CS3383 Unit 3: Dynamic Programming

David Bremner

February 24, 2024

Dynamic Programming

Shortest path in DAG

Background

Dynamic programming DPV 6, CLRS 15

Topological Sort CLRS 22.4, DPV 3.3

Shortest path in DAG DPV 6.1

November Break Hotels

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!

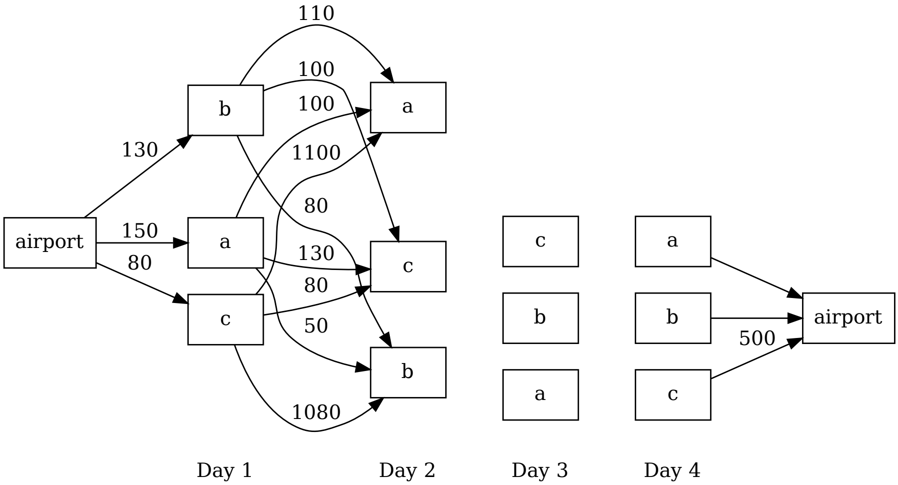
Hotel Price

	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



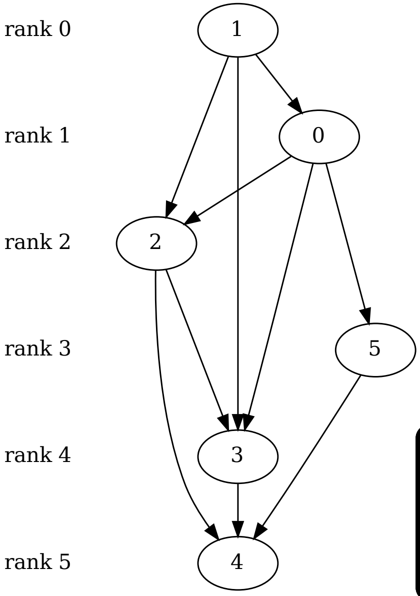
Djikstra considered overkill

- We have a DAG with non-negative edge weights
- So we find a shortest path in linear time after *topological sorting*.
- We can do topological sort by DFS or by (essentially) BFS.

Topological Sort

Input DAG $G = (E, V)$

Output `rank[v]` s.t.
$(u, v) \in E \Rightarrow$
$\text{rank}[u] < \text{rank}[v]$



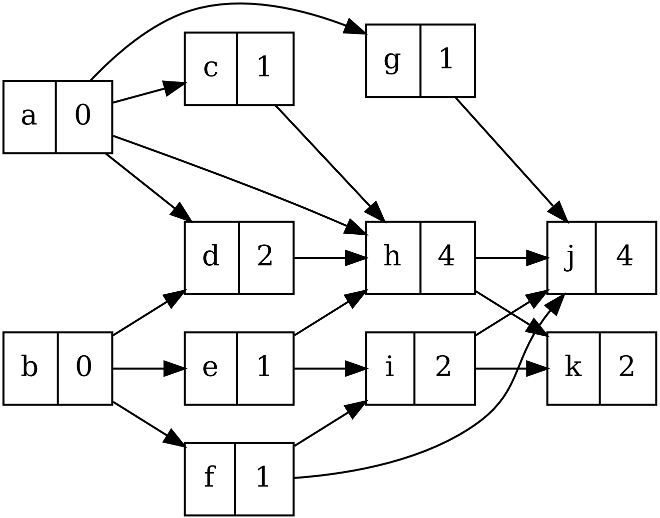
"Recursive" topological sort

Recursive topological sort

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

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.
- *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]
```

So what does this have to do with 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 $j < i$.