Lecture #17
Graph Algorithm (4)

Algorithm

JBNU

Jinhong Jung

In This Lecture

❑ Topological sorting

- Problem definition
- Application

❑ Algorithms for topological sorting

- Kahn's algorithm
- Algorithm based on DFS (Depth First Search)

Outline

- Topological sorting
- Kahn's algorithm
- Algorithm based on DFS

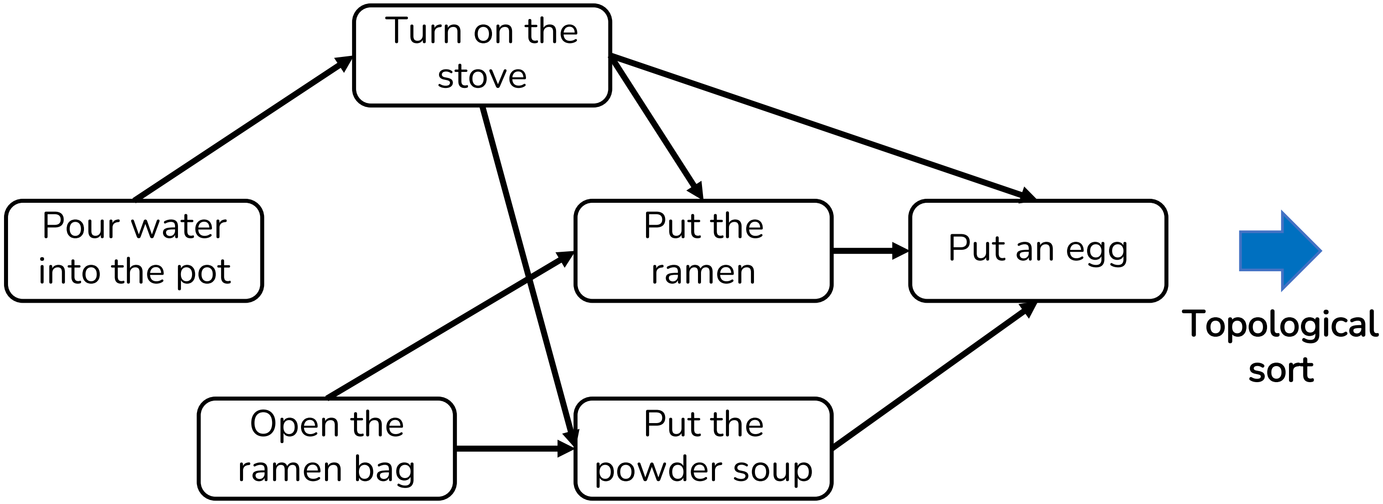
Motivation (1)

- Suppose we are going to make a ramen. There are sub-tasks for the purpose.
  - T1. Turn on the stove
  - T2. Put the egg
  - T3. Put water into the pot
  - T4. Put the ramen
  - T5. Put the powder soup
  - T6. Open the ramen bag

- How to define the order of sub-tasks?

Motivation (2)

- One sub-task must be performed before another.
  - We can put a ramen into a pot after turning on a stove.
  - We can put an egg after putting the ramen.
- Such dependencies are represented as follows:
  - If task i must precede task j, this is represented as a directed edge from task i and task j (i.e., $i \rightarrow j$)



1. Open the ramen bag
2. Put water into the pot
3. Turn on the stove
4. Put the powder soup
5. Put the ramen
6. Put the egg

Directed Acyclic Graph

❑ Topological sort aim to sort nodes in a graph

- For a directed edge $i \rightarrow j$, node i must precede node j in the sorted result!

- What if there a cycle in the directed graph?
  - Suppose we have a cycle $i \rightarrow j \rightarrow k \rightarrow i$.
  - Node j can precede node i in the result while there is an edge $i \rightarrow j$
  - Thus, we cannot do topological sorting in a directed graph having cycles

- Directed graph having no cycles is called **directed acyclic graph (DAG)**

Topological Sorting

❑ Problem definition

- **Input**: a directed acyclic graph (DAG)
- **Output**: sorted nodes in the topological order
  - For $i \rightarrow j$, node i must precede node j in the sorted result

❑ Applications (instruction scheduling)

- **Spreadsheet**: to determine the order of formula cell evaluation
- **Makefile**: to determine the order of compilation tasks
- **Tensorflow**: to determine the order of operations in a computational graph

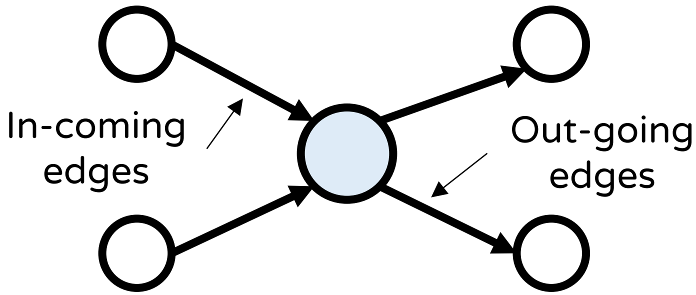
Outline

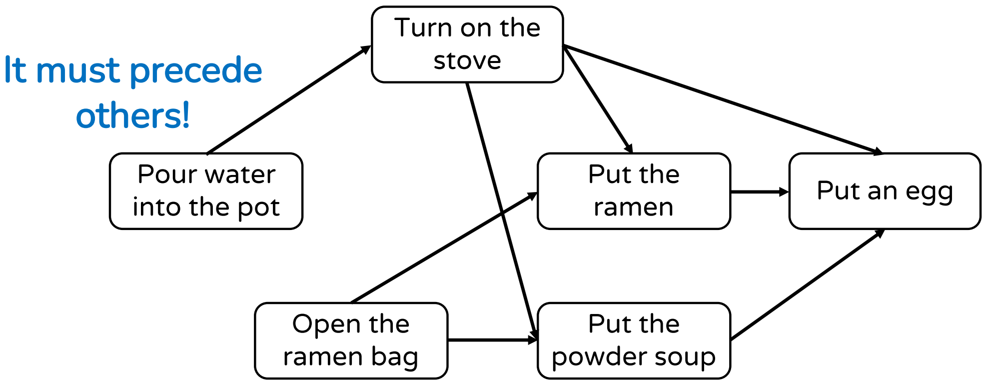
- ❑ Topological sorting
- ❑ **Kahn's algorithm**
- ❑ Algorithm based on DFS

Kahn's Algorithm

❑ Kahn's intuition

- A node not having incoming edges should precede others!





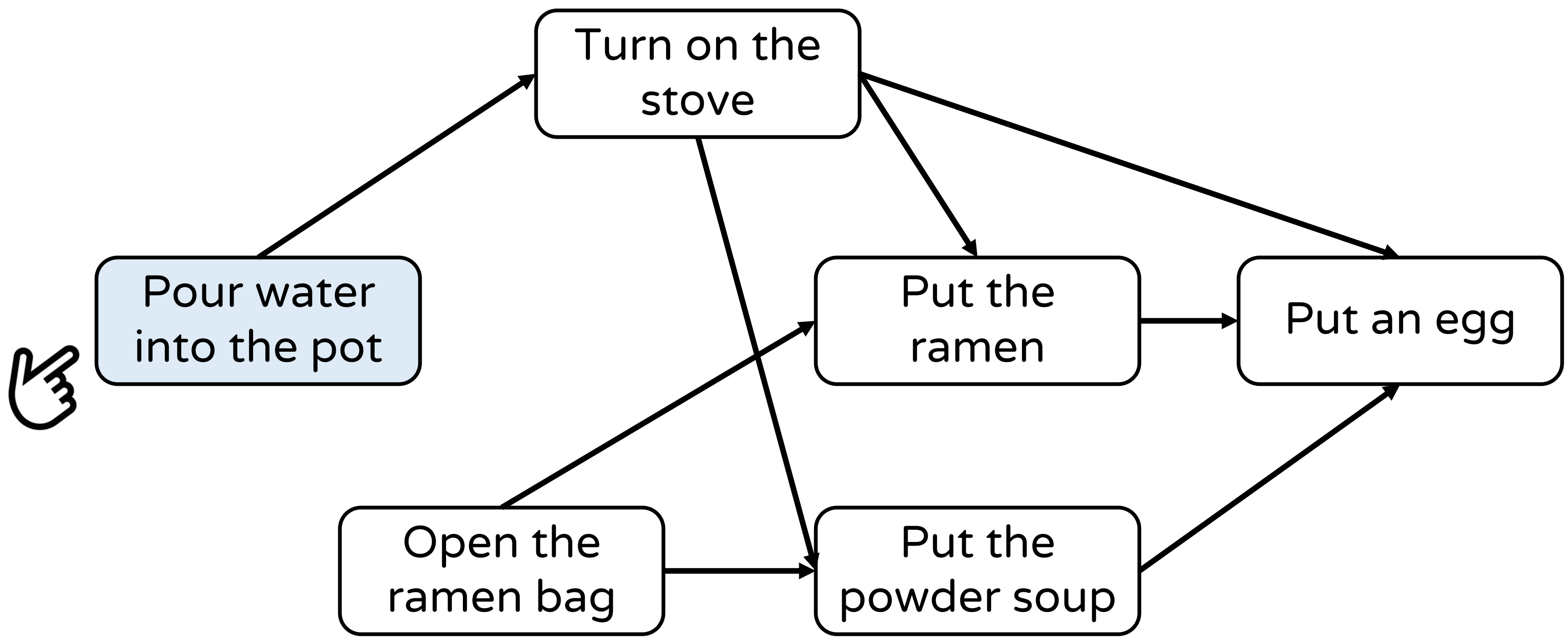
❑ Process of Kahn's Algorithm

- **Step 1.** Pick a node u not having in-coming edges & push u into queue Q (sorted result)
- **Step 2.** Remove node u and out-going edges from u
  - Repeat Steps 1 & 2 until there are no more nodes to be sorted

Example (1)

❑ Step 1. Pick a node u not having in-coming edges

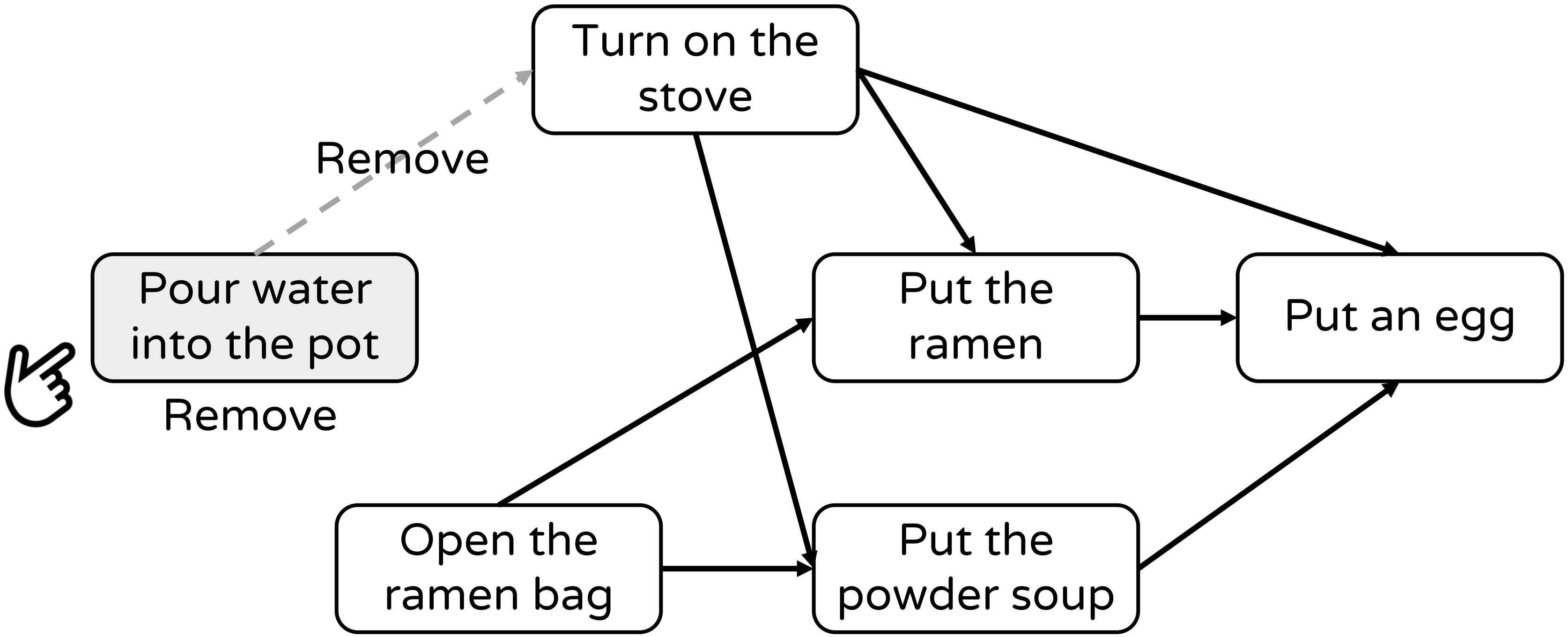
- Then, push u into queue Q



Q Pour wather into the pot

Example (2)

❑ Step 2. Remove node u and its out-going edges

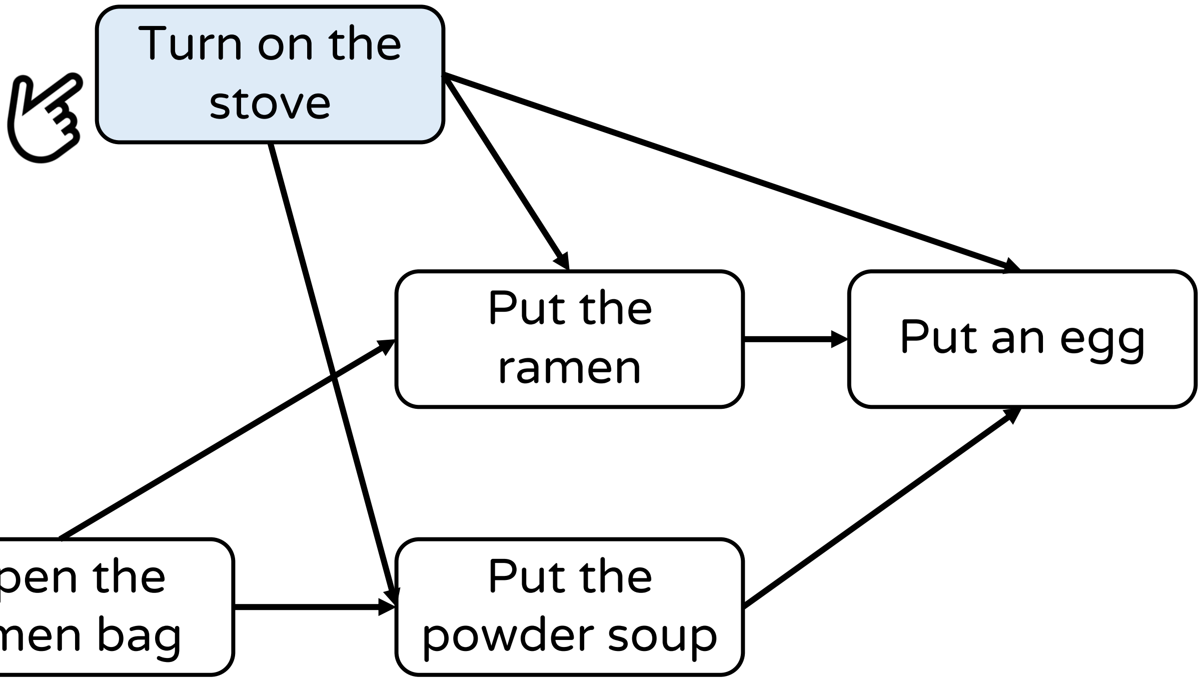


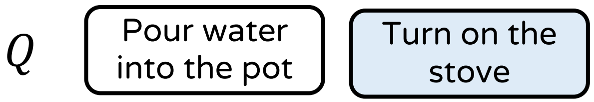
Q Pour water into the pot

Example (3)

❑ Step 1. Pick a node u not having in-coming edges

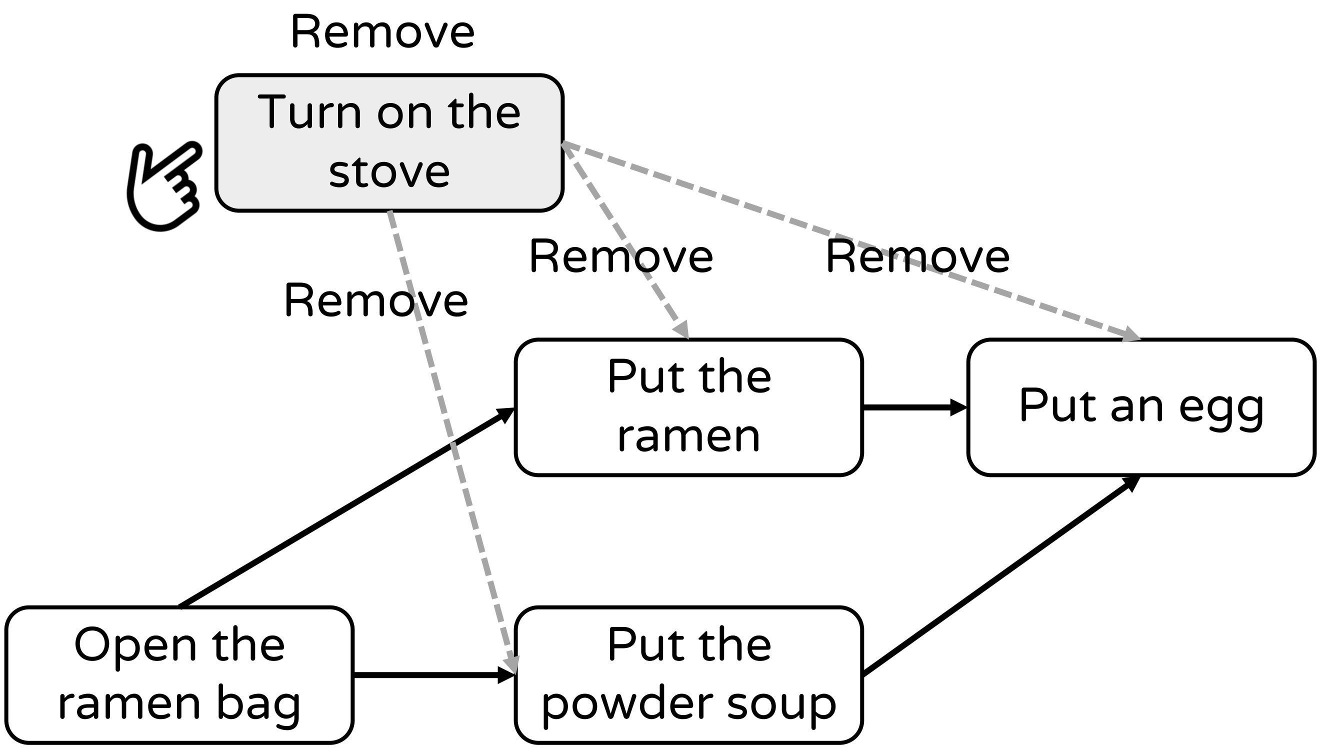
- Then, push u into queue Q



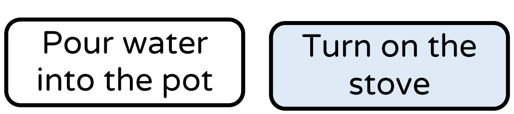


Example (4)

❑ Step 2. Remove node u and its out-going edges



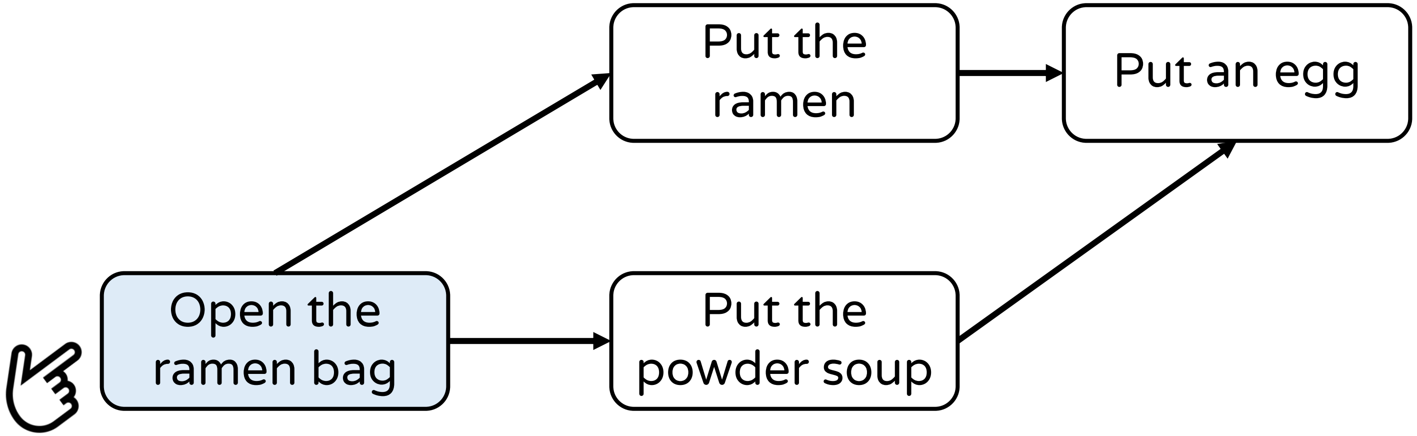
Q

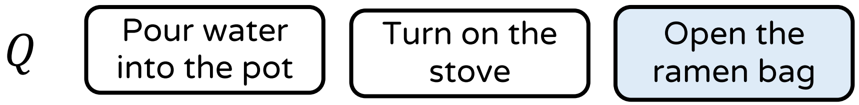


Example (5)

❑ Step 1. **Pick a node u not having in-coming edges**

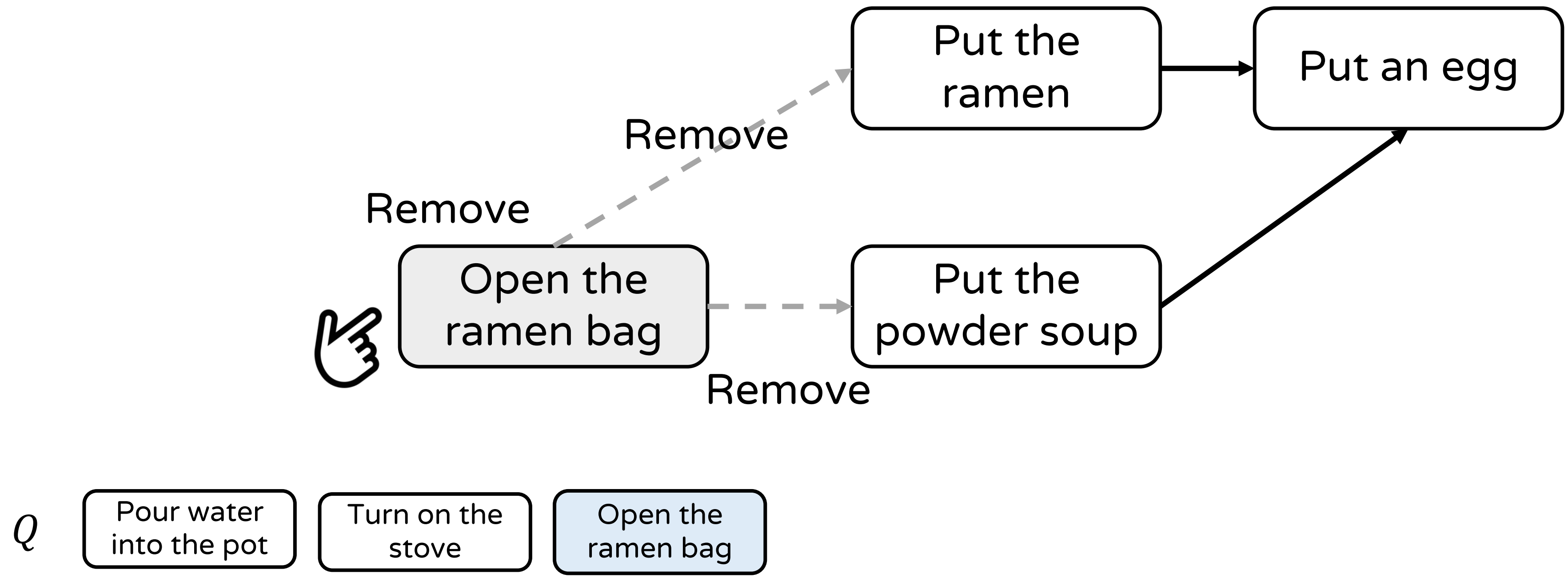
- Then, push u into queue Q





Example (6)

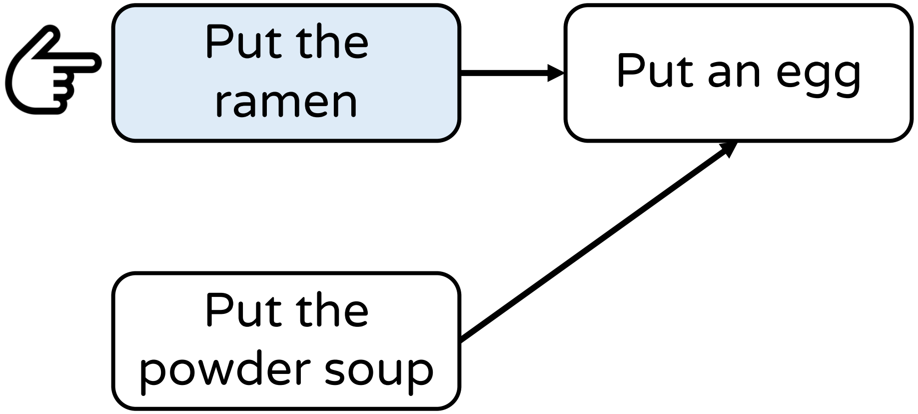
❑ Step 2. Remove node u and its out-going edges

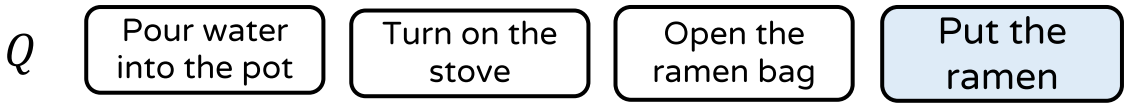


Example (7)

❑ Step 1. Pick a node u not having in-coming edges

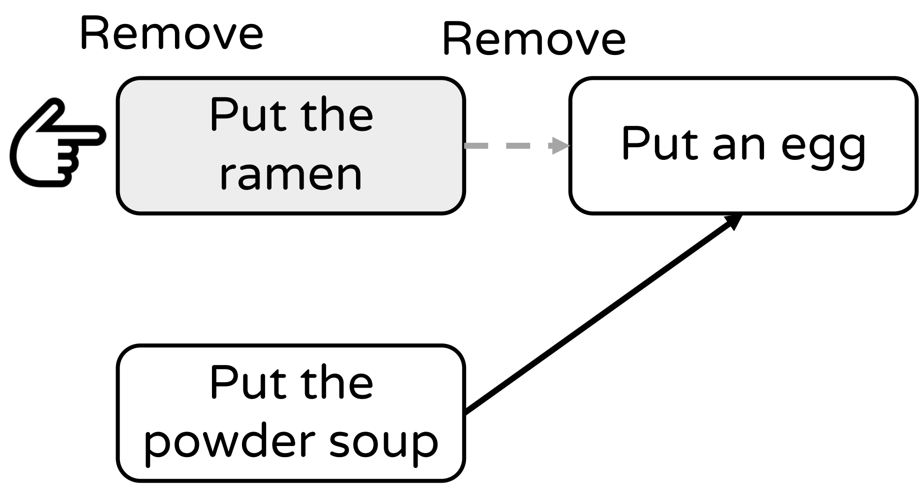
- Then, push u into queue Q



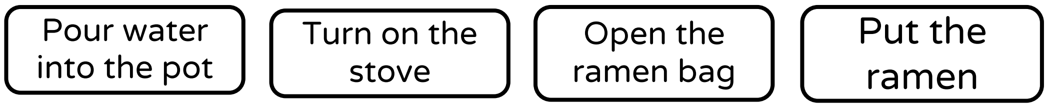


Example (8)

❑ Step 2. Remove node u and its out-going edges

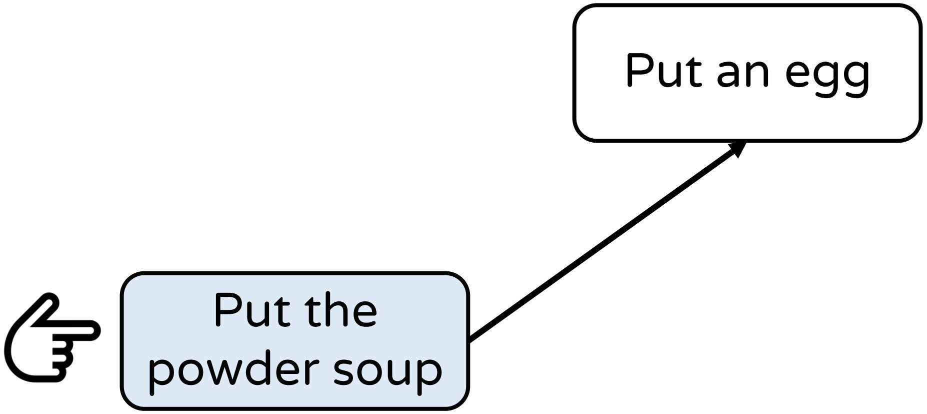


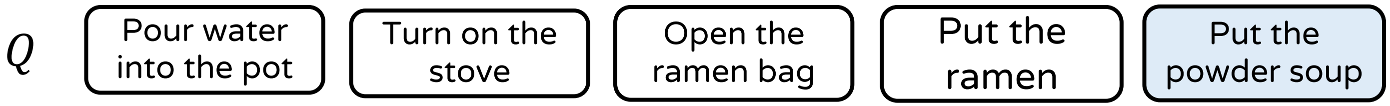
Q



Example (9)

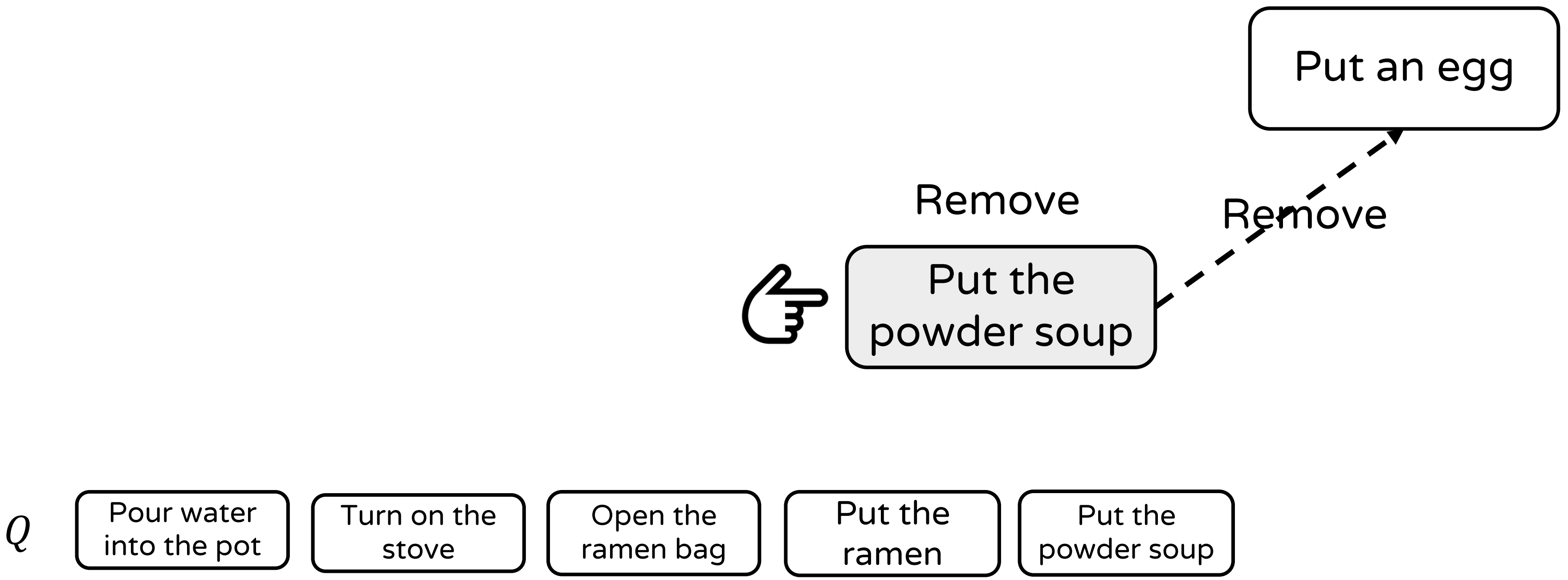
- **Step 1. Pick a node u having no in-coming edges**
  - Then, push u into queue Q





Example (10)

❑ Step 2. Remove node u and its out-going edges



Example (11)

❑ Step 1. Pick a node u not having in-coming edges

- Then, push u into queue Q

Put an egg

Topologically sorted

Q

Pour water into the pot	Turn on the stove	Open the ramen bag	Put the ramen	Put the powder soup	Put an egg

Kahn's Algorithm

❑ Pseudocode

```
def kahn(G):
    Q ← queue()
    S ← set of all nodes not having in-coming edges

    while S is not empty:
        u ← remove a node from S
        Q.enqueue(u)
        for each out-neighbor v of node u in N⃗_u:
            remove edge u → v
            if v hasn't other in-coming edges:
                S.push(v)

    return Q        # topologically sorted order
```

Step 1. Pick a node not having in-coming edges & push it into Q

Step 2. Remove the selected node and its out-going edges

Kahn's Algorithm

Implementation details

- If we use adjacency list and store out-going edges, it is easy to remove an out-going edge (i.e., $O(1)$ time using iterator)
- If we use a queue for S, it is easy to remove and push an item (i.e., $O(1)$ time)
- How can we know if a node has in-coming edges or not?
  - Use a counter variable $c[v]$ to keep tracking the number of in-coming neighbors
    - Increase $c[v]$ by 1 when $u \to v$ is inserted into the adjacency list
    - Decrease $c[v]$ by 1 when $u \to v$ is removed from the adjacency list
  - If $c[v]$ is 0, then node v hasn't in-coming edges.
  - The above operations take $O(1)$ time, respectively.

Kahn's Algorithm

❑ Time complexity analysis

```
def kahn(G):
    Q ← queue()
    S ← set of all nodes not having in-coming edges

    while S is not empty:
        u ← remove a node from S
        Q.enqueue(u)
        for each out-neighbor v of node u in N⃗_u:
            remove edge u → v
            if v hasn't other in-coming edges:
                S.push(v)

    return Q          # topologically sorted order
```

$O(1)$

$O(|\vec{N}_u|)$

$$\sum_{u \in V} (1 + |\vec{N}_u|) = O(n+m)$$

❑ Limitation

- Kahn's algorithm directly modifies the input graph
  - Extra costs occur for checking in-coming edges and removing edges

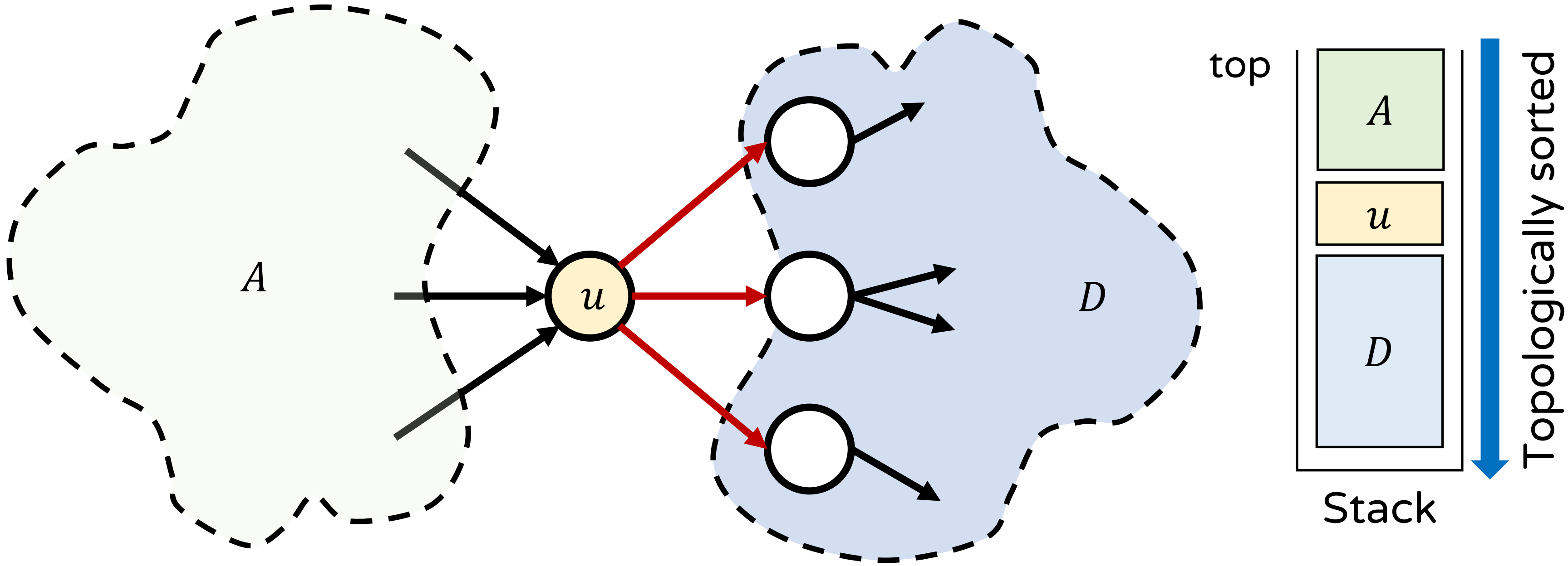
Outline

- ❑ Topological sorting
- ❑ Kahn's algorithm
- ❑ Algorithm based on DFS

Algorithm Based on DFS

❑ Main intuition of algorithm based on DFS

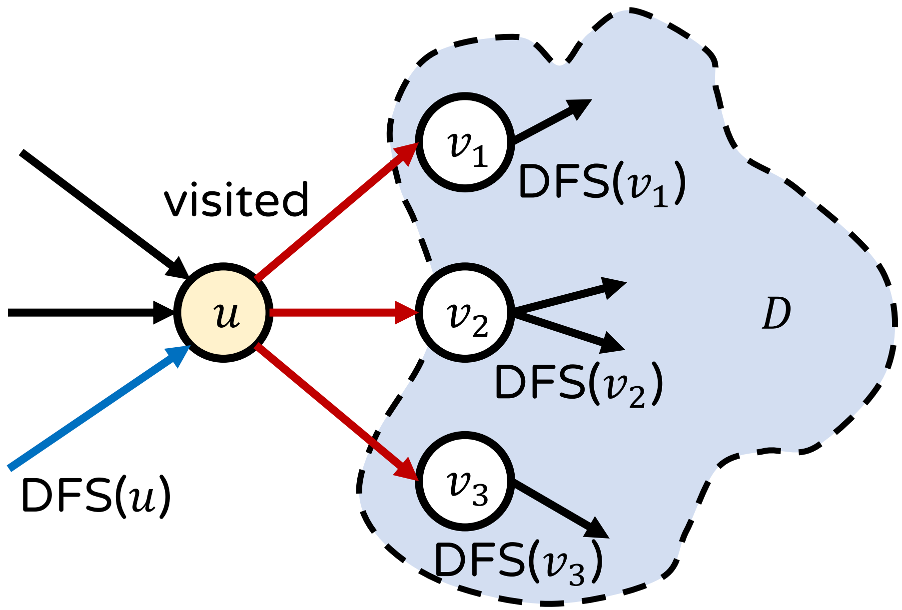
- Suppose we fill nodes reversely in the topological order
- Then, node u can be placed after all its out-going neighbors and descendant are processed
  - Since node u must precede its out-going neighbors

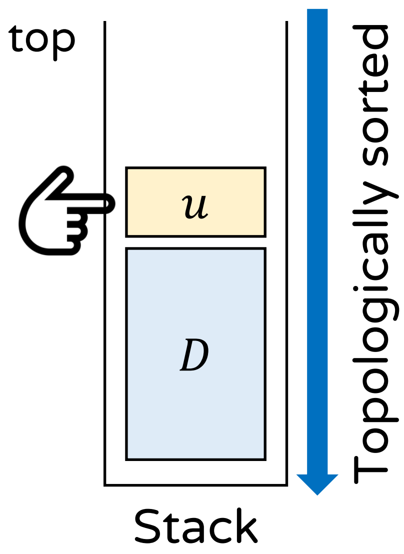


Algorithm Based on DFS

❑ Main intuition of algorithm based on DFS

- During DFS, push node u into the stack after DFS of u's out-neighbors are finished
  - Do not need to modify the input graph!

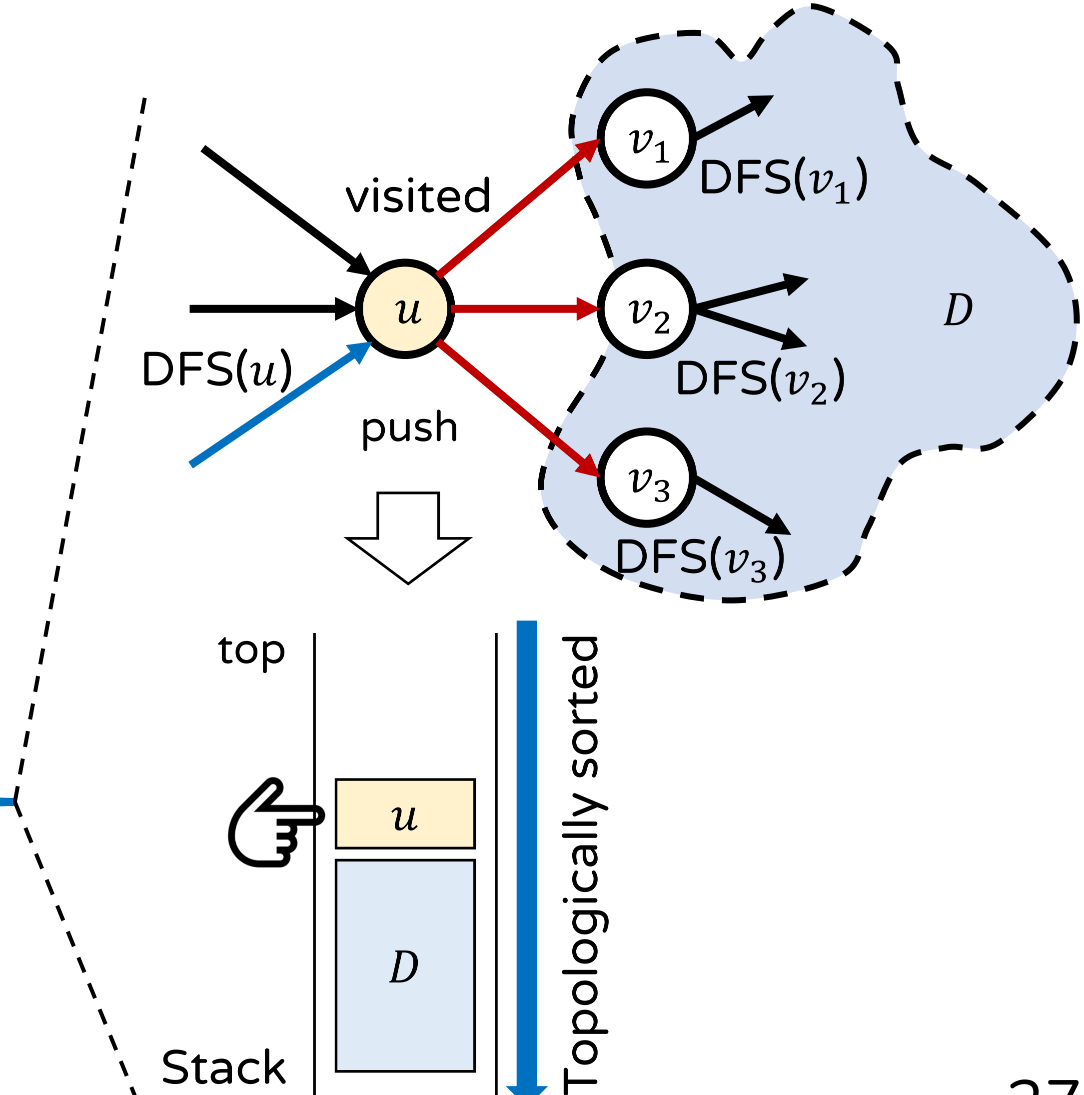




Algorithm Based on DFS

❑ Pseudocode

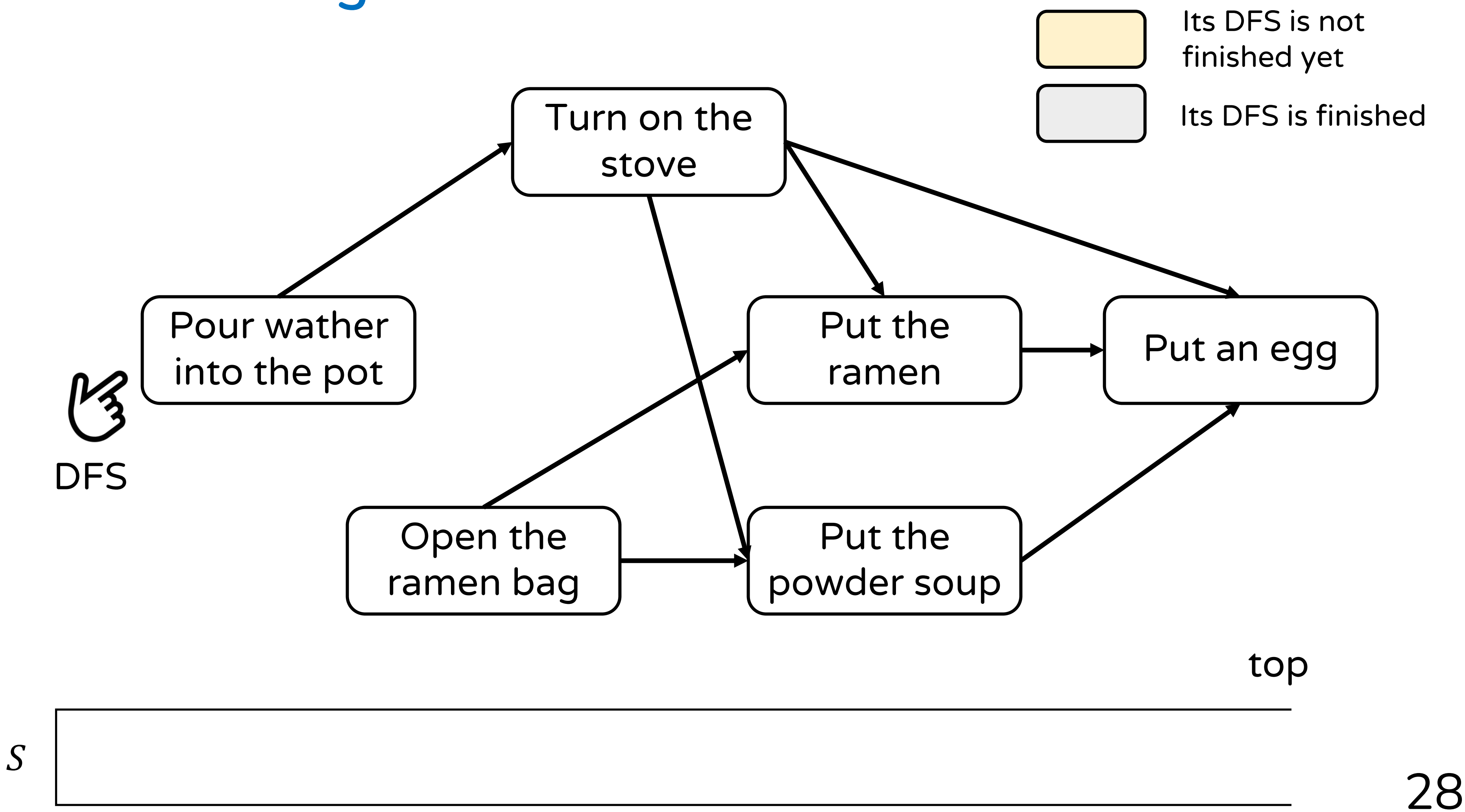
```
def topological-sort(G):
    S ← stack()
    for each v in V:
        visited[v] ← false
    for each v in V:
        if visited[v] is false:
            dfs(v, S)
    return S

def dfs(u, S):
    visited[u] ← true;
    for each v in N⃗_u:
        if visited[v] is false:
            dfs(v, S)
    S.push(u)
```



Time complexity: $O(n+m)$

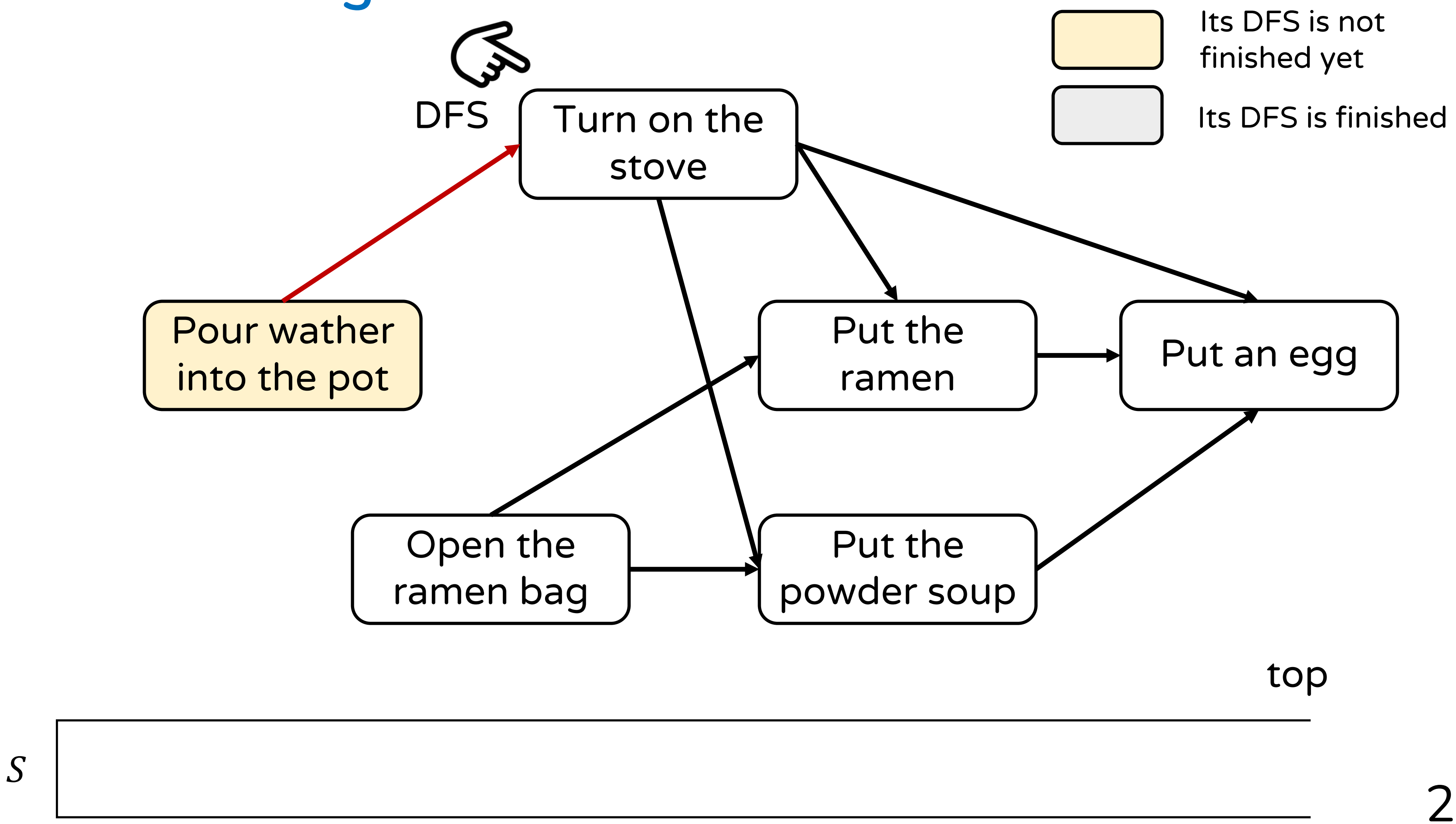
Example (1)

- During DFS, push node u into the stack after DFS of u's out-neighbors are finished



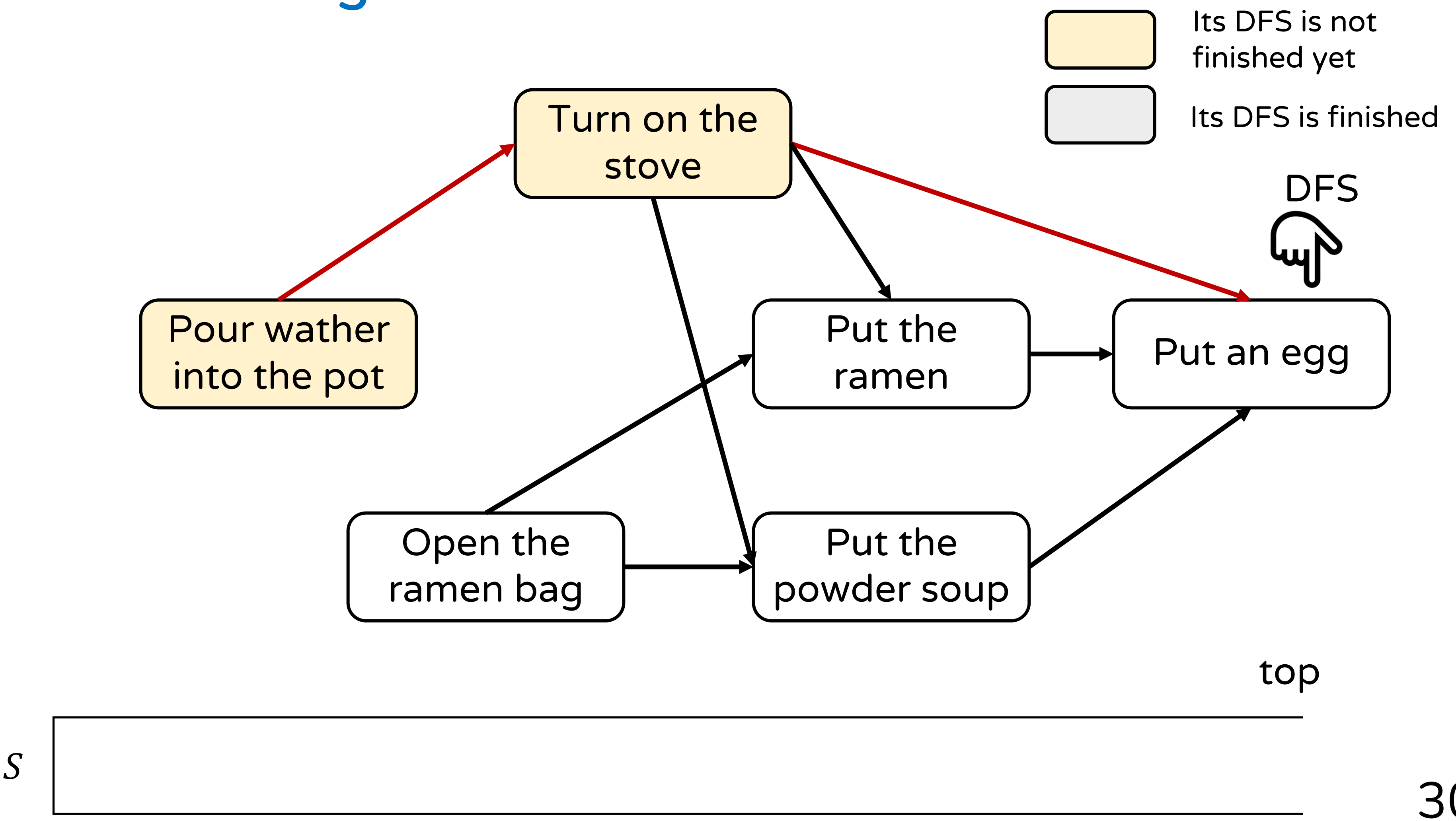
Example (2)

- During DFS, push node u into the stack after DFS of u's out-neighbors are finished



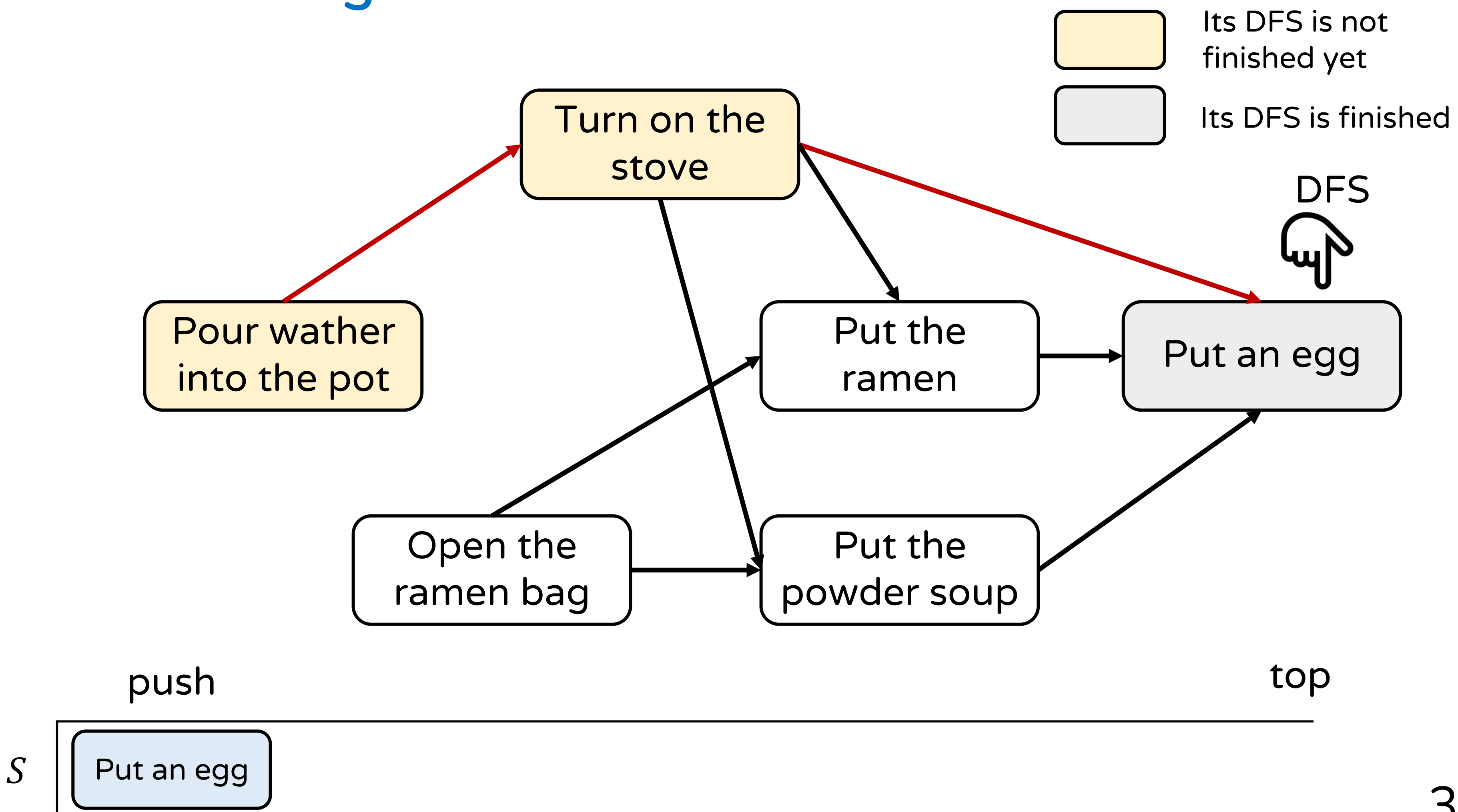
Example (3)

- During DFS, push node u into the stack after DFS of u's out-neighbors are finished



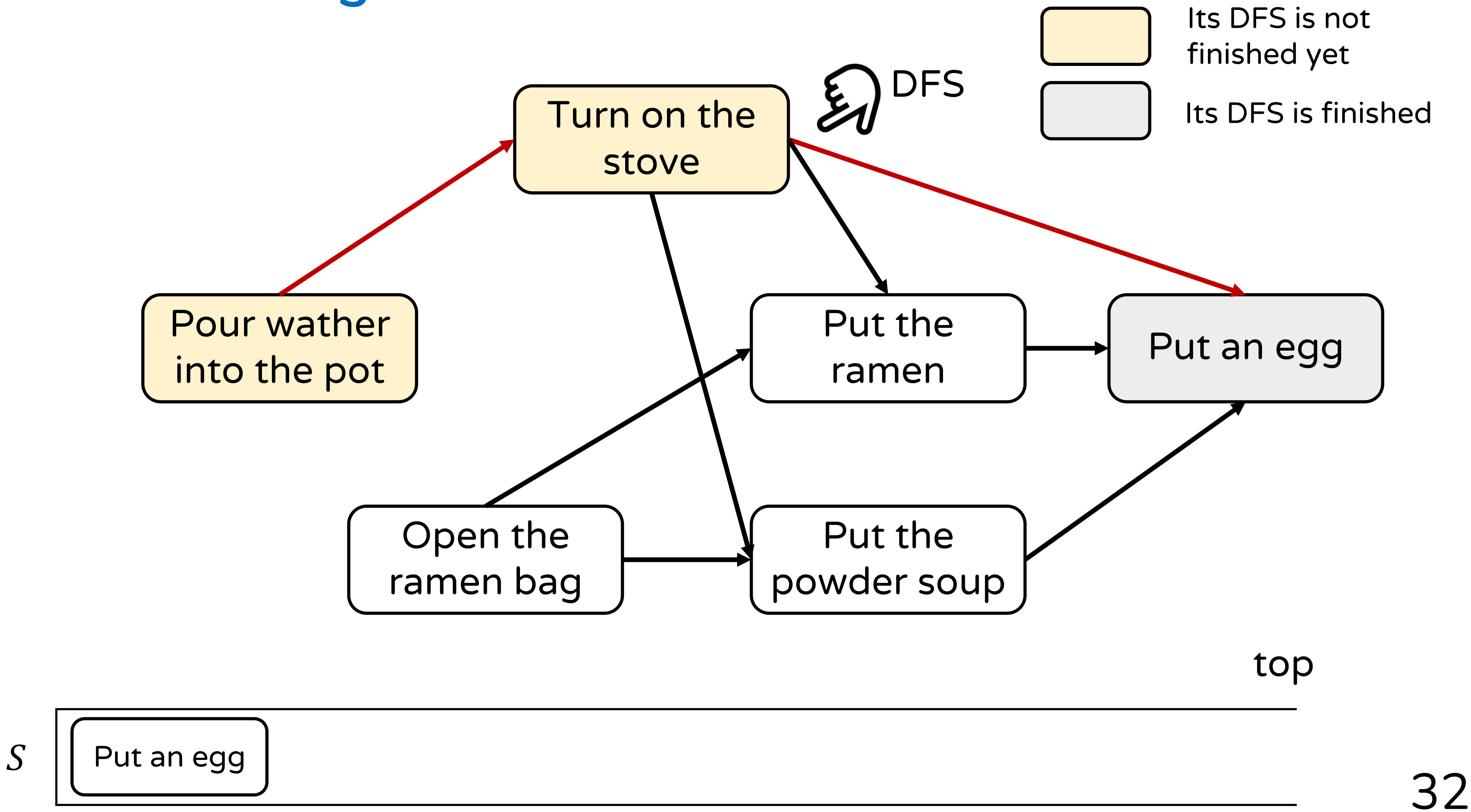
Example (4)

- During DFS, push node u into the stack after DFS of u's out-neighbors are finished



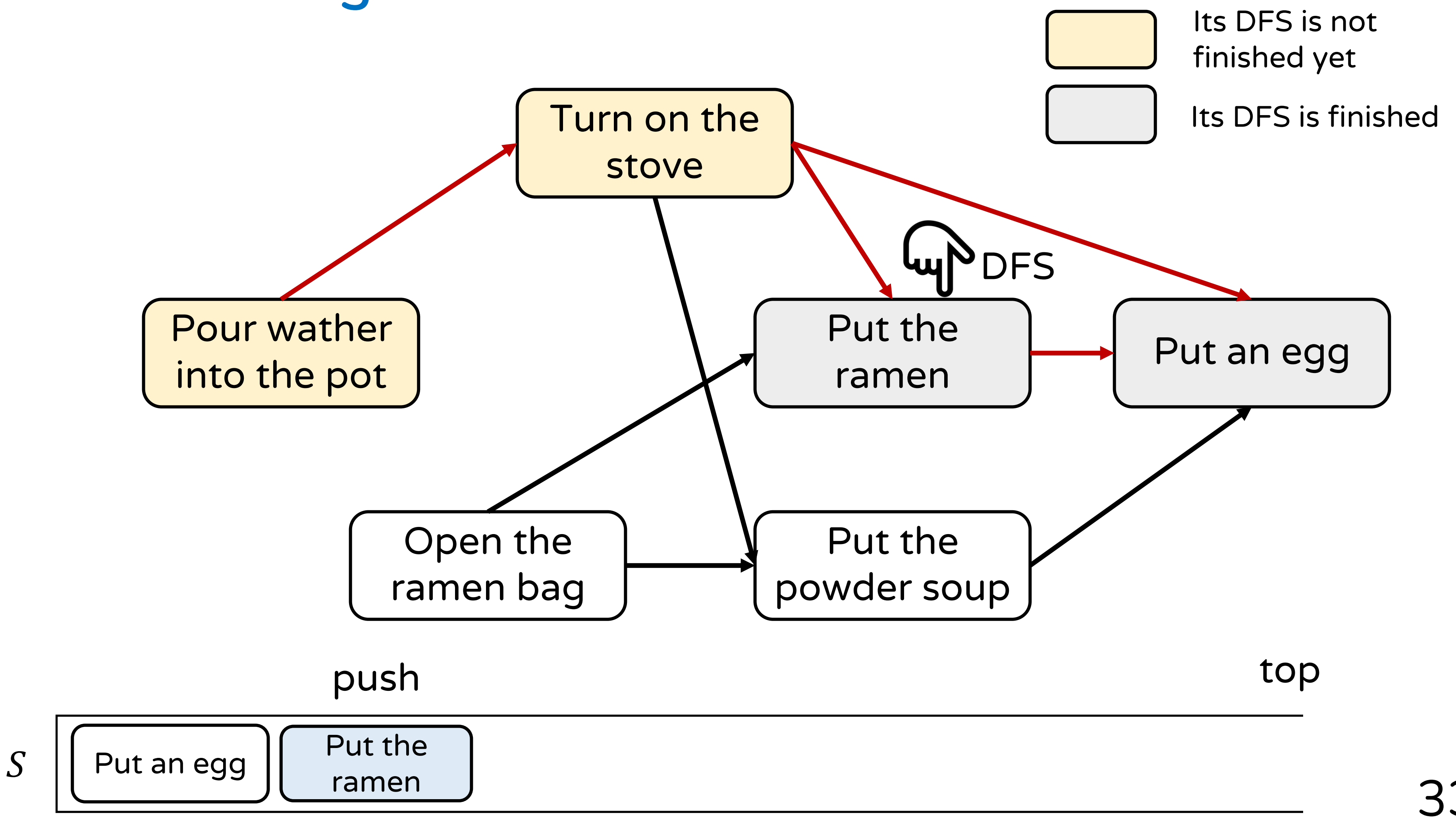
Example (5)

- During DFS, push node u into the stack after DFS of u's out-neighbors are finished



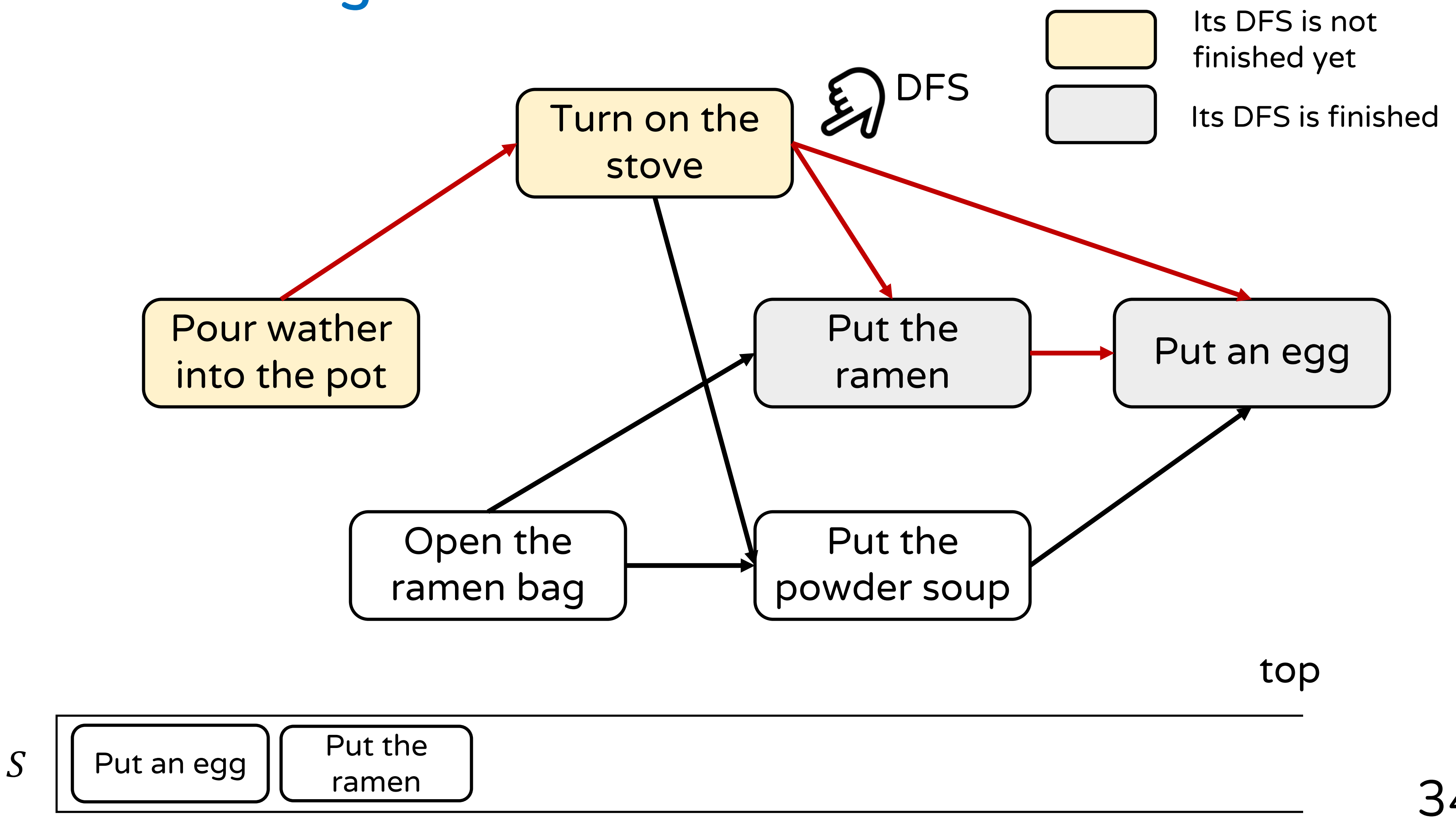
Example (6)

- During DFS, push node u into the stack after DFS of u's out-neighbors are finished



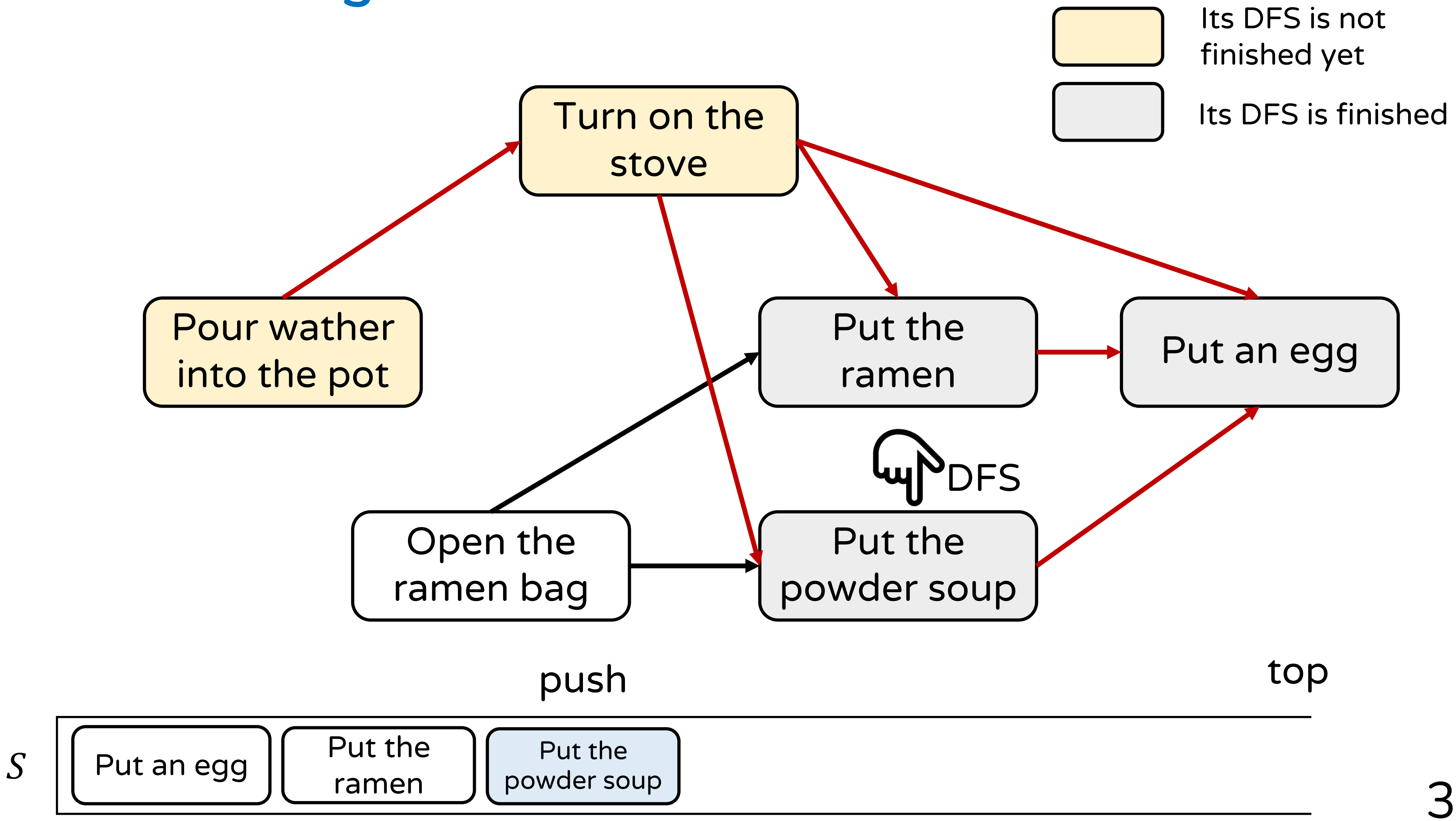
Example (7)

- During DFS, push node u into the stack after DFS of u's out-neighbors are finished



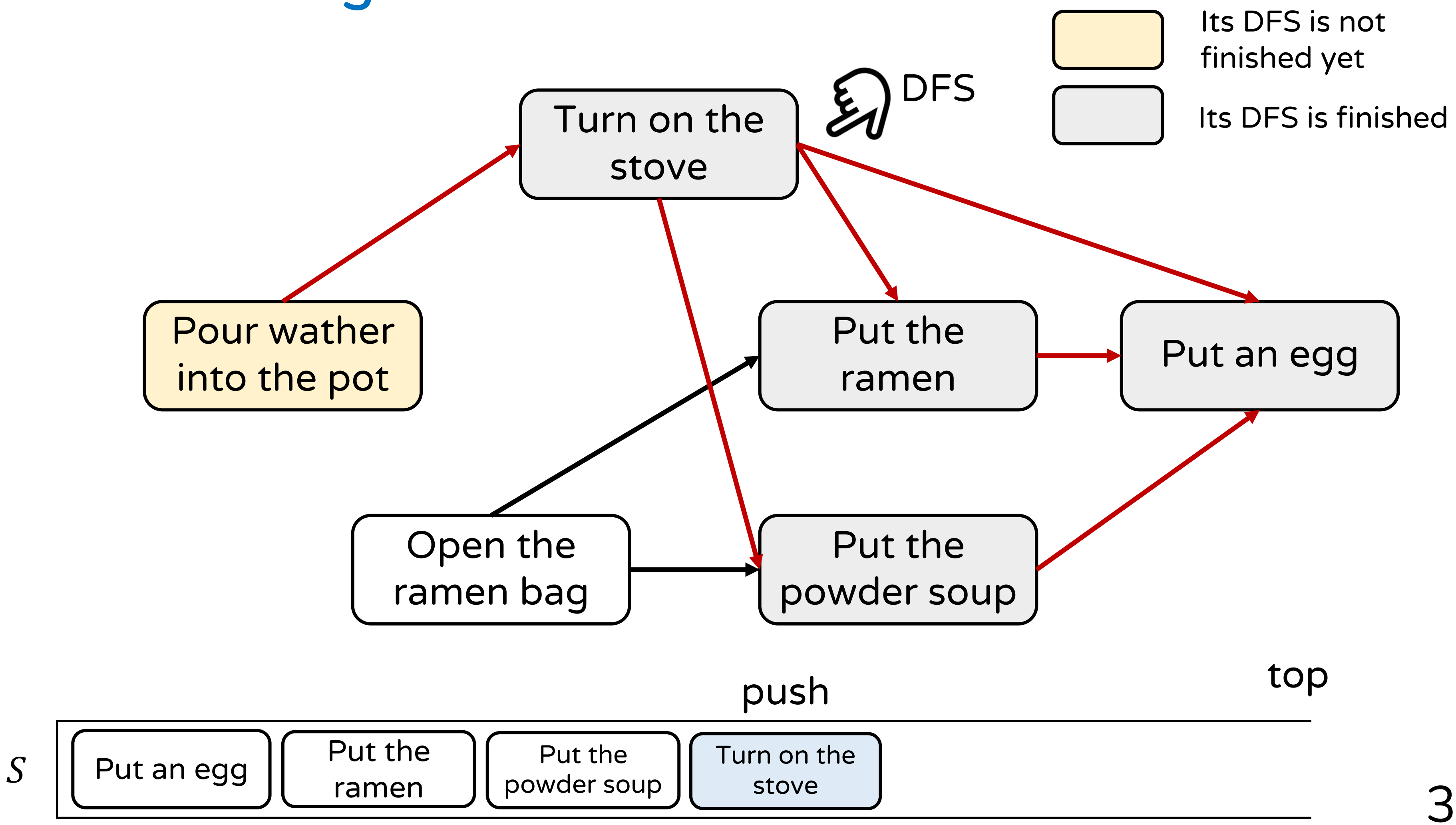
Example (8)

