DYNAMIC PROGRAMMING : MWIS

- Create a recurrence for MWIS on a line [DP1]
- Write correct pseudocode for a dynamic prog. alg [DP2]

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
- Brute Force

We will prove using strong induction that ____________________

correctly returns ____

for all $n \geq$ ________ where $n =$ ________

Base case: when $n \leq$ ________, the base case of the algorithm

correctly ________________________________ because/via

____.

Inductive Step: let $k \geq$ ________, and assume ____________________

is correct for all input sizes j for ________ $\leq j \leq k$. Consider an input of size $k + 1$.

Since $k + 1 \geq$ ________, we skip the base case and go to the divide step.

(d) Complete the proof (except for you may use Lemma 1 without proof)

Fill in the three missing parts (box and two blanks) of the proof below:

Let $M(x, y)$ be the algorithm that checks

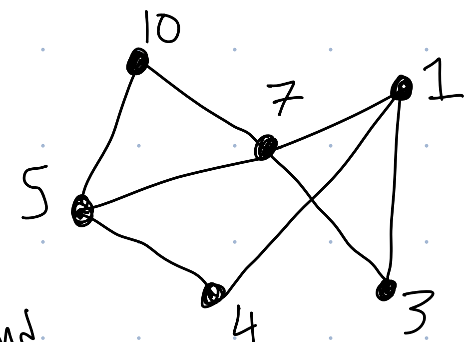
and outputs 1 if all checks pass, and 0 otherwise.

Then if x is a YES instance, then there exists a y that is a ______________________________,
which will cause $M(x, y)$ to output 1, but if x is a NO instance, then for any y, at least one check will fail, resulting in $M(x, y)$ outputting 0.

If the x [redacted] then $|x|$ is of size

$\Omega($__________$)$, and all checks can be done by brute force in $O(\text{poly}(|x|))$ time.

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 $\overset{\text{not}}{\neg}\,(u \in S \overset{\text{and}}{\wedge} v \in S)$ ← Independent Set
- Maximizes $\underbrace{W(S) = \sum_{v \in S} w(v)}_{\text{objective function}}$

Applications:

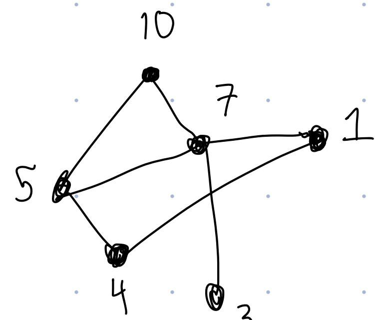
- Cell Tower Transmissions
- Choose franchise location
- Party Invite
- Scheduling
- House robbing (ethical concerns)

<u>General Graph</u>: Hard

<u>Line Graph</u>: 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.

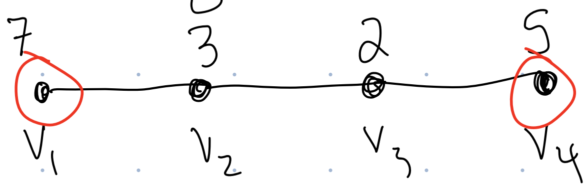
- If $\{u, v\} \in E$, $\neg(u \in S \wedge v \in S)$ ⟸ "Independent Set Condition" "constraint"

