CS3383 Unit 3: Dynamic Programming

David Bremner

March 1, 2024

Outline

Dynamic Programming

Shortest path in DAG

March Break Hotels

Scenario

Wanted	Cheap holiday
Costs	Hotel + Taxi, no charge for inconvenience

March Break Hotels

Scenario

Wanted Cheap holiday

Costs Hotel + Taxi, no charge for inconvenience

Taxi Cost

	a	b	c	aprt
a	0	10	30	50
b	10	0	30	50
c	30	30	0	50
aprt	50	50	50	0

Hotel Price

	1	2	3	4
a	100	100	100	100
b	80	40	120	120
c	50	80	80	80

It's a trap!

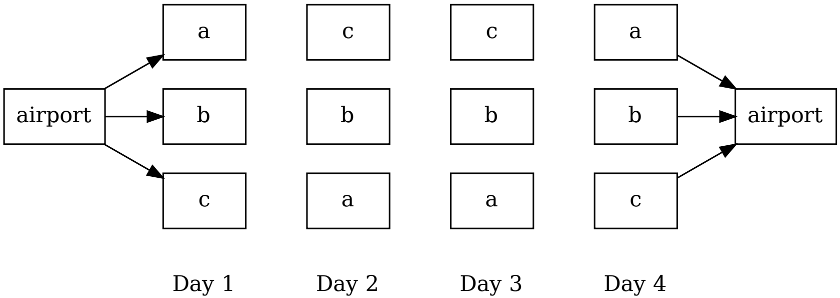
Hotel Price

	1	2	3	4
a	100	100	100	100
b	80	40	120	120
c	50	80	80	80

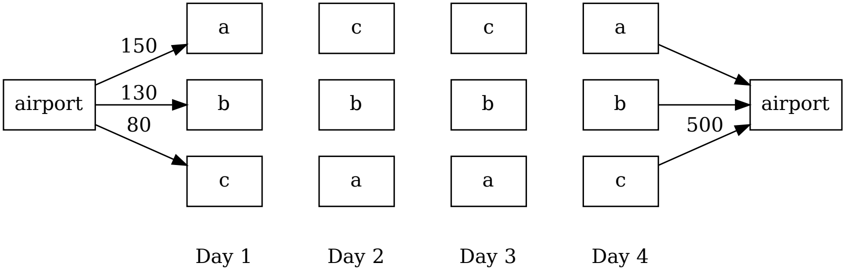
Taxi Cost

	a	b	c	airport
a	0	10	30	50
b	10	0	30	50
c	1000	1000	0	500
airport	50	50	50	0

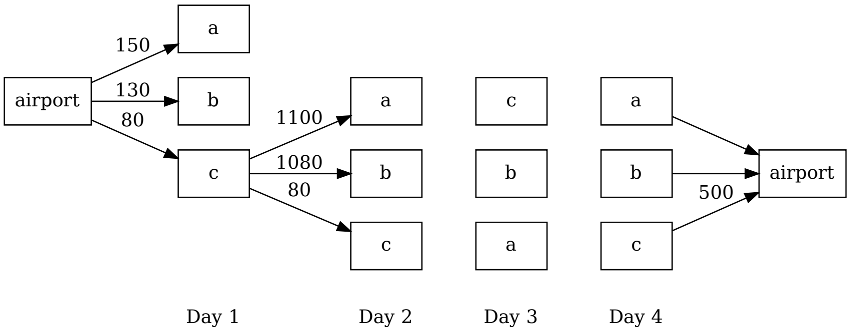
Let's get graphical



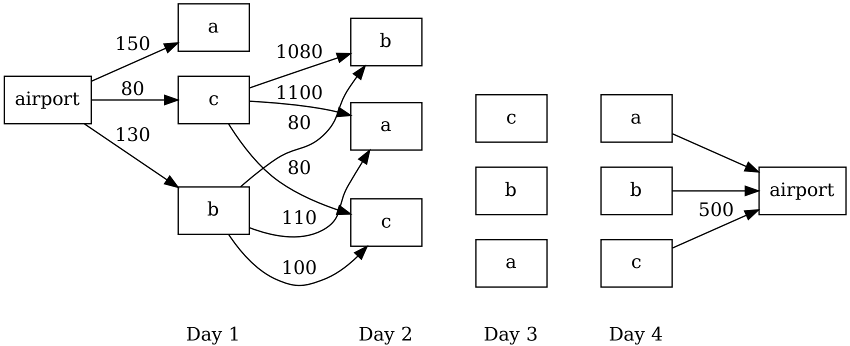
Let's get graphical



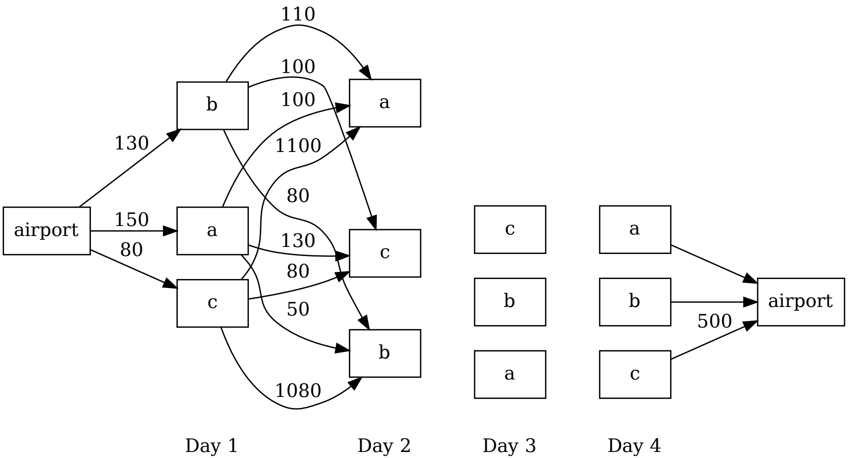
Let's get graphical



Let's get graphical



Let's get graphical



Djikstra considered overkill

- There are no negative edge weights, so shortest path is tractable.
- Even better, we have an *acyclic* graph (why?)
- So we find a shortest path in linear time after *topological sorting*.

