Agenda for today

Quiz

Talk more about independent set + activity

Computing optimal choices (not just value)

A non-recursive formulation for dynamic programming

You should use P(j) in OPT(j)

(a) (4 points) Suppose that the houses are located at (in yards) $x_1 = 100$, $x_2 = 350$, $x_3 = 1000$, $x_4 = 1100$, $x_5 = 1200$, $x_6 = 1300$, and c_1 through c_6 are $5, 8, 2, 12, 3, 7$. What is the maximum amount of candy that you can get? Which houses should you visit?

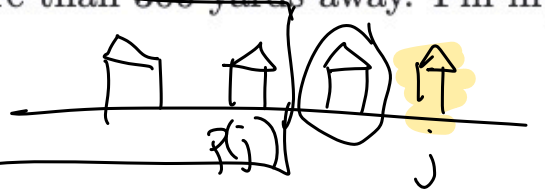
20 pieces = OPT(6)

visit 2 and 4

$c_2 = 8$ $c_4 = 12$

OPT(j) = optimal # candies using a compatible subset of houses 1 to j

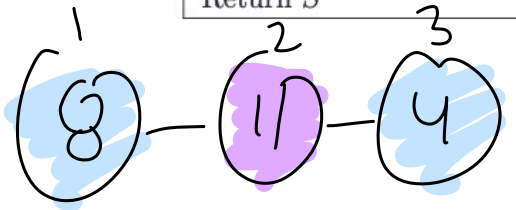
(b) (5 points) Let OPT(j) denote the number of candy pieces that you can get using all houses up to house j. Also, for house located at x_j, let $p(j)$ be the earlier house that is closest to house j but more than 300 yards away. Fill in part of the recursive definition of OPT(j) below.



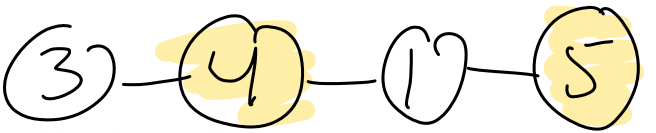
$$\text{OPT}(j) = \begin{cases} 0 & \text{if } j = 0 \\ \max\left(\text{OPT}(j-1),\ c_j + \text{OPT}(p(j))\right) & \text{if } j > 0. \end{cases}$$

(a) (3 points) Give an example to show that the following algorithm *does not* always find an independent set of maximum total weight.

> Start with S equal to the empty set
> While some node remains in G:
> Pick a node v_i of maximum weight and add it to S
> Delete v_i and its neighbors from G
> Return S



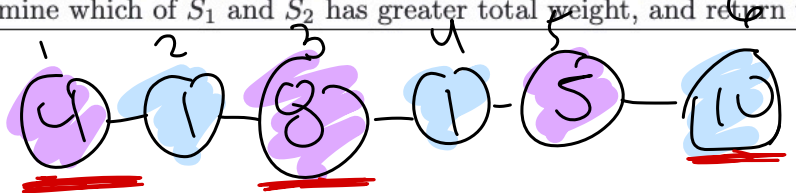
wt 11 wt 12



(b) (3 points) Give an example to show that the following algorithm also *does not* always find an independent set of maximum total weight.

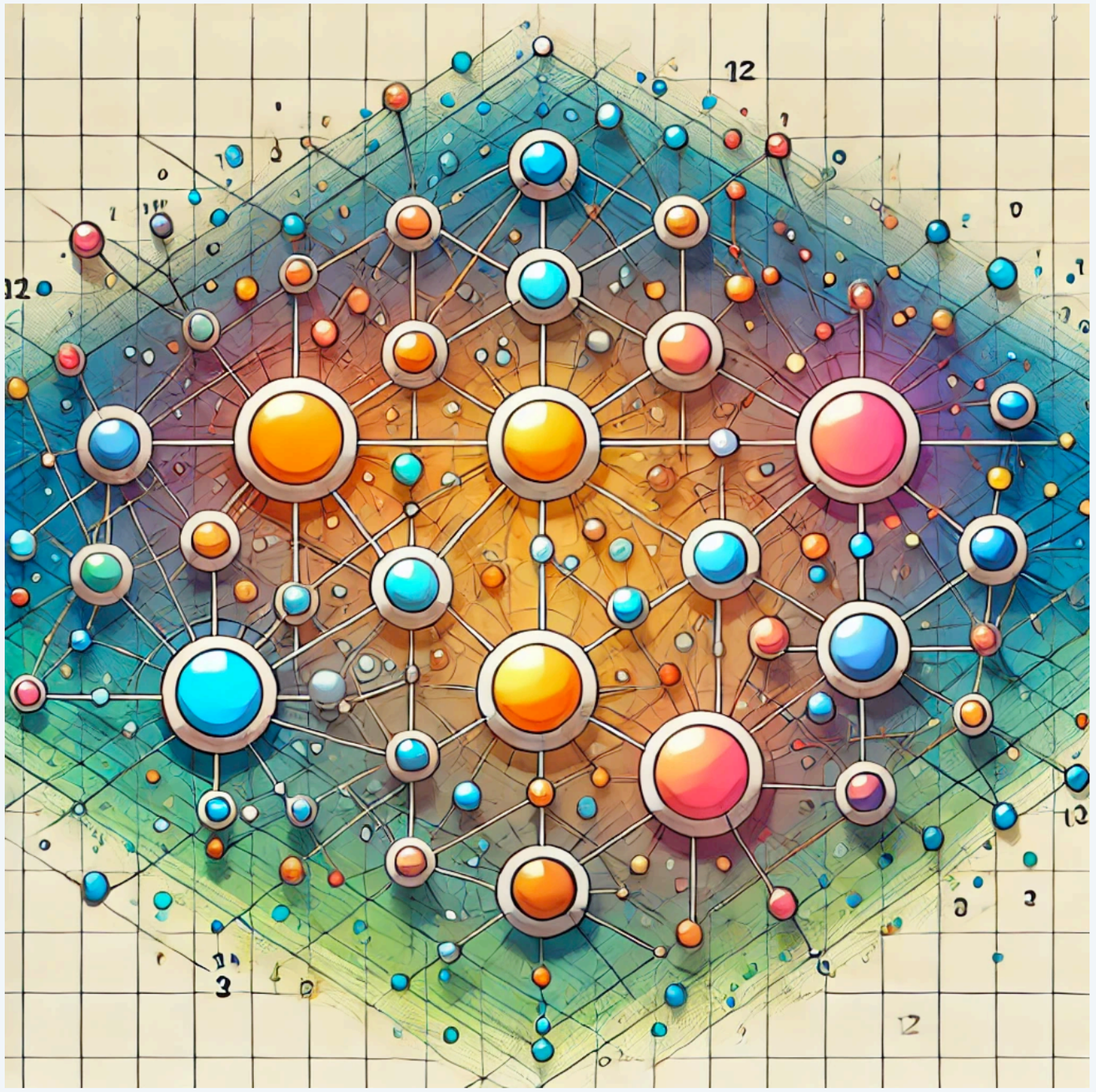
> Let S_1 be the set of all v_i where i is odd
> Let S_2 be the set of all v_i where i is even
> (Note that S_1 and S_2 are both independent sets)
> Determine which of S_1 and S_2 has greater total weight, and return this one

weight S_1 17
weight S_2 12



weight 22

Activity



(a) Give a recurrence relation for the weight, OPT(j), of the first j nodes in an n-node path as input to independent set. Notice that you don't need $p(j)$ for this problem.

(b) baseline weight 1

(c) +2 for every node included

$$OPT(j) = \begin{cases} 1 & \text{if } j=0 \\ \max(OPT(j-1), w_j + 2 + OPT(j-2)) & \text{if } j>0 \end{cases}$$

Weighted interval scheduling: finding a solution

FIND-SOLUTION(j)

IF ($j = 0$)

  RETURN $\varnothing$.

ELSE IF ($w_j + M[p[j]] > M[j-1]$)

  RETURN $\{ j \} \cup$ FIND-SOLUTION($p[j]$).

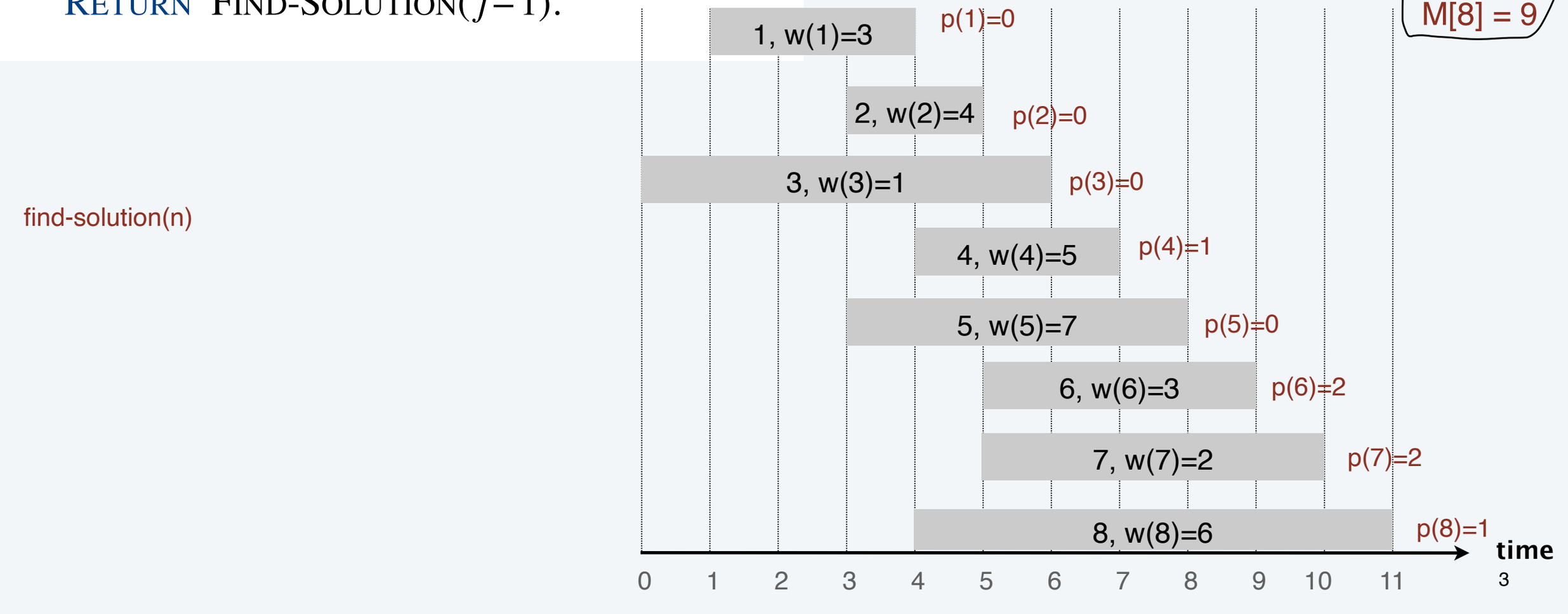
ELSE

  RETURN FIND-SOLUTION($j-1$).

find-solution(n)

{8, 6}

M[0] = 0
M[1] = 3
M[2] = 4
M[3] = 4
M[4] = 8
M[5] = 8
M[6] = 8
M[7] = 8
M[8] = 9



Can there be more than one optimal set of intervals?

FIND-SOLUTION(j)

IF $(j = 0)$

  RETURN $\emptyset$.

ELSE IF $(w_j + M[p[j]] > M[j-1])$

  RETURN $\{ j \} \cup$ FIND-SOLUTION($p[j]$).

ELSE

  RETURN FIND-SOLUTION($j-1$).

1. Yes
2. No

Can there be more than one optimal set of intervals?

```
FIND-SOLUTION(j)

IF (j = 0)
    RETURN ∅.
ELSE IF (w_j + M[p[j]] > M[j−1])
    RETURN { j } ∪ FIND-SOLUTION(p[j]).
ELSE
    RETURN FIND-SOLUTION(j − 1).
```

1. Yes
2. No

With table: which one does this algorithm find?

Weighted interval scheduling: bottom-up dynamic programming

BOTTOM-UP($n, s_1, \ldots, s_n, f_1, \ldots, f_n, w_1, \ldots, w_n$)

Sort jobs by finish time and renumber so that $f_1 \leq f_2 \leq \ldots \leq f_n$.

