Graph Algorithms I

Breadth-First Search (BFS) /
Depth-First Search (DFS)/
Topological Sort /
Strongly Connected Components (SCC)

Terminology

- Edge (u,v) incident to vertices u and v
- Vertex u adjacent to vertex v if there's an edge linking them
- Degree of vertex
- Path, Simple path, Cycle, Simple cycle

Adjacency matrix / Adjacency list

operation	adjacency matrix	adjacency list
Space	$\|V\|^2$	2 $\|E\|$
adjacency check	1	max. degree
list of adjacent vertices	$\|V\|$	max. degree
add edge	2	2
delete edge	2	2 max. degree

Terminology (cont'd)

- A undirected graph is connected if there's a path from every vertex to every other vertex
- A component of a graph is a maximal connected subset of the vertices
- A directed graph is strongly connected if there's a path from every vertex to every other vertex
- A strongly connected component of a directed graph is a vertex u and the collection of all vertices v s.t. there's a path from u to v and a path from v to u

Breadth-First Search

- Input: G=(V,E) (e.g. Adjacency Matrix),

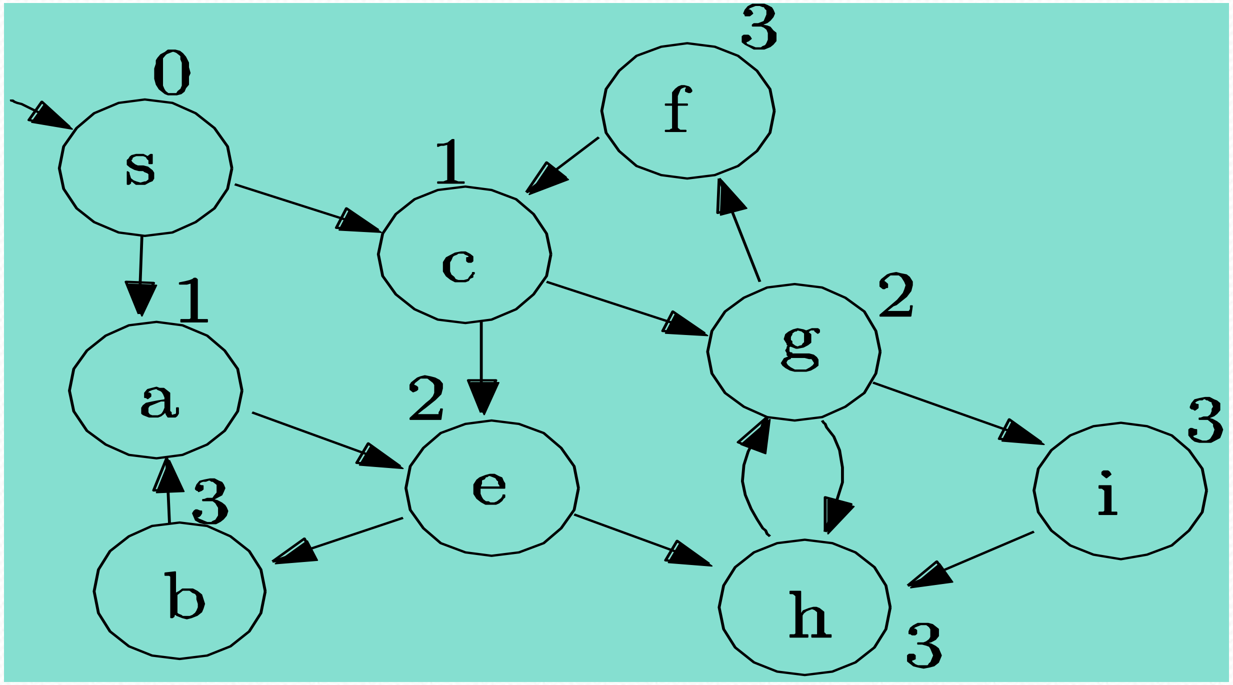
  Source vertex s in V
- Output: for all v in V,

  d[v]=shortest distance from s to v

  prev[v]=predecessor of v

  (-> Breadth-first tree)
- O(V+E):
  - O(V): every vertex v in V enqueued at most once
  - O(E): scan adjacency list of vertex v

    only when v is dequeued



Note that BFS may not visit all vertices of G

Breadth-First Search

```
BFS(G,s){
 for each vertex u in
      V-{s}
   color[u]= WHITE;
   d[u]= \infty;
   pred[u] = NIL;

 color[s] = GRAY;
 d[s] = 0;
 pred[s] = NIL;
 enqueue(Q,s);

 while(Q not empty){
  u = dequeue(Q);
  for each v adjacent to u
      if(color[v] == WHITE){
         color[v] = GRAY;
         d[v] = d[u]+1;
         pred[v] = u;
         enqueue(Q,v);
      }
  color[u] = BLACK;
 }
}
```

