DYNAMIC PROGRAMMING : MWIS

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

Announcements

Exit Tickets

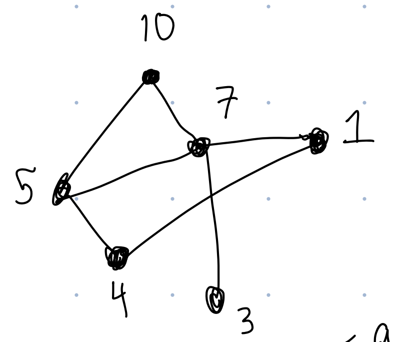
Max Weight Independent Set

Input:

Output:

Applications:

Max Weight Independent Set (MWIS)



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

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

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

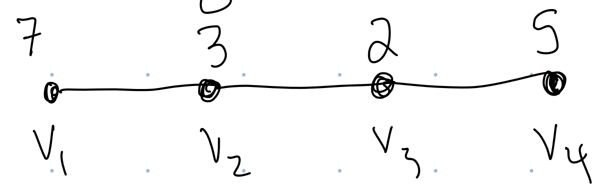
- If $\{u, v\} \in E$, $\neg (u \in S \wedge v \in S)$ ⇐ "Independent Set Condition" "constraint"

  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:



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

Dynamic Prog. Approach

Recall: # of n bit strings with 2 consecutive ones

Needed to identify "final options"

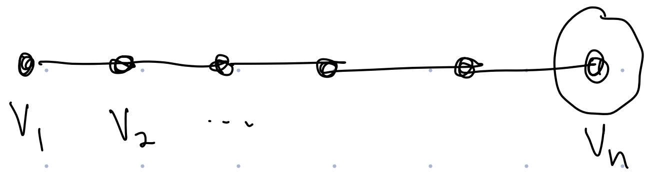
To create a DP alg, (often) need to conceptualize the optimal solution as a sequence of choices

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

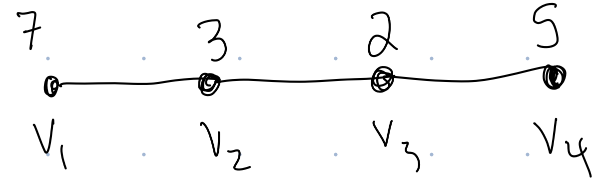
Options: S_{n-1}, S_{n-2}, $S_{n-1} \cup \{v_n\}$, $S_{n-2} \cup \{v_n\}$, $S_{n-2} \cup \{v_{n-1}\}$

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



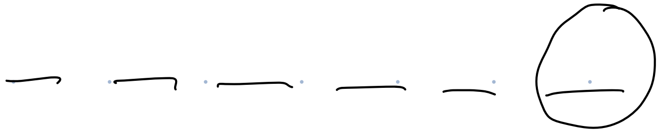

```
MWIS(G=(V,E), w)
maxS ← ∅
MaxW ← 0
For each set S ⊆ V:
    If S is I.S.:
        w ← W(S)
        if w > maxW then
            maxS ← S
            maxW ← w
Return maxS
```

```
S is I.S.:
For i ← 1 to n-1:
    If v_i ∈ S and v_{i+1} ∈ S:
        Return False
Return true
```

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

Dynamic Prog. Approach

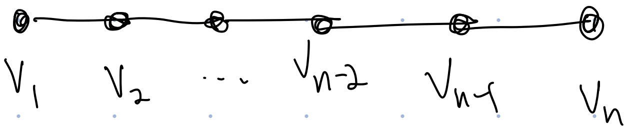
Recall: # of n bit strings with 2 consecutive ones



← 2 cases, 0 or 1

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

MWIS



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

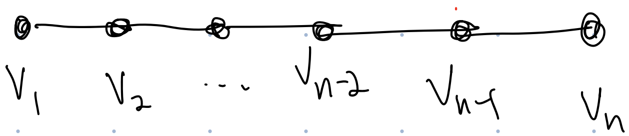
Let's call S_i the MWIS of first i vertices

Options:

