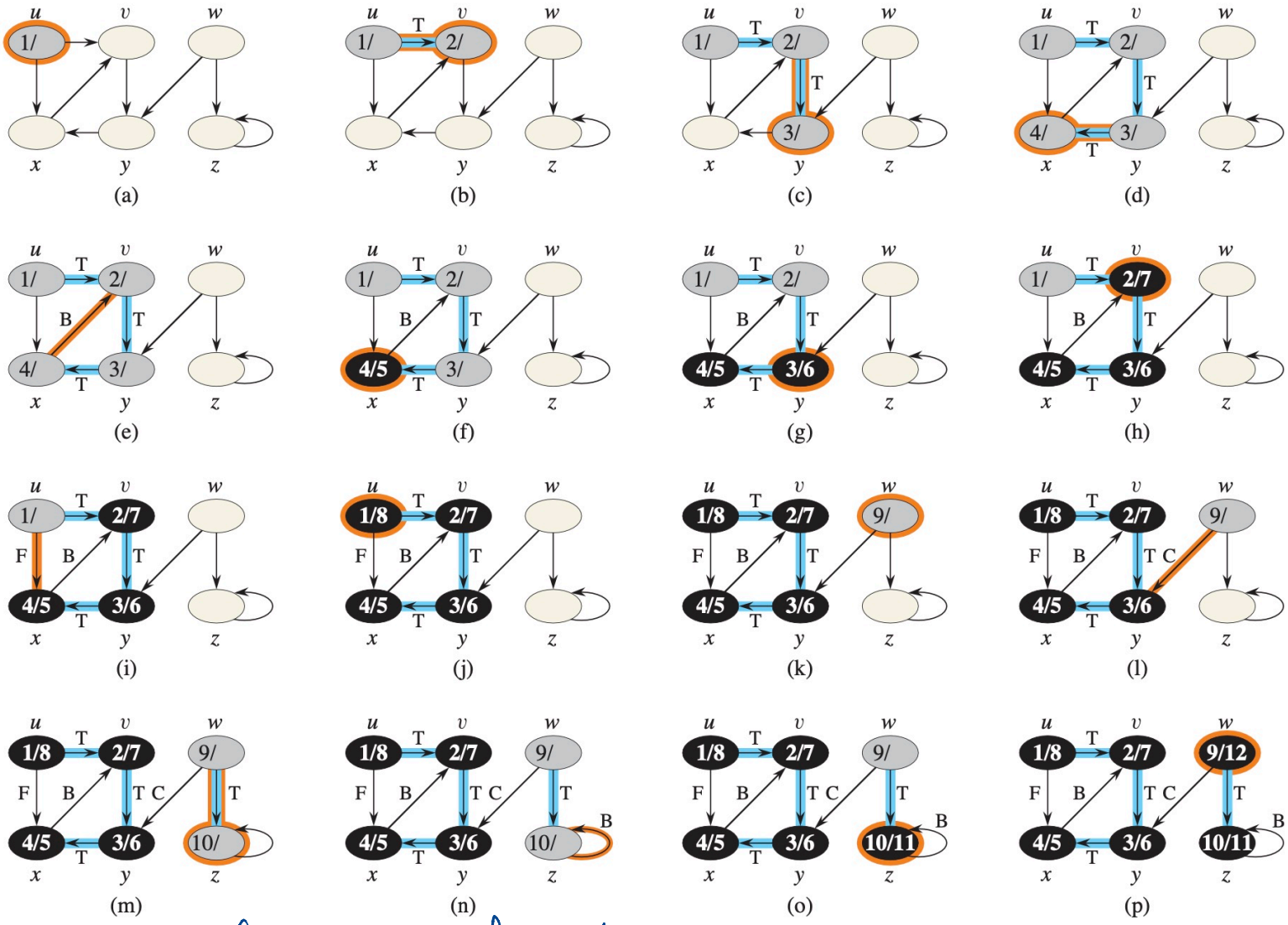
Graphs - III

Depth-First Search

```
DFS (G)
for each vertex u ∈ G.V
    u.color = WHITE
    u.π     = NIL
time = 0  // Global variable
for each vertex u ∈ G.V
    if u.color == WHITE
        DFS-VISIT (G,u)
```

```
DFS - VISIT (G,u)
time = time + 1
u.d = time  // vertex u is discovered
u.color = GRAY
for each vertex v in G.Adj [u]
    if v.color == WHITE
        v.π = u
        DFS-VISIT (G,v)
time = time + 1
u.f = time
u.color = BLACK
```