Depth-First Search

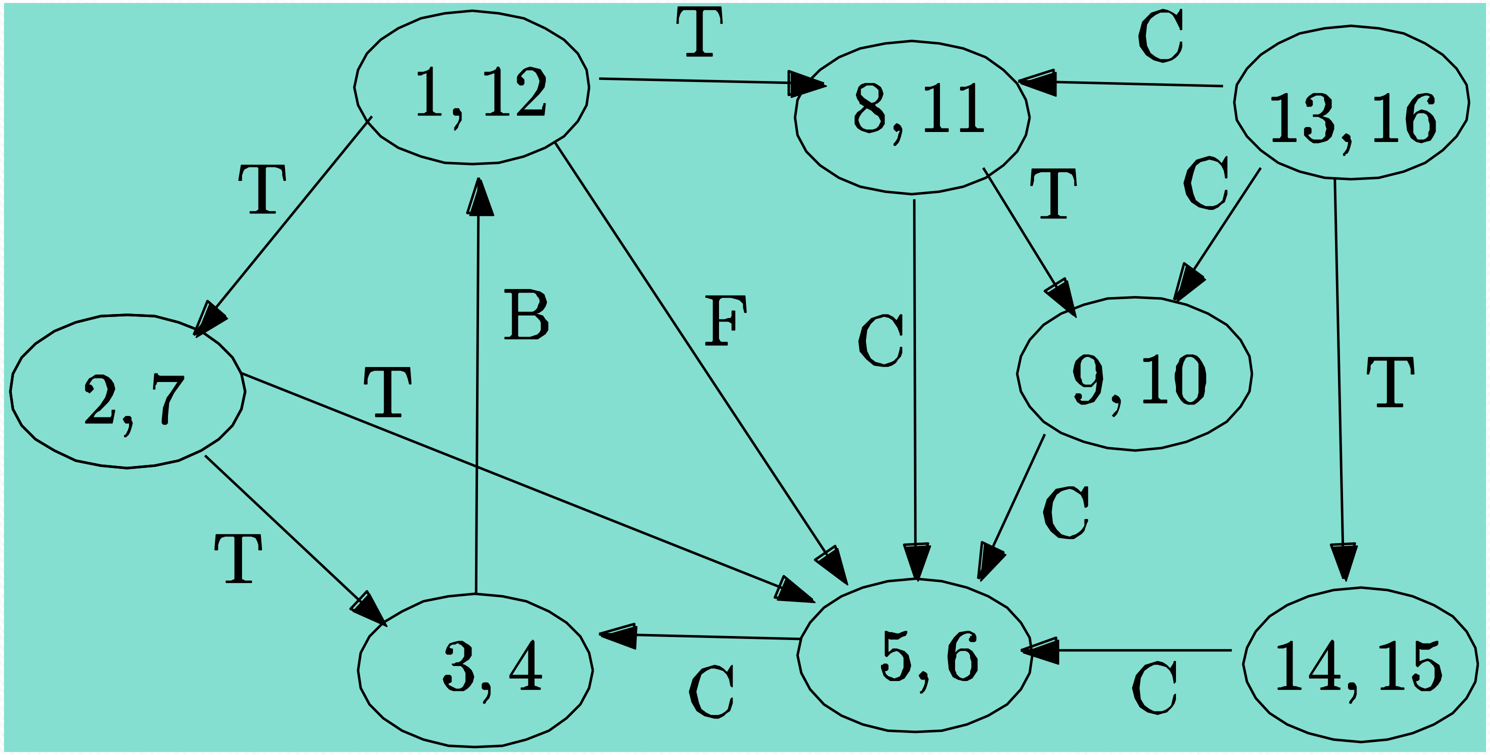
- Input: G=(V,E) (e.g. Adjacency Matrix)
- Output: for all v in V,

  pred[v](->Depth-First Forest)

