highest degree vertex a_1 from the graph and removing corresponding edge from other vertices.

Now the theorem says that either both the sequence are graphic otherwise none are. So we can build new sequences applying the same principle untill we ontain a simple sequence and then we can check if the sequences are valid or not

---

Note :- The sequence after applying theorem may not be non-increasing you have to sort the sequence before applying the algorithm

---

Algorithm

- Pick the vertex with highest degree H
- Connect it with next H highest degree vertices
- Now exhaust this vertex from the graph
- Repeat the above 3 steps untill further reduction of the sequence is not possible

- If the sequence is reduced to all zeroes then it is a valid graphic sequence otherwise it is not graphic

Example

1. Consider a sequence {5, 5, 5,4, 2, 1, 1, 1} If you go through all the steps
   (a) Three point one, point one
   (b) {4, 4, 3, 1, 0, 1, 1}
   (c) {4, 4, 3, 1, 1, 1, 0} after sorting
   (d) {3, 2, 0, 0, 1, 0}
   (e) {3, 2, 1, 0, 0, 0} after sorting
   (f) the sequence can not be reduced further and we are left with a non zero sequence. Hence by Havel-Hakimi Algorithm all the sequence produced are not Graphic including the original one

Theorem 4.2. *If u and v are the only vertices of odd degree in a graph G, Then G must contain u-v Path*

We will use contradiction to prove the theorem
Let us say that u and v are not connected
Then u and v will be contained in separate subgraphs G_1 and G_2 and u will be the only node with odd degree in subgraph G_1 and v will be the only node with odd degree in subgraph G_2
Sum of all degrees in G_1 and G_2 will be odd which is the voilation of the **HandShaking Lemma** in Graph Theory
Hence there has to be a path between u and v in Graph G

4.2.1 Some Questions

Question

Determine the Maximum Number of edges in maximal bipartite subgraphs of P_n, C_n, K_n
For P_n
Maximal bipartite subgraph of P_n is P_n
$P_n \subseteq P_n$ so Maximum Number of edges in maximal bipartite subgraphs of $P_n = n - 1$
For C_n
so Maximum Number of edges in maximal bipartite subgraphs of C_n

$$E = g(x) = \begin{cases} n & \text{if n is even,} \\ n-1 & \text{if n is odd.} \end{cases}$$

For K_n
Maximal bipartite subgraph of K_n is $K_{[n/2,n/2]}$
so Maximum Number of edges in maximal bipartite subgraphs of $C_n = n^2/4$ edges

4.3 Path , Walk, Trail, Cycle and Circuits in a graph

A **walk** is an ordered sequence of vertices or edges of a graph where where end point of one edge is the starting of immediate next edge except last edge. if we traverse a graph we get a walk

Walk can be closed or open. In an open walk, starting and ending vertices are same while in closed walk they are not

In the given figure

$1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 2 \rightarrow 1$ is a closed walk

$1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 2$ is a open walk

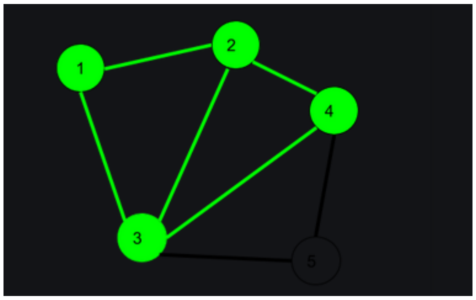


Figure 4.2: Walk:- $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 2 \rightarrow 1 \rightarrow 3$

A **Trail** is a open walk where edges can not repeat but vertices can.

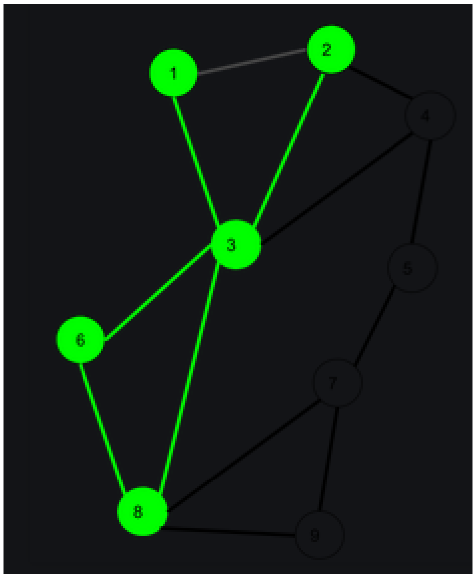


Figure 4.3: Trail:- $1 \rightarrow 3 \rightarrow 8 \rightarrow 6 \rightarrow 3 \rightarrow 2$

A **Circuit** is a closed walk where edges can not repeat but vertices can.

