§5 Dynamic Programming

§5.1 Shortest path in acyclic graphs

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

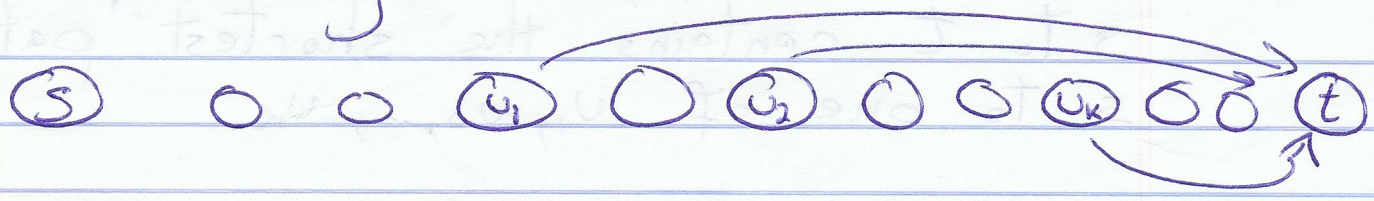
Topological sorting: vertices are numbered $v_1, v_2, \ldots, v_n$ such that for each edge (v_i, v_j) : $i < j$.

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

How to compute the shortest path from s to t?

Step 1: Structure of the optimal solution

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



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

If we know this index i:

shortest path from s to t

= path from s to v_i followed by the edge (v_i, t)

this must be the shortest path from s to v_i

But we do not know the index i.

Shortest path from s to t =

minimum over $i = 1, \ldots, k$ of

shortest path from s to v_i + $wt(v_i, t)$

In other words, the shortest path from s to t contains the shortest path from s to one of $v_1, v_2, \ldots, v_k$.

Step 2: Set up a recurrence for the optimal solution

For $j = 1, 2, \ldots, n$ define

$d(v_j)$ = length of a shortest path from s to v_j.

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

Recurrence:

$$d(v_1) = 0$$

for $2 \le j \le n$:

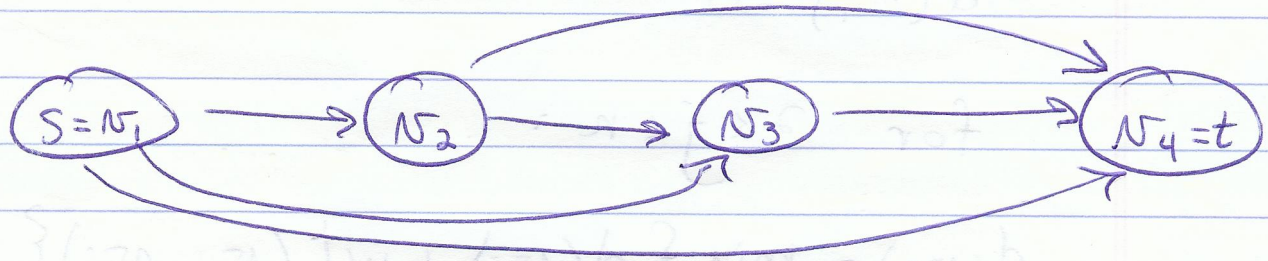
$$d(v_j) = \min_{(v_i, v_j) \in E} \{ d(v_i) + wt(v_i, v_j) \}$$

Step 3 : Solve the recurrence bottom-up

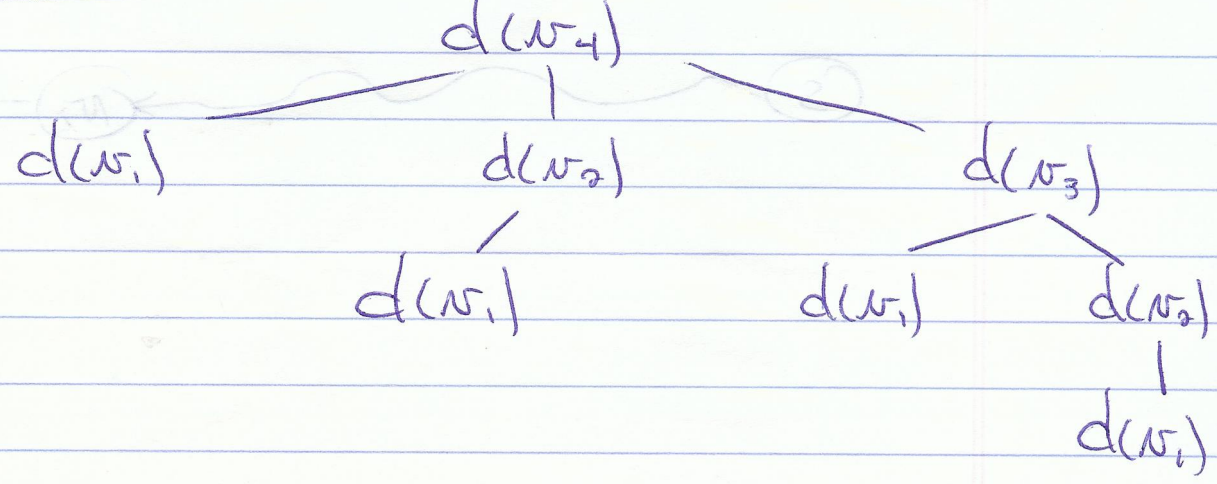
First idea : To compute $d(v_n)$, take all edges $(v_1, t), (v_2, t), \ldots, (v_k, t)$, and recursively compute $d(v_1), d(v_2), \ldots, d(v_k)$. From this, compute $d(v_n)$ as

$$d(v_n) = \min_{1 \le i \le k} \{ d(v_i) + wt(v_i, t) \}$$

Example :



Recursion tree :



$d(v_1)$ is computed 4 times

$d(v_2)$ is computed 2 times

In general, this leads to an exponential running time (see pages 16-18).

