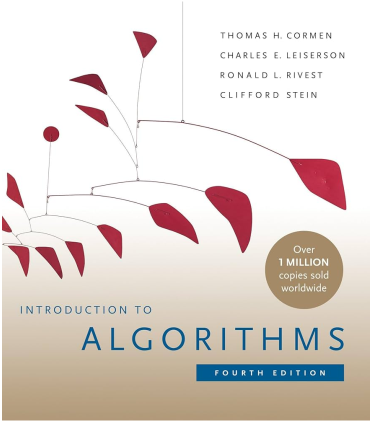
Introduction to Algorithms
Lecture 5: Dynamic Programming (DP)

Prof. Charles E. Leiserson and Prof. Erik Demaine
Massachusetts Institute of Technology

April 10, 2025

Introduction to Algorithms



Content has been extracted from *Introduction to Algorithms*, Fourth Edition, by Cormen, Leiserson, Rivest, and Stein. MIT Press. 2022.
Visit `https://mitpress.mit.edu/9780262046305/introduction-to-algorithms/`.
Original slides from *Introduction to Algorithms 6.046J/18.401J*, Fall 2005 Class by Prof. Charles Leiserson and Prof. Erik Demaine. MIT OpenCourseWare Initiative available at
`https://ocw.mit.edu/courses/6-046j-introduction-to-algorithms-sma-5503-fall-2005/`.

Introduction

Different types of algorithms can be used to solve the all-pairs shortest paths problem:

- Dynamic programming
- Matrix multiplication
- Floyd-Warshall algorithm
- Johnson's algorithm
- Difference constraints

Single-Source Shortest Paths

- Given directed graph $G = (V, E)$, vertex $s \in V$ and edge weights $w : E \to \mathbb{R}$
- Find $\delta(s, v)$, equal to the shortest-path weight $s \to v, \forall v \in V$ (or $-\infty$ if negative weight cycle along the way, or ∞ if no path)

Single-Source Shortest Paths

- Given directed graph $G = (V, E)$, vertex $s \in V$ and edge weights $w : E \to \mathbb{R}$
- Find $\delta(s, v)$, equal to the shortest-path weight $s \to v, \forall v \in V$ (or $-\infty$ if negative weight cycle along the way, or ∞ if no path)

Situtation	**Algorithm**	**Time**
unweigthed ($w = 1$)	BFS	$O(V + E)$
non-negative edge weights	Dijkstra	$O(E + V \lg V)$
general	Bellman-Ford	$O(VE)$
acyclic graph (DAG)	Topological sort + one pass of B-F	$O(V + E)$

All of the above results are the best known. We achieve a $O(E + V \lg V)$ bound on Dijkstra's algorithm using **Fibonacci heaps**.

All-Pairs Shortest Paths (APSP)

- Given edge-weighted graph, $G = (V, E, w)$.
- Find $\delta(u, v)$ for all $u, v \in V$.
- A simple way of solving APSP problems is by running a single-source shortest path algorithm from each of the V vertices in the graph.

All-Pairs Shortest Paths (APSP)

- Given edge-weighted graph, $G = (V, E, w)$.
- Find $\delta(u, v)$ for all $u, v \in V$.
- A simple way of solving APSP problems is by running a single-source shortest path algorithm from each of the V vertices in the graph.

Situtation	**Algorithm**	**Time**	$E = \Theta(V^2)$
unweigthed ($w = 1$)	$\|V\|$+ BFS	$O(VE)$	$O(V^3)$
non-negative edge weights	$\|V\|$+ Dijkstra	$O(VE + V^2 \lg V)$	$O(V^3)$
general	$\|V\|$+ Bellman-Ford	$O(V^2 E)$	$O(V^4)$
general	Johnson's	$O(VE + V^2 \lg V)$	$O(V^3)$

These results (apart from the third) are also best known –don't know how to beat $|V| \times Dijkstra$.

Algorithms to solve APSP*

Dynamic Programming (attempt 1):

1. **Sub-problems**: $d_{uv}^{(m)}$ = weight of shortest path $u \to v$ using $\leq$ edges.
2. **Guessing**: What's the last edge (x, v)?
3. **Recurrence**:

$$d_{uv}^{(m)} = min(d_{uv}^{(m-1)} + w(x, v) \qquad \text{for } x \in V)$$

$$d_{uv}^{(0)} = \begin{cases} 0 & \text{if } u = v. \\ \infty & \text{otherwise.} \end{cases}$$

4. **Topological ordering**: for $m = 0, 1, 2, ..., n-1$: for u and v in V.
5. **Original problem**: If graph contains no negative-weight cycles (by Bellman-Ford analysis), then shortest path is simple
$\Rightarrow \delta(u, v) = d_{uv}^{(n-1)} = d_{uv}^{(n)} = \cdots$

*For all the algorithms described, we assume that $w(u, v) = \infty$ if $(u, v) \in E$

Bottom-up via Relaxation Steps[†]

for $m \leftarrow 1$ **to** n **by** 1
    for u in V
        for v in V
            for x in V
                if $d_{uv} > d_{ux} + d_{xv}$
                    $d_{uv} = d_{ux} + d_{xv}$

[†] In the above pseudocode, we omit superscripts because more relaxation can never hurt.

Time complexity

- In this Dynamic Program, we have $O(V^3)$ total sub-problems.
- Each sub-problem takes $O(V)$ time to solve, since we need to consider V possible choices.
- This gives a total runtime complexity of $O(V^4)$.
- Note that this is no better than $|V|\times$ Bellman-Ford.

Matrix Multiplication

Recall the task of standard matrix multiplication:

- Given $n \times n$ matrices A and B, compute $C = A \cdot B$, such that $c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}$.

Matrix Multiplication

Recall the task of standard matrix multiplication:

- Given $n \times n$ matrices A and B, compute $C = A \cdot B$, such that $c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}$.
  1. $O(n^3)$ using standard algorithm.

Matrix Multiplication

Recall the task of standard matrix multiplication:

- Given $n \times n$ matrices A and B, compute $C = A \cdot B$, such that $c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}$.
  1. $O(n^3)$ using standard algorithm.
  2. $O(n^{2.807})$ using Strassen algorithm.

Matrix Multiplication

Recall the task of standard matrix multiplication:

- Given $n \times n$ matrices A and B, compute $C = A \cdot B$, such that $c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}$.
  1. $O(n^3)$ using standard algorithm.
  2. $O(n^{2.807})$ using Strassen algorithm.
  3. $O(n^{2.376})$ using Coppersmith-Winograd algorithm.

Matrix Multiplication

Recall the task of standard matrix multiplication:

- Given $n \times n$ matrices A and B, compute $C = A \cdot B$, such that $c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}$.
    1. $O(n^3)$ using standard algorithm.
    2. $O(n^{2.807})$ using Strassen algorithm.
    3. $O(n^{2.376})$ using Coppersmith-Winograd algorithm.
    4. $O(n^{2.3728})$ using Vassilevska-Williams algorithm.

Connection to Shortest Paths

- Let's define $\oplus = \min$ and $\odot = +$.
- Then, $C = A \odot B$ produces $c_{ij} = \min_k(a_{ik} + b_{kj})$.
- Define $D^{(m)} = (d_{ij}^{(m)})$, $W = (w(i,j))$, $V = \{1, 2, \ldots, n\}$

With the above definitions, we see that $D^{(m)}$ can be expressed as $D^{(m-1)} \odot W$. In other words, $D^{(m)}$ can be expressed as the circle-multiplication of W with itself m times.

Matrix Multiplication Algorithm

- $n-2$ multiplications $\Rightarrow O(n^4)$ time (still no better).

Matrix Multiplication Algorithm

- $n - 2$ multiplications $\Rightarrow O(n^4)$ time (still no better).
- Repeated squaring: $((W^2)^2)^{2\cdots}$

Matrix Multiplication Algorithm

- $n-2$ multiplications $\Rightarrow O(n^4)$ time (still no better).
- Repeated squaring: $((W^2)^2)^{2\cdots} = W^{2^{\lg n}}$

Matrix Multiplication Algorithm

- $n - 2$ multiplications $\Rightarrow O(n^4)$ time (still no better).
- Repeated squaring: $((W^2)^2)^{2\cdots} = W^{2^{\lg n}} = W^{n-1} = (\delta(i,j))$ if no negative-weight cycles.

Matrix Multiplication Algorithm

- $n-2$ multiplications $\Rightarrow O(n^4)$ time (still no better).
- Repeated squaring: $((W^2)^2)^{2\cdots} = W^{2^{\lg n}} = W^{n-1} = (\delta(i,j))$ if no negative-weight cycles.
- Time complexity of this algorithm is now $O(n^3 \lg n)$.

Floyd-Warshall Algorithms

Dynamic Programming (attempt 2):

1. **Sub-problems**: $c_{uv}^{(k)}$ = weight of shortest path $u \to v$ whose intermediate vertices $\in \{1, 2, \ldots, k\}$

Floyd-Warshall Algorithms

Dynamic Programming (attempt 2):

1. **Sub-problems**: $c_{uv}^{(k)}$ = weight of shortest path $u \to v$ whose intermediate vertices $\in \{1, 2, \ldots, k\}$
2. **Guessing**:

Floyd-Warshall Algorithms

Dynamic Programming (attempt 2):

1. **Sub-problems**: $c_{uv}^{(k)}$ = weight of shortest path $u \rightarrow v$ whose intermediate vertices $\in \{1, 2, \ldots, k\}$
2. **Guessing**: Does shortest path use vertex k?

