**Harold’s Undirected Graphs and Trees**

**Cheat Sheet**

12 December 2024

**Definitions**

|  |  |  |
| --- | --- | --- |
| **Term** | **Definition** | **Example** |
| **Vertices**  **(Nodes)** | An individual element of V is called a vertex.  A graph is **finite** if the vertex set is finite. | Set  ① or ● |
| **Edges**  **(Arcs)** | An edge (u, v) ∈ E, is pictured as an arrow going from one vertex to another. | Set E ⊆ V x V |
| **Self-Loop (Loop)** | An edge that connects a vertex to itself. |  |
| **Undirected Graph** | A graph whose edges are unordered pairs of vertices. | G = (V, E) |
| **Simple Graph** | A graph with no parallel edges or self-loops. | |Cycle| ≥ 3 |
| **Adjacent** | There is an edge between two vertices. | Two vertices are connected. |
| **Endpoints** | Vertices b and e are the **endpoints** of edge {b, e} | The two vertices of an edge. |
| **Incident** | The edge {b, e} is **incident** to vertices b and e. | The edge of two vertices. |
| **Neighbor** | A vertex c is a **neighbor** of vertex b if and only if {b, c} is an edge. | Has an edge to it. |
| **Degree** | The **degree** of a vertex is the number of neighbors it has. |  |
| **Total Degree** | The sum of the degrees of all of the vertices. |  |
| **Regular Graph** | All the vertices have the same degree. |  |
| **d-Regular Graph** | All the vertices have degree d. | 3-Regular Graph: |
| **Subgraph** | A graph H = (VH, EH) is a ***subgraph*** of a graph G = (VG, EG) if VH ⊆ VG and EH ⊆ EG.  Any graph G is a subgraph of itself. | 2-Regular Graph: |
| **Common Graphs** | **K6**: Complete Graph (Clique)    Has an edge between every pair of vertices. | **C7**: Cycle |
| **Q3**: 3-Dimentional Hypercube    Has 2n vertices. | **K3,4**:    No edges between vertices in the same set. |
| **P5**: A path | **S5**: Star |

**Graph Representation**

|  |  |
| --- | --- |
| **Term** | **Description** |
| **Adjacency List** | Each vertex has a list of all its neighbors. |
| **Matrix** | A ‘1’ means an edge is present. Is symmetrical about the diagonal (Mi,j = Mj,i). |
| **Isomorphic** | There is a one-to-one correspondence between each of the edges of two graphs (bijection). |
| **"Efficient" Algorithm** | An algorithm that runs in worst-case polynomial time. |
| **Theorem: Vertex degree preserved under isomorphism** | Consider two graphs, G and G'. Let f be an isomorphism from G to G'. For each vertex v in G, the degree of vertex v in G is equal to the degree of vertex f(v) in G'.  If one graph has a vertex of degree 1 and the other graph does not, then not isomorphic. |
| **Degree Sequence** | A list of the degrees of all of the vertices in non-increasing order. |
|  | |
| **Theorem: Degree sequence preserved under isomorphism** | Degree sequence is preserved under isomorphism.  If two graphs are isomorphic, they have the same degree sequence. |
| **Graph Theory** | Concerned with properties of graphs that are preserved under isomorphism.  Preserved:   * Number of vertices (|V|) * Number of edges (|E|) * Degree sequence (degrees listed high to low) * Total degree (2·|E|)   Not Preserved:   * The lowest numbered vertex has degree 3 * Every even numbered vertex has odd degree |

**Graph Types**

|  |  |  |  |
| --- | --- | --- | --- |
| **Term** | **Description** | **Example** | **Graph** |
| **Walk** | A sequence of alternating vertices and edges that starts and ends with a vertex. |  |  |
| **Open Walk** | A walk in which the first and last vertices are not the same. |  |  |
| **Closed Walk** | A walk in which the first and last vertices are the same. |  |  |
| **Length** | *l*, the number of edges in the walk, path, or cycle. | if a sequence |  |
| **Trail** | An open walk in which no edge occurs more than once. |  |  |
| **Circuit** | A closed walk in which no edge occurs more than once. |  |  |
| **Path** | A trail in which no vertex occurs more than once. |  |  |
| **Cycle** | A circuit of length at least 1 in which no vertex occurs more than once, except the first and last vertices which are the same. |  |  |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | |  | **Repeat Vertex** | **Repeat Edge** | **Closed** | **Open** | | **Walk** |  |  |  |  | | **Trail** |  |  |  |  | | **Circuit** |  |  |  |  | | **Path** |  |  |  |  | | **Cycle** |  |  |  |  | | | | |

**Connectivity**

|  |  |  |
| --- | --- | --- |
| **Term** | **Description** | **Example** |
| **Connected** | If there is a path from vertex v to vertex w, then there is also a path from w to v. The two vertices, v and w, are said to be connected. |  |
| **Disconnected** | A graph is said to be connected if every pair of vertices in the graph is connected, and is disconnected otherwise. |
| **Connected Component** | A maximal set of vertices that is connected. | See graph above for examples. |
| **Isolated Vertex** | A vertex that is not connected with any other vertex is called an isolated vertex and is therefore a connected component with only one vertex. | ● |
| **k-Vertex-Connected** | The graph contains at least k + 1 vertices and remains connected after any k - 1 vertices are **removed** from the graph. (mesh network) | 2-vertex-connected: |
| **Vertex Connectivity** | The largest k such that the graph is k-vertex-connected. | κ(G) |
| **k-Edge-Connected** | The graph remains connected after any k - 1 edges are removed from the graph. | 3-edge-conncted: |
| **Edge Connectivity** | The largest k such that the graph is k-edge-connected. | λ(G) |
| **Theorem: Upper bound for vertex and edge connectivity** | Let G be an undirected graph. Denote the minimum degree of any vertex in G by *δ(G)*. Then *κ(G) ≤ δ(G)* and *λ(G) ≤ δ(G)*. | The minimum degree of any vertex is at least an upper bound for both the edge and vertex connectivity of a graph. |
| **Complete Graph** | There is no set of vertices whose removal disconnects the graph. | Full mesh network. |

**Euler Circuits and Trails**

|  |  |  |
| --- | --- | --- |
| **Term** | **Description** | **Example** |
| **Euler Circuit** | An undirected graph circuit that contains every edge and every vertex.  Every vertex reached.  Every edge occurs exactly once. |  |
| **Theorem: Required conditions for an Euler circuit in a graph** | If an undirected graph G has an Euler circuit, then G is 1) connected and 2) every vertex in G has an even degree. | where |
| **Theorem: Sufficient conditions for an Euler circuit in a graph** | If an undirected graph G is connected and every vertex in G has an even degree, then G has an Euler circuit. | |
| **Theorem: Characterization of graphs that have an Euler circuit** | An undirected graph G has an Euler circuit if and only if G is connected and every vertex in G has even degree. | |
| Procedure | Find circuit C in G.  Repeat until C is an Euler circuit:  Create new graph G' :  Remove edges in C from G  Remove isolated vertices  Find vertex w in G' and C (select any)  Find circuit C' in G' starting at w  Combine C and C'  Follow edges in C to w  Follow edges in C' back to w  Follow remaining edges in C  Rename new circuit to be C | |
| **Euler Trail** | An undirected graph open trail that includes each edge exactly once. |  |
| **Theorem: Characterizations of graphs that have an Euler trail** | An undirected graph G has an Euler trail if and only if G is 1) connected and 2) has exactly two vertices with odd degree. | Euler trail begins and ends with vertices of odd degree. |

**Tree Term Definitions**

|  |  |  |
| --- | --- | --- |
| **Term** | **Description** | **Example** |
| **Tree** | An undirected graph that is connected and has no cycles. | Computer file system |
| **Free Tree** | There is no particular organization of the vertices and edges |  |
| **Rooted Tree** | The vertex at the top is designated as the **root** of the tree. |  |
| **Level** | The **level** of a vertex is its distance (number of edges in the shortest path between the two vertices) from the root. | The root is the only level 0 vertex. |
| **Height** | The **height** of a tree is the highest level of any vertex. | Most hops to bottom. |
| **Parent** | The first vertex after v encountered along the path from v to the root. (One vertex above v.) | The parent of vertex g is h. |
| **Child** | The vertex below the parent. | Vertices c and g are the children of vertex h. |
| **Ancestor** | All vertices up in path. | The ancestors of vertex g are h, d, and b. |
| **Descendant** | All vertices down in path. | The descendants of vertex h are c, g, and k. |
| **Leaf** | Rooted: A vertex which has no children.  Free: A vertex of degree 1. | The leaves are a, f, c, k, i, and j. |
| **Sibling** | Vertices with the same parent. | Vertices h, i, and j are siblings of parent d. |
| **Subtree** | A tree consisting of new root v and all v's descendants. | The subtree rooted at h includes h, c, g, and k and the edges between them. |
| **Game Tree** | Shows all possible playing strategies of both players in a game.  Games can be deterministic (tic-tac-toe) or chance (dice). | *vi* = game configuration |
| **Variable Length Codes** | The number of bits for each character can vary. | ‘a’ = 1, ‘e’ = 01, etc. |
| **Prefix Code** | The code for one character cannot be a prefix of the code for another character. | Leaf nodes guarantee the prefix property. |
| **ASCII** | 8-Bit characters (256 max.) | UTF-8 |
| **Unicode** | 16-Bit characters (64K max.) | UTF-16 |
| **Internal Vertex** | Free: The vertex has degree at least two. |  |
| **Forest** | A graph that has no cycles and that is not necessarily connected.  |E|= |V| – |C| (connected components) |  |

**Tree Theorems**

|  |  |  |
| --- | --- | --- |
| **Term** | **Description** | **Example** |
| **Theorem: Unique paths in trees** | Let T be a tree and let u and v be two vertices in T. There is exactly one path between u and v.  There is a unique path between every pair of vertices in a tree. |  |
| **Theorem: Number of edges in a tree** | Let T be a tree with n vertices and m edges, then m = n - 1. |  |
| **Theorem: Number of leaves in a tree** | Any free tree with at least two vertices has at least two leaves. | Lower bound |
| **Theorem: Prim's Algorithm** | Prim's algorithm finds a minimum spanning tree of the input weighted graph. | See Spanning Trees below |

**Tree Traversals**

|  |  |  |
| --- | --- | --- |
| **Term** | **Description** | **Example** |
| **Traversal** | Systematically visiting each vertex. | Hit a node. |
| **Pre-Order Traversal** | A vertex is visited before its descendants. | First hit (left side) of tree vertex |
| **In-Order Traversal** | A vertex is visited after its first descendant. | 2nd hit of tree vertex |
| **Post-Order Traversal** | A vertex is visited after its descendants. | Last hit (right side) of tree vertex |

**Spanning Trees**

|  |  |  |
| --- | --- | --- |
| **Term** | **Description** | **Example** |
| **Spanning Tree** | For a connected graph G. a subgraph of G which contains all the vertices in G and is a tree. | Fewest edges possible to visit all vertices |
| **Depth-First Search (DFS)** | Favors going deep into the graph.  Produces trees with longer paths. | Explorer ventures far away from home |
| **Breadth-First Search (BFS)** | Explores the graph by distance from the initial vertex, starting with its neighbors and expanding the tree to neighbors of neighbors.  Produces trees with shorter paths. | Explorer ventures close to home |
| **Weighted Graph** | A graph G = (V ,E), along with a function w: E → ℝ. | The function w assigns a real number to every edge. |
| **Weight w(G)** |  | w(G) is the sum of the weights of the edges in G. |
| **Minimum Spanning Tree (MST)** | A spanning tree T of G whose weight is no larger than any other spanning tree of G. | Goal: Min. weight |
| **Prim's Algorithm** | A classic algorithm for finding minimum spanning trees developed by mathematician Robert Prim in 1957. | Always choose min. edge in queue. |
| Input: An undirected, connected, weighted graph G.  Output: T, a minimum spanning tree for G.  T := ∅.  Pick any vertex in G and add it to T.  For j = 1 to n-1  Let C be the set of edges with one endpoint inside T and one endpoint outside T.  Let e be a minimum weight edge in C.  Add e to T.  Add the endpoint of e not already in T to T.  End-for | |

**Hamiltonian Cycle**

|  |  |
| --- | --- |
| **Term** | **Description** |
| **Hamiltonian Path** | A [path](https://en.wikipedia.org/wiki/Path_(graph_theory)) in an undirected or directed graph that visits each [vertex](https://en.wikipedia.org/wiki/Vertex_(graph_theory)) exactly once. |
| **Hamiltonian Cycle** | A [cycle](https://en.wikipedia.org/wiki/Cycle_(graph_theory)) that visits each vertex exactly once. |
| **Orthographic projections and Schlegel diagrams with Hamiltonian cycles** |  |

**Dijkstra’s Algorithm**

|  |  |
| --- | --- |
| **Term** | **Description** |
| **Dijkstra's Algorithm** | An algorithm for finding the shortest paths between nodes in a weighted graph. |
| **Fundamentals of Dijkstra's Algorithm** | 1. Dijkstra's Algorithm begins at the node we select (the source node), and it examines the graph to find the shortest path between that node and all the other nodes in the graph. 2. The Algorithm keeps records of the presently acknowledged shortest distance from each node to the source node, and it updates these values if it finds any shorter path. 3. Once the Algorithm has retrieved the shortest path between the source and another node, that node is marked as 'visited' and included in the path. 4. The procedure continues until all the nodes in the graph have been included in the path. In this manner, we have a path connecting the source node to all other nodes, following the shortest possible path to reach each node. |
|  | |

**Sources**:

* [SNHU MAT 230](https://www.snhu.edu/admission/academic-catalogs/coce-catalog#/courses/4kVhSZLtg) - Discrete Mathematics, zyBooks.
* See also “Harold’s Directed Graphs Cheat Sheet”.
* Towards AI (2024). <https://towardsai.net/p/l/the-value-of-graph-theory-within-sustainability>
* Wikipedia (2024). Hamiltonian Path. <https://en.wikipedia.org/wiki/Hamiltonian_path>