CSI - 3105 Design & Analysis of Algorithms
Course 13

Jean-Lou De Carufel

Fall 2020

Chapter 5: Dynamic Programming

Section 5.1: Shortest Paths in Acyclic Graphs

Let $G = (V, E)$ be a directed acyclic graph, where each edge (u, v) has a weight $wt(u, v) > 0$.

Topological sorting: vertices are numbered $v_1, v_2, ..., v_n$ such that for each edge (v_i, v_j), we have $i < j$.

Let $s = v_1$ and $t = v_n$.

How do we compute the shortest path from s to t?

Chapter 5: Dynamic Programming

Section 5.1: Shortest Paths in Acyclic Graphs

Let $G = (V, E)$ be a directed acyclic graph, where each edge (u, v) has a weight $wt(u, v) > 0$.

Topological sorting: vertices are numbered $v_1, v_2, \ldots, v_n$ such that for each edge (v_i, v_j), we have $i < j$.

Let $s = v_1$ and $t = v_n$.

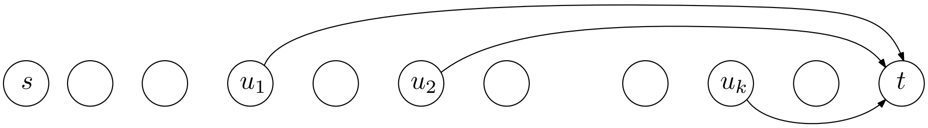
How do we compute the shortest path from s to t?

Can we do better than Dijkstra algorithm, using the topological ordering of G?

Step 1: Structure of the Optimal Solution

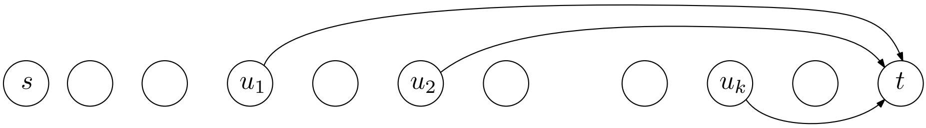
Step 1: Structure of the Optimal Solution

Let $u_1, u_2, \ldots, u_k$ be all the vertices that have an edge to t.



Step 1: Structure of the Optimal Solution

Let $u_1, u_2, \ldots, u_k$ be all the vertices that have an edge to t.

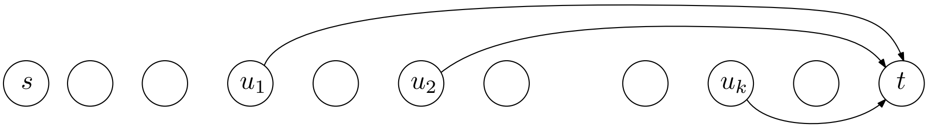


The last edge on the shortest path from s to t is (u_i, t) for some $1 \leq i \leq k$.

If we know this index i, then the shortest path from s to t is equal to

Step 1: Structure of the Optimal Solution

Let $u_1, u_2, \ldots, u_k$ be all the vertices that have an edge to t.



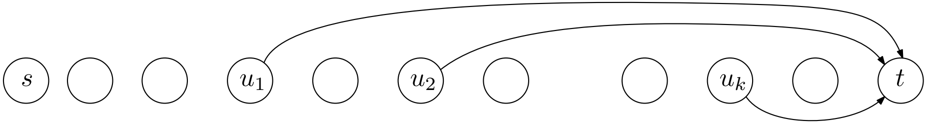
The last edge on the shortest path from s to t is (u_i, t) for some $1 \leq i \leq k$.

If we know this index i, then the shortest path from s to t is equal to

$$\text{path from } s \text{ to } u_i \text{ followed by the edge } (u_i, t)$$

Step 1: Structure of the Optimal Solution

Let $u_1, u_2, \ldots, u_k$ be all the vertices that have an edge to t.