Floyd-Warshall Algorithms

Dynamic Programming (attempt 2):

1. **Sub-problems**: $c_{uv}^{(k)}$ = weight of shortest path $u \to v$ whose intermediate vertices $\in \{1, 2, \ldots, k\}$
2. **Guessing**: Does shortest path use vertex k?
3. **Recurrence**:

$$c_{uv}^{(0)} = w(u, v)$$
$$c_{uv}^{(k)} = \min(c_{uv}^{(k-1)}, c_{ux}^{(k-1)} + c_{xv}^{(k-1)})$$

Floyd-Warshall Algorithms

Dynamic Programming (attempt 2):

1. **Sub-problems**: $c_{uv}^{(k)}$ = weight of shortest path $u \to v$ whose intermediate vertices $\in \{1, 2, \ldots, k\}$
2. **Guessing**: Does shortest path use vertex k?
3. **Recurrence**:

$$c_{uv}^{(0)} = w(u, v)$$
$$c_{uv}^{(k)} = \min(c_{uv}^{(k-1)}, c_{ux}^{(k-1)} + c_{xv}^{(k-1)})$$

4. **Topological order**: for k: for u and v in V:
5. **Original problem**: $\delta(u, v) = c_{uv}^{(n)}$. Negative weight cycle $\Leftrightarrow$ negative $c_{uu}^{(n)}$

Time Complexity

This Dynamic Program contains $O(V^3)$ problems as well. However, in this case, it takes only $O(1)$ time to solve each sub-problem, which means that the total runtime of this algorithm is $O(V^3)$.

Bottom-up via Relaxation

$C = (w(u, v))$
for $k \leftarrow 1$ **to** n
    for u in V
        for v in V
            if $c_{uv} > c_{uk} + c_{kv}$
                $c_{uv} = c_{uk} + c_{kv}$

Johnson's algorithm

1. Find function $h : V \to \mathbb{R}$ such that $w_h(u, v) = w(u, v) + h(u) - h(v) \geq 0$ for all $u, v \in V$ or determine that a negative-weight cycle exists.
2. Run Dijkstra's algorithm on (V, E, w_h) from every source vertex $s \in V \Rightarrow$ get $\delta_h(u, v)$ for all $u, v \in V$.
3. Given $\delta_h(u, v)$, it is easy to compute $\delta(u, v)$.

Proof

Claim. $\delta(u, v) = \delta_h(u, v) - h(u) + h(v)$.

Proof. Look at any $u \to v$ path p in the graph G:

- Say p is $v_0 \to v_1 \to v_2 \to \cdots \to v_k$, where $v_0 = u$ and $v_k = v$.

$$\begin{aligned} w_h(p) &= \sum_{i=1}^{k} w_h(v_{i-1}, v_i) \\ &= \sum_{i=1}^{k} [w(v_{i-1}, v_i) + h(v_{i-1}) - h(v_i)] \\ &= \sum_{i=1}^{k} w(v_{i-1}, v_i) + h(v_0) - h(v_k) \\ &= w(p) + h(u) - h(v) \end{aligned}$$

- Hence all $u \to v$ paths change in weight by the same offset $h(u) - h(v)$, which implies that the shortest path is preserved.

∎

How to find h?

We know that

$$w_h(u,v) = w(u,v) + h(u) - h(v) \geq 0$$

This is equivalent to,

$$h(v) - h(u) \leq w(u,v)$$

for all $(u,v) \in V$. This is called a **system of difference constraints**.

Theorem.

If (V,E,w) has a negative-weight cycle, then there exists no solution to the above system of difference constraints.

Proof

Say $v_0 \to v_1 \to \cdots \to v_k \to v_0$ is a negative weight cycle.
Let us assume to the contrary that the system of difference constraints has a solution; let's call it h.
This gives us the following system of equations,

$$\begin{aligned} h(v_1) - h(v_0) &\leq w(v_0, v_1) \\ h(v_2) - h(v_1) &\leq w(v_1, v_2) \\ h(v_3) - h(v_2) &\leq w(v_2, v_3) \\ &\vdots \\ h(v_k) - h(v_{k-1}) &\leq w(v_{k-1}, v_k) \\ h(v_0) - h(v_k) &\leq w(v_k, v_0) \end{aligned}$$

Summing all these equations gives us

$$0 \leq w(cycle) < 0$$

which is obviously not possible.
From this, we can conclude that no solution to the above system of difference constraints exists if the graph (V, E, w) has a negative weight cycle.
∎

Theorem.

If (V, E, w) has no negative-weight cycle, then we can find a solution to the difference constraints.

Proof. Add a new vertex s to G, and add edges (s, v) of weight 0 for all $v \in V$.

