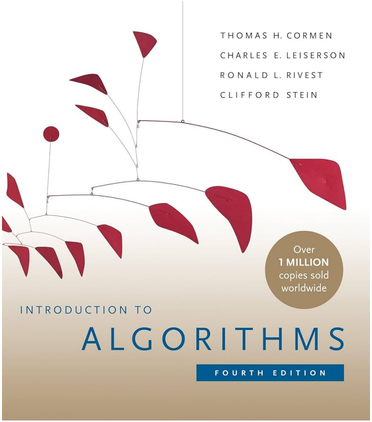
Introduction to Algorithms
Lecture 5: Dynamic Programming (DP)

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

April 8, 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 ($w = 1$)	$\lvert V \rvert$+ BFS	$O(VE)$	$O(V^3)$
non-negative edge weights	$\lvert V \rvert$+ Dijkstra	$O(VE + V^2 \lg V)$	$O(V^3)$
general	$\lvert V \rvert$+ 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 ← 1 to n by 1
    for u in V
        for v in V
            for x in V
                if d_uv > d_ux + d_xv
                    if 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):

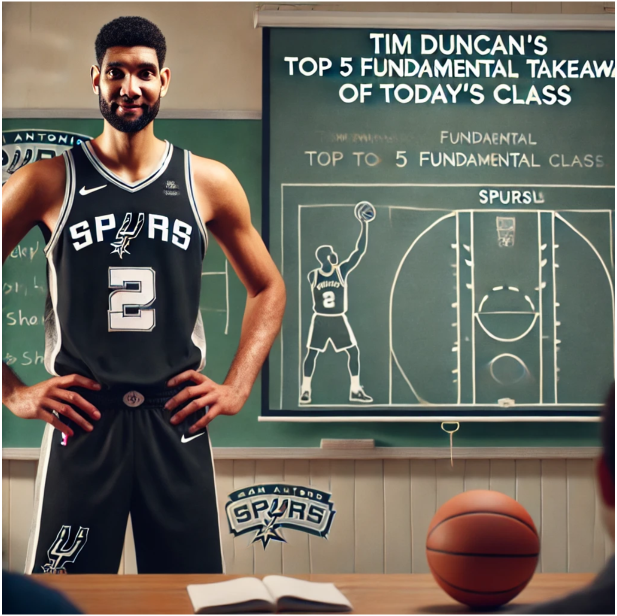
1. **Sub-problems**: $c_{uv}^{(k)}$ = weight of shortest path $u \to v$ whose intermediate vertices $\in \{1, 2, \ldots, k\}$
2. **Guessing**: Does shortest path use vertex k?
3. **Recurrence**:

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

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 $c_{uu}^{(n)}$

End of Lecture 5.

TDT5FTOTTC



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

Top 5 Fundamental Takeaways

Top 5 Fundamental Takeaways

5. Computing Fibonacci numbers or finding the longest palindromic subsequence becomes efficient when subproblem results are cached to prevent repeated computation.

Top 5 Fundamental Takeaways

5 Computing Fibonacci numbers or finding the longest palindromic subsequence becomes efficient when subproblem results are cached to prevent repeated computation.

4 The LCS problem uses a table-based approach to find the length and content of the longest common subsequence in quadratic time.

Top 5 Fundamental Takeaways

5 Computing Fibonacci numbers or finding the longest palindromic subsequence becomes efficient when subproblem results are cached to prevent repeated computation.

4 The LCS problem uses a table-based approach to find the length and content of the longest common subsequence in quadratic time.

3 DP Has Four Key Steps: identifying the structure, defining recurrence, computing solutions bottom-up, and reconstructing the optimal result.

Top 5 Fundamental Takeaways

5 Computing Fibonacci numbers or finding the longest palindromic subsequence becomes efficient when subproblem results are cached to prevent repeated computation.

4 The LCS problem uses a table-based approach to find the length and content of the longest common subsequence in quadratic time.

3 DP Has Four Key Steps: identifying the structure, defining recurrence, computing solutions bottom-up, and reconstructing the optimal result.

2 DP optimizes problems that have **overlapping subproblems** and **optimal substructure**.

Top 5 Fundamental Takeaways

5 Computing Fibonacci numbers or finding the longest palindromic subsequence becomes efficient when subproblem results are cached to prevent repeated computation.

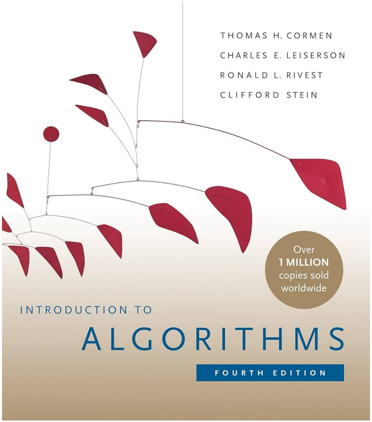
4 The LCS problem uses a table-based approach to find the length and content of the longest common subsequence in quadratic time.

3 DP Has Four Key Steps: identifying the structure, defining recurrence, computing solutions bottom-up, and reconstructing the optimal result.

2 DP optimizes problems that have **overlapping subproblems** and **optimal substructure**.

1 Dynamic Programming = recursion + memoization.

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/`.