DYNAMIC PROGRAMMING : MWIS

- Create a recurrence for MWIS on a line [DP1]

Announcements

Exam on Thurs.
- 20 min
- DC1, DC2

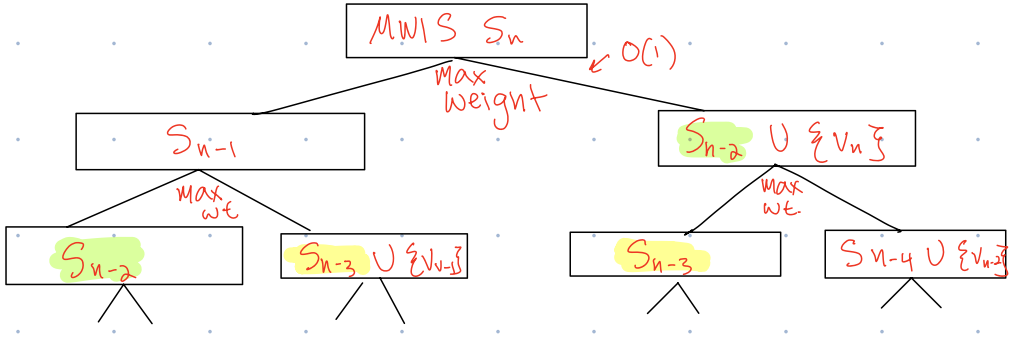
Exit Tickets

Warm Up:

Explain this recurrence:

$$S_n = \begin{cases} S_{n-2} \cup \{v_n\} & \text{if } v_n \in S_n \\ S_{n-1} & \text{if } v_n \notin S_n \end{cases}$$

Why is this recursive alg. bad?



$S_n, S_{n-1}, S_{n-2}, S_{n-3}, S_{n-4} \cdots S_1$

$\Omega(2^{n/2})$

How many unique subproblems are there?

A) $\sqrt{n}$ B) $\frac{n}{2}$ C) n D) n^2

Warm-Up:

decreasing ↓ w_i/t_i — will prove later ↓

The greedy algorithm with $f(i) = w_i/t_i$ is optimal for minimizing $A(\sigma) = \sum_{i=1}^{n} w_i C_i(\sigma)$. Write pseudocode for this greedy scheduling alg + analyze the runtime.

```
for i ← 1 to n:        ← O(n)
    f[i] ← w_i - t_i   ← O(1)      } O(n)
Sort by decreasing f + return  O(nlogn)
```

Runtime $O(n \log n)$

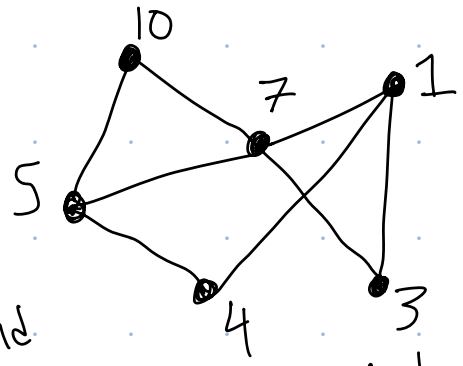
Why do greedy if frequently not optimal? Heuristics

More Practice: website, textbooks

How to create a scoring function? ✓

Why greedy? (Short-sighted?) ✓

Max Weight Independent Set



Input: Graph $G = (V, E)$ (undirected)

Weights $w : V \to \mathbb{Z}^+$

Output: $S \subseteq V$ s.t.

- If $\{u, v\} \in E$ then $\neg(u \in S \wedge v \in S)$ ← Independent Set
  (not ↓, and ↓)
- Maximizes $W(S) = \sum_{v \in S} w(v)$ ← objective function

Applications:

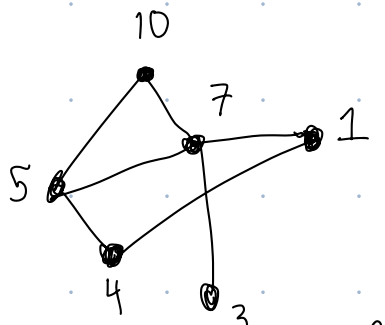
- Cell Tower Transmissions
- Choose franchise location
- Party Invite
- Scheduling
- House robbing (ethical concerns)

General Graph: Hard

Line Graph: Easy if use dynamic programming

Max Weight Independent Set (MWIS)

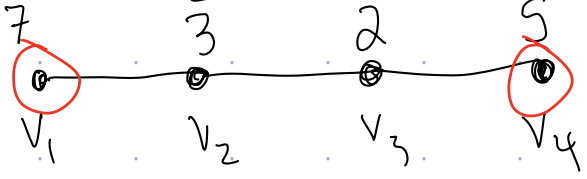
Input: Graph $G = (V, E)$
weights $w: V \to \mathbb{Z}^+$



Output: $S \subseteq V$ s.t.

- If $\{u, v\} \in E$, $\neg(u \in S \wedge v \in S)$ ⇐ "Independent Set Condition" ("Constraint")
  (not ↓, and ↓)
- Maximizes $W(S) = \sum_{v \in S} w(v)$ ⇐ "Weight of S"

What is max weight $W(S)$ for MWIS of this line graph:

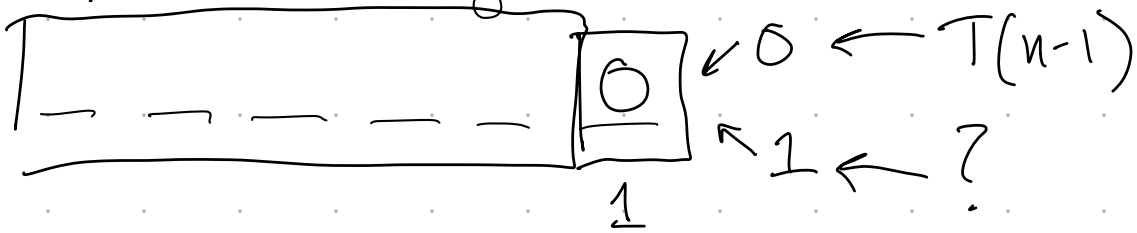


→ $\{v_1, v_4\}$

A) 0 B) 8 C) 9 D) 12

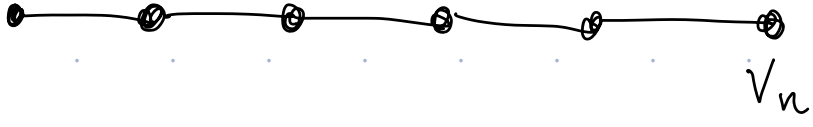
Dynamic Prog. Approach

Recall: # of n bit strings with 2 consecutive ones $\} T(n)$



Needed to identify "final options"

To create a DP alg, (often) need to conceptualize the optimal solution as a sequence of choices

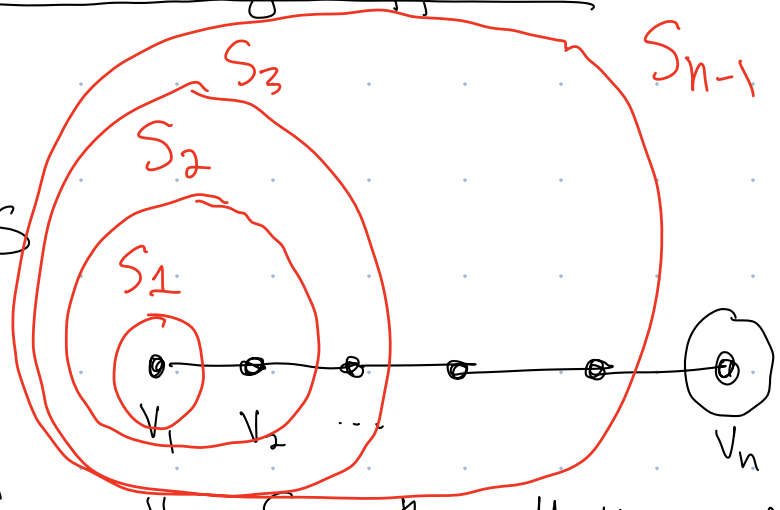


"Final choice"