  not ↓ ; and

objective function

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

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

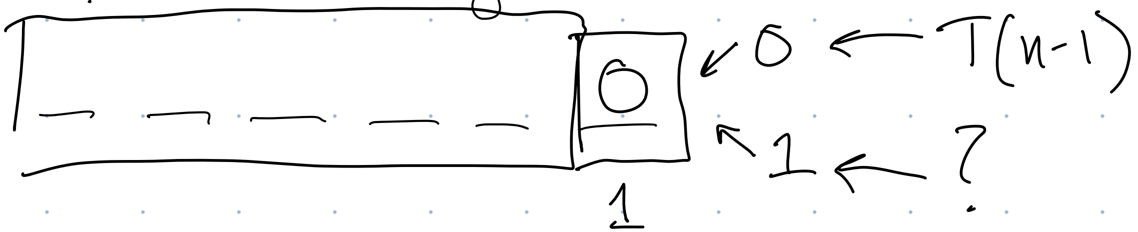


↘ $\{v_1, v_4\}$

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

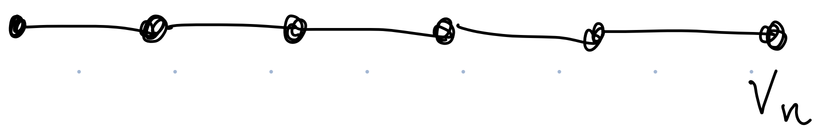
Dynamic Prog. Approach

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

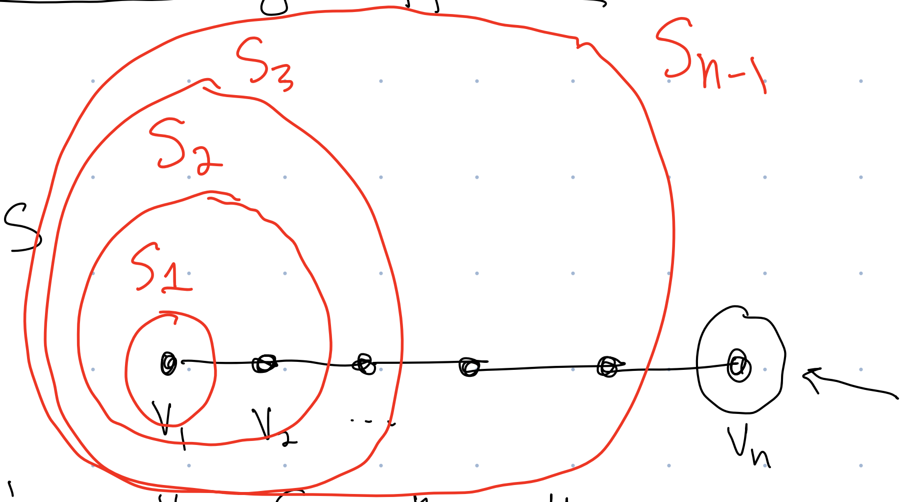


"Final choice"

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

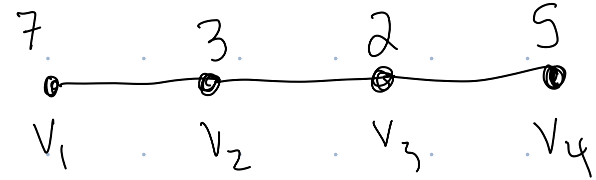
Options: S_{n-1}, S_{n-2}, $S_{n-1} \cup \{v_n\}$, $S_{n-2} \cup \{v_n\}$, $S_{n-2} \cup \{v_{n-1}\}$

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 $S \leftarrow \emptyset$
Max $W \leftarrow 0$
For each set $S \subseteq V$: $O(2^n)$
  If S is I.S.:
    $w \leftarrow W(S)$
    if $w >$ max W then
      max $S \leftarrow S$
      max $W \leftarrow w$
Return max S

$O(n)$

000
001
010
011
100
101
110
111

$O(2^n n)$

S is I.S.:

For $i \leftarrow 1$ to $n-1$:
  If $v_i \in S$ and $v_{i+1} \in S$:
    Return False
Return true

$O(n)$

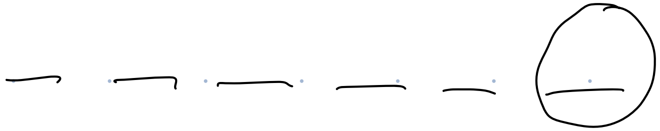
$W(S)$:

$W \leftarrow 0$
For $i \leftarrow 1$ to n
  If $v_i \in S$
    $W \leftarrow W + w(v_i)$
return W

$O(n)$

Dynamic Prog. Approach

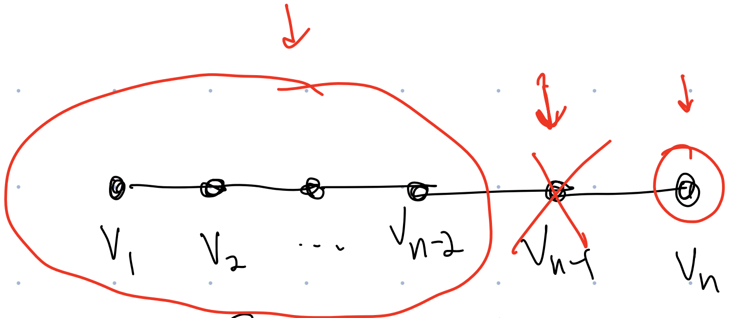
Recall: # of n bit strings with 2 consecutive ones