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

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

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

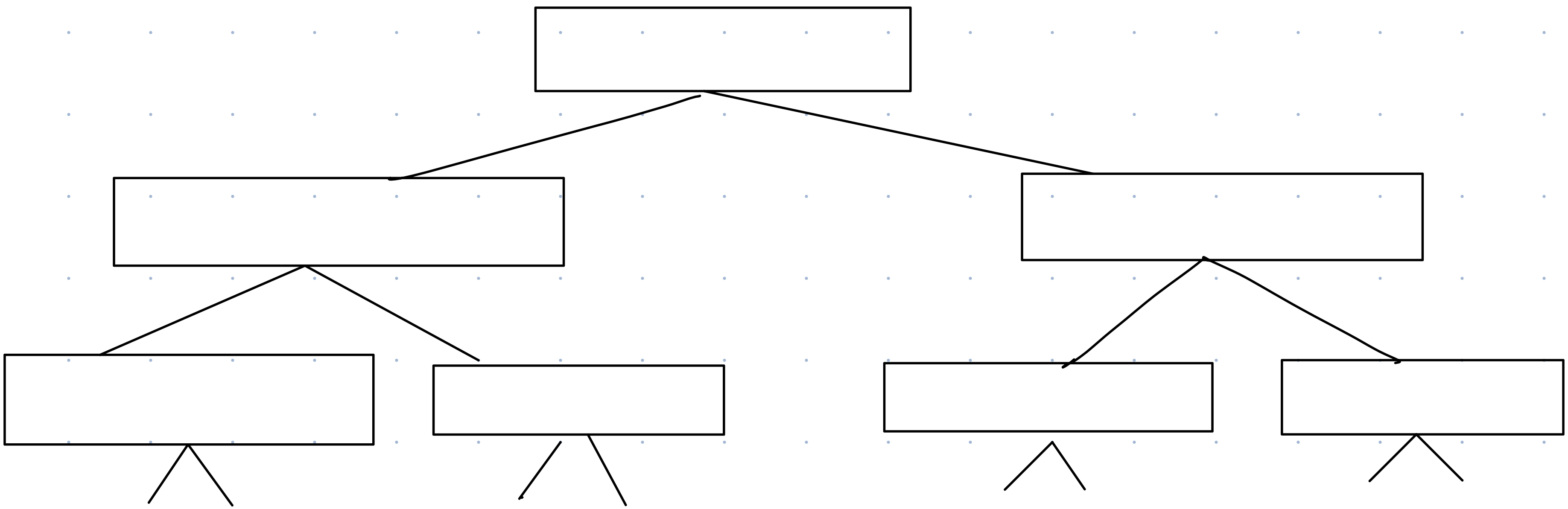


$$S_n = \begin{cases} \underline{\hspace{4cm}} & \text{if } n \in S_n \\ \underline{\hspace{4cm}} & \text{if } n \notin S_n \end{cases}$$

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

☆ And base case Later. .

Recursive Algorithm:



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

(Base Case)

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

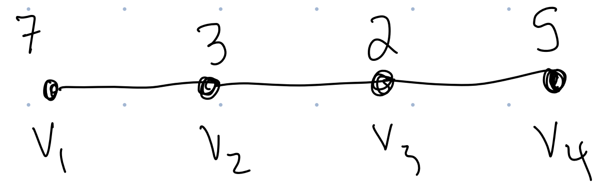
Recursive Algorithm:

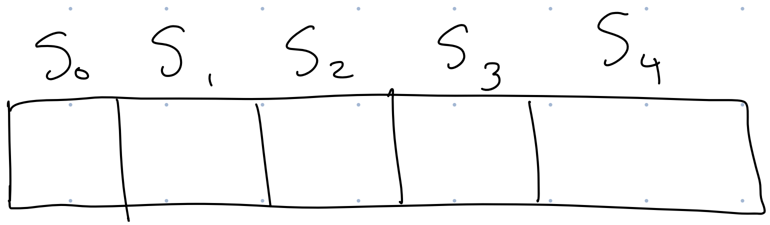
How many unique subproblems are there?

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

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

$$S_n = \begin{cases} \underline{\qquad\qquad\qquad} \\ \underline{\qquad\qquad\qquad} \end{cases}$$





Trick 1:

Trick 0: