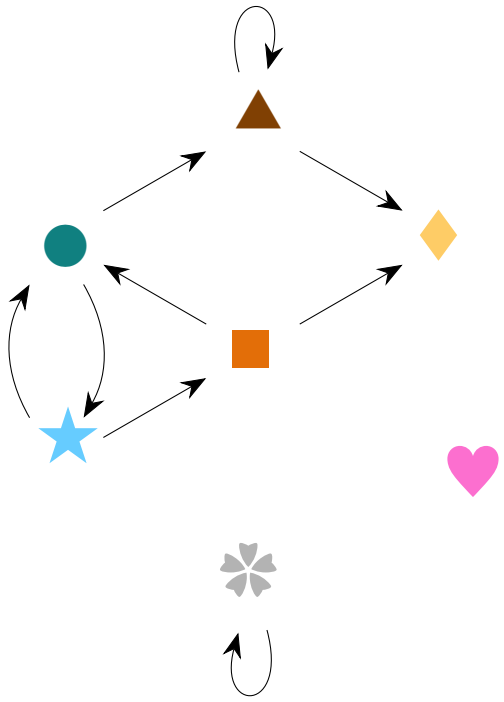
Lecture 14: Digraphs and DAGs

1 Directed analogs of all our defs/theorems

Definition 1. *A **digraph** or **directed graph** $G = (V, E)$ is a set V of **vertices** or **nodes** together with a subset $E \subseteq V \times V$ of **directed edges**.*

Example:



An edge $(u, v) \in E$ (where $u \in V$ and $v \in V$), drawn and sometimes written as $u \to v$, represents a directed edge from u to v.

Only difference from undirected graph def: edges are (u, v) (order matters), instead of $\{u, v\}$.

Some interesting consequences:

- Self-loops are allowed! (v, v)
- Antiparallel edges are allowed! (u, v) and (v, u) are different edges.

Note: still not allowing parallel edges; can't have two arrows from u to v.

Another note: we will never mix undirected and directed edges in the same graph! Make sure you know which kind of graph we're talking about in any given problem.

More examples:

- State machines were secretly digraphs the whole time! States are vertices, and transitions are directed edges.
- Social networks: Facebook friend graph is undirected, but Twitter follower graph is directed.
- Road networks, Google Maps. Some roads are bidirectional: model with two directed edges, one in each direction. (Recall: don't mix undirected with directed!)
- Web graph: which pages link to other pages on the internet. Idea of Page Rank, the core of Google's search engine, is that the graph structure itself can reveal crucial information. Idea: a site gains high score by being linked to from lots of pages with high score. A self-referential system, solved with complicated linear algebra and distributed algorithms, still an active area of research, lots of applications outside of Google. (Named not just for web pages, but for co-founder Larry Page! Like German Chocolate cake named for an American baker, Samuel German. And MySQL, named for the dev's daughter, My. Twitter has some less credible examples.[1])

Lots of analogs of concepts and theorems we've discussed.

1.1 Degree

Undirected: degree $\deg(v)$ is the number of incident edges.

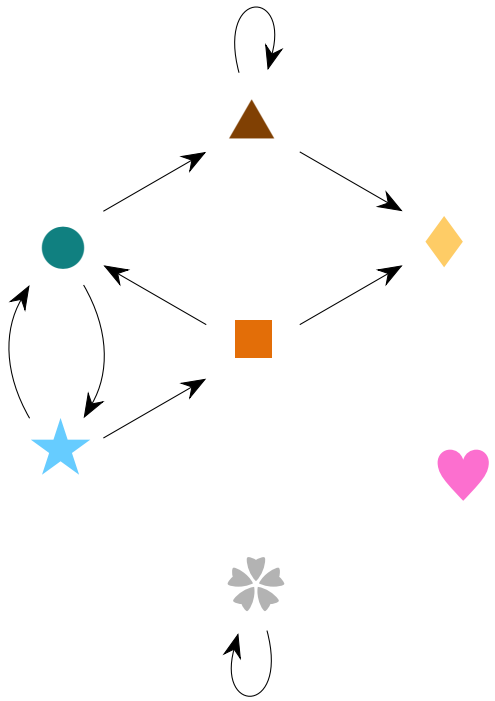
Directed: in-degree $\deg^-(v)$ (or $\deg_{\text{in}}(v)$) is the number of edges (u, v) pointing at v, and out-degree $\deg^+(v)$ (or $\deg_{\text{out}}(v)$) is the number of edges (v, w) coming from v. There is not a combined "degree" for digraphs.

Handshake lemma: $\sum_{v \in V} \deg^-(v) = |E| = \sum_{v \in V} \deg^+(v)$.

1.2 Walks

Directed case: a (directed) walk is a sequence of vertices $v_0, \ldots, v_k$ such that $(v_i, v_{i+1}) \in E(G)$ for each i. Only difference is that order across the edge matters.

[1] https://twitter.com/fred_delicious/status/1265758435175534594?lang=en
https://twitter.com/TheAndrewNadeau/status/851534371949600768



Example: 

Example:  (Walk of length 1)

Example:  (Walk of length 0)

Non-example:  (First edge goes the wrong way)

Trails, paths, closed walks, tours are all defined exactly as in undirected case. Still have:

Theorem 1. *If there exists a directed walk from u to v, then there exists a directed path from u to v.*

Same proof: the shortest walk from u to v must be a path.

1.3 Cycles

A cycle is still a tour that has positive length and does not repeat vertices (except at the very start/end), just as we defined in Lecture 13, Definition 9.

Recall: in the undirected case, we noted (Definition 14) that a cycle must have length at least 3, which is not true in the directed case. This is because self-loops are cycles of length 1! And a pair of antiparallel edges are a cycle of length 2! (You may interpret ! as factorial or excitement.)

1.4 Connectivity

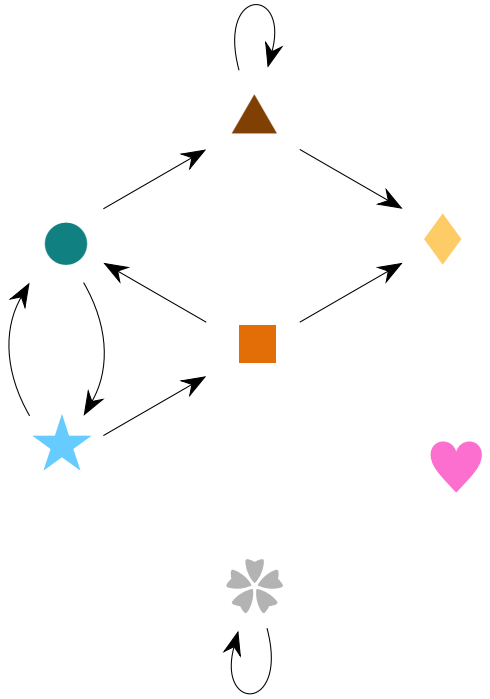
Undirected: u and v are connected iff there exists a walk from u to v.

In directed land, this isn't symmetric! There might be a walk from u to v but not from v to u, so direction matters.

Definition 2. *A vertex u can **reach** v, or v is **reachable** from u, iff there is a u-v walk.*

Definition 3. *Two vertices u and v are **strongly connected** iff there is a directed walk from u to v* and *a directed walk from v to u.*

Definition 4. *A graph G is **strongly connected** iff every pair of vertices of G is strongly connected.*



Example:  can reach both  and , but not  .

Example: Every vertex v can reach itself.

Example:  and  are strongly connected.

Example:  and  are not strongly connected.

Example: Every vertex v is strongly connected with itself.

1.5 Euler Tours

An Euler Tour is still a tour that uses every edge exactly once (and visits all vertices). Can prove a similar theorem:

Theorem 2. *A directed graph has an Euler tour iff it is strongly connected, and every vertex has in-degree equal to its out-degree.*

Proof is exactly as the proof of the corresponding theorem for undirected graphs. We will see an application of this in recitation.

1.6 Condensation Graphs

Definition 5. *The **strongly connected component (SCC)** of v, often denoted $[v]$, is the induced subgraph on the set of vertices that are strongly connected with v.*

Note: we often abuse notation and equate $[v]$ (a graph) with its vertex set.
Like connected components in undirected graphs, SCCs partition the *vertices* of a digraph, but be warned that some *edges* travel between components! This gives an interesting concept that we did not observe in the undirected case:

Definition 6. *The **condensation graph** of a graph $G = (V, E)$ is $H = (C, E')$ where*

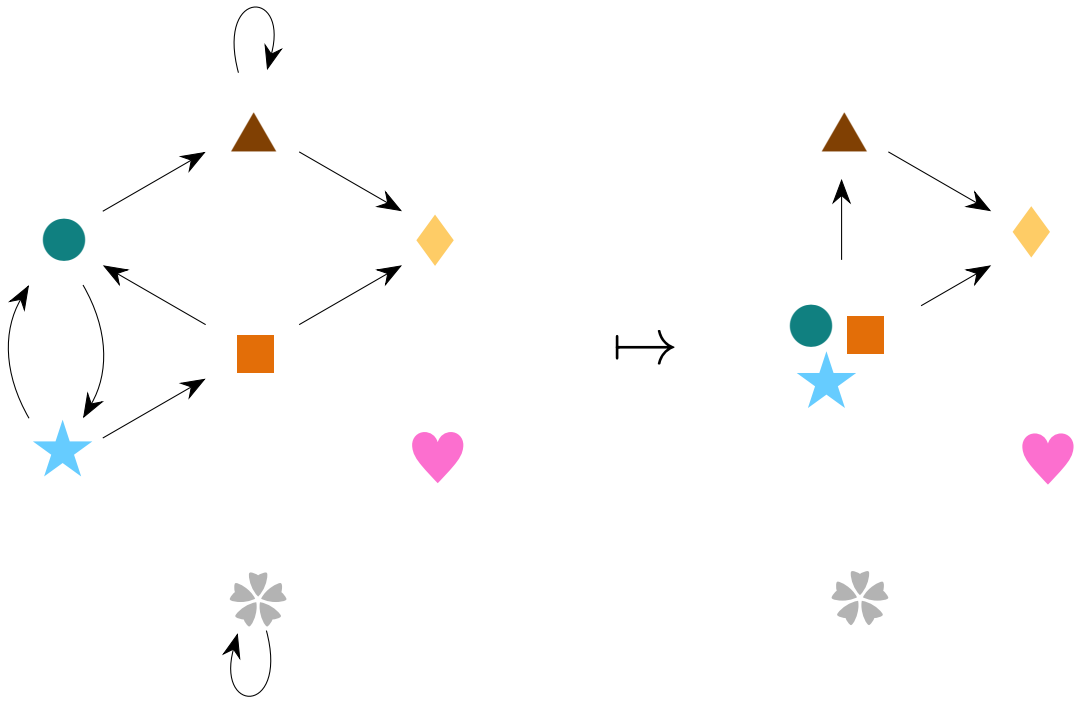
$$\begin{aligned} C &= \{[v] : v \in V\}, \text{ and} \\ E' &= \{([u], [v]) : [u] \neq [v] \text{ and } (u, v) \in E\}. \end{aligned}$$

Essentially, this is the graph produced by collapsing every SCC into a single vertex, and then removing all self-loops.

Example: Left graph has 5 SCCs:

- $[\,\star\,] = [\,\blacksquare\,] = [\,\bullet\,] = \{\star, \blacksquare, \bullet\}$
- $[\,\blacktriangle\,] = \{\blacktriangle\}$
- $[\,\blacklozenge\,] = \{\blacklozenge\}$
- $[\,\heartsuit\,] = \{\heartsuit\}$
- $[\,\text{✿}\,] = \{\text{✿}\}$

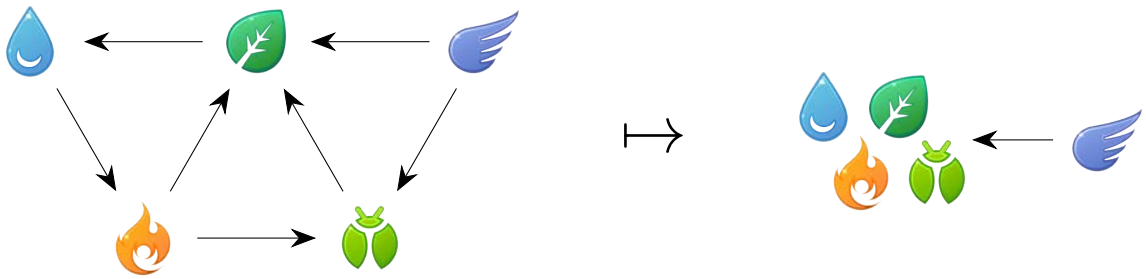
Right graph is condensation of left graph.



Example: Left graph has 2 SCCs:

- $\left[\ \right] = \left[\ \right] = \left[\ \right] = \left[\ \right] = \{\ ,\ ,\ ,\ \}$
- $\left[\ \right] = \{\ \}$

Right graph is condensation of left graph.

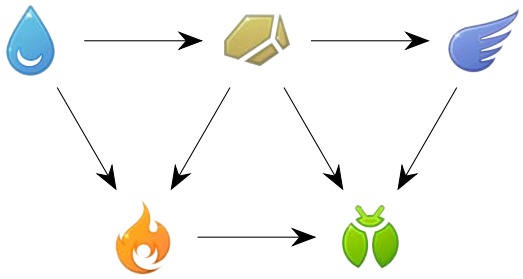


2 DAGs

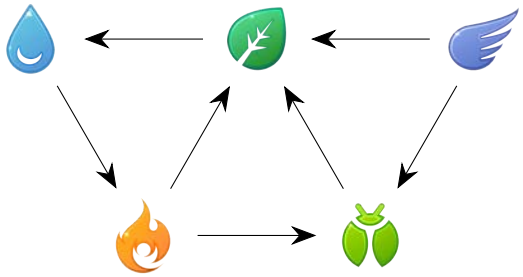
In undirected land, an acyclic graph is a forest. The directed analog is:

Definition 7. *A **Directed Acyclic Graph**, or **DAG**, is a digraph with no cycles.*

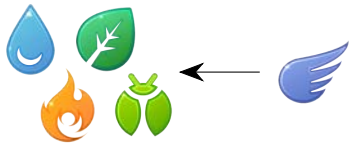
Example:



Non-Example:



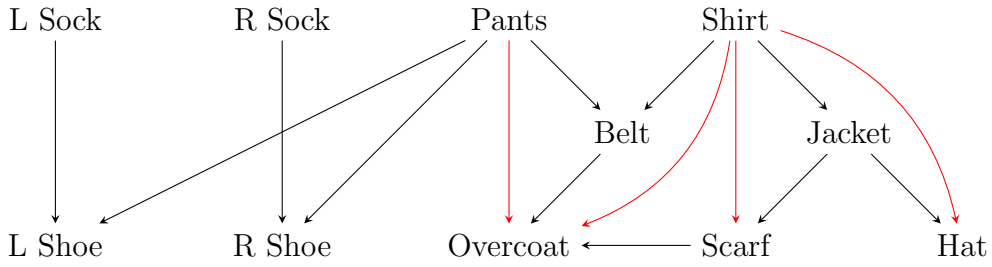
Example:



In fact, every condensation graph is a DAG!

Many uses: scheduling and precedence, concurrency control, data processing, etc.

Example: Getting Dressed! directed edge $u \to v$ when we must put on u before v. E.g., pants before belt, shirt before jacket, etc.



Observe: shirt must precede jacket because of edge. But also, shirt must precede scarf because of longer path "shirt to jacket to scarf". Belt and hat are unrelated, so they can be put on in either order.

In general, u must precede v when v is reachable from u.

Edge from shirt to scarf is not needed, since it is already encoded by the path "shirt to jacket to scarf". Often want to remove those redundant edges. An edge is called a ***covering edge*** iff it is the *only* path between its endpoints. Otherwise, it is redundant. This pared down DAG formed from just the covering edges encodes the same precedence relation, but often with far fewer edges. Sometimes called the ***Hasse Diagram*** for the DAG.

Example: Red edges are redundant above, black edges are covering edges, and black subgraph is the Hasse diagram.

Why must the graph be a DAG in scheduling problems? Which comes first, chicken or egg?

(job vs work experience?)

2.1 Topological Orders

Dress up! In what order should we get dressed?

Can we start with belt? No, need pants first. In general, must start with a *minimal* element. Otherwise: https://www.youtube.com/watch?v=aYYTYCofdXs

Definition 8. *A **source** (resp. **sink**) in a digraph is a vertex with in-degree (resp. out-degree)* 0.

Definition 9. *A source (resp. sink) in a DAG is a **minimal** (resp. **maximal**) element.*

In our example, that's pants, L sock, R sock, and shirt.

After putting on shirt, jacket becomes minimal! Still not belt, though, b/c we haven't put on pants yet.

Does it matter which minimal element we pick! No! There can be many orders. Let's pick a good one...

Definition 10. *A **topological order** (also called **topological sort**) of a DAG is a list of all the vertices such that every vertex appears earlier in the list than* every *other vertex reachable from it.*

Theorem 3. *Every finite DAG has a topological order!*

Proof. Idea: Greedy alg: repeatedly pick and remove a minimal element, just like in the example. Must show

Lemma 4. *Every finite DAG has a minimal element.*

Proof. Similar to proof that leaves exist. Can show that the longest path starts at a minimal element and ends at a maximal element; otherwise, we could either make it longer or would find a cycle. □

Now we can use induction. Let $P(n)$ be the predicate "Every n-vertex graph has a topological order."

- Base case ($n = 0$): The empty sequence is a topological order
- Inductive step: Assume $P(n-1)$. We wish to prove $P(n)$. To this end, let $G = (V, E)$ be a graph with n vertices. Let v be a minimal element of G, and let H be the induced subgraph on $V \setminus \{v\}$. H has $n-1$ vertices, so has a topological order. Prepend v to this topological order to get a topological order for G.

□

2.2 Parallel Task Scheduling

Goal: suppose we have as many helpers as we could want, and can do many tasks in parallel, as long as all of their prereqs are already finished. What's the fewest number of stages needed?

Could consider this getting dressed "à la Downton Abbey", but Wallace and Gromit have a better video clip: https://www.youtube.com/watch?v=EGSyw2dHhrc.

Definition 11. *Say that two vertices u, v are **comparable** if u can reach v or v can reach u.*

*A **chain** is a subset of vertices in which every pair is comparable.*

Vertices in a chain must fall along a single path (but don't need to be all of the path!).

E.g., {Pants, Overcoat, Belt} is a chain. How about {Shirt, Hat}? Yes.
How about {L Sock, R Sock, L Shoe}? No, L sock and R sock are not comparable.

No pair in a chain can be processed at the same time, since one will always be a prereq for the other. Therefore,

Fact 1. *The shortest parallel schedule is $\geq$ the length of the longest chain, no matter how many tasks you can do at the same time!*

Definition 12. *A maximum-size chain is called a* ***critical path****.*

Example: {Shirt, Jacket, Scarf, Overcoat}

Warning: A critical path is a *set* of vertices. Its size is the number of vertices. It is *not* a path, but it does correspond to an obvious path whose length differs from the size of the critical path by 1.

By contrast,

Definition 13. *An* ***antichain*** *is a subset of vertices in which every pair of distinct vertices is* in*comparable.*

Example: {L Sock, R Sock, Pants, Hat, Scarf} is a maximum-size antichain.

We can perform a subset of tasks together iff they form an antichain.

Note: since the largest antichain has size 5, extra helpers after the fifth don't actually have anything to do!

Can now prove a more surprising fact:

Fact 2. *The shortest parallel schedule is* ***equal to*** *the length of the longest chain.*

Proof. Idea: do *all* minimal elements at the same time, then repeat.

Let $depth(v)$ be the length of the longest path that ends at v. Note v is minimal iff $depth(v) = 0$. Claim depths go from $0, 1, \ldots, c-1$, where c is the length of the critical path.

Note: if u can reach $v \neq u$, then $depth(u) < depth(v)$. Take a path to u with length $depth(u)$. Append to this a path from u to v, to find a longer path ending at v. So $depth(v) =$ longest path to $v \geq$ length of this path to $v > depth(u)$.

In particular, this means: all prereqs for v have strictly smaller depths than v. Also, V_i is an antichain.

Now, on step i, we can do all of V_i in parallel, since all prereqs for these tasks have already been completed. This schedule has exactly c steps, as needed. □

In the case of the dress-up graph:

$$\begin{aligned} V_1 &= \{\text{L Sock, R Sock, Pants, Shirt}\} \\ V_2 &= \{\text{L Shoe, R Shoe, Belt, Jacket}\} \\ V_3 &= \{\text{Scarf, Hat}\} \\ V_4 &= \{\text{Overcoat}\} \end{aligned}$$

This shows that if c is the size of the largest chain, then we can partition $V(G)$ into c antichains, and no fewer.

Note: We knew that we need at most five helpers, but our construction gives a parallel schedule that we can implement using only four helpers instead of five! The size of the largest antichain gives an upper bound on the number of helpers we need for any parallel schedule, but we may be able to get away with fewer. Note also that this greedy approach schedules every task as soon as possible, but it doesn't necessarily use the fewest possible number of helpers. In fact, there is a schedule with four steps and only three helpers:

$$\begin{aligned} V_1 &= \{\text{L Sock, Pants, Shirt}\} \\ V_2 &= \{\text{L Shoe, R Sock, Jacket}\} \\ V_3 &= \{\text{R Shoe, Belt, Scarf}\} \\ V_4 &= \{\text{Overcoat, Hat}\} \end{aligned}$$

Can we use fewer helpers and still get a 4-step schedule? No, because we have 11 items of clothing, but 2 helpers can only handle 8 items between them in 4 steps.

2.3 Dilworth's Theorem (fun fact, but not examined)

Theorem 5. *For every threshold $t > 0$, every DAG on n vertices has a chain of size $> t$ or an antichain of size $\geq n/t$.*

Proof. Let G be a DAG on n vertices, and let $t > 0$. Assume for sake of contradiction that the biggest chain (critical path) has size (vertex count) $c \leq t$, and the biggest antichain has size $\ell < n/t$. We know that we can partition the vertex set by depth. This gives a partition into c antichains. Each of these c antichains has fewer than ℓ vertices. In total, they have fewer than $c\ell < t \cdot n/t = n$ vertices. Contradiction! □

Dilworth's Theorem says that there is always a large chain or a large antichain. Often, we use the threshold $t = \sqrt{n}$.

Corollary 6. *Every DAG on n vertices has a chain of size $> \sqrt{n}$ or an antichain of size $\geq \sqrt{n}$.*

6.1200J Mathematics for Computer Science

Spring 2024