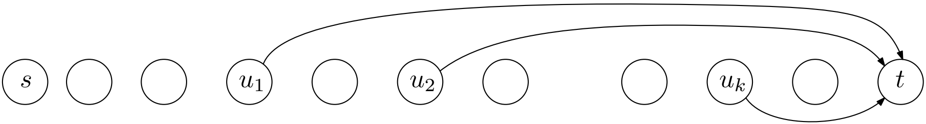


The last edge on the shortest path from s to t is (u_i, t) for some $1 \leq i \leq k$.

If we know this index i, then the shortest path from s to t is equal to

$$\underbrace{\textbf{path from } s \textbf{ to } u_i}_{\substack{\text{this must be the} \\ \text{shortest path from} \\ s \text{ to } u_i}} \text{ followed by the edge } (u_i, t)$$

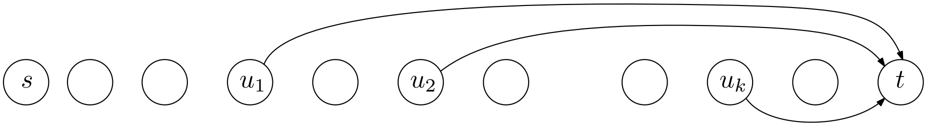
Let $u_1, u_2, \ldots, u_k$ be all the vertices that have an edge to t.



The last edge on the shortest path from s to t is (u_i, t) for some $1 \leq i \leq k$.

But we do not know the index i !

Let $u_1, u_2, \ldots, u_k$ be all the vertices that have an edge to t.



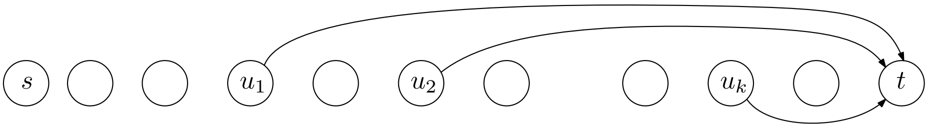
The last edge on the shortest path from s to t is (u_i, t) for some $1 \leq i \leq k$.

But we do not know the index i !

So the length of the shortest path from s to t is equal to

$$\min_{1 \leq i \leq k} \{(\text{length of the shortest path from } s \text{ to } u_i) + wt(u_i, t)\}$$

Let $u_1, u_2, \ldots, u_k$ be all the vertices that have an edge to t.



The last edge on the shortest path from s to t is (u_i, t) for some $1 \leq i \leq k$.

But we do not know the index i !

So the length of the shortest path from s to t is equal to

$$\min_{1 \leq i \leq k} \{(\text{length of the shortest path from } s \text{ to } u_i) + wt(u_i, t)\}$$

In other words, the shortest path from s to t contains the shortest path from s to one of $u_1, u_2, \ldots, u_k$.

Step 2: Set Up a Recurrence for the Optimal Solution

Step 2: Set Up a Recurrence for the Optimal Solution

For $j = 1, 2, ..., n$, define

$$d(v_j) = \text{length of a shortest path from } s \text{ to } v_j$$

Since $v_n = t$, we want to compute $d(v_n)$.

Step 2: Set Up a Recurrence for the Optimal Solution

For $j = 1, 2, ..., n$, define

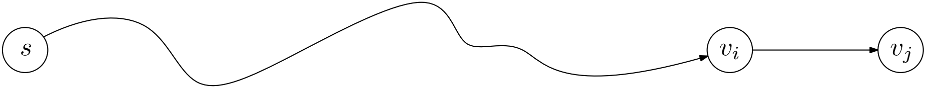
$$d(v_j) = \text{length of a shortest path from } s \text{ to } v_j$$

Since $v_n = t$, we want to compute $d(v_n)$.