  & 2 timestamps

  d[v]=discovery time

  f[v]= finishing time
- d[v], f[v]
  - Unique integers from 1 to 2|V|
  - d[v] < f[v]


1, 12
8, 11
13, 16
2, 7
9, 10
3, 4
5, 6
14, 15
T
C
T
T
C
B
F
C
T
T
C
T
C
C

Depth-First Search

```
DFS(G){

for each vertex u in V
  color[u] = WHITE;
  pred[u]=NIL;
time = 0;// global variable
for each vertex u in V
  if(color[u] == WHITE)
     DFS_Visit(u)


}
```

```
DFS_Visit(u){
 color[u] = GRAY;
 time = time + 1;
 d[u] = time;
 for each v adjacent to u
   if (color[v] == WHITE){
        pred[v] = u;
        DFS_Visit(v);
    }
 color[u] = BLACK;
 time = time + 1;
 f[u] = time;
}
```

Analysis

- Time: $\Theta(V+E)$
  - Exhaustive search of vertices and edges
  - Remember that BFS = $O(V+E)$
- Timestamps & colors
  - v is a descendant of u:
    - $d[u]<d[v]<f[v]<f[u]$
    - At time d[u], there is a path from u to v of only white vertices
  - Neither of u and v is a descendant of the other:
    - $d[u]<f[u]<d[]<f[v]$ or $d[v]<f[v]<d[u]<f[u]$

Classify edges in directed graph

- Tree edges (u,v): v white and d[u]<d[v]<f[v]<f[u]
  - (prev[v],v) where DFS calls are made;
  - v is first discovered by exploring this edge
- Back edges: v gray
  - (u,v) where v is an ancestor of u in the search
- Forward edges (none in undirected graph)
  - nontree edge (u,v) connecting a vertex u to a descendant v
  - v black and d[u]<d[v]<f[v]<f[u]
- Cross edges (none in undirected graph)
  - All other edges: v black and f[v]<d[u]

Topological Sort

- Directed acyclic graph (or Dag):

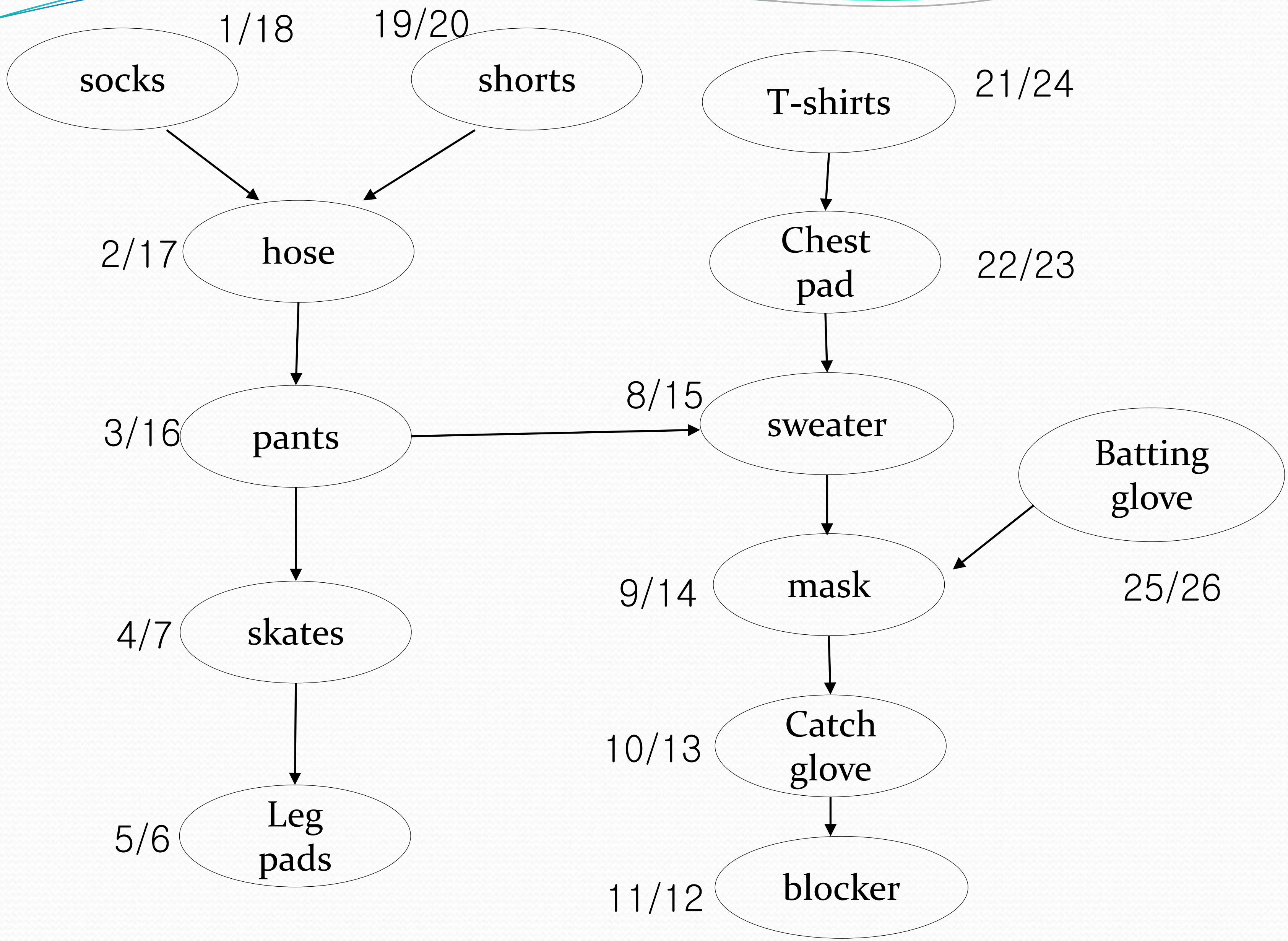
  directed graph with no cycles
  - Good for modeling processes and structures that have a partial order
  - Partial order
    - if a>b and b>c, then a>c
    - May have a and b s.t. neither a>b nor a<b
- Given a directed acyclic graph, is it possible to have a total order keeping this partial order?