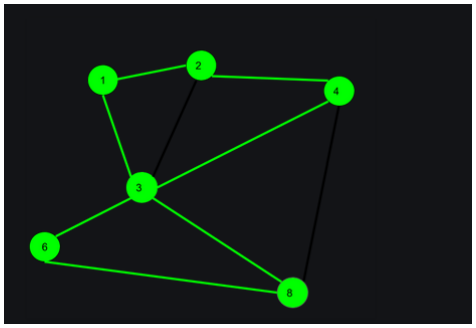


Figure 4.4: Circuit:- $1 \rightarrow 2 \rightarrow 4 \rightarrow 3 \rightarrow 6 \rightarrow 8 \rightarrow 3 \rightarrow 1$

A **Cycle** is a closed walk where edges and vertices are distinct except the end point vertices.

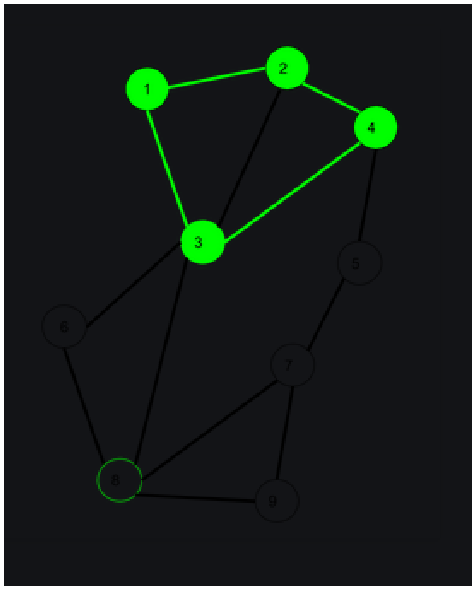


Figure 4.5: Cycle:- $1 \rightarrow 2 \rightarrow 4 \rightarrow 3 \rightarrow 1$

A **Path** is a trail where all the edges and vertices are distinct.

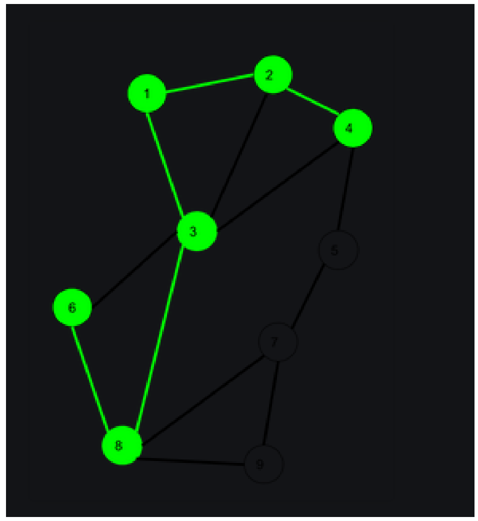


Figure 4.6: Path:- 6 → 8 → 3 → 1 → 2 → 4

4.4 Directed Graphs

A **directed Graph** also known as **Digraph** is a triple consisting of vertex Set **V(G)**, an edge Set **E(G)** and a **function** assigning each edge an ordered pair of vertices. First vertex in Edge is called as **Tail**, second as **Head** and together they are called as **Endpoints**. We say that an edge from **Tail** to **Head**

For Example :- In the given Figure

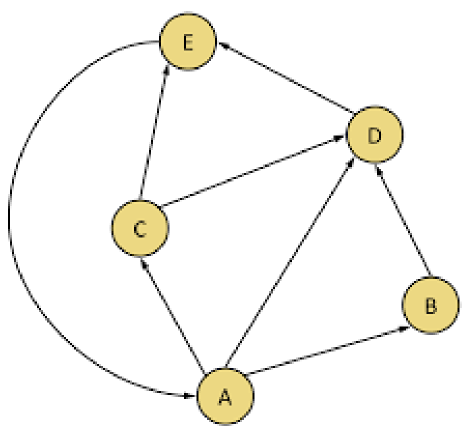


Figure 4.7: Directed Graph

A $\rightarrow B$ is a directed Edge
where A is a tail
B is a head
and together A, B are endpoints

4.4.1 Underlying Graph

UnderLying Graph of a Directed Graph D is a graph G obtained by treating every edge of F as unordered pairs

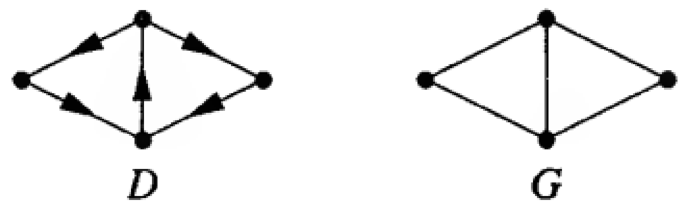


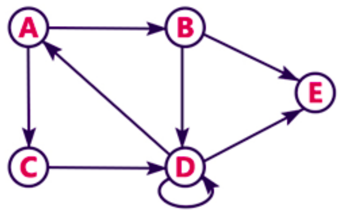
Figure 4.8: Corresponding Underlying graph G of Directed Graph D

4.4.2 Representing Directed Graphs

Graph Representation is a technique to store graph into the memory of computer There are different techniques for representing a graph

Adjacency matrix

Adjacency matrix is a sequential 2D representation of a graph in the from of a matrix where A_{ij} represent an edge from node i to j. In case of weighted graph A_{ij} represent the weight of an edge

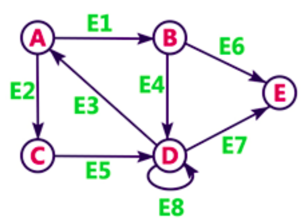


$$
\begin{array}{c} \\ A \\ B \\ C \\ D \\ E \end{array}
\begin{array}{c}
\begin{array}{ccccc} A & B & C & D & E \end{array} \\
\begin{bmatrix}
0 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0
\end{bmatrix}
\end{array}
$$

Figure 4.9: Corresponding adjacency matrix of a Directed Graph

Incident matrix

Incident Matrix is also a sequential 2D representation of a graph in the form of a matrix where where A_{ij} represent an edge j whose one of the endpoint is i



$$
\begin{array}{c} \\ A \\ B \\ C \\ D \\ E \end{array}
\begin{array}{c}
\begin{array}{cccccccc} E1 & E2 & E3 & E4 & E5 & E6 & E7 & E8 \end{array} \\
\begin{bmatrix}
1 & 1 & -1 & 0 & 0 & 0 & 0 & 0 \\
-1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\
0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & -1 & -1 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & -1 & -1 & 0
\end{bmatrix}
\end{array}
$$

Figure 4.10: Corresponding incident matrix of a Directed Graph

4.4.3 Weakly and Strongly connected Digraphs

A digraph is **weakly connected** is its underlying graph is connected. Whereas a Digraph is **strongly Connected** is for every ordered pair u and v of vertices, there is a path from u to v.
The **strong Components** of a Digraph are its **maximal strong subgraphs**.

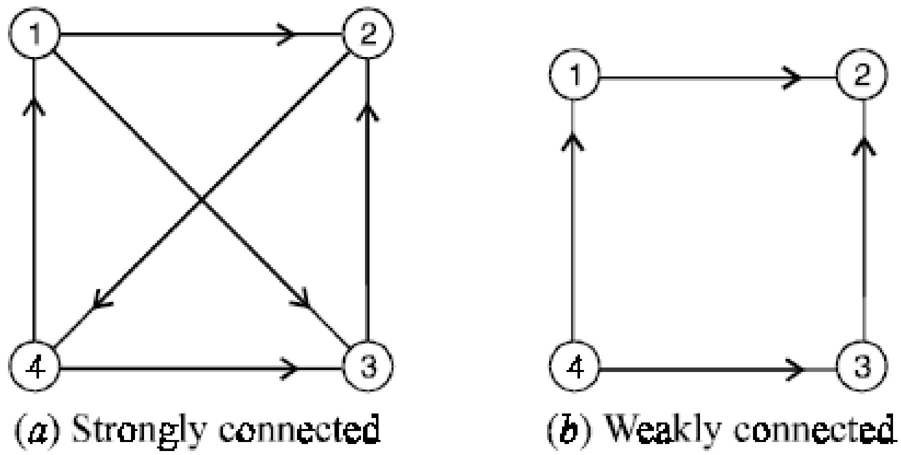


Figure 4.11: Strongly and weakly connected Digraphs

4.4.4 Kernel Of Digraph

A kernel in a diagraph D is a set S $\subseteq V(D)$ such that S induces no edges and every vertex outside S has a successor in S.

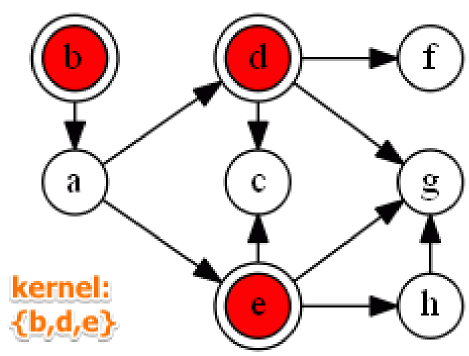


Figure 4.12: Kernel of a Directed Graph