

COL 351 : ANALYSIS & DESIGN OF ALGORITHMS

## LECTURE 21

INTRODUCTION TO DYNAMIC PROGRAMMING  
AND  
MID-TERM REVIEW

SEPT 11, 2024

|

ROHIT VAISH

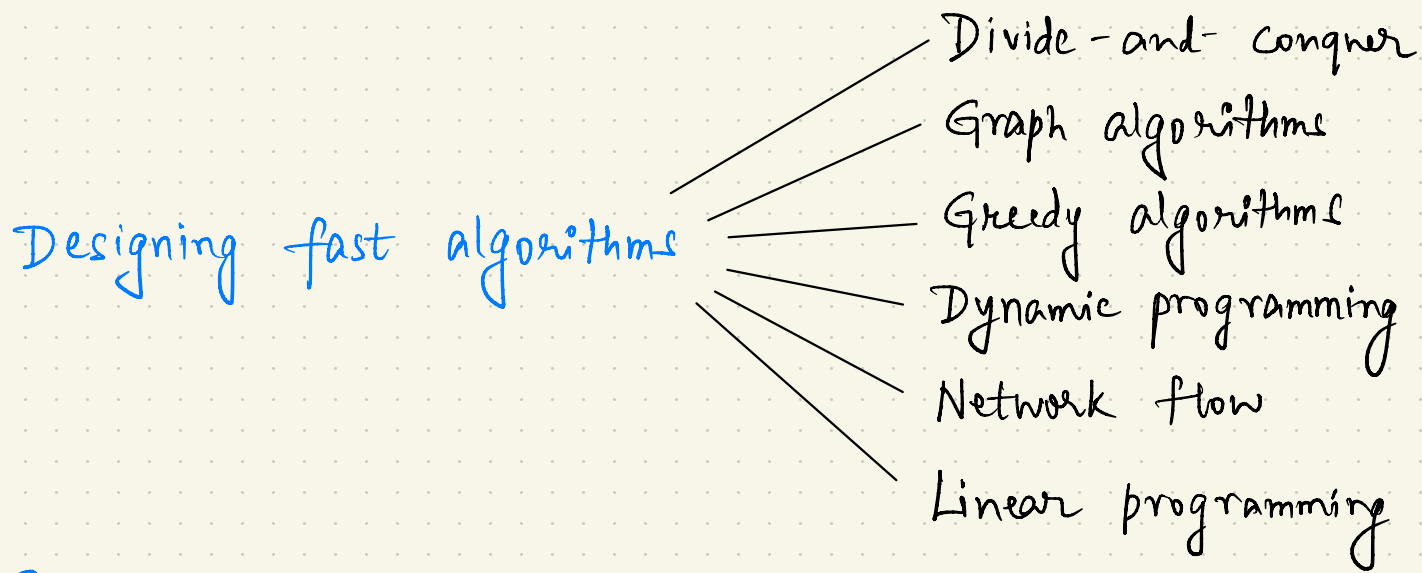
STORY SO FAR

# STORY SO FAR



A recipe for solving some computational problem.

Designing fast algorithms



Proving that a fast algorithm probably doesn't exist

NP-completeness

Coping with NP-completeness

Approximation

Randomization

Parameterization

Designing fast algorithms

Divide-and-conquer

Graph algorithms

Greedy algorithms

Dynamic programming

Network flow

Linear programming

} so far

Proving that a fast algorithm probably doesn't exist

NP-completeness

Coping with NP-completeness

Approximation

Randomization

Parameterization

Designing fast algorithms

Divide-and-conquer

Graph algorithms

Greedy algorithms

Dynamic programming → today

Network flow

Linear programming

} so far

Proving that a fast algorithm probably doesn't exist

NP-completeness

Coping with NP-completeness

Approximation

Randomization

Parameterization

The algorithm design space is surprisingly rich!

