**Harold’s Directed Graphs**

**Cheat Sheet**

22 October 2022

**Definitions**

|  |  |  |
| --- | --- | --- |
| **Term** | **Definition** | **Example** |
| **Vertices****(Nodes)** | An individual element of V is called a vertex. | Set $V=\{a, b, c, d, e\}$① or ● |
| **Edges****(Arcs)** | A directed edge (u, v) ∈ E, is pictured as an arrow going from one vertex to another. | Set E ⊆ V x V$$E=\{(a, b), (a, c), ..., (d, e)\}$$ |
| **Directed Graph****(Digraph)** | A finite set of dots called vertices (or nodes) that are connected by links called edges (or arcs). Consists of a pair (V, E).A sequence of vertices in which there is a (directed) edge pointing from each vertex in the sequence to its successor in the sequence, with no repeated edges. | Anatomy of a Graph |
| **Self-Loop (Loop)** | An edge that connects a vertex to itself. |  |
| **In-Degree** | The number of edges pointing into, to, or with v as their terminal vertex. | $$in-degree\left(v\right)=$$$$|\{ u | (u, v) \in E \}|$$ |
| **Out-Degree** | The number of edges pointing out of, from, or with v as their initial vertex. | $$out-degree\left(v\right)=$$$$|\{ u | (v, u) \in E \}|$$ |
| **Walk** | A sequence of alternating vertices and edges that starts and ends with a vertex. | $$\left〈v\_{0}, (v\_{0}, v\_{1}), v\_{1}, (v\_{1}, v\_{2}), v\_{2}, ..., v\_{l}\right〉$$$$\left〈v\_{0}, v\_{1}, v\_{2}, ..., v\_{l}\right〉$$ |
| **Open Walk** | A walk in which the first and last vertices are not the same. | $$\left〈a, ... , z\right〉$$ |
| **Closed Walk** | A walk in which the first and last vertices are the same. | $$\left〈a, ... , a\right〉$$ |
| **Length** | l, the number of edges in the walk, path, or cycle. | $$I=\left|E\right|$$ |
| **Trail** | An open walk in which no edge occurs more than once. | $$\left〈a, b, c, d, c, b, a\right〉$$ |
| **Circuit** | A closed walk in which no edge occurs more than once. | $$\left〈a, b, a, c, a\right〉$$ |
| **Path** | A trail in which no vertex occurs more than once. | $$\left〈a, b, c, d\right〉$$ |
| **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. | $$\left〈a, b, c, a\right〉$$ |
| **DAG** | A directed acyclic graph (or DAG) is a digraph with no directed cycles. | Directed Acyclic Graph (DAG) Overview & Use Cases | Hazelcast |

**Digraph Theorems**

|  |  |
| --- | --- |
| **Theorem** | **Definition and Examples** |
| **Graph Power Theorem (Gk)** | Let G be a directed graph. Let u and v be any two vertices in G. There is an edge from u to v in Gk if and only if there is a walk of length k from u to v in G. |
|  |
| **Transitive Closure** | The union of Gk for all k ≥ 1 (denoted G+) represents reachability by walks of any length in G.G+ = G1 U G2 U G3 U G4 … (infinite or up to |V|)G+ = G1 U G2 U G3 U … U Gn (finite with n vertices)R+ = R1 U R2 U R3 U … U Rn (finite with n elements) |
|  |
| **Procedure to find the transitive closure of a relation R on a set A** | Repeat the following step until no pair is added to R:● If there are three elements x, y, z ∈ A such that (x, y) ∈ R, (y, z) ∈ R and (x, z) ∉ R, then add (x, z) to R. |
|  |

**Boolean Matrix Operations**

|  |  |
| --- | --- |
| **Term** | **Description** |
| **Adjacency Matrix** | A directed graph G with n vertices that is represented by an n × n matrix over the set {0, 1}.Ai,j = 1 if there is an edge from vertex i to vertex j in G, otherwise, Ai,j = 0. |
|  |
| **Boolean Matrix****{0, 1}** | A matrix whose entries are from the set {0, 1}.**Purpose:** Matrix addition and multiplication for square Boolean matrices are used to compute the transitive closure of a graph. |
| **Dot Product** | For Boolean matrices, if the dot product (sum of products) ≥ 1, then dot product = 1. |
| **Matrix Product****(AB)** | The product of two matrices, A and B, is **well defined** only if the number of columns in A is equal to the number of rows in B.Associative, but not commutative. |
| **kth Power of a Matrix****(Ak)** | $$A^{2}=A•A$$$$A^{3}=A^{2}•A$$ |
| **Matrix Ak is the Adjacency Matrix for Graph Gk** | Let G be a directed graph with n vertices and let A be the adjacency matrix for G. Then for any k ≥ 1, Ak is the adjacency matrix of Gk, where Boolean addition and multiplication are used to compute Ak.There is a walk of length k in G from vertex v to vertex w if and only if the entry in row v, column w in Ak is 1. |
| **How to read it:**There is a walk of length 3 in G from vertex 1 to vertex 3 if and only if there is an edge from 1 to 3 in G3. If row 1, column 3 of A3 is 0, then no such walk exists. |
| **Matrix Sum****(A+B)** | The sum of two matrices A and B is **well defined** if A and B have the same number of rows and the same number of columns. |
| For Boolean matrices, if the sum ≥ 1, then sum = 1. |
| **Addition and Graph Union** | Let G and H be two directed graphs with the same vertex set. Let A be the adjacency matrix for G and B the adjacency matrix for H. Then the adjacency matrix for **G U H = A + B**, where Boolean addition is used on the entries of matrices A and B. |
| **Transitive Closure of G+** | Includes both Boolean multiplication and addition.G+ = G1 U G2 U G3 U … U GnA+ = A1 U A2 U A3 U … U AnA+ shows every possible walk in G+ up to length n. |

