Lecture 10: Directed Graphs

CAB203 Discrete Structures

Matthew McKague

Queensland University of Technology

cab203@qut.edu.au

Outline

Readings

This week:

- No readings this week

Next week:

- No readings next week

Outline

Directed graphs

A directed graph is like a graph, except that the edges have a *direction*.

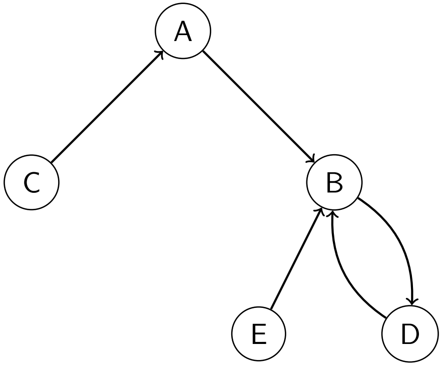
- A directed graph is a irreflexive relation
- $G = (V, E)$ with $E \subseteq V \times V$ and

$$\forall v \in V \ (v, v) \notin E$$

- It is allowed to have both (u, v) and (v, u), but this is now *two* edges!

Visualisation

Directed graphs are visualised like graphs, except that the edges have arrowheads to indicate the direction of the edge. Edge (u, v) will have the arrow pointing from u to v.



Directed graph examples

- Sometimes airlines fly routes between three cities in a cycle, so that the are direct flights one way, but not backwards. We can take this into account with a directed graph.
- Some roads are one-way. We can encode this into a directed graph for a road network.
- We can form a *dependency graph* where the vertices are actions, and (u, v) is an edge if u must be performed before v can be performed. (Example, you must put your socks on before your shoes.)

Outline

DAGs

Just as graphs can have cycles, so do directed graphs.

- A *directed cycle* is a cycle in a directed graph where all the edges point the same way around the cycle.
- That is, a directed cycle is a sequence of vertices $v_1, v_2, \ldots, v_j$ such that (v_1, v_2), $\ldots$, (v_{j-1}, v_j), (v_j, v_1) are all in E.
- A *directed acyclic graph* (DAG) is a directed graph without any directed cycles.

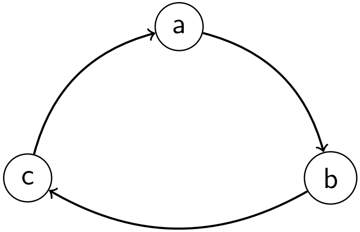
Topological ordering

A *topological ordering* is a total ordering R on the vertices of a directed graph (V, E) such that if $(u, v) \in E$ then uRv.

- Put another way, a topological ordering is a total ordering R on the vertices such that $E \subseteq R$.
- Recall that a total ordering is:
  - Reflexive ($\forall a \in A\ aRa$)
  - Anti-symmetric ($\forall a, b \in A\ (aRb \wedge bRa) \rightarrow a = b$)
  - Transitive ($\forall a, b, c \in A\ (aRb \wedge bRc) \rightarrow aRc$)
  - All elements are comparable ($\forall a, b \in A\ aRb \vee bRa$)

Existence of topological orderings

Not every directed graph allows for topological orderings.



- Since aRb and bRc for any topological ordering, we also must have aRc by transitivity
- We have aRc and cRa, but $a \neq c$, so we break anti-symmetry.

To allow a topological ordering, a directed graph must be *acyclic.* That is, it must not have any directed cycles.

Total orderings and ordered lists

A total ordering R over a finite set S implies a linear order S. That is, we can write down the elements of S in a list like $s_1 s_2 s_3 \dots s_{|S|}$ so that $s_j R S_k$ if and only if $j \leq k$.
This can be done as follows:

1. Set $j = 1$ and $T = \emptyset$
2. Set s_j to the smallest element in $S \setminus T$, i.e. the unique element s such that $\forall t \in S \; s \leq t$
3. Let $T = T \cup s_j$
4. Repeat steps 2-4 until $T = S$

Lists to total orderings

Similarly, any list of elements $s_1, \ldots, s_{|S|}$ can be turned into a total ordering R given by:

$$R = \bigcup_{j=1}^{|S|} \{(s_j, s_k) : k \in \{j, \ldots, |S|\}\}$$

Hence we can think of a topological ordering on a directed graph (V, E) as a list $v_1, \ldots, v_{|V|}$ such that whenever $(v_j, v_k) \in E$ we have $j \leq k$.

Topological orderings in DAGs

It is possible to build a topological ordering on any DAG.
Start with an empty list of vertices.

1. Find all vertices that have no edges coming in
2. Add these vertices to the list any order that you like
3. Remove these vertices and associated edges from the graph
4. Repeat all steps until the graph is empty

When finished, the list will be a topological ordering of the vertices.
This is the idea behind Khan's algorithm for topological ordering finding.

Recursive formulation for finding topological orderings in DAGs

- $G_0 = \emptyset$, $V_0 = V$
- $G_j = \{u \in V_{j-1} : \forall v \in V_{j-1}\ (v, u) \notin E\}$
- $V_j = V_{j-1} \setminus G_j$
- Write down the vertices in a sequence starting with those in G_1, then those in G_2, etc.
- Any order is allowed for vertices within a particular G_j

Outline

Weighted graphs

We can add additional information to graphs in various ways. A common way is to add *weights* to a graph.

- Define a function $w : E \to \mathbb{R}$ (sometimes we use $\mathbb{Z}$ or $\mathbb{N}$ or another co-domain instead.)
- Then for $e \in E$, the *weight* of e is $w(e)$.

Weights are often interpreted as the *cost* or *capacity* of an edge.

Examples of weights

- A graph of roads and intersections, with weight equal to the distance along a road between two intersections
- A graph of network nodes and physical layer connections, with weight equal to the bandwidth along a physical layer
- A graph of airports and direct flights, with weight equal to the cost of a seat on a flight
- A graph of roads and intersections, with weight equal to the capacity (cars per hour) of a road segment
- A graph of towns and possible power lines, with weight equal to the cost of building a line between two towns.

Questions about weighted graphs

- Find the lowest weight path between two vertices (weight of a path is the sum of the weights of the edges along the path)
- Find a minimum weight spanning tree (weight of a tree is the sum of the weights of its edges)
- Find a maximum flow (maximum capacity of a graph between two vertices, more on this later)
- Find a minimum cut (minimum capacity of a bottleneck in a graph, more on this later)

The travelling salesman problem is to find a minimum weight cycle in a graph that visits every vertex in the graph (a Hamiltonian cycle). This problem is famously NP-hard.

Flows

A *flow* on a weighted graph consists of a *source* $s \in V$, a *drain* $d \in D$, and a function $f : E \to \mathbb{R}$ such that

- $\forall (u,s) \in E \; f((u,s)) = 0$
- $\forall (d,u) \in E \; f((d,u)) = 0$
- $\forall e \in E$
$$0 \leq f(e) \leq w(e)$$
- $\forall v \in V \setminus \{s,d\}$
$$\sum_{(u,v)\in E} f((u,v)) = \sum_{(v,u)\in E} f((v,u))$$

Flows

We can restate the definition of a flow by

- The flow has a source and a drain
- There is no flow into the source, and there is no flow out of the drain
- The flow on an edge is non-negative and less than its capacity (weight)
- The flow into any vertex is equal to the flow out of that vertex (except for the source and drain)

We can visualise a flow as water moving through a network of pipes from the source to the drain. The amount of water coming into a pipe joint is always equal to the amount of water going out of the joint. The amount of water moving through a pipe cannot exceed some maximum, which depends on the pipe.

Maximum flows

Define the *value* of a flow by the sum of the flows into the drain:

$$\sum_{(u,d)\in E} f((u,d))$$

This is always equal to the sum of the flows out of the source:

$$\sum_{(s,u)\in E} f((s,u))$$

- The *maximum flow problem* on a weighted graph is to find a flow whose value is at least as high as any other flow on the graph.