DIVIDE & CONQUER



# DIVIDE & CONQUER

Integer multiplication : Grade school v/s Karatsuba

Matrix multiplication : Grade school v/s Strassen

# DIVIDE & CONQUER

Integer multiplication : Grade school v/s Karatsuba

Matrix multiplication : Grade school v/s Strassen

Merge sort : Recursion tree analysis

# DIVIDE & CONQUER

Integer multiplication : Grade school v/s Karatsuba

Matrix multiplication : Grade school v/s Strassen

Merge sort : Recursion tree analysis

Counting inversions : Piggyback on merge sort

# DIVIDE & CONQUER

Integer multiplication : Grade school v/s Karatsuba

Matrix multiplication : Grade school v/s Strassen

Merge sort : Recursion tree analysis

Counting inversions : Piggyback on merge sort

Binary search : Halving the search space

# DIVIDE & CONQUER

Integer multiplication : Grade school v/s Karatsuba

Matrix multiplication : Grade school v/s Strassen

Merge sort : Recursion tree analysis

Counting inversions : Piggyback on merge sort

Binary search : Halving the search space

Master theorem : General tool for analyzing recurrences

# GRAPH ALGORITHMS

# GRAPH ALGORITHMS

Applications: Road networks, social networks, Web, ...

# GRAPH ALGORITHMS

Applications : Road networks, social networks, Web, ...

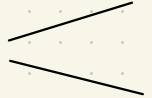
Representation : adjacency list, adjacency matrix.



# GRAPH ALGORITHMS

Applications : Road networks, social networks, Web, ...

Representation : adjacency list, adjacency matrix.

BFS  shortest paths  
connected components

# GRAPH ALGORITHMS

Applications: Road networks, social networks, Web, ...

Representation: adjacency list, adjacency matrix.

BFS

- shortest paths
- connected components

DFS

- topological ordering
- strongly connected components & Kosaraju's algo.

# GRAPH ALGORITHMS

Applications: Road networks, social networks, Web, ...

Representation: adjacency list, adjacency matrix.

BFS

- shortest paths
- connected components

Linear time!

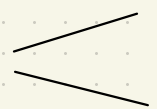
DFS

- topological ordering
- strongly connected components & Kosaraju's algo.

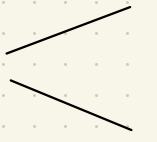
# GRAPH ALGORITHMS

Applications: Road networks, social networks, Web, ...

Representation: adjacency list, adjacency matrix.

BFS  shortest paths  
connected components

Linear time!

DFS  topological ordering  
strongly connected components & Kosaraju's algo.

Dijkstra's algorithm: single-source shortest paths in weighted directed graphs (+ heaps)

# GREEDY ALGORITHMS

# GREEDY ALGORITHMS

Correctness analysis is non-trivial

- exchange argument
- induction

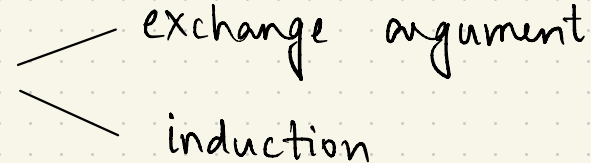
# GREEDY ALGORITHMS

Correctness analysis is non-trivial

- exchange argument
- induction

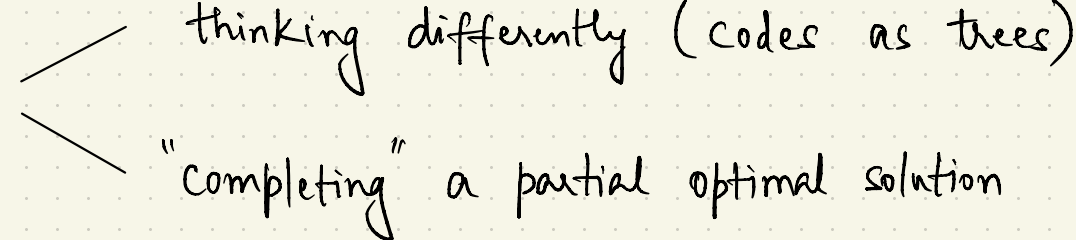
Job scheduling : using special cases to build general solution

# GREEDY ALGORITHMS

Correctness analysis is non-trivial 

- exchange argument
- induction


Job scheduling : using special cases to build general solution

Huffman coding 

- thinking differently (codes as trees)
- "completing" a partial optimal solution

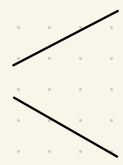


# GREEDY ALGORITHMS


Correctness analysis is non-trivial 

- exchange argument
- induction

Job scheduling : using special cases to build general solution

Huffman coding 

- thinking differently (codes as trees)
- "completing" a partial optimal solution

Minimum spanning trees 

- Prim's and Kruskal's algorithms
- Empty cuts, lonely edges, Cut property

# DYNAMIC PROGRAMMING

I spent the Fall quarter (of 1950) at RAND. My first task was to find a name for multistage decision processes. An interesting question is, "Where did the name, dynamic programming, come from?" The 1950s were not good years for mathematical research. We had a very interesting gentleman in Washington named Wilson. He was Secretary of Defense, and he actually had a pathological fear and hatred of the word "research". I'm not using the term lightly; I'm using it precisely. His face would suffuse, he would turn red, and he would get violent if people used the term research in his presence. You can imagine how he felt, then, about the term mathematical. The RAND Corporation was employed by the Air Force, and the Air Force had Wilson as its boss, essentially. Hence, I felt I had to do something to shield Wilson and the Air Force from the fact that I was really doing mathematics inside the RAND Corporation. What title, what name, could I choose? In the first place I was interested in planning, in decision making, in thinking. But planning, is not a good word for various reasons. I decided therefore to use the word "programming". I wanted to get across the idea that this was dynamic, this was multistage, this was time-varying. I thought, let's kill two birds with one stone. Let's take a word that has an absolutely precise meaning, namely dynamic, in the classical physical sense. It also has a very interesting property as an adjective, and that is it's impossible to use the word dynamic in a pejorative sense. Try thinking of some combination that will possibly give it a pejorative meaning. It's impossible. Thus, I thought dynamic programming was a good name. It was something not even a Congressman could object to. So I used it as an umbrella for my activities.

— Richard Bellman, *Eye of the Hurricane: An Autobiography* (1984, page 159)

I spent the Fall quarter (of 1950) at RAND. My first task was to find a name for multistage decision processes. An interesting question is, "Where did the name, dynamic programming, come from?" The 1950s were not good years for mathematical research. We had a very interesting gentleman in Washington named Wilson. He was Secretary of Defense, and he actually had a pathological fear and hatred of the word "research". I'm not using the term lightly; I'm using it precisely. His face would suffuse, he would turn red, and he would get violent if people used the term research in his presence. You can imagine how he felt, then, about the term mathematical. The RAND Corporation was employed by the Air Force, and the Air Force had Wilson as its boss, essentially. Hence, I felt I had to do something to shield Wilson and the Air Force from the fact that I was really doing mathematics inside the RAND Corporation. What title, what name, could I choose? In the first place I was interested in planning, in decision making, in thinking. But planning, is not a good word for various reasons. I decided therefore to use the word "programming". I wanted to get across the idea that this was dynamic, this was multistage, this was time-varying. I thought, let's kill two birds with one stone. Let's take a word that has an absolutely precise meaning, namely dynamic, in the classical physical sense. It also has a very interesting property as an adjective, and that is it's impossible to use the word dynamic in a pejorative sense. Try thinking of some combination that will possibly give it a pejorative meaning. It's impossible. Thus, I thought dynamic programming was a good name. It was something not even a Congressman could object to. So I used it as an umbrella for my activities.

— Richard Bellman, *Eye of the Hurricane: An Autobiography* (1984, page 159)



# MAX WEIGHT INDEPENDENT SET ON PATHS

# MAX WEIGHT INDEPENDENT SET ON PATHS

input: a path graph  $G = (V, E)$  with nonnegative weights on vertices  $\{w_v\}_{v \in V}$

# MAX WEIGHT INDEPENDENT SET ON PATHS

**input:** a path graph  $G = (V, E)$  with nonnegative weights on vertices  $\{w_v\}_{v \in V}$

**output:** an independent set  $S \subseteq V$  of  $G$  with maximum  $\sum_{v \in S} w_v$

# MAX WEIGHT INDEPENDENT SET ON PATHS

**input:** a path graph  $G = (V, E)$  with nonnegative weights on vertices  $\{w_v\}_{v \in V}$

**output:** an independent set  $S \subseteq V$  of  $G$  with maximum  $\sum_{v \in S} w_v$

subset of non-adjacent vertices

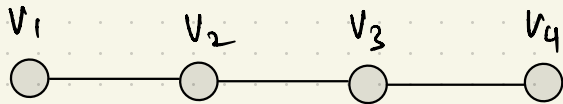


# MAX WEIGHT INDEPENDENT SET ON PATHS

**input:** a path graph  $G = (V, E)$  with nonnegative weights on vertices  $\{w_v\}_{v \in V}$

**output:** an independent set  $S \subseteq V$  of  $G$  with maximum  $\sum_{v \in S} w_v$

subset of non-adjacent vertices

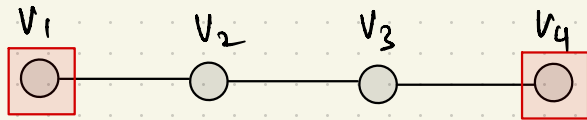


# MAX WEIGHT INDEPENDENT SET ON PATHS

**input:** a path graph  $G = (V, E)$  with nonnegative weights on vertices  $\{w_v\}_{v \in V}$

**output:** an independent set  $S \subseteq V$  of  $G$  with maximum  $\sum_{v \in S} w_v$

subset of non-adjacent vertices

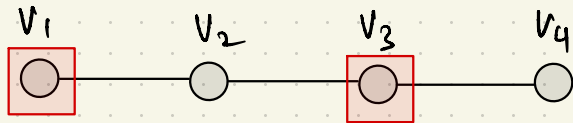


# MAX WEIGHT INDEPENDENT SET ON PATHS

**input:** a path graph  $G = (V, E)$  with nonnegative weights on vertices  $\{w_v\}_{v \in V}$

**output:** an independent set  $S \subseteq V$  of  $G$  with maximum  $\sum_{v \in S} w_v$

subset of non-adjacent vertices

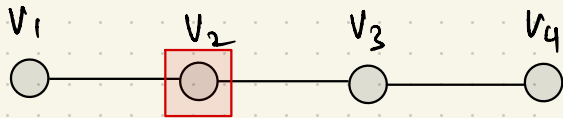


# MAX WEIGHT INDEPENDENT SET ON PATHS

**input:** a path graph  $G = (V, E)$  with nonnegative weights on vertices  $\{w_v\}_{v \in V}$

**output:** an independent set  $S \subseteq V$  of  $G$  with maximum  $\sum_{v \in S} w_v$

subset of non-adjacent vertices



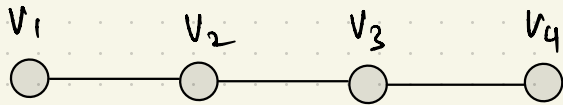
# MAX WEIGHT INDEPENDENT SET ON PATHS

**input:** a path graph  $G = (V, E)$  with nonnegative weights on vertices  $\{w_v\}_{v \in V}$

**output:** an independent set  $S \subseteq V$  of  $G$  with maximum  $\sum_{v \in S} w_v$

subset of non-adjacent vertices

How many independent sets?

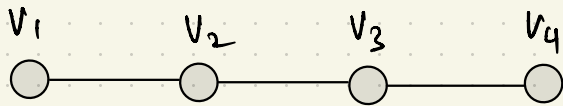


# MAX WEIGHT INDEPENDENT SET ON PATHS

**input:** a path graph  $G = (V, E)$  with nonnegative weights on vertices  $\{w_v\}_{v \in V}$

**output:** an independent set  $S \subseteq V$  of  $G$  with maximum  $\sum_{v \in S} w_v$

subset of non-adjacent vertices



How many independent sets? 8

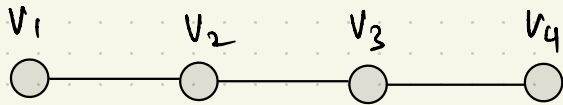
$\emptyset$ , four singletons,  $\{v_1, v_3\}$ ,  $\{v_2, v_4\}$ ,  $\{v_1, v_4\}$

# MAX WEIGHT INDEPENDENT SET ON PATHS

**input:** a path graph  $G = (V, E)$  with nonnegative weights on vertices  $\{w_v\}_{v \in V}$

**output:** an independent set  $S \subseteq V$  of  $G$  with maximum  $\sum_{v \in S} w_v$

subset of non-adjacent vertices



How many independent sets? 8

$\emptyset$ , four singletons,  $\{v_1, v_3\}$ ,  $\{v_2, v_4\}$ ,  $\{v_1, v_4\}$

in general: exponential in  $n$  😞