**Order Properties of Binary Relations with Two Sets**

|  |  |  |
| --- | --- | --- |
| **Property** | **Logical Statement** | **Description** |
| **Reflexive** | xRx(x, x) ∈ R∀x ∈ A (xRx)∀x ∈ A ((x, x) ∈ R) | * iA ⊆ R

where iA is the identity relation of set A or iA = {(x, x) | x ∈ A}* **Directed graph**: Loop

Icon  Description automatically generated |
| **Anti-Reflexive** | ¬ (xRx)∀x ∈ A ¬ (xRx) | * **Directed graph**: No loops
 |
| **Symmetric** | xRy ⟶ yRx∀x ∈ A ∀y ∈ A (xRy ⟶ yRx) | * R = R-1
* **Directed graph**: 2-way arrow (edges come in pairs) or no arrows
 |
| **Anti-Symmetric** | (xRy ∧ yRx) ⟶ (x = y)(x ≠ y) ⟶ ¬ (xRy) ∨ ¬ (yRx)∀x ∈ A ∀y ∈ A ((xRy ∧ yRx) ⟶ (x = y)) | * Equivalence
* **Directed graph**: An arrow from x to y implies that there is no arrow from y to x

Icon  Description automatically generated |
| **Asymmetric** | xRy ⟶ ¬ (yRx)∀x ∈ A ∀y ∈ A ∀z ∈ A (xRy ⟶ ¬ (yRx)) | * Fails the vertical line test, so not a proper function, f(x)
* **Directed graph**: 1-way arrow
 |
| **Transitive** | (xRy ∧ yRz) ⟶ xRz∀x ∀y ∀z ((xRy ∧ yRz) ⟶ xRz)∀x ∈ A ∀y ∈ A ∀z ∈ A ((xRy ∧ yRz) ⟶ xRz) | * R ◦ R ⊆ R
* Similar to S ◦ R
* **Directed graph**: Two routes from every vertex A to every vertex B, 1-hop and 2-hops

See the source image |
| **Total** | xRy ∨ yRx∀x ∈ A ∀y ∈ A (xRy ∨ yRx)  | * Either-or
 |
| **Density** | xRy ⟶ ∃z | xRz ∧ zRy∀x ∈ A ∀y (xRy) ⟶ ∃z | xRz ∧ zRy | * A middle-man exists
 |
| **Binary** | R-1 ◦ R = Relation on set AR ◦ R-1 = Relation on set C | * Relation on set <set>
* Binary relation on set <set>
 |
| **Identity** | iA = {(x, y) ∈ A × A | x = y}iA = {(x, x) | x ∈ A} | * Similar to a diagonal matrix
 |
| **Composition****(S ∘ R)** | S ◦ R = (a, c) ∈ S ◦ R ⟷ ∃b | (a, b) ∈ R and (b, c) ∈ S{(a, c) ∈ A × C | ∃b ∈ B ((a, b) ∈ R and (b, c) ∈ S)}aRb and bSc{(a, c) ∈ A × C | ∃b ∈ B (aRb ∧ bSc)} | * The composition of S and R is the relation S ◦ R from A to C
* aRb and bSc, meaning R:a ⟶ R:b ⟶ S:b ⟶ S:c, so (R:a, S:c)
* Ring operator
 |

**Partial Orders**

|  |  |  |
| --- | --- | --- |
| **Term** | **Description** | **Additional** |
| **Partial Order** | A relation R on a set A that is **reflexive**, **transitive**, and **anti**-**symmetric**.A partial order acts like a ≤ operator on the elements of A. | aRb = a ⪯ b"a is at most b" |
| Example | The ≤ operator acting on the set of integers is a partial order, denoted by (**Z**, ≤). The relation is:1. Reflexive (x ≤ x)
2. Anti-symmetric (if x ≤ y and y ≤ x then x = y).
3. Transitive (x ≤ y and y ≤ z implies that x ≤ z).
 |  |
| **Poset** | Partially Ordered SetThe domain along with a partial order defined on it is denoted (A, ⪯). | (A, ⪯) |
| **Comparable** | If x ⪯ y or y ⪯ x. | Example: (**Z**, ≤) |
| **Incomparable** | Not comparable.Neither x ⪯ y nor y ⪯ x.Either they are not connected at all by a path of line segments or the only paths between x and y require a change in direction from up to down or from down to up. | c and f are incomparable. |
| **Total Order** | If every two elements in the domain are comparable. | Example: (**Z**, ≤) |
| **Minimal** | An element x is a **minimal** element if there is no y ≠ x such that y ⪯ x.Vertex x has in-degree = 0. | All edges are leaving the vertex. |
| **Maximal** | An element x is a **maximal** element if there is no y ≠ x such that x ⪯ y.Vertex x has out-degree = 0. | All edges are entering the vertex. |
| **Hasse Diagram** | Ordered from top to bottom to identify if comparable. Incomparable if up then down or vice versa is needed.Incomparable if “air gaped”. |  |

