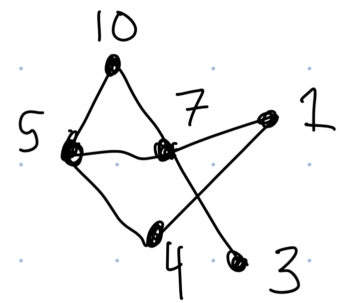
DYNAMIC PROGRAMMING : MWIS

- Create a recurrence for MWIS on a line [DP1]
- Write correct pseudocode for a dynamic prog. alg [DC3]

Announcements

- Boilerplate for NP, Induction
- Might not put full Closest Pts proof, on exam but best way to prepare is to be able to write proof from scratch + do other DC3 as on PSET.
- Resubmissions of Programming Assignments
- Better than 7 pts

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 $\underset{\text{not}}{\neg}(u \in S \underset{\text{and}}{\wedge} v \in S)$ ← "independent set"
- Maximizes $W(S) = \sum_{v \in S} w(v)$ ← "weight of "S""

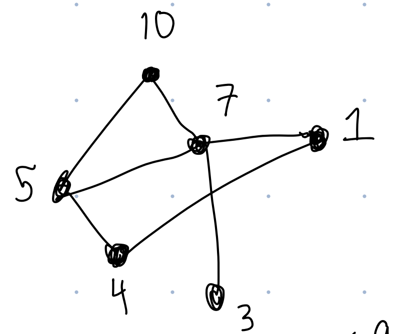
Applications:

- Cell Tower Transmission
- Choosing Franchise Location
- Party Invites
- Scheduling
- House robbing (ethically bad)

General Graph: Hard

Line Graph: Easy, using Dynamic Programming

Max Weight Independent Set (MWIS)



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

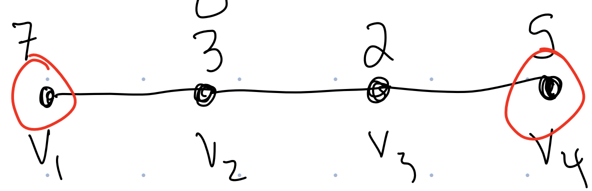
weights $w: V \to \mathbb{Z}^+$

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

- If $\{u,v\} \in E$, $\overset{\text{not}}{\neg}(u \in S \overset{\text{and}}{\wedge} v \in S)$ ⇐ "Independent Set Condition" "constraint"

- Maximizes $W(S) = \sum_{v \in S} w(v)$ ⟸ "Weight of S"

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

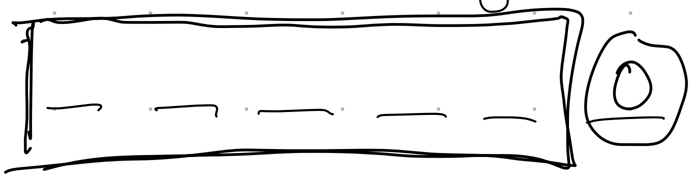


A) 0 B) 8 C) 9 D) 12

Dynamic Prog. Approach

$T(n)$

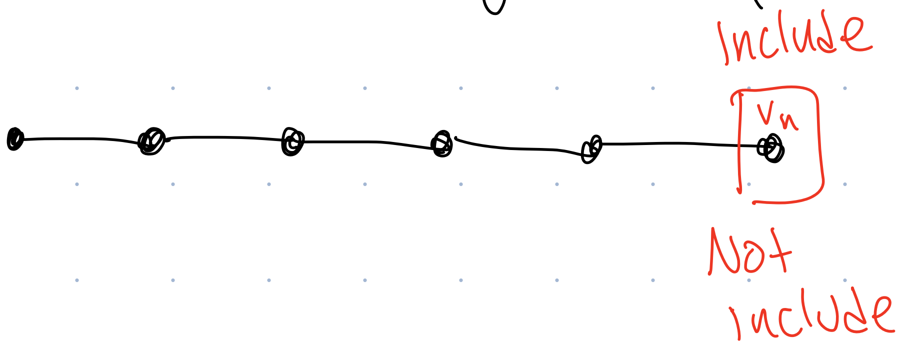
Recall: # of n bit strings with 2 consecutive ones



$$T(n) = \begin{cases} \underline{\qquad} & \text{if ends } 1 \\ \underline{T(n-1)} & \text{if ends } 0 \end{cases}$$

Needed to identify "final options"

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

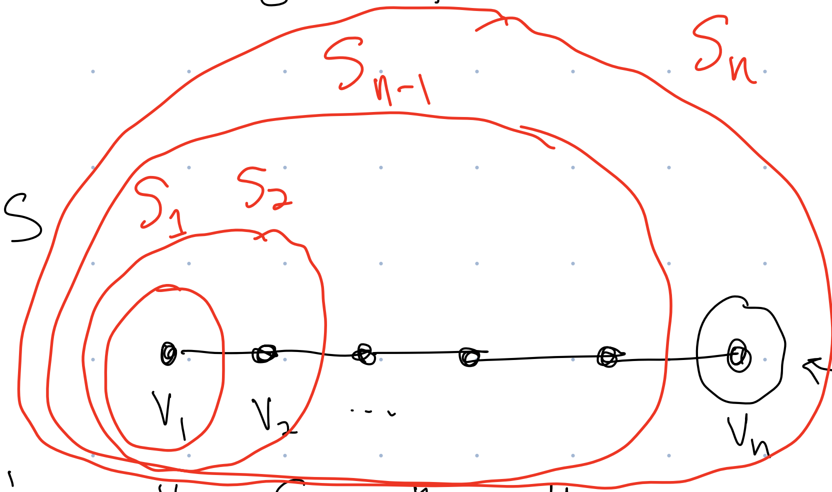


"Final choice"

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 $S_n = \begin{cases} \underline{\hspace{3cm}} & \text{if } v_n \in S_n \\ \underline{\hspace{4cm}} & \text{if } v_n \notin S_n \end{cases}$

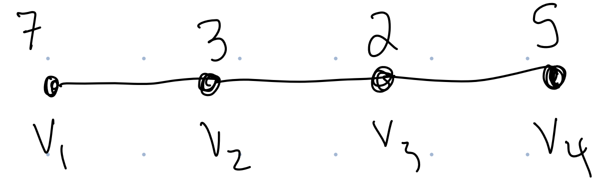
Options: S_{n-1}, S_{n-2}, $S_{n-1} \cup \{v_n\}$, $S_{n-2} \cup \{v_n\}$, $S_{n-2} \cup \{v_{n-1}\}$

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.:
        w ← W(S)
        if w > max W then
            max S ← S
            max W ← w
Return max S
```

O(n)

```
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
```

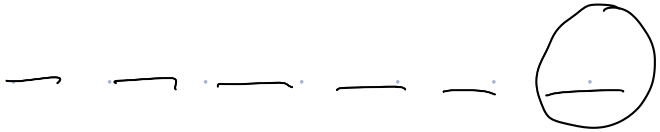
```
W(S):
W ← 0
For i ← 1 to n                 ← O(n)
    If v_i ∈ S
        W ← W + w(v_i)
return W
```

000
001
010
011
100
101
110

$O(2^n \cdot n)$

Dynamic Prog. Approach

Recall: # of n bit strings with 2 consecutive ones

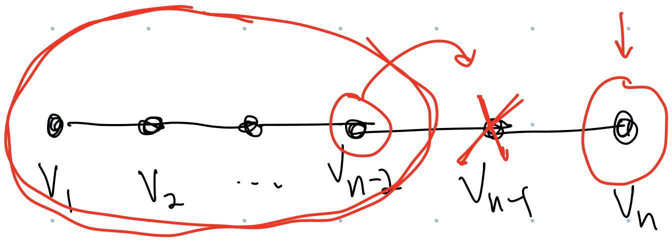


← 2 cases, 0 or 1

0 → $T(n-1 \ldots) \ldots$

1 → $T(n-2) \ldots$

MWIS



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

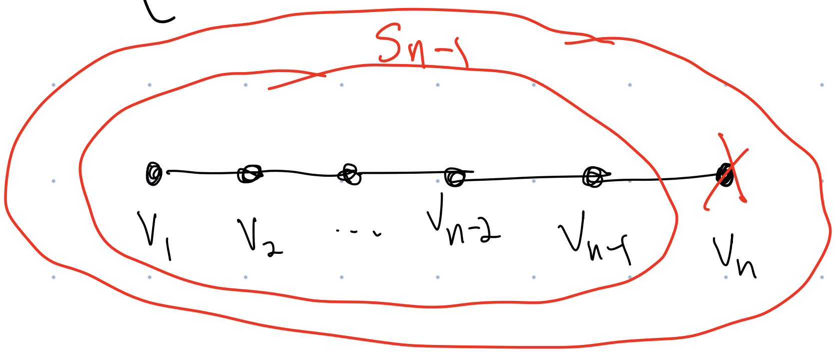
Let's call S_i the MWIS of first i vertices

Options:

S_{n-1}, S_{n-2}, $S_{n-1} \cup \{v_n\}$ ← might not be ind. set

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

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

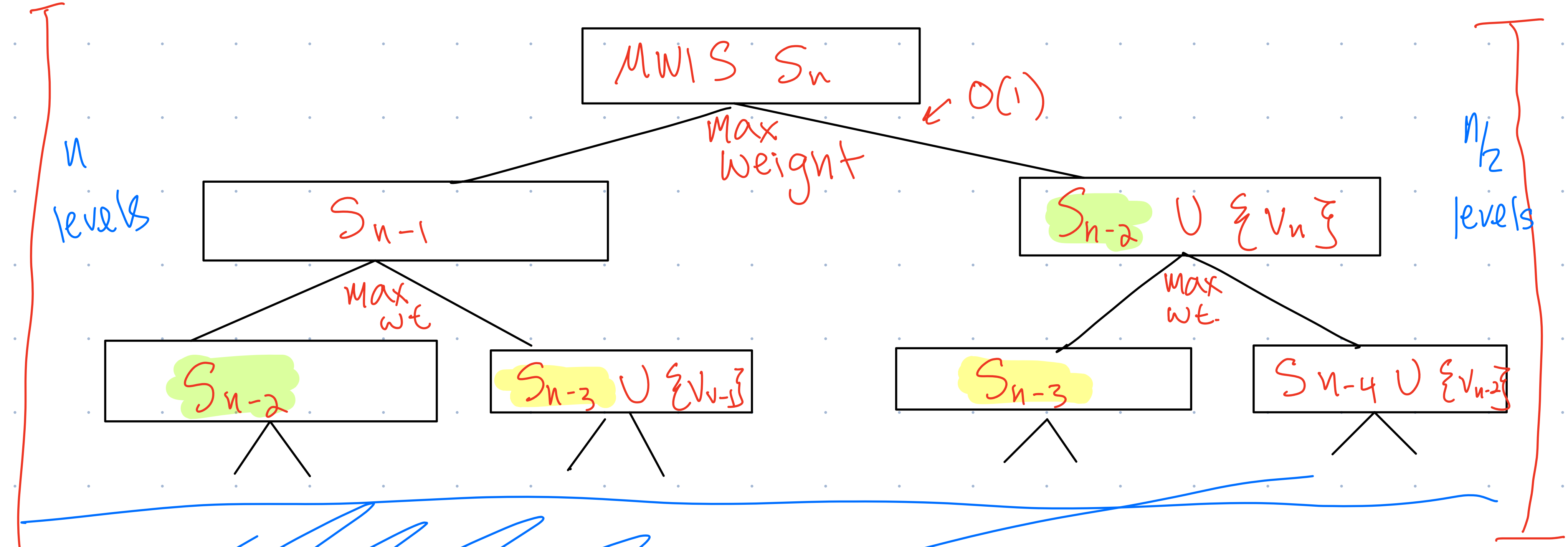


$$S_n = \begin{cases} S_{n-2} \cup \{v_n\} & \text{if } v_n \in S_n \\ S_{n-1} & \text{if } v_n \notin S_n \end{cases}$$

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

* And base case Later. . .

Recursive Algorithm:



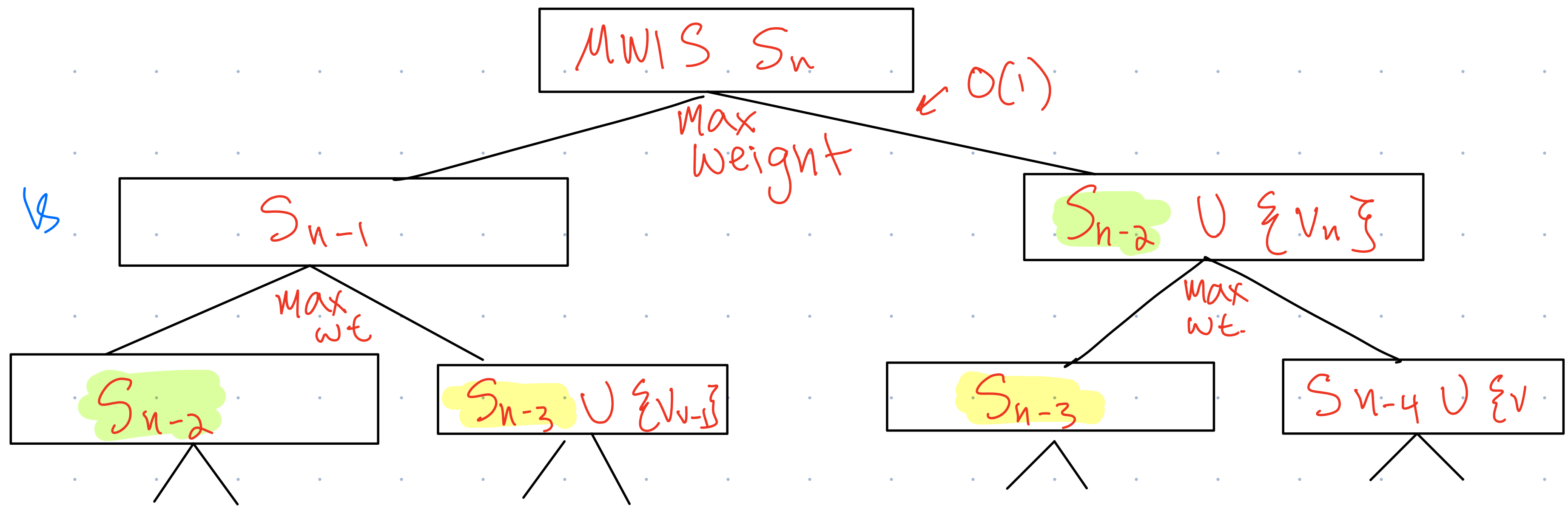
# of nodes:

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

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

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

Recursive Algorithm:



How many unique subproblems are there? $S_n, S_{n-1}, S_{n-2} \cdots, S_1$

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

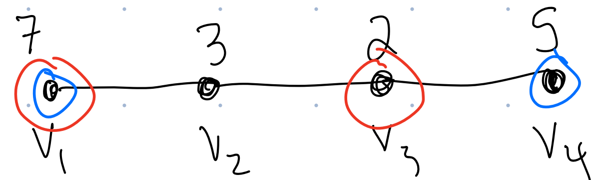
Dynamic Programming Idea: Store subproblems in an array + look up

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

Base Case

ϕ if $n = 0$

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



$S_2 = \max\{\{v_1\}, \phi \cup \{v_2\}\}$

$S_3 = \max\{\{v_1\}, \{v_1\} \cup \{v_3\}\}$

$S_4 = \max\{\{v_1 v_3\}, \{v_1\} \cup \{v_4\}\}$

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

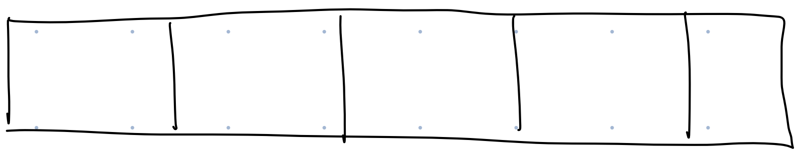
Trick 1: Fill up from bottom

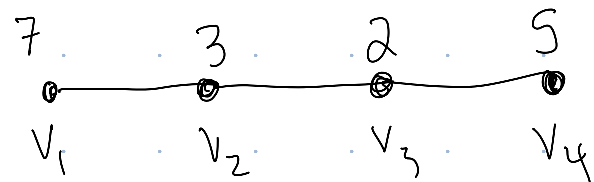
Trick 0: Include empty subproblem (S_0)

Trick 2: Store Objective Function Value $A(n) =$

$$S_n = \begin{cases} S_{n-2} \cup \{v_n\} \\ S_{n-1} \\ \text{Base Case} \\ \quad \emptyset \quad \text{if} \quad n=0 \\ \quad \{v_1\} \quad \text{if} \quad n=1 \end{cases}$$

array A





MWIS on a Line (G, w): [DP2]

// Create array of objective function values

1.

2.

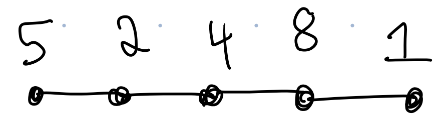
3.

4.

// Determine optimal Set

5.

6.

ex: 

A

S = { }

Runtime:

7. // Work backwards through FOR loop code

While $i \geq \boxed{}$ // include vertex v_i?

if $A[i] = A[i-1]$:

$\boxed{}$

else

$\boxed{}$

$A[i] \leftarrow \max\{A[i-1], A[i-2] + w(v_i)\}$

8 // Base case(s)

If $i == \boxed{}$:

$\boxed{}$

9 Return S

Why "dynamic programming"