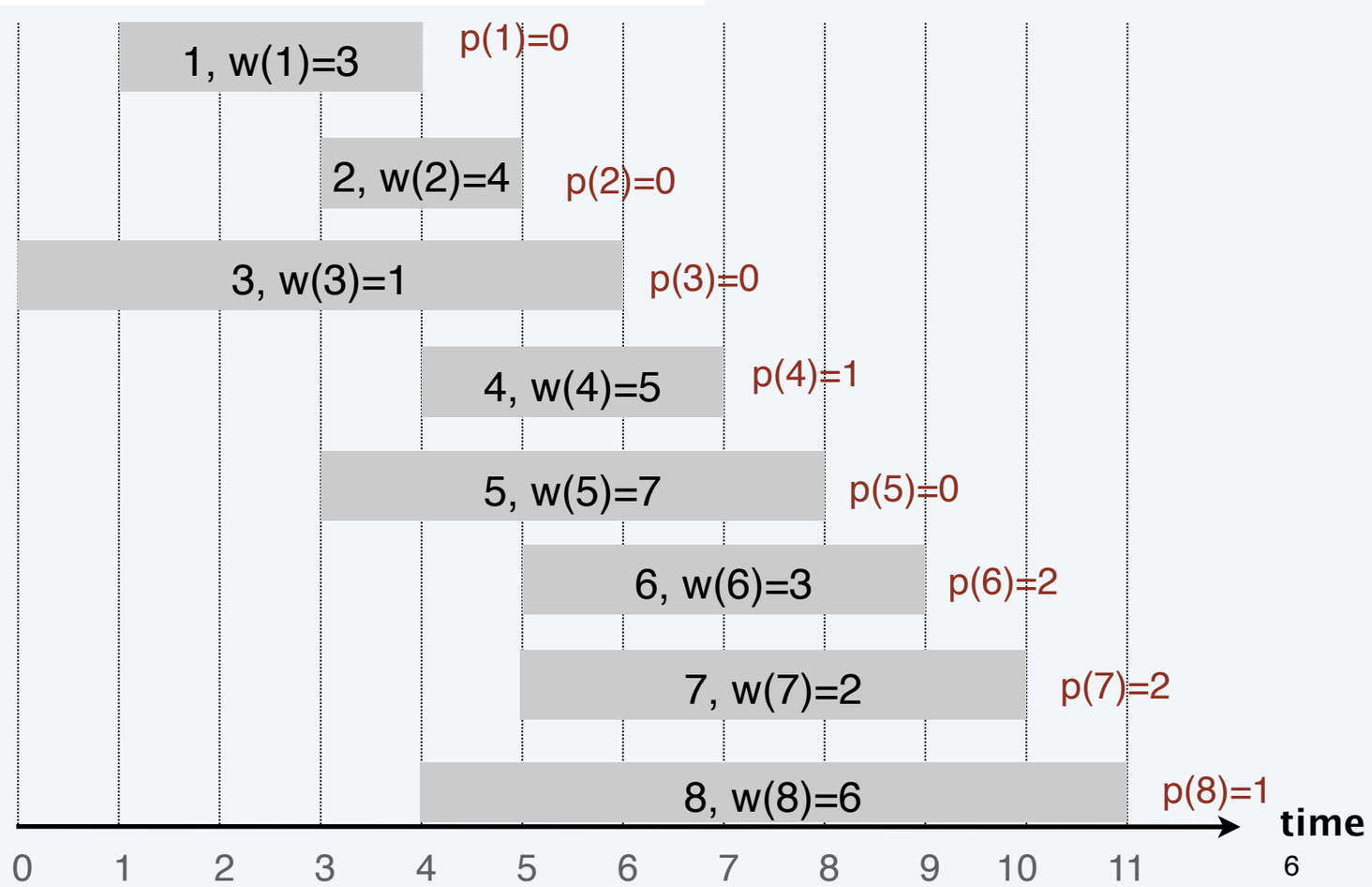
Compute $p[1], p[2], \ldots, p[n]$.

$M[0] \leftarrow 0$.

FOR $j = 1$ TO n

  $M[j] \leftarrow \max \{ M[j-1],\ w_j + M[p[j]] \}$.

M[0] = 0
M[1] =
M[2] =
M[3] =
M[4] =
M[5] =
M[6] =
M[7] =
M[8] =



Memoization allowed us to go from $O(2^n)$ to $O(n)$...

Can we memoize merge sort?

mergesort(L):

```
L_1 = first half of L
L_2 = first half of L
sorted_L_1 = mergesort(L_1)
sorted_L_2 = mergesort(L_2)
return merged L_1 and L_2
```

Memoization allowed us to go from $O(2^n)$ to $O(n)$...

Can we memoize merge sort?

mergesort(L):

```
  L_1 = first half of L
  L_2 = first half of L
  sorted_L_1 = mergesort(L_1)
  sorted_L_2 = mergesort(L_2)
  return merged L_1 and L_2
```

No-the key was overlapping subproblems