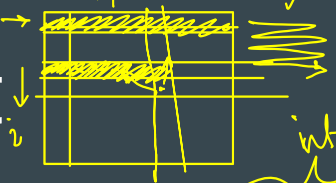
CSE525 Lec9: Dynamic Programming

Weighted Subset Sum with target

time complexity : O(1) × O(nT) = O(nT)
space " : Keep two adj. rows O(T)

memo : 2D array of dimensions i=0 to n, t=0 to T
(n+1)×(T+1) = O(nT)
top to bottom, left to right
integer valued

WSS problem = S(n, W) = bottom right element



Find largest value of any subset of S={s1,...,sn} with wts w(1),...,w(n) & target W

Find the value of the ~~heaviest~~ largest valued subset from {1,2,3,4,5,6} with wts 12,23,11,24,21,15 and total weight at most 40.

1,3,6 — 38; 2,6; 2,4; 4,6 — 39

- S(i) = ~~optimal value for~~ value of the largest valued subset of {s1 ... si}
  - Backtracking algorithm does not pick s1 if wt(s1) > W
  - Else, it either picks s1 or does not pick s1 ...
- S(i,t) = optimal value for {s1 ... si} with target = t
- S(i,t) = optimal value for {si ... sn} with target = t

$$S(i,t) = \begin{cases} S(i-1,t) & \text{if } wt(s_i) > W \\ \max\{S(i-1,t),\ S(i-1, t-W(s_i)) + s_i\} & \text{o/w} \end{cases}$$

s_i is not included; s_i is included; $<t$

$S(1,0) = 0$
$S(i,0) = 0\ \forall i$
$S(0,t) = 0\ \forall t$
$S(1,t) = ?$

Exercise

opt. solution for $\{S_1 \cdots S_i\}$ = opt. solution for $\{S_1 \cdots S_{i-1}\}$ → feasible solution of $\{S_1 \cdots S_i\}$
val(S_i) + opt. solution for $\{S_1 \cdots S_{i-1}\}$, target = W − w(S_i)
S(i-1) ≠

Min-wt Vertex Cover

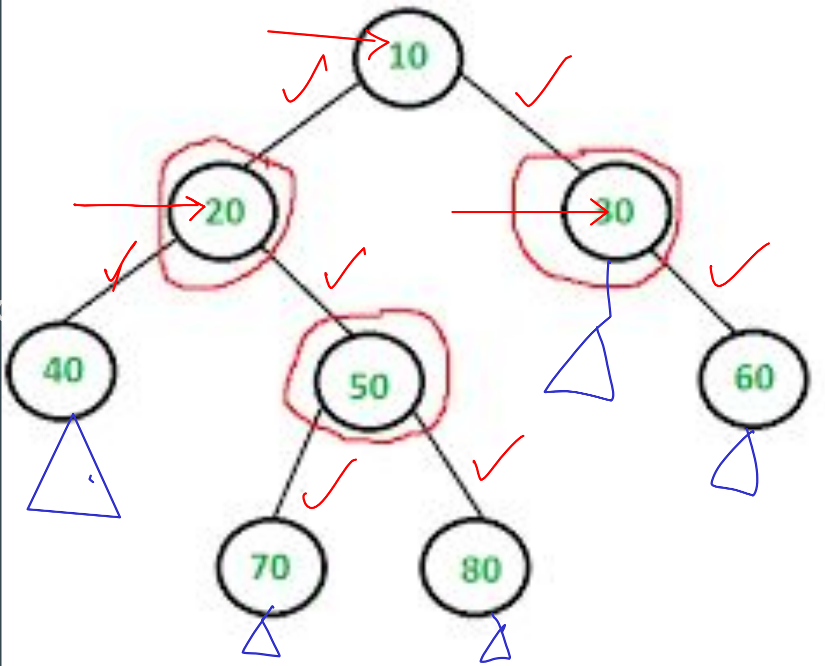
VC = subset of vertices which cover **every** edge

wt of the min vertexcover

Define VC(node) = ~~solution~~ of the subtree rooted node.

Can you compute VC(10) using solutions for VC(20) and VC(30)?

=110 =70 X



Think of a backtracking algorithm ! What are all the possible cases needed to compute solution for subtree(10)?