← 2 cases, 0 or 1

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

MWIS



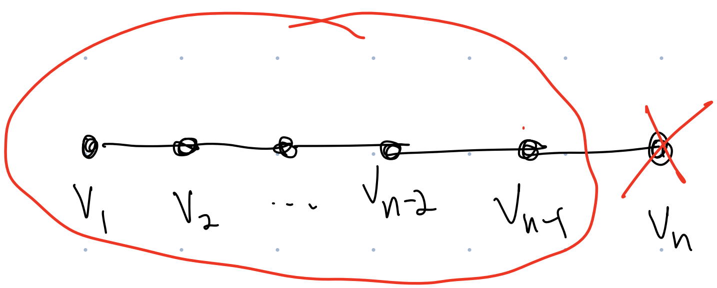
← 2 cases : $v_n \in S$ or $v_n \notin S$

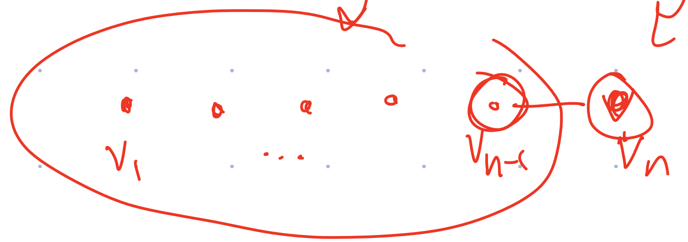
Let's call S_i the MWIS of first i vertices

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

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

Ind. Set



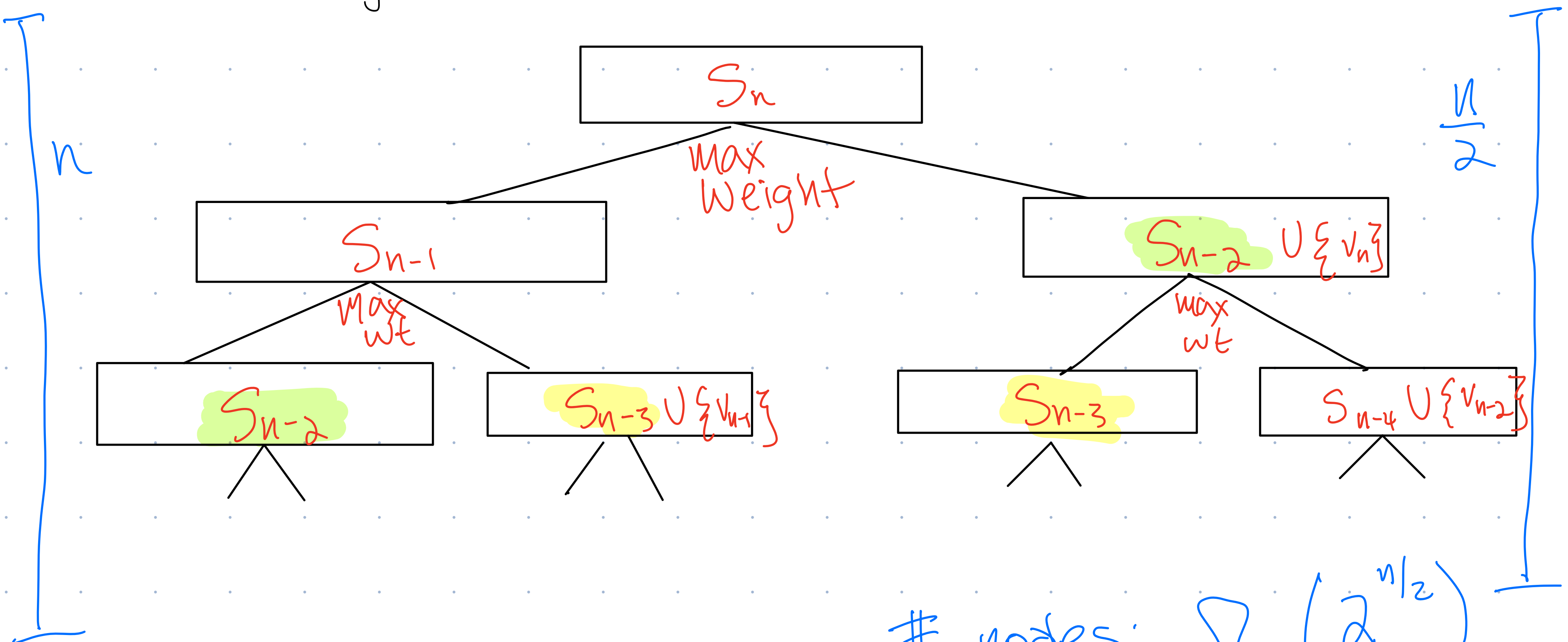


$$S_n = \begin{cases} S_{n-2} \cup \{v_n\} & \text{if } v_n \in S_n \\ S_{n-1} & \text{if } v_n \notin S_n \end{cases}$$

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

★ And base case Later..

