Data Structures and Algorithms (Adv)

Graphs: DFS on directed graphs

Julian Mestre

School of Computer Science
The University of Sydney

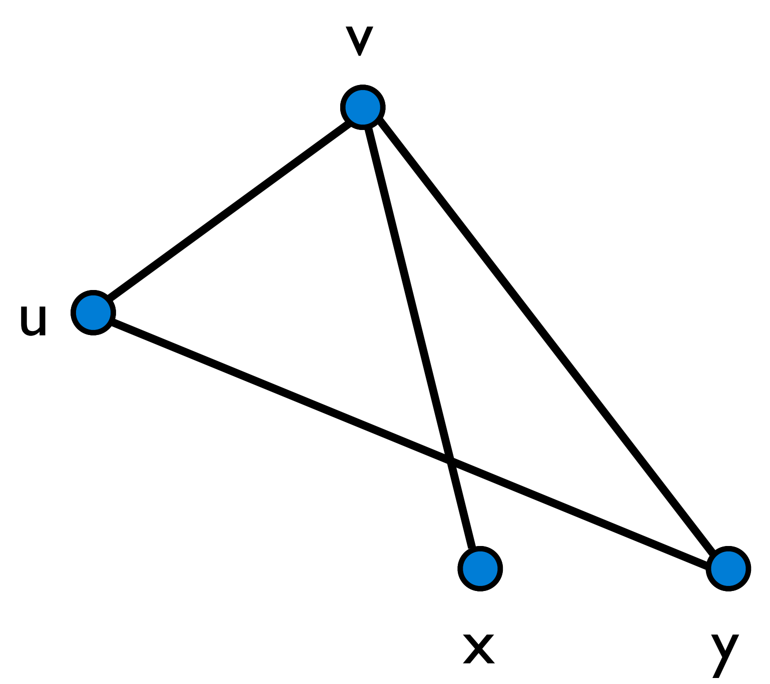
Undirected graphs

Let G=(V,E) be an undirected graph:

- V = set of vertices (a.k.a. nodes)
- E = set of edges

Some notation

- deg(u) = # edges incident to u
- deg(G) = max u deg(u)
- N(u) = neighborhood of u
- n = |V|
- m = |E|



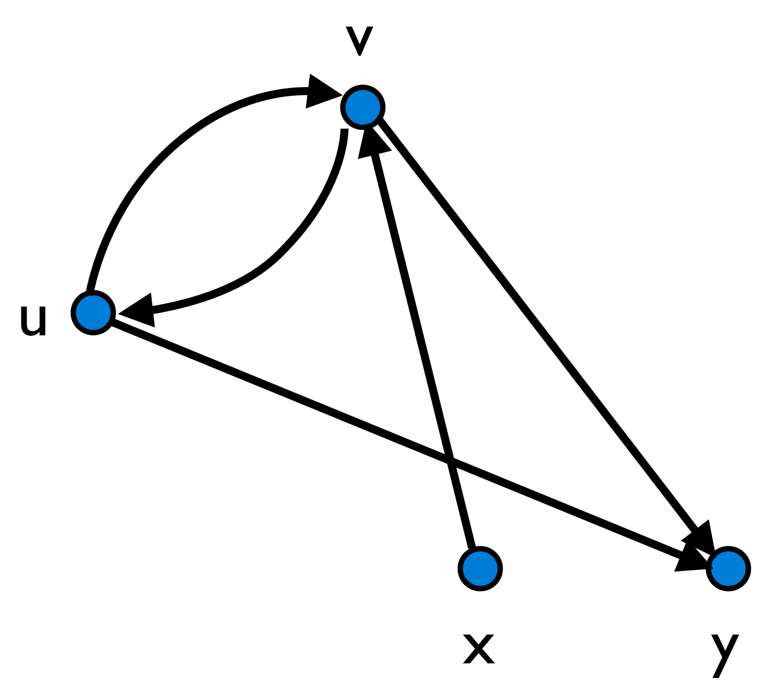
Directed graphs

Let G=(V,E) be a directed graph:

- V = set of vertices (a.k.a. nodes)
- E = set of directed edges (a.k.a. arcs)

Some notation

- $deg^{out}(u)$ = # arcs out of u
- $deg^{in}(u)$ = # arcs into u
- $N^{out}(u)$ = out neighborhood of u
- $N^{in}(u)$ = in neighborhood of u



DFS on directed graphs

```
def DFS(G):
  for u in G
    visited[u] = false
    parent[u] = None
  time = 0
  for u in G:
    if not visited[u]:
      DFS_visit(u)
  return parent
```

```
def DFS_visit(u):
  visited[u] = true
  time = time + 1
  discovery[u] = time
  for v in G[u]:
    if not visited[v]:
      parent[v] = u
      DFS_visit(v)
  time = time + 1
  finish[u] = time
```