Memo: Tree, order: Start at leaf, go up

Min-wt Vertex Cover (T) = A(root, maybe)

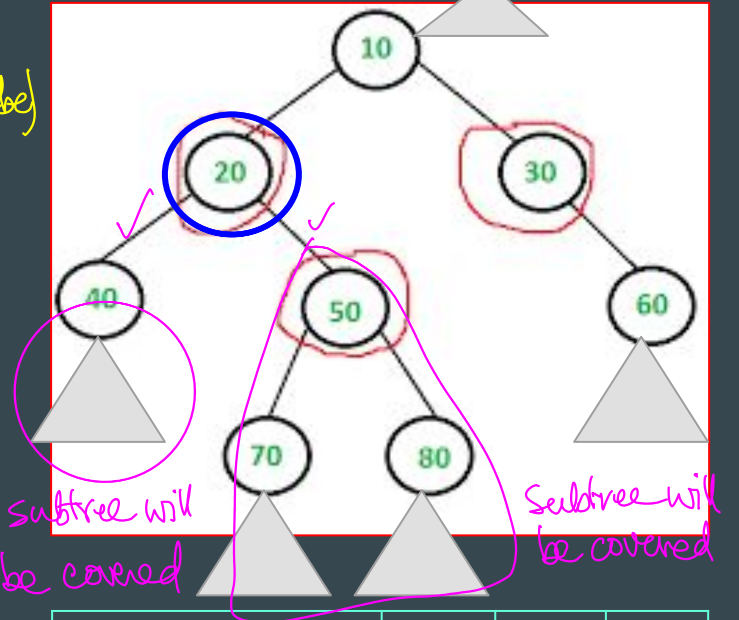
Define problems A(node,b) where b = yes/no/maybe

value of

A(v,yes) = min-wt-VC of subtree under v in which v **must be** included

A(v,no) = min-wt-VC of subtree under v in which v **must-not be** included

A(v,maybe) = min-wt-VC of subtree under v in which v **may or may not be** included

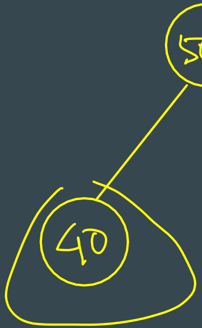


v=	40	50	20
A(v)	400	100	?
Ayes(v)	400	200	? 20+ A(40, maybe) + A(50, maybe) 520
Ano(v)	600	100	? 600 A(40, yes) + A(50, yes)
Amaybe(v)	400	100	? min(A(20, yes), A(20, no)) 520

Q: Time-complexity & Space-complexity? $O(1) \times V = O(V)$, $O(V)$ $O(1)$

Q: What would be the base cases? A(l, yes) = wt(l) A(l, no) = ?. Ex. A(l, maybe) = min(yes, no)

Hint: Look at the parent of a leaf node.



$A(40, \text{yes}) = 40$

$A(40, \text{no}) = ?$

$A(50, \text{yes}) = 50$

$= 50 + A(40, \text{maybe})$

$\therefore A(40, \text{maybe}) = 0$

recursive form.

$A(40, \text{no}) = 0$

defn. of A -

Longest Balanced Subsequence

A =][ )]][[ )][ ) (
(([ ]) [ ] ( ) ( ) [[]]) [[]]

Describe and analyze an algorithm to compute the length of a longest balanced subsequence of a given string of parentheses ( ) and brackets [ ]. Your input is an array A[1 .. n], where A[i] ∈ {(,),[,]} for every index i.

A string s consisting of (,),[,] is balanced if it is of the form (S1) or [S1] or S1.S2 where S1 and S2 are balanced themselves.

LBS(i,j) = length of longest balanced subsequence of A[i ... j]

$$= \begin{cases} A_i = (\quad A_j =) \quad \max \begin{bmatrix} 2 + LBS(i+1, j-1), \\ \max\limits_{s,t} \; LBS(i,s) + LBS(t,j) \end{bmatrix} \end{cases}$$

s < t

A[s] =)

A[t] = (

A_i - - - - A_j

k

$LBS(A_i \cdots A_k)$ $\quad A_{k+1} \cdots A_j$

( ) ( ) ( )

s t

① (LBS of A[i+1 ... j-1]) = Longest BS of A[i...j] ?

② Longest BS of A[i...j] = (. . . . .) . . . (. . . .)

1   s   t   20

A = ( ) ( )

Suppose we used only ① . LBS(( ) ( )) = $\underbrace{(\text{ LBS( ) ( ) })}_{0}$ = ( ) ✗

( ) ( )

LBS( ) ( ) = 0

LBS(( ) ( )) = 2

③ Longest BS of A[i...j] = (. . . . .))

A = ( ))
1 2 3

( )
1 2
SUB1

( )
1 3

(( )))
1 2 3 4 5

(( ))
1 2 3 5

(( ))
1 2 3 4

Exercise

(( )
1 15 20

(. . . .) ( ))
1 15 17 18 20

(. . . . .) ( ))
1 15 17 18 20

(( )))
1 7 15 17 20