Recursive Algorithm:



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

n = 10, 8, 6, 4, 2

n = 10, 9, 8, 7, 6

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

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

Recursive Algorithm:

How many unique subproblems are there?

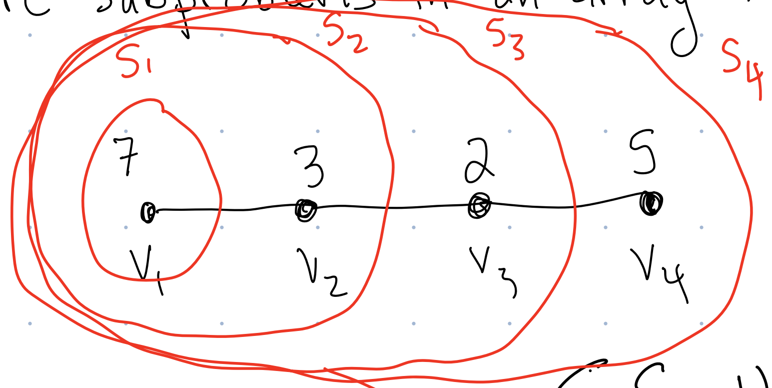
A) $\sqrt{n}$ B) $\frac{n}{2}$ C) n D) n^2

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

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

$\emptyset$ if $n=0$

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



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

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

Trick 2: Store Objective Function Value

$A(n)$ = weight of S_n

$A(i)$ = weight of S_i

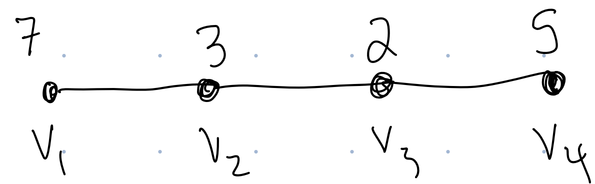
$$S_n = \begin{cases} S_{n-2} \cup \{v_n\} & \text{if } \boxed{v_n \in S_n} \\ S_{n-1} & \text{if } \boxed{v_n \notin S_n} \\ \text{Base Case} & \\ \emptyset & \text{if } n = 0 \\ \{v_1\} & \text{if } n = 1 \end{cases} \implies A(n) = \begin{cases} \max\{\boxed{A(n-2) + w(v_n)}, \boxed{A(n-1)}\} & \text{if } n \geq 2 \\ 0 & \text{if } n = 0 \\ w(v_1) & \text{if } n = 1 \end{cases}$$

array A

$A(0)$	$A(1)$	$A(2)$	$A(3)$	$A(4)$
$W(S_0)$	$W(S_1)$	$W(S_2)$	...	
0	7	7	9	12

9

7+5



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

// Create array of objective function values

1. Initialize array A of size $n+1$ // starting index @ 0
2. $A[0] \leftarrow 0$
3. $A[1] \leftarrow w(v_1)$
4. For $i \leftarrow 2$ to n

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

// Determine optimal set

5. $S \leftarrow \emptyset$ // store optimal set
6. $i \leftarrow n$ // index to walk backwards

ex: 5 — 2 — 4 — 8 — 1

A	0	5	5	9	13	13

$S = \{ \qquad \}$

Runtime: $O(n)$

7. // Work backwards through FOR loop code

While $i \geq \boxed{2}$ // include vertex v_i?

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

    $\boxed{i \leftarrow i-1}$

  else

    $\boxed{\begin{array}{l} S \leftarrow S \cup \{v_i\} \\ i \leftarrow i-2 \end{array}}$

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

8 // Base case(s)

  If $i == \boxed{1}$:

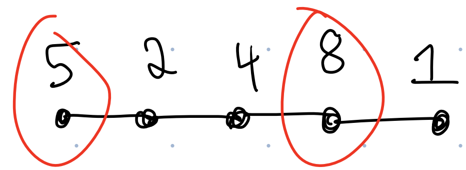
    $\boxed{S \leftarrow S \cup \{v_1\}}$

9 Return S

$i = n$

	0	1	2	3	4	5
A	0	5	5	9	13	13

$S = \{1, 4\}$

ex: 

Why "dynamic programming"