- Clearly, these new edges do not introduce any new negative weight cycles to the graph.
- Adding these new edges ensures that there now exists at least one path from s to v. This implies that $\delta(s, v)$ is finite for all $v \in V$
- We now claim that $h(v) = \delta(s, v)$. This is obvious from the triangle inequality:
  $\delta(s, u)+w(u, v) \geq \delta(s, v) \Leftrightarrow \delta(s, v)-\delta(s, u) \leq w(u, v) \Leftrightarrow h(v)-h(u) \leq w(u, v)$.

∎

Time Complexity

1. The first step involves running Bellman-Ford from s, which takes $O(VE)$ time. We also pay a pre-processing cost to reweight all the edges ($O(E)$).

Time Complexity

1. The first step involves running Bellman-Ford from s, which takes $O(VE)$ time. We also pay a pre-processing cost to reweight all the edges ($O(E)$).
2. We then run Dijkstra's algorithm from each of the V vertices in the graph; the total time complexity of this step is $O(VE + V^2 \lg V)$.

Time Complexity

1. The first step involves running Bellman-Ford from s, which takes $O(VE)$ time. We also pay a pre-processing cost to reweight all the edges ($O(E)$).
2. We then run Dijkstra's algorithm from each of the V vertices in the graph; the total time complexity of this step is $O(VE + V^2 \lg V)$.
3. We then need to reweight the shortest paths for each pair; this takes $O(V^2)$ time.

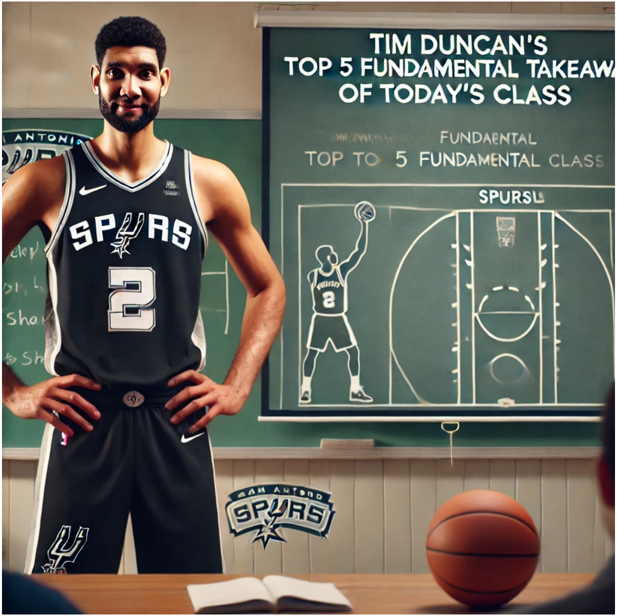
Time Complexity

1. The first step involves running Bellman-Ford from s, which takes $O(VE)$ time. We also pay a pre-processing cost to reweight all the edges ($O(E)$).
2. We then run Dijkstra's algorithm from each of the V vertices in the graph; the total time complexity of this step is $O(VE + V^2 \lg V)$.
3. We then need to reweight the shortest paths for each pair; this takes $O(V^2)$ time.

The total running time of this algorithm is $O(VE + V^2 \lg V)$.

End of Lecture 5.

TDT5FTOTTC

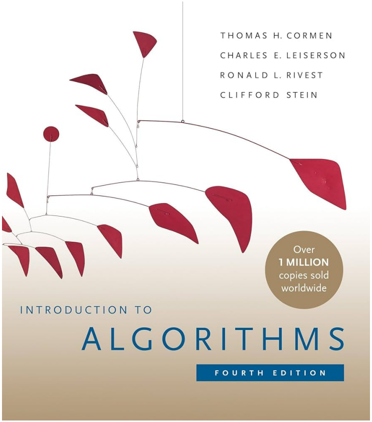

TIM DUNCAN'S
TOP 5 FUNDAMENTAL TAKEAW
OF TODAY'S CLASS
FUNDAENTAL
TOP TO 5 FUNDAMENTAL CLASS
SPURSL
SPURS
2

Top 5 Fundamental Takeaways

Top 5 Fundamental Takeaways

5

Introduction to Algorithms



Content has been extracted from *Introduction to Algorithms*, Fourth Edition, by Cormen, Leiserson, Rivest, and Stein. MIT Press. 2022.
Visit `https://mitpress.mit.edu/9780262046305/introduction-to-algorithms/`.
Original slides from *Introduction to Algorithms 6.046J/18.401J*, Fall 2005 Class by Prof. Charles Leiserson and Prof. Erik Demaine. MIT OpenCourseWare Initiative available at
`https://ocw.mit.edu/courses/6-046j-introduction-to-algorithms-sma-5503-fall-2005/`.