- During DFS, push node u into the stack after DFS of u's out-neighbors are finished



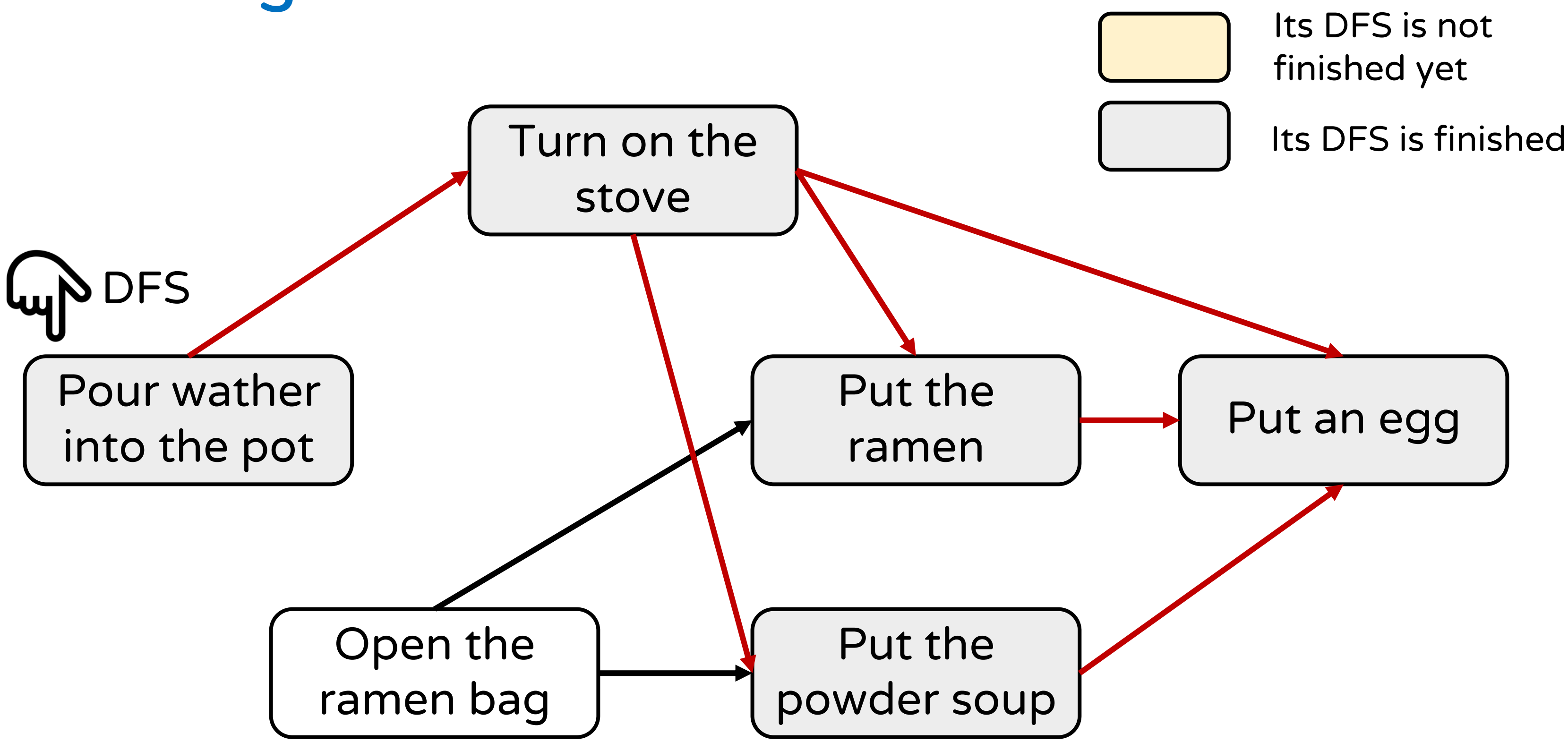
Example (9)

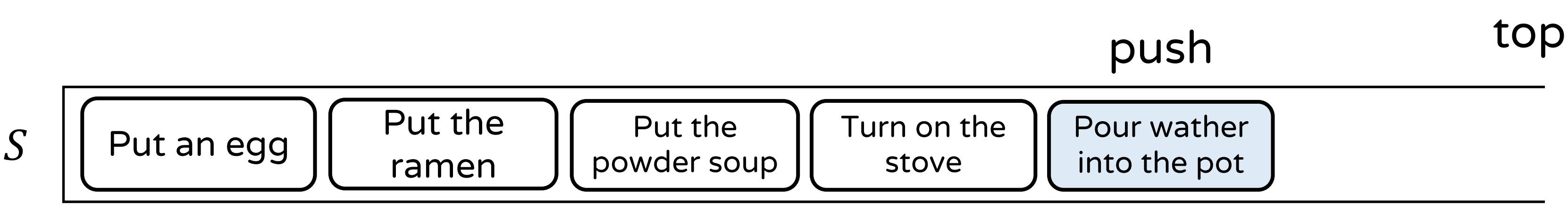
- During DFS, push node u into the stack after DFS of u's out-neighbors are finished



Example (10)

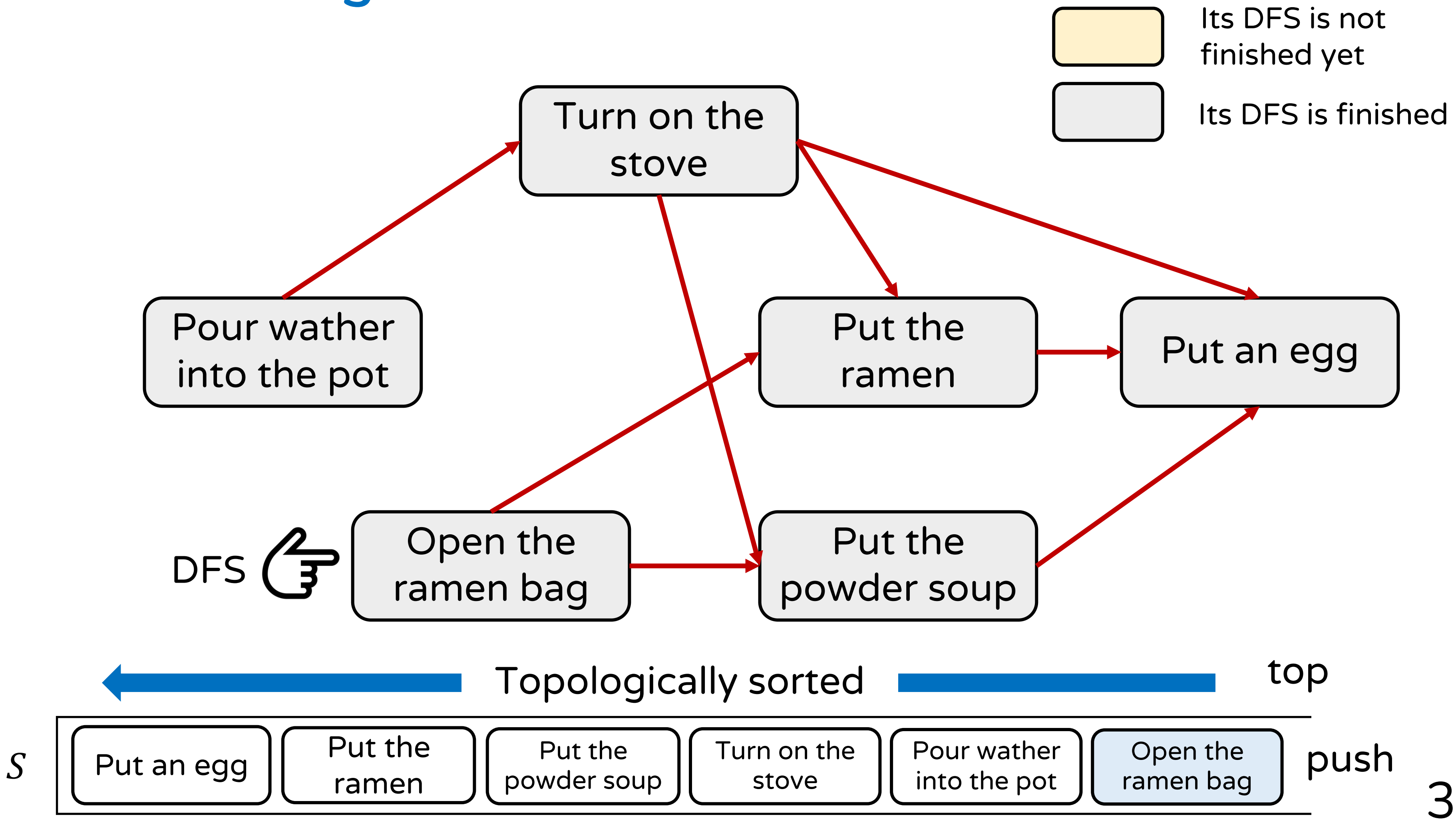
- During DFS, push node u into the stack after DFS of u's out-neighbors are finished





Example (11)

- During DFS, push node u into the stack after DFS of u's out-neighbors are finished



What You Need To Know

❑ Topological sorting

- Sort nodes of a DAG in the topological order
  - For $i \to j$, node i must precede node j in the sorted result

❑ Kahn's algorithm

- Remove a node with no in-coming edges step by step
- Time complexity is $O(n+m)$

❑ Algorithm based on DFS

- During DFS, push node u into the stack after DFS of u's out-neighbors are finished
  - Without modifying the input graph
- Time complexity is $O(n+m)$

In Next Lecture

- String matching - How can we efficiently search for "algorithm" in a document?

String-searching algorithm

From Wikipedia, the free encyclopedia

In computer science, **string-searching algorithms**, sometimes called **string-matching algorithms**, are an important class of string algorithms that try to find a place where one or several strings (also called patterns) are found within a larger string or text.

A basic example of string searching is when the pattern and the searched text are arrays of elements of an alphabet (finite set) Σ. Σ may be a human language alphabet, for example, the letters *A* through *Z* and other applications may use a *binary alphabet* (Σ = {0,1}) or a *DNA alphabet* (Σ = {A,C,G,T}) in bioinformatics.

Thank You