Topological sort of a Dag G

- Problem: Find a linear ordering of vertices such that if (u,v) is an edge of G, then

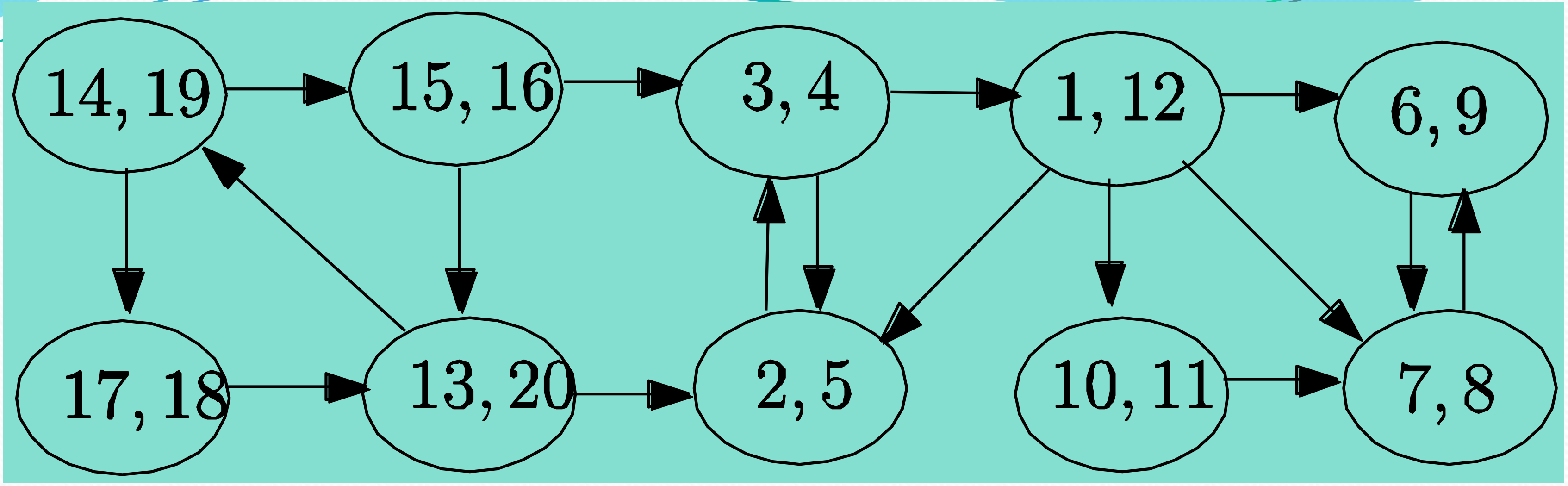
u must appear somewhere before v

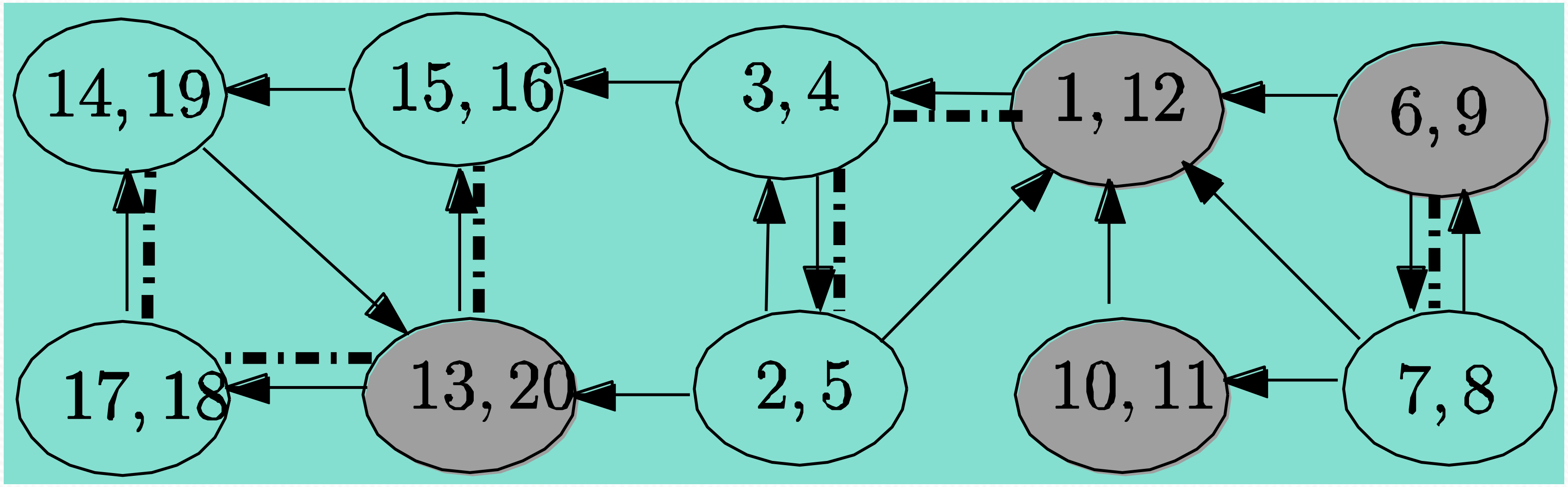
- `Topological-Sort(G,E)`
  - **Call DFS(V,E) to compute finishing time f[v] for all v in V**
  - **Output vertices in order of decreasing finishing time (method: as each vertex is finished, insert it onto the front of a linked list)**


socks
1/18
shorts
19/20
T-shirts
21/24
hose
2/17
Chest pad
22/23
pants
3/16
sweater
8/15
Batting glove
25/26
skates
4/7
mask
9/14
Catch glove
10/13
Leg pads
5/6
blocker
11/12

Finding Strongly Connected Components (SCC)

- **Strongly-Connected-Components(G)**
  - call DFS(G) to compute finishing times f[u] for each vertex
  - compute GT, inversing all edges in G
  - call DFS(GT), but in the main loop of DFS, consider the vertices in order of decreasing f[u] computed in step 1
  - output the vertices of each tree in the depth-first forest of step 3 as a separate strongly connected component


14, 19
15, 16
3, 4
1, 12
6, 9
17, 18
13, 20
2, 5
10, 11
7, 8


14, 19
15, 16
3, 4
1, 12
6, 9
17, 18
13, 20
2, 5
10, 11
7, 8

Correctness Proof

- Component Graph $G^{scc}=(V^{scc},E^{scc})$
  - V^{scc} contains exactly one vertex for each SCC
  - (u,v) in E^{scc} if G contains a directed edge (x,y) for some x, y in different SCCs
- G^{scc} is also DAG
- Let $f[C]=\max\{f[v]: v \text{ in } C\}$.
- For u in C and v in C',
  - If (u,v) in E, then (u,v) not in ET and $f[C] > f[C']$.