Elementary Graph Algorithms

Recap on Graphs

- $G = (V, E)$ vertices graphs
- Directed, undirected, weighted, unweighted, path (simple), cycle
- Adjacency list / matrix

  i.e: adj[u] nodes adjacent to u

Breath-first search (BFS)

Input: $G = (V, E)$, directed or undirected

source vertex S

Output: $d[u]$ = distance from S to u $\forall u \in V$

$\pi[u]$: u's predecessor on path $S \rightsquigarrow u$

Idea: send a "wave" out from S

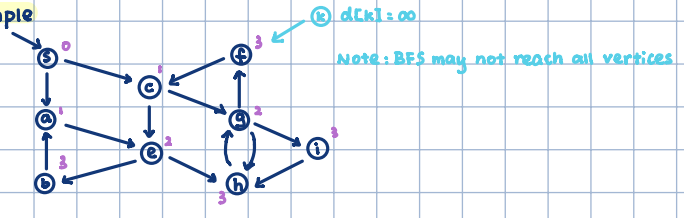
- 1st hit all vertices 1 edge away
- 2nd hit all vertices 2 edges away
- ⋮ etc.

Note: FIFO Q to maintain "wavefront"

- $V \in Q$ iff wave has hit V but has not come out of V yet

```
BFS(V,E,S)
    for each u ∈ V-{s}
        do d[u] ← ∞
    d[s] ← 0
    Q ← ∅ (empty)
    ENQUEUE(Q,s)
    while Q ≠ ∅
        do u ← DEQUEUE(Q)
            for each v ∈ adj[u]
                do if d[v] = ∞    // haven't discovered before
                   then d[v] ← d[u] + 1
                        ENQUEUE(Q,v)
```

Example



iter 0: s

iter 1: a, c

iter 2: e, g

iter 3: b, h, i, f

iter 4: done

Time: $O(V+E)$

Depth-first Search (DFS)

Input: $G = (V, E)$, directed or undirected

no source vertex!

Output: $d[v]$ = discovery time

$f[v]$ = finish time

$\pi[v]$ as well

Idea: unlike BFS, in DFS the moment we discover vertex we immediately explore from it.

Each vertex has a color.

white: undiscovered

grey: discovered but not finished

black: finished (found everything reachable from it)

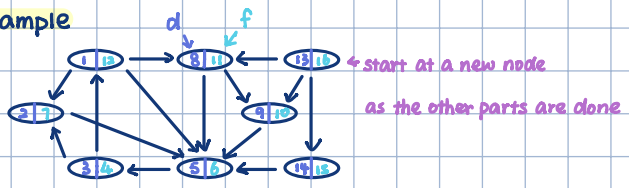
DFS(V, E)

```
for each u ∈ V
    do color[u] ← white
time ← 0
for each u ∈ V
    do if color[u] = white
        then DFS-VISIT(u)
```

DFS-VISIT(u)

```
color[u] ← Gray
time ← time + 1
d[u] ← time
for each v ∈ Adj[u]
    do if color[v] = White
        then DFS-VISIT(v)
color[u] ← Black
time ← time + 1
f[u] ← time
```

Example



← start at a new node as the other parts are done

Time: $\theta(V+E)$

Definitions & Theorems

- **White-path theorem**

  v is a descendant of u iff at time d[u], ∃ a path u⇝v consisting of only white vertices.

- **Parenthesis theorem**

  ∀u,v exactly one of the following is true:

  1. d[u] < f[u] < d[v] < f[v] or
     d[v] < f[v] < d[u] < f[u] and neither of u and v are descendant of the other
  2. d[u] < d[v] < f[v] < f[u] and v is descendant of u.
  3. d[v] < d[u] < f[u] < f[v] and u is descendant of v.

  ★: d[u] < d[v] < f[u] < f[v] cannot happen!

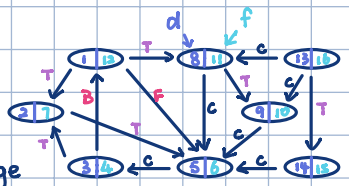
- **Classification of Edges**

  Tree Edge: in DFS forest, found by exploring (u,v)

  Back Edge: (u,v) where u is descendant of v

  Forward Edge: (u,v) where v is descendant of u but not Tree Edge

  Cross Edge: any other edge



- **Theorem (no proof)**

  In DFS of undirected graph, we got only Tree and Back edges

Topological Sort

Partial order ⇒ Total Order

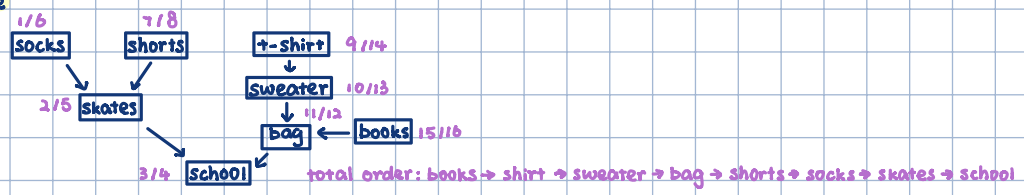
(DAG) — either a ≥ b or b ≥ a ∀ a, b

- a ≥ b and b ≥ c ⇒ a ≥ c
- may have a, b s.t. neither a ≥ b or b ≥ c

[a] → [b] : a precede b (come before sth.)

"a must happen before b"

Example



total order: books → shirt → sweater → bag → shorts → socks → skates → school

Topological Sort (V,E)

call DFS(V,E)

outputs vertices in decreasing order of finishing times

Time: $\Theta(V+E)$

Lemma: A directed graph G is acyclic iff DFS yields no back edges

Proof: 1. cycle ← if back edge (easy to proof)

2. cycle → back edge

proof: suppose G contains cycle C. let V is the first vertex in C that is discovered.

Let (u,v) is preceding edge in cycle C.

At time d[v], vertices in C from a white path $v \rightsquigarrow u$

By white-path theorem, u is descendant of v. ⇒ (u,v) is a back edge

Correctness proof: need to show if $(u,v) \in E$ then $f[v] < f[u]$

proof: When exploring (u,v) what are colors of u and v?

- u is gray
- Is v also gray? No! Otherwise, v would be an ancestor of u → (u,v) is a back edge → contradiction.
- Is v white? Yes, possible. v becomes descendant of u. → by parenthesis theorem: $d[u] < d[v] < f[v] < f[u]$
- Is v black? Yes, possible. v is already finished! Since we are currently exploring (u,v) → u is not finished → $f[v] < f[u]$

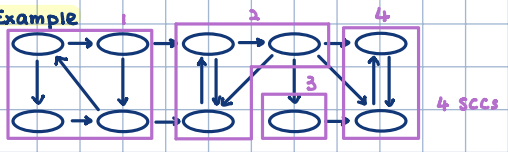
Strongly connected component (SCC)

Input: directed $G=(V,E)$

Output: all SCCs of G

SCC: maximal set of vertices $C \subseteq V$ s.t. $\forall u,v \in C$ both $u \rightsquigarrow v$ and $v \rightsquigarrow u$ exist.

Example



Algorithm uses G^T = transpose of G

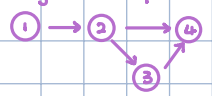
- $G^T = (V, E^T)$, $E^T = \{(u,v), (v,u) \in E\}$
- G^T is G with edges reversed

observe: G^T and G have same SCCs

Component graph

- $G^{scc} = (V^{scc}, E^{scc})$
- V^{scc} has one vertex per SCC in G
- E^{scc} has edge if ∃ edge between SCCs in G

using the example above:

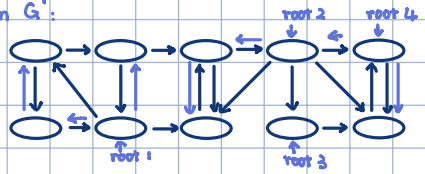


SCC (G)

1. call DFS(G) → finish times f[u]
2. compute G^T
3. call DFS (G^T) in main loop of DFS take nodes in decreasing order of f[u]'s (as computed in first DFS)
4. output vertices in each tree of DFS forest created at step 3 as separate SCC

Time: $\Theta(V+E)$

In G^T:



output: