Cosc 201 Algorithms and Data Structures Lecture 12 (2/4/2025) Graph Paths

> Brendan McCane brendan.mccane@otago.ac.nz And Michael Albert





1

### Paths in graphs

- One of the standard metaphors in graph theory is to think of the edges as physical connections between locations represented by their endpoints.
- Indeed, the Seven Bridges of Königsberg problem resolved in 1736 by Euler "laid the foundations of graph theory".
- A path in a simple graph is a sequence of *distinct* vertices v<sub>0</sub>, v<sub>1</sub>, ..., v<sub>k</sub> such that every consecutive pair of them (i.e., v<sub>0</sub> and v<sub>1</sub>, v<sub>1</sub> and v<sub>2</sub>, ..., v<sub>k-1</sub> and v<sub>k</sub>) are connected by an edge.
- The length of that path is k the number of edges it uses (in particular, paths of length 0 exist and are just vertices!)
- A <u>walk</u> is a sequence of vertices  $v_0, v_1, \ldots, v_k$  such that every consecutive pair of them are connected by an edge. The vertices need not be distinct.

### Connectedness again

- A graph, G, is <u>connected</u> if, given any two vertices v and w of G there is a walk from v to w.
- If we make a union-find instance on the vertices of G and then do a union operation for the endpoints of every edge, then that's equivalent to saying that all the vertices belong to the same group (i.e., there's a single representative) at the end.
- Is it? That should probably be proven.
- More interesting, is it also true that if there's a walk from v to w there's necessarily a path?
- Yes! If there is a walk, then there's a shortest walk, and the shortest walk must be a path.
- Because, if a walk contains some vertex, z, two or more times, then we can cut out the parts between the first and last occurrences of z (replacing them by a single occurrence) and get a shorter walk. So a shortest walk contains no repeated vertices i.e., it's a path.

# Finding shortest paths

That highlights as an interesting problem:

Single-pair shortest path problem

Given a graph G and vertices v and w find a shortest path from v to w (or show that no path exists).

More generally:

Single-source shortest path problem

Given a graph G and a vertex v find shortest paths from v to every vertex that it can reach.

Hint: We already know how to do this!

#### Breadth-first traversal revisited

- The <u>distance</u> between two vertices is the length of the shortest path between them.
- ▶ In a BFT in *G* from *v* we first add *v* to a queue.
- ▶ That's the only vertex at distance 0 from *v*.
- > Then we add all its neighbours definitely the vertices at distance one.
- Then we add their neighbours unless already seen, these are clearly at distance two!
- And that "obviously" persists.
- So we could certainly compute an array of distances but we want *paths*.
- Let's look at the edges xy that are used when y is added to the queue because it's an unseen neighbour of x.

Dijkstra's shortest-path algorithm (unweighted case)

- Start a BFT from the source vertex *v*.
- At the moment when a vertex y is added to the queue it's added because it's a neighbour of some vertex x.
- ▶ Record this information *y*.*parent*  $\leftarrow$  *x*.
- Stop as usual when the queue is empty (or when w is added to the queue if you're only interested in a path from v to w).
- ► To find the shortest path from *v* to (any vertex) *w*:
  - If w.parent is not defined, there isn't one.
  - Otherwise, read backwards w, w.parent, then the parent of that vertex, ..., until you reach v.
- We've added a constant amount of work in the vertex-processing part of a BFT so the complexity is still O(|V| + |E|).

# Dijkstra's shortest-path algorithm (weighted case)

- In most shortest-path problems the edges are weighted and/or directed and the length (or weight) of a path is defined to be the sum of the weights of its edges.
- If the weights are always positive, can we cope?
- ▶ What about the absolutely shortest (non-zero) path from *v*.
- It must consist of a single edge (because the weights are positive).
- ▶ In fact it must be the edge of least weight that starts from *v*.
- In a basic BFT its other endpoint will be one of the ones added in the first round of additions to the queue - but we probably want to look at it next.
- What kind of queue allows removal based on some other criterion?
- We can replace the standard queue with a min-priority queue!

# Weighted Dijkstra correctness

- We know the *weight* of the least-weight paths to some vertices (initially just the base).
- We have a min-priority queue whose keys are edges representing the last edge of a potential path. The source of that edge will always have a known weight.
- The priority associated to the key is the weight of its source plus the weight of the edge itself.
- When we poll the queue, the edge returned will have a target which is either:
  - Already of known weight, in which case we just continue, or
  - Of unknown weight, in which case we can compute and set its weight, assign its parent, and add all the outedges from it to the queue.
- We will look in more detail next time.