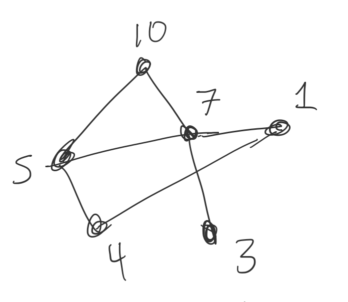
4. MWIS (Dynamic Programming)

Friday, March 12, 2021 11:44 AM

Max Weight Independent Set Problem (MWIS)

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



Output: $S \subseteq V$ s.t.
- If $\{u, v\} \in E$, $u \notin S$ or $v \notin S$ ⟸ Independent Set
- $W(S) = \sum_{v \in S} w(v)$ is maximized ⟸ Max weight

← objective function

Applications
- Cell Tower Transmissions
- Choosing franchise locations
- Party Invites
- Scheduling

} Will discuss when we learn about reductions

* General graph → very hard to solve optimally
* Path/Line graph → easier

What is $W(S)$ for MWIS S of

7 — 3 — 2 — 5 (v_1, v_2, v_3, v_4)

A) 0 B) 8 C) 9 D) 12 ↑

Ind Set	Weight
$\emptyset$	0
$\{v_1\}$	7
$\{v_2\}$	3
⋮	9
$\{v_1, v_3\}$	
$\{v_1, v_4\}$	12 ⇐
$\{v_2, v_4\}$	8

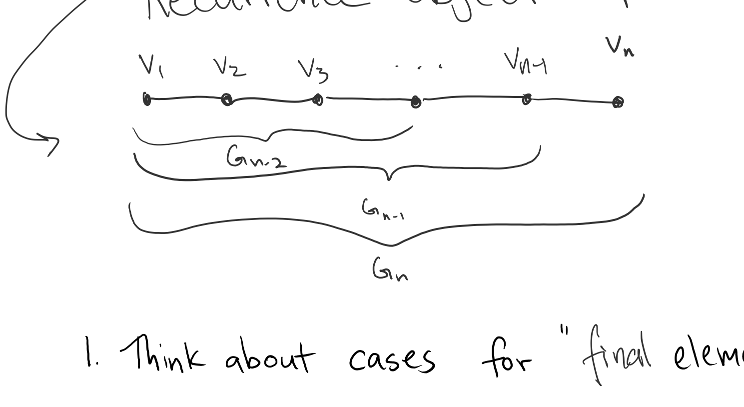
Brute Force Alg (n vertices)
For each set $S \subseteq V$ ← $O(2^n)$
 if I.S. } $O(n)$ (on line)
  check weight } $O(n)$
  store if largest seen } $O(1)$
 } $O(n)$
Return max weight

$O(n2^n)$ TERRIBLE

Divide + Conquer... better but not best

Designing a Dynamic Programming Alg.

0. [Create series of increasingly smaller subproblems
Recurrence object: Optimal output of each subproblem]



S_i = MWIS of G_i

1. Think about cases for "final elements" of recurrence object

Consider S_n. Two options for final vertex:

ONLY OPTIONS [i) $v_n \notin S_n$
ii) $v_n \in S_n$

(similar to last bit of string being 0 or 1)

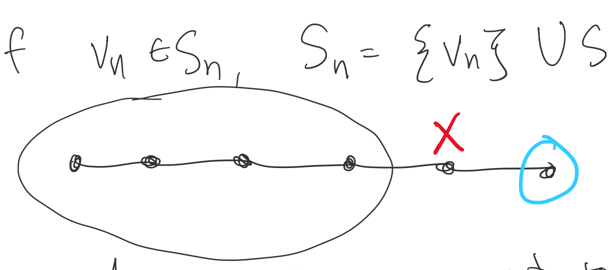
2. For each case, create recurrence

ONLY OPTIONS [i) If $v_n \notin S_n$, $S_n = S_{n-1}$
ii) If $v_n \in S_n$, $S_n = S_{n-2} \cup \{v_n\}$