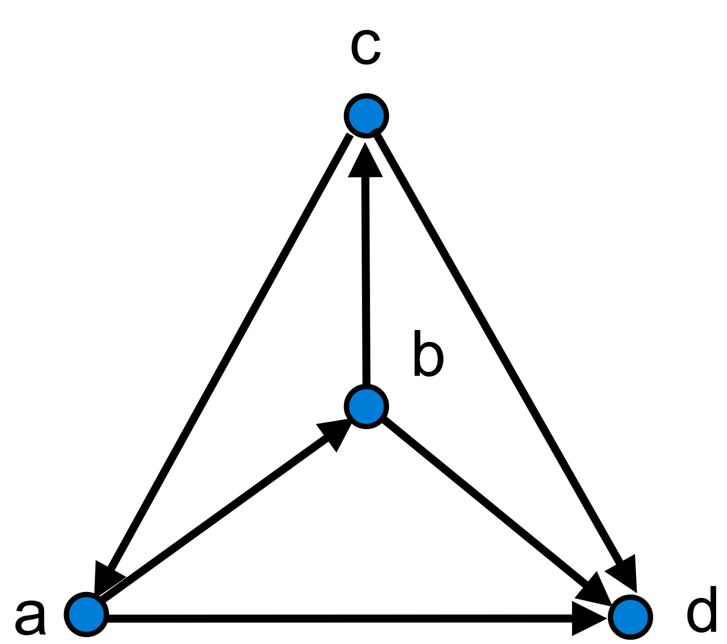
Visit the out neighborhood of u

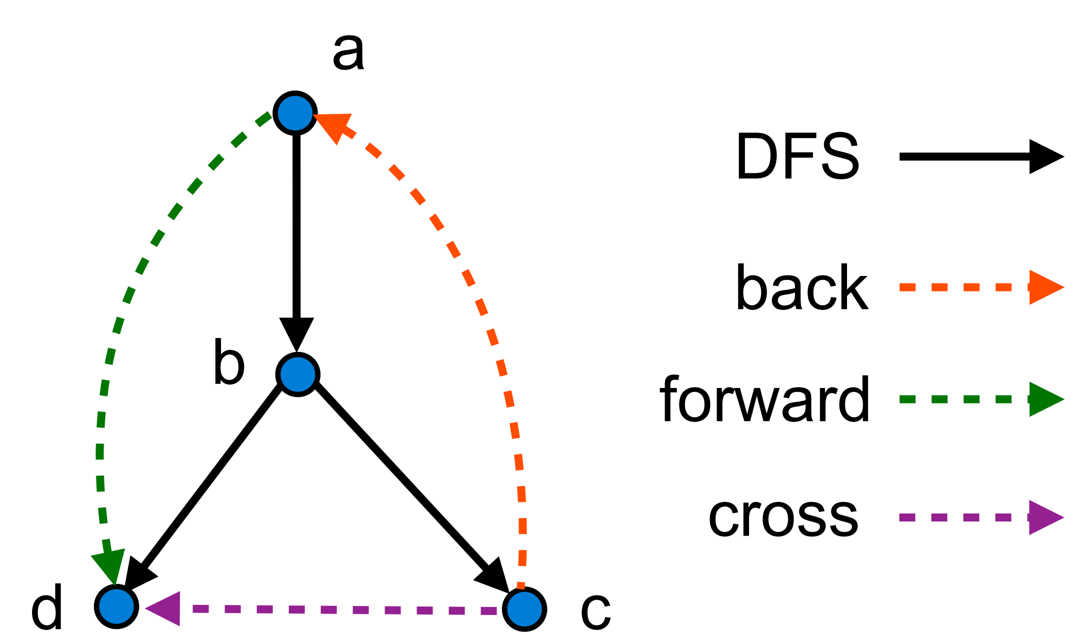
DFS on directed graphs: Properties

Some things don’t change, e.g., running time. But some of the properties of DFS are slightly different, e.g., edge types.

Non-tree edges can be

- back edge
- forward edge
- cross edge

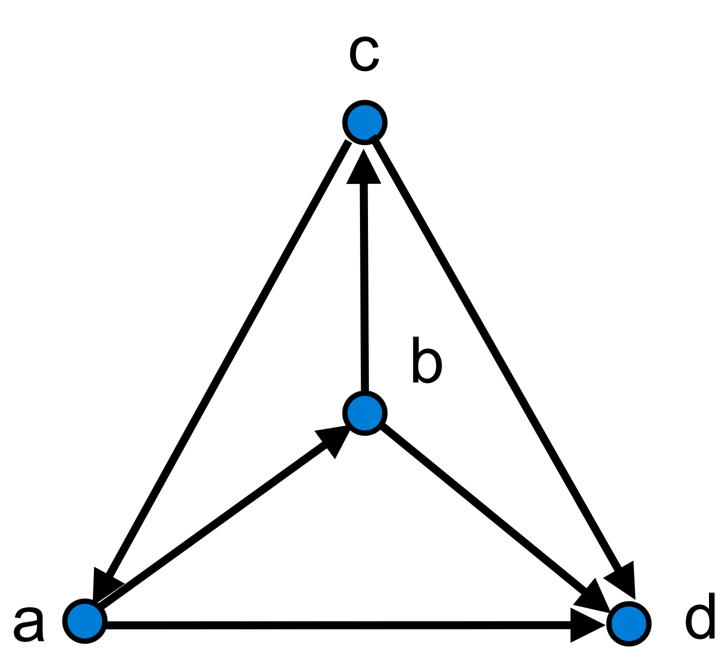




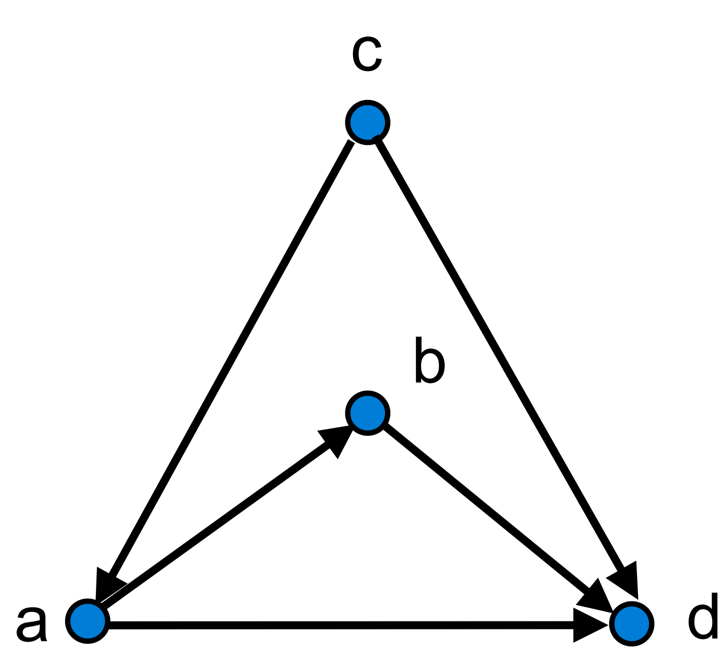
Directed acyclic graphs (DAG)

Def.: A directed graph is acyclic if it does not have any cycles.

Every DAG can be topologically sorted: Vertices can be laid out from left to right in such all edges go left to right as well.



not a DAG



DAG

Topological sort

Time complexity:

- DFS takes in O(n+m) time
- Back edge check takes O(m) time
- Sorting takes O(n) time

Correctness:

- In every cycle there must be at least one back edge
- If there is an edge from right to left, then it must be back edge

```
def topo_sort(G):

  d,f,parent = DFS(G)
  for edge (u,v) in G:
    if “(u,v) is a back edge”:
       return None
  order = [ u : for u in G ]
  “sort order in decreasing f-value”
  return order
```

Thm. There is an O(m+n) time algorithm to topologically sort vertices of a DAG

Strongly connected components

<u>Def.</u>: Let $G=(V, E)$ be a directed graph. A strongly connected component (SCC) of G is a subset C of vertices such that

- For any $u,v \in C$, there is a u-v path in G
- No superset of C has the above property

<u>Def.</u>: The SCC graph of G is $G^{SSC} = (V^{SCC}, E^{SCC})$ where

- $V^{SCC} = \{ C : C$ is a SCC of $G \}$
- $E^{SCC} = \{ (C, C') :$ there is $u \in C$ and $v \in C'$ such that $(u,v) \in E \}$

The SCC problem

Given a directed graph G, compute G^{SCC}. Notice that we only need to compute V^{SCC}

What is the trivial algorithm for this problem?

- Find one SCC
- Remove
- Iterate

The running time of the trivial algorithm is $O(n\ (m+n))$, where $O(m+n)$ is the time it takes to compute a single SSC

DFS based algorithm for SCC

Time complexity:

- DFS takes in O(n+m) time
- Building F takes O(n+m) time
- "Reading" components from DFS forest takes O(n) time

Correctness:

- Not at all obvious!

```
def SCC(G):

  d,f,parent = DFS(G)
  F = copy of G with reversed edges
  run DFS on F but process vertices
      in decreasing order of f-value
  components = []
  for tree T in second DFS forest:
     components.append(vertices in T)

  return components
```

Thm. There is an O(m+n) time algorithm to find all SCCs of an input directed graph