Example:



- Classification of edges: Depth-first search classifies edges of G into 4 categories:

→ The DFS forest

1. Tree edges: edges in G_π. An edge (u,v) is a tree-edge if v was discovered by exploring edge (u,v).

2. Back edges: edges (u,v) that connects a vertex u to ancestor v in a DF tree.

3. Forward edges: Those non-tree edges (u,v) connecting vertex u to a proper descendent v in a DF tree

4. Cross edges: All other edges. Can go between vertices of same DF tree (e.g. siblings) or b/w different trees.

- In a depth-first search of an undirected graph G every edge is either a tree-edge or a back edge.

- Cycle detection: A directed graph is acyclic iff a depth-first search yields no "back" edges.

- Topological sort:

Given a Directed Acyclic Graph (DAG)

→ A digraph with no cycles

$G = (V, E)$, a topological sort is a linear ordering of all its vertices s.t. if G contains an edge (u, v) then u appears before v in the ordering.

→ Think of topological sort of a DAG as an ordering of its vertices in a horizontal line so that all edges go left to right

Topological-Sort (G)

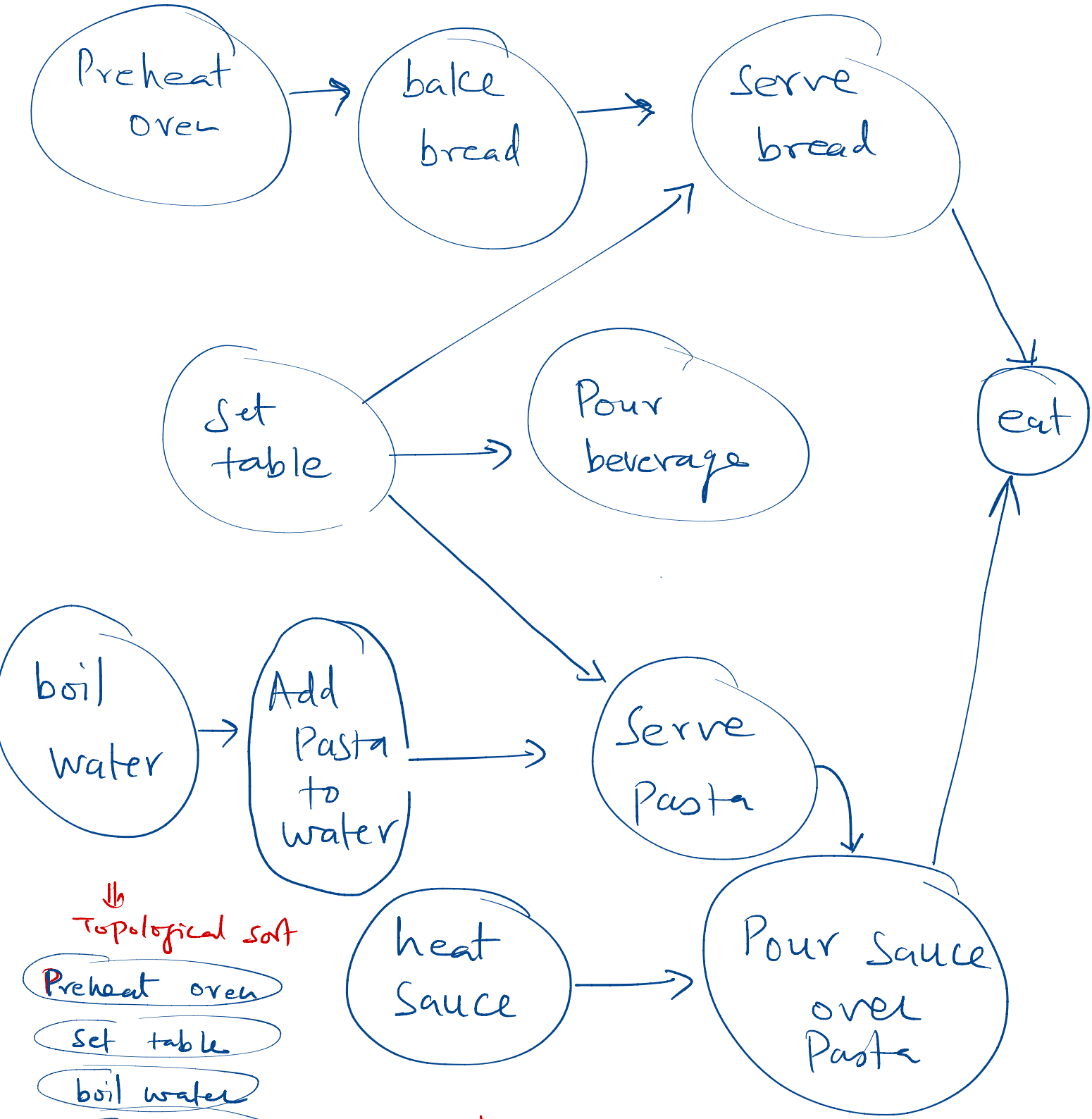
Call DFS(G) to compute finish times v.f for each vertex v