Dynamic Prog. Approach

[DP1]

MWIS



← 2 cases: $v_n \in S$ or $v_n \notin S$

Let's call S_i the MWIS of first i vertices

② $S_n = \begin{cases} \_\_\_\_\_ & \text{if } v_n \in S_n \\ \_\_\_\_\_ & \text{if } v_n \notin S_n \end{cases}$

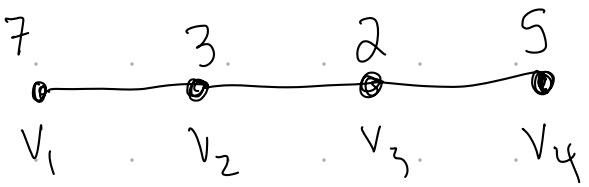
Options: S_{n-1}, S_{n-2}, $S_{n-1} \cup \{v_n\}$, $S_{n-2} \cup \{v_n\}$, $S_{n-2} \cup \{v_{n-1}\}$

① Name, pronouns, best thing watched/listened to

③ Write pseudocode for brute force approach, analyze runtime

④ Brainstorm greedy, divide + conquer approaches

Brute Force



MWIS($G=(V,E)$, w)

```
max S ← ∅
max W ← 0
For each set S ⊆ V:                    O(2^n)
    If S is I.S.:                      O(n)
        w ← W(S)
        if w > max W then
            max S ← S
            max W ← w
Return max S
```

S is I.S.:

```
For i ← 1 to n-1:                      O(n)
    If v_i ∈ S and v_{i+1} ∈ S:
        Return False
Return true
```

W(S):

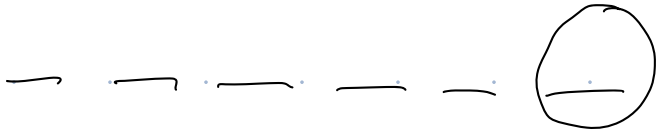
```
W ← 0
For i ← 1 to n                         O(n)
    If v_i ∈ S
        W ← W + w(v_i)
return W
```

000
001
010
011
100
101
110
111

$O(2^n n)$

Dynamic Prog. Approach

Recall: # of n bit strings with 2 consecutive ones

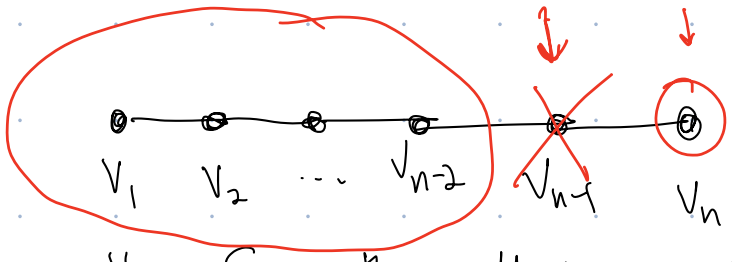


← 2 cases, 0 or 1

↓ ↓

$T(n-1\ldots)$ ..... $T(n-2)\ldots$

MWIS



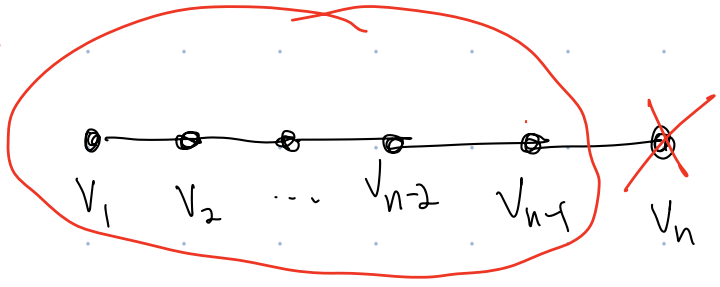
← 2 cases : $v_n \in S$ or $v_n \notin S$

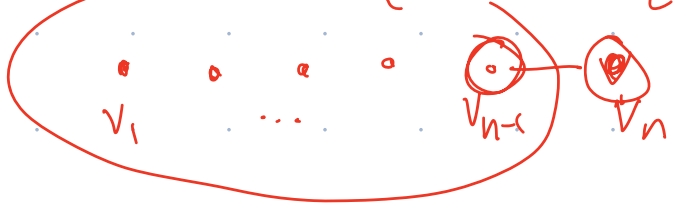
Let's call S_i the MWIS of first i vertices

$$S_n = \begin{cases} S_{n-2} \cup \{v_n\} & \text{if } v_n \in S_n \\ S_{n-1} & \text{if } v_n \notin S_n \end{cases}$$

Options:
S_{n-1}, S_{n-2}, $S_{n-1} \cup \{v_n\}$,
$S_{n-2} \cup \{v_n\}$, $S_{n-2} \cup \{v_{n-1}\}$

Ind. Set



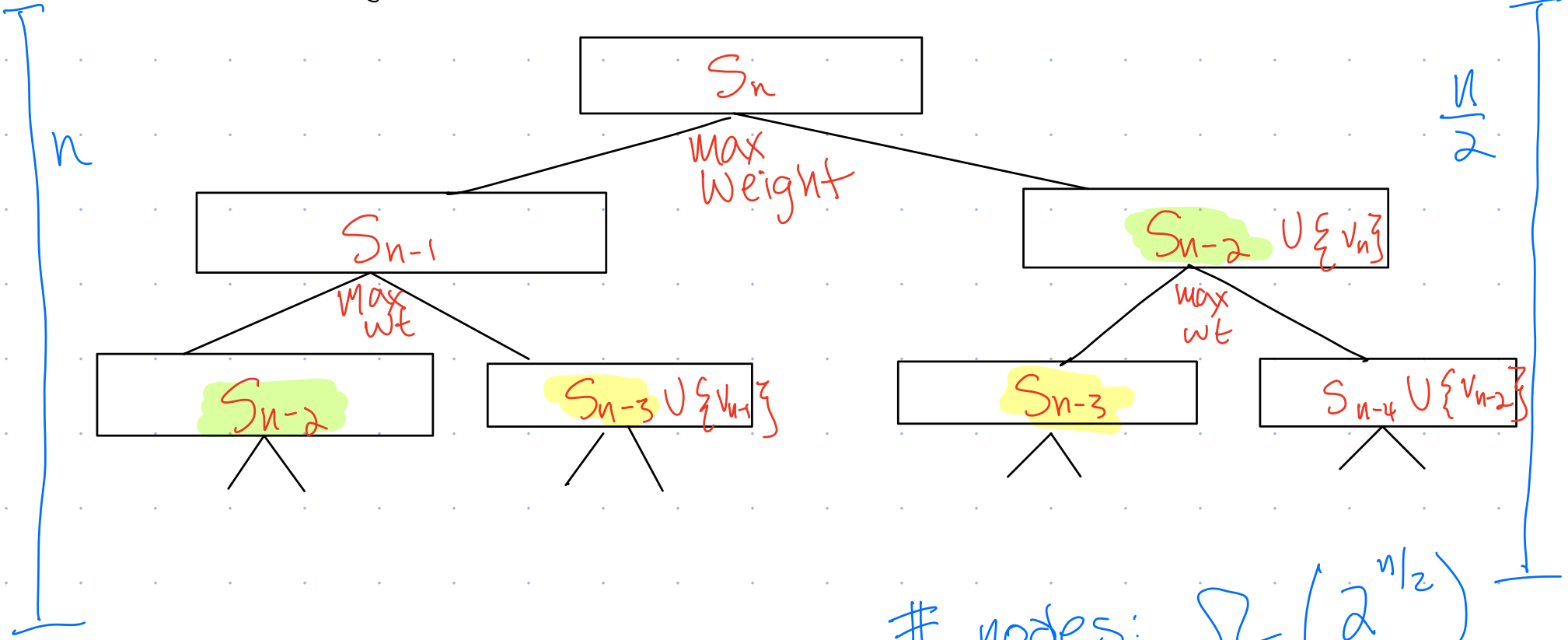


$$S_n = \begin{cases} S_{n-2} \cup \{v_n\} & \text{if } v_n \in S_n \\ S_{n-1} & \text{if } v_n \notin S_n \end{cases}$$

Only 2 possible options, check both + take larger weight set.

* And base case Later...

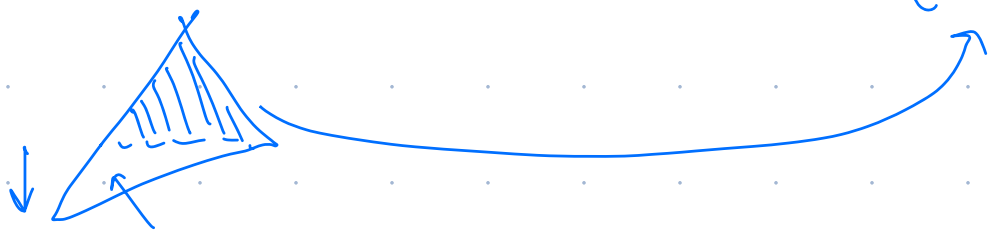
Recursive Algorithm:



n = 10, 8, 6, 4, 2

n = 10, 9, 8, 7, 6

nodes: $\Omega(2^{n/2})$



$$S_n = \begin{cases} \_\_\_\_\_ & \text{if } v_n \in S_n \\ \_\_\_\_\_ & \text{if } v_n \notin S_n \\ \text{(Base case)} \end{cases}$$

Only 2 possible options, check both + take larger weight set.

Recursive Algorithm:

How many unique subproblems are there?

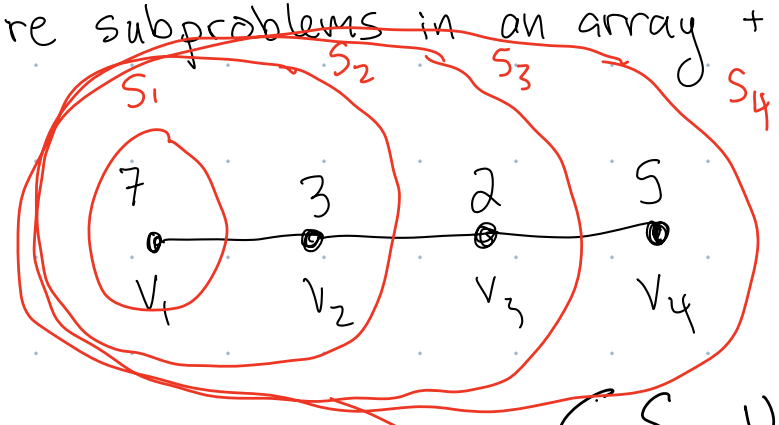
A) $\sqrt{n}$ B) $\frac{n}{2}$ C) n D) n^2

Dynamic Programming Idea: Store subproblems in an array + look up

$$S_n = \begin{cases} S_{n-2} \cup \{v_n\} \\ S_{n-1} \end{cases} \text{ max wt}$$

$$\emptyset \quad \text{if } n=0$$

$$\{v_1\} \quad \text{if } n=1$$



$$S_1 = \max \begin{cases} S_{-1} \cup \{v_1\} \\ S_0 \end{cases}$$

S_0	S_1	S_2	S_3	S_4
$\emptyset$	$\{v_1\}$	$\{v_1\}$	$\{v_1, v_3\}$	$\{v_1, v_4\}$

$$S_4 = \max \begin{cases} S_2 \cup \{v_4\} \\ S_3 \end{cases}$$

Trick 1: Fill array from bottom

Trick 0: Start with empty problem