Better: compute, in this order, $d(v_1)$, $d(v_2)$, ..., $d(v_n)$.

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

Algorithm:

$d(v_1) = 0$
for $j = 2$ to n

$K =$ indegree(v_j)
let $u_1, \ldots, u_K$ be all vertices that have an edge to v_j.
$d(v_j) = \infty$
for $i = 1$ to K
if $d(u_i) + wt(u_i, v_j) < d(v_j)$
$d(v_j) = d(u_i) + wt(u_i, v_j)$

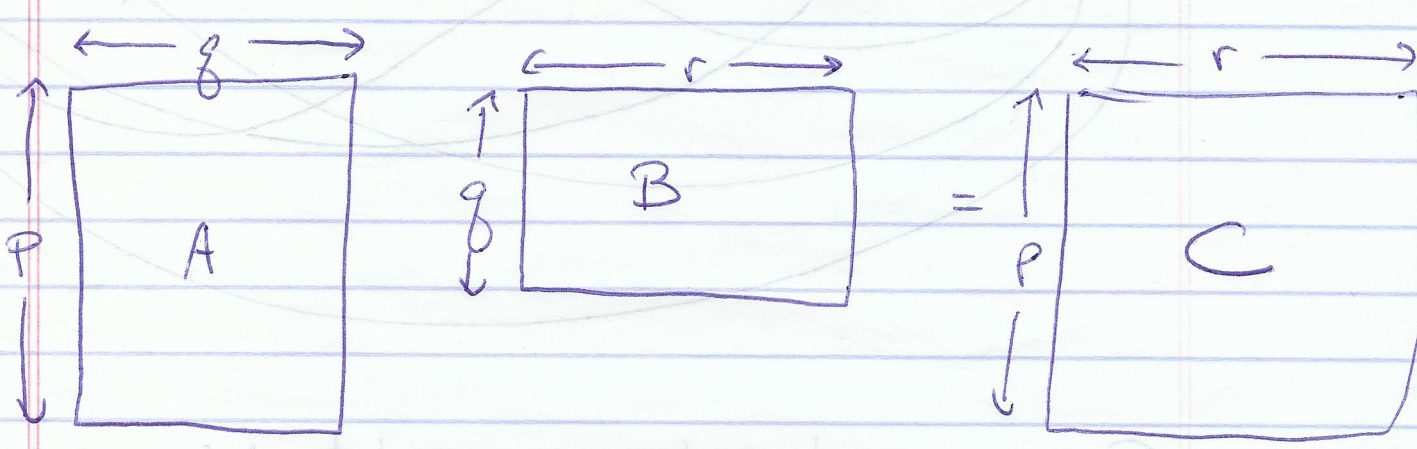
return $d(v_n)$

Running time

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

§5.3 Matrix chain multiplication

A: $p \times q$ matrix
B: $q \times r$ matrix
} $C = AB$ $p \times r$ matrix



C has pr entries, each of which can be computed in $O(q)$ time. So C can be computed in $O(pqr)$ time. We define the cost of multiplying A and B to be pqr.

Consider 3 matrices

A_1 : 10×100

A_2 : 100×5

A_3 : 5×50

How to compute $A_1 A_2 A_3$

① Compute $A_1 A_2$, cost $= 10 \times 100 \times 5 = 5000$

$\oplus$

Compute $\underbrace{(A_1 A_2)}_{10 \times 5} \underbrace{A_3}_{5 \times 50}$, cost $= 10 \times 5 \times 50 = 2500$

7500

② Compute $A_2 A_3$, cost $= 100 \times 5 \times 50 = 25\,000$

$\oplus$

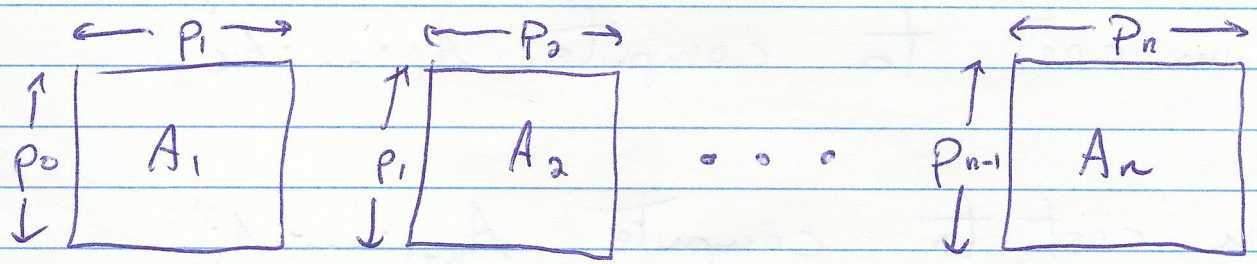
Compute $\underbrace{A_1}_{10 \times 100} \underbrace{(A_2 A_3)}_{100 \times 50}$, cost $= 10 \times 100 \times 50 = 50\,000$

75 000

Given matrices $A_1, A_2, \ldots, A_n$,

positive integers $p_0, p_1, \ldots, p_n$,

matrix A_i has p_{i-1} rows and p_i columns



Compute the best order to compute $A_1 A_2 \cdots A_n$, i.e. minimize the total cost.