“Recursive” topological sort

Recursive ”algorithm”

1. Remove a *source* from the DAG, and put it first.
2. Topologically sort the remaining graph.

- how to quickly find a source?

“Recursive” topological sort

Recursive ”algorithm”

1. Remove a *source* from the DAG, and put it first.
2. Topologically sort the remaining graph.

- how to quickly find a source?
- Use some auxilary data structure to track sources across iterations

Using a Queue

BFS-like topological sort

```
function TopSort(G)
    Q ← All Sources
    while !empty(Q) do
        v ← deq(Q)


    end while
end function
```

Using a Queue

BFS-like topological sort

```
function TopSort(G)
    Q ← All Sources
    while !empty(Q) do
        v ← deq(Q)
        Output v
        Remove v, add new sources to Q
    end while
end function
```

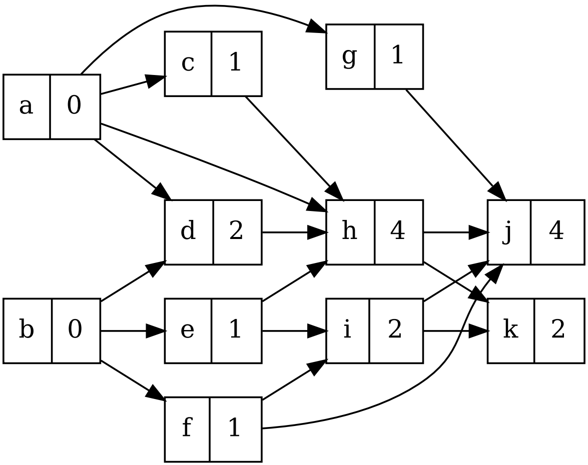
Using a Queue

BFS-like topological sort

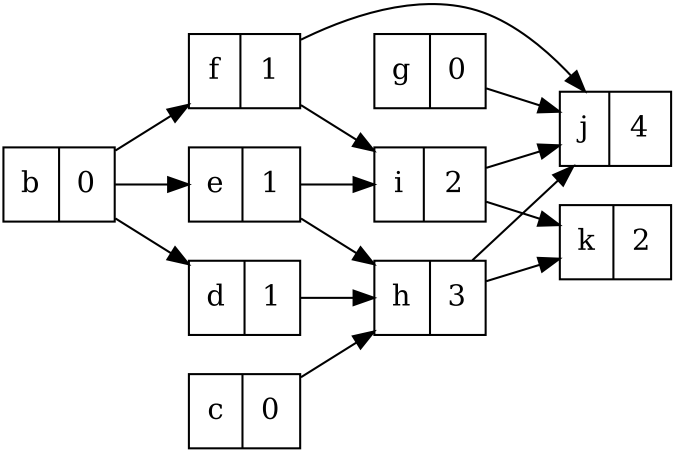
1: **function** TopSort(G)
2: $Q \leftarrow$ All Sources
3: **while** !empty(Q) **do**
4: $v \leftarrow \mathrm{deq}(Q)$
5: Output v
6: Remove v, add new sources to Q
7: **end while**
8: **end function**

- What is the complexity of step 6?

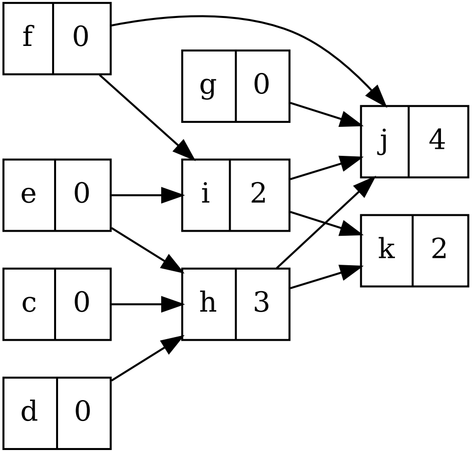
Topological sort with counters



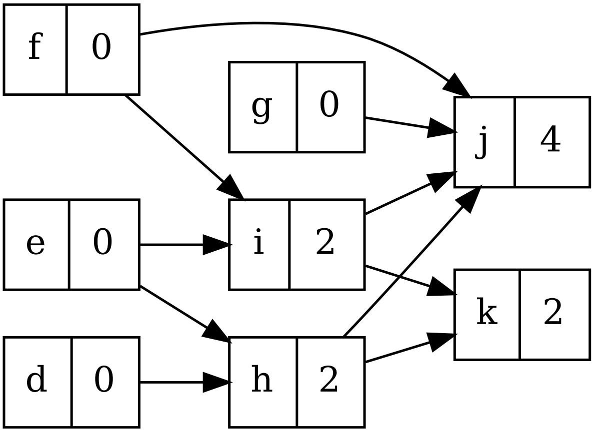
Topological sort with counters



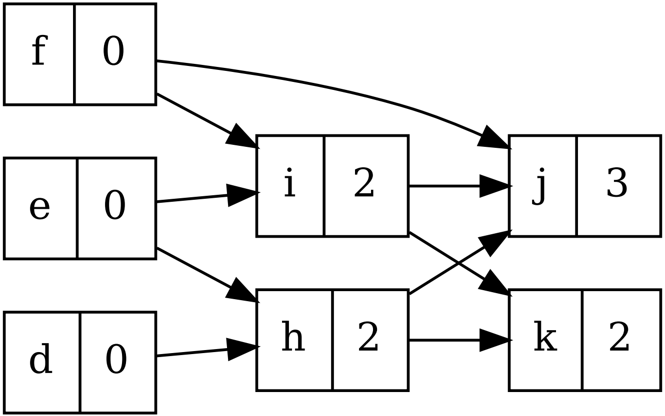
Topological sort with counters



Topological sort with counters



Topological sort with counters



No priority queue needed

```
while len(Q) > 0:
    v = Q.popleft()
    rank[v]=len(output)
    output.append(v)
    for (u,_) in G[v]:
        count[u] -= 1
        if count[u] == 0:
            Q.append(u)
```

Shortest Paths in DAGs

- Every path in a DAG goes through nodes in linearized (topological sort) order.

```
for j in range(rank[root]+1,n):
    v = order[j]
    for (prev,w) in In[v]:
        if w+dist[prev] < dist[v]:
            dist[v]=w+dist[prev]
```

Shortest Paths in DAGs

- Every path in a DAG goes through nodes in linearized (topological sort) order.
- *every node is reached via its predecessors*

```
for j in range(rank[root]+1,n):
    v = order[j]
    for (prev,w) in In[v]:
        if w+dist[prev] < dist[v]:
            dist[v]=w+dist[prev]
```

Shortest Paths in DAGs

- Every path in a DAG goes through nodes in linearized (topological sort) order.
- *every node is reached via its predecessors*
- So we need a single loop after sorting.

```
for j in range(rank[root]+1,n):
    v = order[j]
    for (prev,w) in In[v]:
        if w+dist[prev] < dist[v]:
            dist[v]=w+dist[prev]
```

What makes this *Dynamic Programming*?

Ordered Subproblems

In order to solve our problem in a single pass, we need

- An ordered set of subproblems $L(i)$

What makes this *Dynamic Programming*?

Ordered Subproblems

In order to solve our problem in a single pass, we need

- An ordered set of subproblems $L(i)$
- Each subproblem $L(i)$ can be solved using only the answers for $L(j)$, for $j < i$.