Recurrence:

- $d(v_1) = 0$
- For $2 \leq j \leq n$,

$$d(v_j) = \min_{(v_i, v_j) \in E} \{d(v_i) + wt(v_i, v_j)\}$$



Step 3: Solve the Recurrence "Bottom-Up"

Step 3: Solve the Recurrence "Bottom-Up"

First idea: To compute $d(v_n) = d(t)$, take all edges $(u_1, t), (u_2, t), \ldots, (u_k, t)$, and recursively compute $d(u_1), d(u_2), \ldots, d(u_k)$. From this, compute $d(v_n)$ as

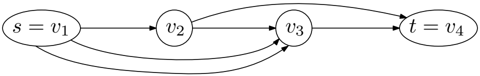
$$d(v_n) = \min_{1 \leq i \leq k} \{d(u_i) + wt(u_i, t)\}.$$

Step 3: Solve the Recurrence "Bottom-Up"

First idea: To compute $d(v_n) = d(t)$, take all edges $(u_1, t), (u_2, t), \ldots, (u_k, t)$, and recursively compute $d(u_1), d(u_2), \ldots, d(u_k)$. From this, compute $d(v_n)$ as

$$d(v_n) = \min_{1 \leq i \leq k} \{d(u_i) + wt(u_i, t)\}.$$

EXAMPLE:

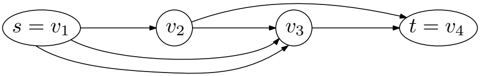


Step 3: Solve the Recurrence "Bottom-Up"

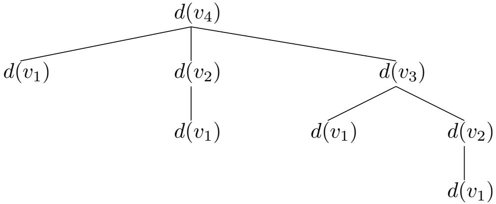
First idea: To compute $d(v_n) = d(t)$, take all edges $(u_1, t), (u_2, t), \ldots, (u_k, t)$, and recursively compute $d(u_1), d(u_2), \ldots, d(u_k)$. From this, compute $d(v_n)$ as

$$d(v_n) = \min_{1 \leq i \leq k} \{d(u_i) + wt(u_i, t)\}.$$

EXAMPLE:



Récursion tree:



$d(v_1)$ is computed 4 times

$d(v_2)$ is computed 2 times

In general, this leads to exponential running time.

$d(v_1)$ is computed 4 times

$d(v_2)$ is computed 2 times

In general, this leads to exponential running time.

Better: compute in this order:

$$d(v_1), d(v_2), ..., d(v_n).$$

When computing $d(v_j)$, we already know $d(v_1), d(v_2), ..., d(v_{j-1})$. From these values, we obtain $d(v_j)$ without recursive calls.

$d(v_1)$ is computed 4 times

$d(v_2)$ is computed 2 times

In general, this leads to exponential running time.

Better: compute in this order:

$$d(v_1), d(v_2), \ldots, d(v_n).$$

When computing $d(v_j)$, we already know $d(v_1), d(v_2), \ldots, d(v_{j-1})$. From these values, we obtain $d(v_j)$ without recursive calls.

Do you remember Fibonacci?

Algorithm:

- $d(v_1) = 0$
- For $j = 2$ to n:
  - $k = indegree(v_j)$
  - Let $u_1, u_2, ..., u_k$ be all the vertices that have an edge to v_j.
  - $d(v_j) = \infty$
  - For $i = 1$ to k
    - $d(v_j) = \min\{d(v_j), d(u_i) + wt(u_i, v_j)\}$
- return $d(v_n)$

Algorithm:

- $d(v_1) = 0$
- For $j = 2$ to n:
  - $k = indegree(v_j)$
  - Let $u_1, u_2, ..., u_k$ be all the vertices that have an edge to v_j.
  - $d(v_j) = \infty$
  - For $i = 1$ to k
    - $d(v_j) = \min\{d(v_j), d(u_i) + wt(u_i, v_j)\}$
- return $d(v_n)$

Running time

$$O\left(\sum_{j=1}^{n} (1 + indegree(v_j))\right) = O(|V| + |E|)$$