CS 341: Algorithms

Lecture 12: Directed graphs

Éric Schost

based on lecture notes by many other CS341 instructors

David R. Cheriton School of Computer Science, University of Waterloo

Fall 2024

Directed graphs

Directed graphs basics

Definition:

- $G = (V, E)$ as in the undirected case, with the difference that edges are **(directed)** pairs (v, w)
  - edges also called **arcs**
  - we allow **loops**, with $v = w$
- walks, paths and cycles as before; here, cycles have at least one edge
- a **directed acyclic graph** (DAG) is a directed graph with no cycle

BFS and DFS for directed graphs

The algorithms work **without any modification**.

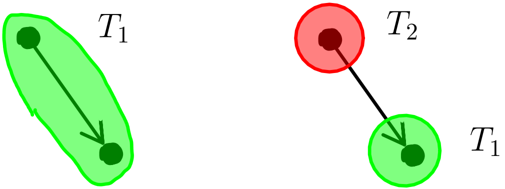
BFS: still get shortest paths

DFS: still have

- a partition of V into **vertex-disjoint trees** $T_1, \ldots, T_k$
- white path lemma (when we start exploring a vertex v, any w with an **unvisited path** $v \rightsquigarrow w$ becomes a descendant of v)
- properties of start and finish times

New for DFS:

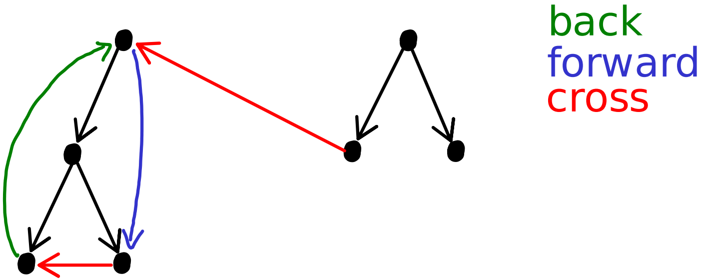
- there can exist edges connecting the trees T_i



Classification of edges

Suppose we have a DFS forest. Edges of G are one of the following:

- **tree edges**
- **back edges:** from descendant to ancestor
- **forward edges:** from ancestor to descendant (but not tree edge)
- **cross edges:** all others



(depends on the order of vertices we chose in the main DFS loop)

Classification of edges

```
explore(v)
    visited[v] = true
    start[v] = t, t++
    for all w neighbour of v do
        if visited[w] = false
            explore(w)                    (v, w) tree edge
    finish[v] = t, t++
```

If w **was visited**:

- if w not finished, (v, w) **back edge**
- else if **start**$[v] <$ **start**$[w] <$ **finish**$[w]$, (v, w) **forward edge**
- else, **start**$[w] <$ **finish**$[w] <$ **start**$[v]$, (v, w) **cross edge**

Testing acyclicity

Claim

G has a cycle if and only if there is a back edge in the DFS forest

Proof

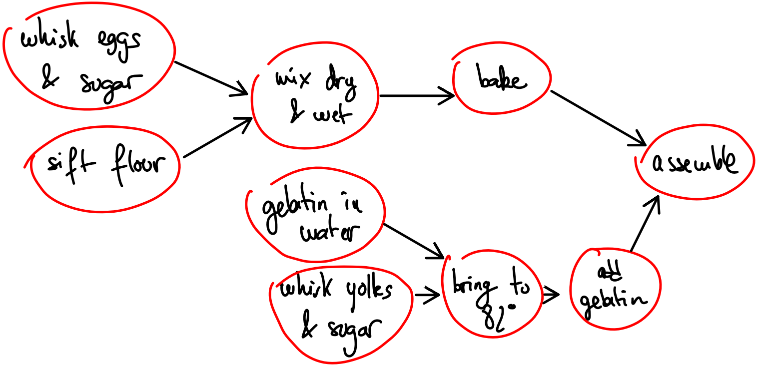
- Suppose there is a back edge (v, w). Then v is a descendant of w, so there is a path $w \rightsquigarrow v$, and a cycle $w \rightsquigarrow v \to w$
- Suppose there is a cycle $\boldsymbol{v_1, \dots, v_k, v_1}$. Up to renumbering, assume we find v_1 first in the DFS.

  Starting from v_1, we will reach v_k (white path lemma). We check the edge (v_k, v_1), and v_1 is not finished. So back edge.

Consequence: acyclicity test in $\boldsymbol{O(n+m)}$

Topological ordering

Definition: Suppose $G = (V, E)$ is a DAG. A **topological order** is an ordering $<$ of V such that for any edge (v, w), we have $v < w$.



Remark: exists a topological order **iff** G is a DAG.

From a DFS forest

Observation:

- start times do not help
- finish times do, but we have to reverse their order

From a DFS forest

Claim

Assume G is a DAG. Suppose that V is ordered using the reverse of the finishing times: $v < w \iff \textsf{finish}[w] < \textsf{finish}[v]$.

This is a topological order.

Proof. Have to prove: for any edge (v, w), $\textsf{finish}[w] < \textsf{finish}[v]$.

- if we discover v **before** w, w will become a descendant of v (white path lemma), and we finish exploring it before we finish v.
- if we discover w **before** v, because there is no path $w \rightsquigarrow v$ (G is a DAG), we will finish w before we start v.

Consequence: topological order in $O(n + m)$.

Testing strong connectivity

Definition. A directed graph G is **strongly connected** if for all v, w in G, there is a path $\boldsymbol{v \rightsquigarrow w}$ (and thus a path $\boldsymbol{w \rightsquigarrow v}$).

Algorithm:

- call **explore twice**, starting from a same vertex s
- edges reversed the second time

Correctness:

- first run tells whether for all v, there is a path $s \rightsquigarrow v$
- second one tells whether for all v, there is a path $s \rightsquigarrow v$ in the reverse graph (which is a path $v \rightsquigarrow s$ in G)

Consequence: test in $\boldsymbol{O(n+m)}$

Structure of directed graphs

Definition: a **strongly connected component** of G is

- a subgraph of G
- which is strongly connected
- but not contained in a larger strongly connected subgraph of G.

Exercise

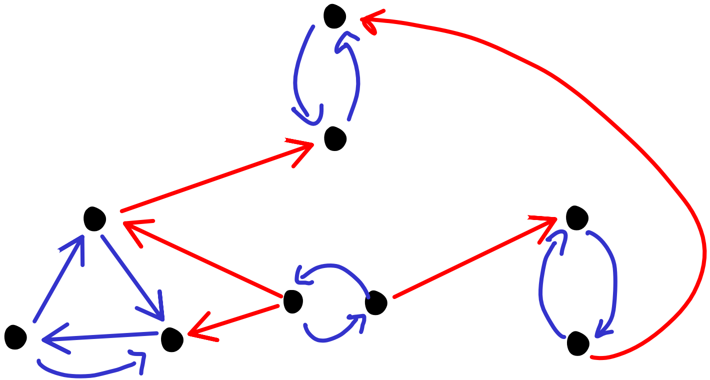
v and w are in the same strongly connected component if and only if there are paths $v \leadsto w$ and $w \leadsto v$.

Exercise

The vertices of strongly connected components form a partition of V.

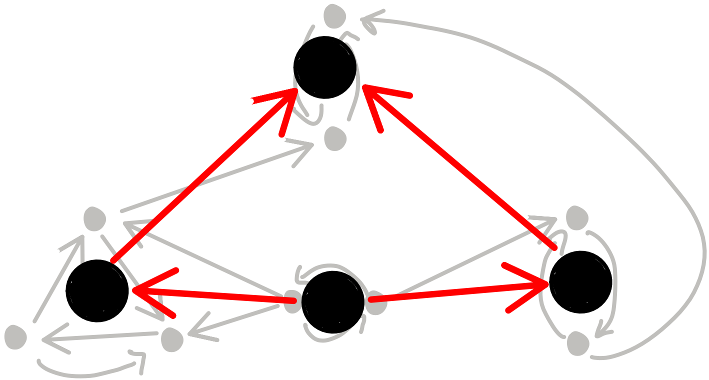
Structure of directed graphs

A directed graph G can be seen as a **DAG** of disjoint **strongly connected components**.



Structure of directed graphs

A directed graph G can be seen as a **DAG** of disjoint **strongly connected components**.



Kosaraju's algorithm for strongly connected components

Definition: for a directed graph $G = (V, E)$, the **reverse** (or **transpose**) graph $G^T = (V, E^T)$ is the graph with same vertices, and reversed edges.

SCC(G)
1. run a DFS on G and record finish times
2. run a DFS on G^T, with vertices ordered in **decreasing finish time**
3. return the trees in the DFS forest of G^T

Complexity: $\boldsymbol{O(n+m)}$ (don't forget the time to reverse G)

Exercise

check that the strongly connected components of G and G^T are the same

The idea behind the algorithm

Claim

If S and T are two strongly connected components of G and there is an edge $S \to T$,
latest finish time in S > latest finish time in T

Proof:

- if we visit a vertex in S first, all vertices in T will be its descendants
- if we visit a vertex in T first, we won't reach S before T is finished.

Consequence:

- start second run from the last-finished vertex s
- in G^T, every vertex reachable from s is in the same strongly connected component
- continue