Ford-Fulkerson method

The Ford-Fulkerson method is a method for finding maximum flows. The basic idea is to find paths from the source to the drain, and add some flow along the path.

- Given a directed graph G, weights (edge capacities) w and a flow f an *augmenting path* is a sequence of vertices $s = v_1, v_2, \ldots, v_n = d$ (with no vertex repeated) such that
  - For each $j = 1 \ldots n-1$, either $(v_j, v_{j+1}) \in E$ or $(v_{j+1}, v_j) \in E$.
  - If $e = (v_j, v_{j+1}) \in E$ then $a(e) := w(e) - f(e) > 0$.
  - If $e = (v_{j+1}, v_j) \in E$ then $a(e) := f(e) > 0$.
- The *capacity* of an augmenting path is the minimum $a(e)$ among all e in the path

Using augmenting paths

- Given a flow f and an augmenting path with capacity a, we can create a new flow g with a higher value

$$g((u,v)) := \begin{cases} f((u,v)) + a & \text{the path goes forwards along } (u,v) \\ f((u,v)) - a & \text{the path goes backwards along } (u,v) \\ f((u,v)) & \text{otherwise} \end{cases}$$

- If there is no augmenting path then the flow is maximum
- The flow $f(e) = 0$ is always a valid flow, and serves as a starting point

Uses of flows

Some uses:

- Find routing of data through a network to maximise bandwidth
- Determining the total capacity of a road network between two points
- Find capacity of an electrical network to deliver power to a city
- Determine optimal logistics for factories

Directed paths

A *directed path* in a directed graph $G = (V, E)$ is a sequence of distinct vertices $v_1, \ldots, v_j$ such that $(v_k, v_{k+1}) \in E$ for each $k = 1 \ldots j - 1$.

- Directed paths are like regular paths in undirected graphs, but they must respect the direction of the edges

Edge cuts

Given a directed graph G, a source $s \in V$ and a drain $d \in V$, an *edge cut* or *cut* is a set of edges T such that there is no directed path from s to d in the graph $(V, E \setminus T)$.

- Put differently, any directed path from s to d must go along some edge in T
- Cuts define sets of edges whose removal prevents flow from s to d

Minimum cuts

The *capacity* of a cut is the sum of the weights of allow of its edges.

- A *minimum cut* is a cut who's capacity is minimum among all possible cuts
- We sometimes say minimum cut for the capacity of a minimum cut
- A minimum cut is a bottleneck in a network that limits the amount of flow possible

Max-flow min-cut

Theorem (Max-flow min-cut)

For any directed graph $G = (V, E)$ with source $s \in V$, drain $d \in V$, and weights w, the maximum value of a flow on G is equal to the minimum value of a cut of G.

- If we find a flow with value v and a cut with capacity v, then we know that the flow is maximum and the cut is minimum.

Outline

Bipartite graphs

Recall that a *bipartite* graph is a graph $G = (V, E)$ such that V can be partitioned into A and B such that there are no edges within A or B.

- $A \cup B = V$
- $A \cap B = \emptyset$
- $\forall u, v \in A \ (u, v) \notin E$
- $\forall u, v \in B \ (u, v) \notin E$

Matchings

A *matching* on a bipartite graph $G = (V, E)$ is a subset $M \subseteq E$ such that no vertex of V is incident with more than one edge in M.

- A *maximum matching* on a graph is a matching such that no other matching on the graph has more edges.

Finding maximum matchings

We can reduce the problem of finding maximum matchings to finding maximum flows!

- Add additional vertices s and d
- Add edges (s, a) for all $a \in A$
- Add edges (b, d) for all $b \in B$
- Add weight 1 for every edge

A maximum flow will use as many edges as possible, which gives a maximum matching.

Uses of matchings

- Assign processes to processors
- Assign classrooms to units