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

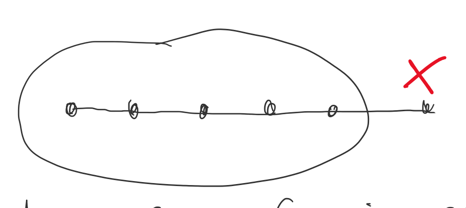
Explain

ii) If $v_n \in S_n$, $S_n = \{v_n\} \cup S_{n-2}$



Any I.S. here is good to create overall I.S. (with v_n)
S_{n-2} produces overall I.S. with max weight

i) If $v_n \notin S_n$, $S_n = S_{n-1}$



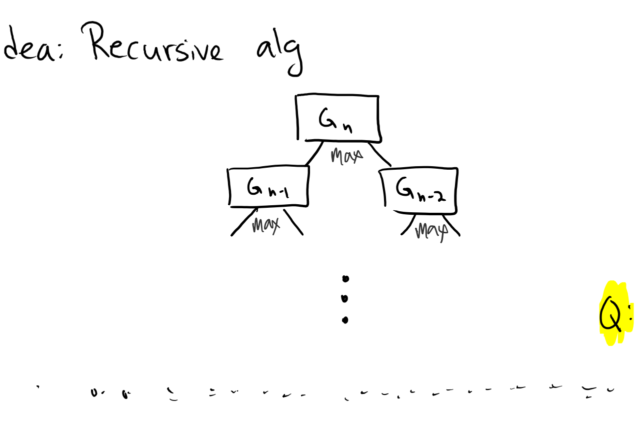
Any I.S. on G_{n-1} is good to create overall I.S. (without v_n)
S_{n-1} is I.S. with max weight

Conclusion

$S_n = \begin{cases} S_{n-1} \text{ OR} \\ S_{n-2} + V_n \end{cases}$ } Only possibilities. Take whichever has larger weight

Recurrence:
$S_n = S_{n-1}$ or $S_{n-2} \cup \{v_n\}$
$S_0 = \emptyset$, $S_1 = v_1$

First idea: Recursive alg

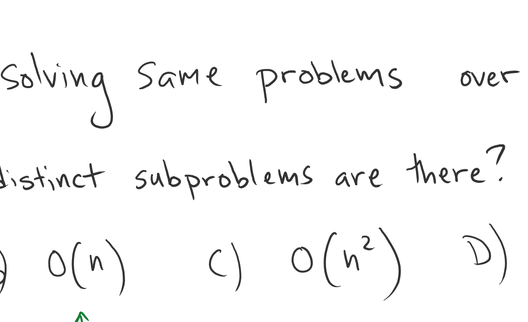


} O(n) levels ×2 for each

Q: How many leaves are there in this tree? (Approximately)

A) n B) n^2 C) 2^n

↑ BAD! $O(1)$ time/leaf ... at least 2^n time for algorithm



* Actually solving same problems over and over!

Q. How many distinct subproblems are there?

A) $O(1)$ B) $O(n)$ C) $O(n^2)$ D) $O(2^n)$

↑ $\{G_1, G_2, \ldots G_n\}$

Idea: Instead of solving recursively, store solutions in an array, look up.

Don't Build

S_0	S_1	S_2	S_3	...	S_n
$\emptyset$	v_1				

Trick 1: Build up instead of look back
Trick 2: Store objective function value instead of set/strategy object (faster)

Build

A[0] = $W(S_0)$	A[1] = $W(S_1)$			
0	w_1	.	.	

$S_i = \begin{cases} S_{i-1} \\ \text{OR} \\ S_{i-2} \cup \{v_i\} \end{cases} \iff W(S_i) = \begin{cases} W(S_{i-1}) \\ \text{OR} \\ W(S_{i-2}) + w_i \end{cases}$ take max one

$= \max\{W(S_{i-1}), W(S_{i-2}) + w_i\}$

MWIS on Line

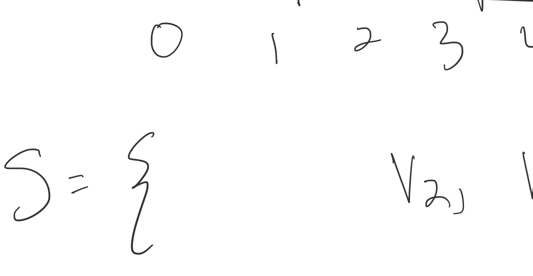
// Determine $W(S_i)$'s:

1. A[0] ← 0 } Base cases
2. A[1] ← w_1
3. For $i = 2$ to n:
 A[i] ← max{ A[i-1], A[i-2] + w(v_i) }

// Determine MWIS

4. $S_n \leftarrow \emptyset$
5. $i \leftarrow n$
6. while $i \geq 1$:
 if A[i] = A[i-1]: // $v_n \notin S_n$
  $i \leftarrow i-1$
 else: // $v_n \in S_n$
  $S_n \leftarrow$ Append v_i to S_n
  $i \leftarrow i-2$

ex:



0	5	7	9	11	11
0	1	2	3	4	5

A = [0, 5, 7, 9, 11, 11]

$S = \{ V_2, V_4 \}$

Runtime: $O(n)$
Proof: Explanation of recurrence relation
Ethics:
Why called dynamic programming?