**Strict Orders**

|  |  |  |
| --- | --- | --- |
| **Term** | **Description** | **Additional** |
| **Strict Order** | A relation R on a set A that is **transitive** and **anti**-**reflexive**. Every strict order is anti-symmetric (assumed).A strict order acts like a < operator on the elements of A. | aRb = a ≺ b"a is less than b" |
| Example | The real numbers (**R**) along with the < relation is a strict order. The relation is:1. Transitive (if a < b and b < c, then a < c)
2. Anti-reflexive (there is no real a such that a < a)
 | Examples:(**R**, <)(P(A), ⊂) |
| **Comparable** | Same as a partial order above, except:Partial Order: ⪯Strict Order: ≺ | The arrow diagram for a strict order is basically an arrow diagram for a partial order without the self-loops. |
| **Incomparable** |
| **Total Order** |
| **Minimal** |
| **Maximal** |

**Directed Acyclic Graphs (DAG)**

|  |  |  |
| --- | --- | --- |
| **Term** | **Description** | **Additional** |
| **DAG** | Directed Acyclic Graph (DAG)A directed graph (digraph) that has no directed cycles or positive length cycles.Note that since a single vertex is a cycle of zero length.Acyclic = No Cycles |  |
| Example | Useful for representing precedence relationships or constraints. | College course prerequisites graph or software module dependencies. |
| **Theorem: DAGs and Strict Orders** | Let G be a directed graph. G has no positive length cycles if and only if G+ is a strict order. | If G is a DAG, then G+ is a strict order.If G+ is a strict order, then G is a DAG. |
| **Topological Sort** | If there is an edge (u, v), then u appears earlier than v. | A topological sort for a DAG G is also a topological sort for G+. |
| Example | One way to construct a topological sort for a DAG G is to:1) Pick a vertex x with in-degree 0 and remove x from G. 2) Then pick another vertex with in-degree 0 from among the remaining vertices. 3) Keep selecting vertices until there are no vertices left. |

**Equivalence Relations**

|  |  |  |
| --- | --- | --- |
| **Term** | **Description** | **Additional** |
| **Equivalence Relation** | A relation R is an equivalence relation if R is **reflexive**, **symmetric**, and **transitive**. | aRb = a ~ b"a is equivalent to b" |
| Example | The domain is the set of all people. Define relation B such that xBy if person x and person y have the same birthday. The relation B is:1. Reflexive since every person has the same birthday as himself/herself.
2. Symmetric because if x has the same birthday as y, then y has the same birthday as x.
3. Transitive because if x and y share a birthday and y and z share a birthday, then x and z must also share a birthday.
 |
| **Equivalence Class** | If A is the domain of an equivalence relation and a ∈ A, then [a] is defined to be the set of all x ∈ A such that a ~ x. | The set [a].a ∈ A ⟶ [a] ⊆ A |
| **Theorem: Structure of Equivalence Relations** | Consider an equivalence relation on a set A. Let x, y ∈ A:* If x ~ y then [x] = [y] (identical)
* If it is not the case that x ~ y, then [x] ∩ [y] = ∅ (completely disjoint)
 | The vertices of the network can be partitioned into sets of vertices that can all communicate with each other. |
| **Partition** | Consider an equivalence relation over a set A. The set of all distinct equivalence classes defines a **partition** of A. The term "distinct" means that if there are two equal equivalence classes [a] = [b], the set [a] is only included once. | Equivalence Relation ⟶ Equivalent Class ⟶ Partition ⟶ Set A |
| Example |  |
| **Pairwise Disjoint** | The intersection of any pair of the sets is empty. |  |
| **Strong Connectivity** | Strong connectivity is an equivalence relation on the set of vertices:1. **Reflexive**: Every vertex v is strongly connected to itself.
2. **Symmetric**: If v is strongly connected to w, then w is strongly connected to v.
3. **Transitive**: If v is strongly connected to w and w is strongly connected to x, then v is also strongly connected to x.
 |  |

**Sources**:

* [SNHU MAT 230](https://www.snhu.edu/admission/academic-catalogs/coce-catalog#/courses/4kVhSZLtg) - Discrete Mathematics, zyBooks.
* See also “Harold’s Undirected Graphs and Trees Cheat Sheet”.
* See also pages 9 & 10 of “Harold’s Sets Cheat Sheet” for Relations.