As each vertex is finished, insert it onto the front of a linked list (LL)

Return the LL of vertices

$\Theta(V+E)$

Example: Cooking Pasta as a DAG



Topological sort

1. Preheat oven
2. set table
3. boil water
4. bake bread
5. Add pasta to water
6. Pour beverage
7. serve bread
8. heat sauce
9. Serve Pasta
10. Pour sauce over Pasta
11. eat

Minimum Spanning Trees

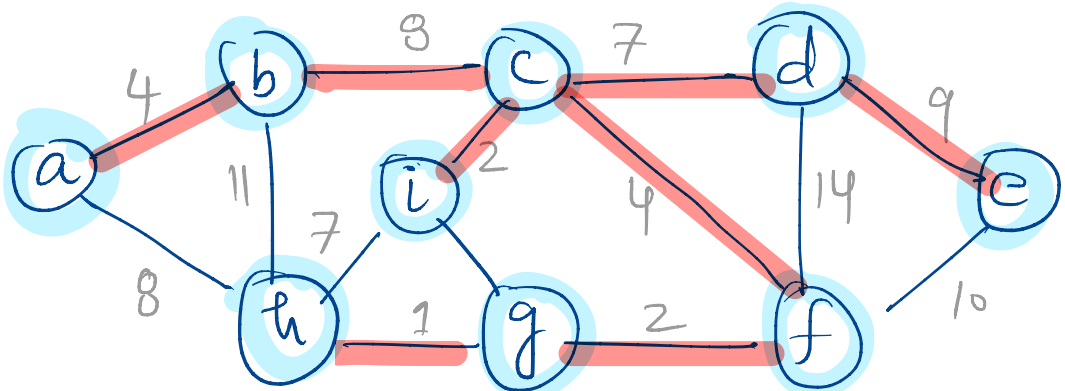
→ Given an undirected graph $G = (V, E)$, where every edge $(u, v) \in E$ has a weight $w(u, v) \in \mathbb{R}$, we wish to find out an acyclic subset $T \subseteq E$ that connects all vertices where the total weight

$$w(T) = \sum_{(u,v) \in T} w(u, v) \quad \text{is minimized}$$

→ Since T is acyclic & connects all vertices, it is a tree.

→ Such a tree is called a Minimum Spanning Tree (MST)

Ex



Two algorithms: → Kruskal's: $O(E \lg V)$ bin heap

→ Prim's: $O(E \lg V)$ bin heap

$O(E + V \lg V)$ fib. heap

Greedy algorithms:

An algorithm that makes a choice that looks best at the moment, ⇒ a locally optimal choice

↳ Don't guarantee global optima.

— General strategy to find MST for a graph $G = (V, E)$ with weight function $W: E \to \mathbb{R}$ using a general greedy approach:

a) Prior to each iteration set A to a subset of MST

b) At each step, we determine edge (u,v) that can be safely added to A without violating the invariant:

> $A \cup \{(u,v)\}$ is a subset of MST

↳ (u,v) is called a safe edge

Generic-MST (G, w)

$A = \emptyset$

while A does not form an MST

  find an edge (u,v) that is safe for A

  $A = A \cup \{(u,v)\}$

return A
