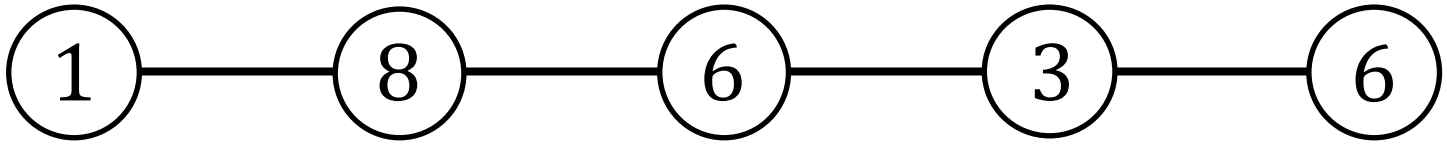
Dynamic Programming
CSCI 432

Problem Statement

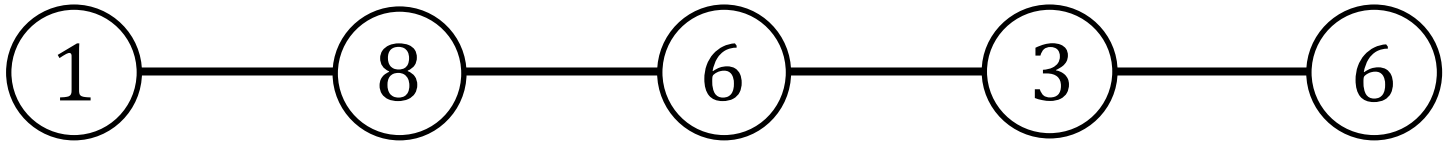
Suppose we have a weighted graph of this form:



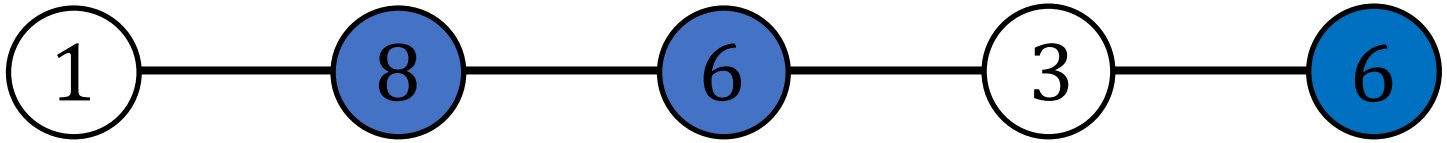
Goal: Find a subset of vertices such that they do not share any edges with each other, and the sum of their weights (w_i) is maximized.

Problem Statement

Suppose we have a weighted graph of this form:

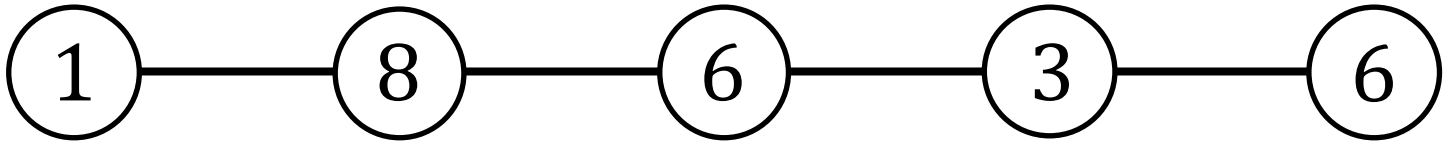


Goal: Find a subset of vertices such that they do not share any edges with each other, and the sum of their weights (w_i) is maximized.

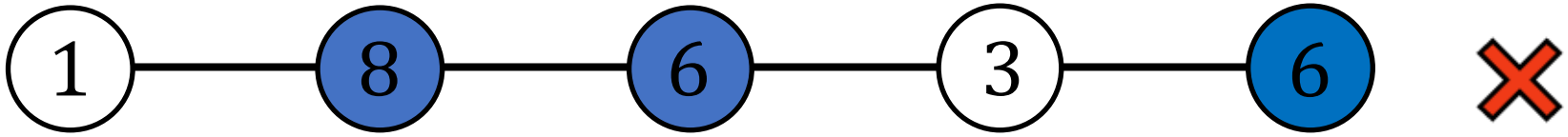


Problem Statement

Suppose we have a weighted graph of this form:

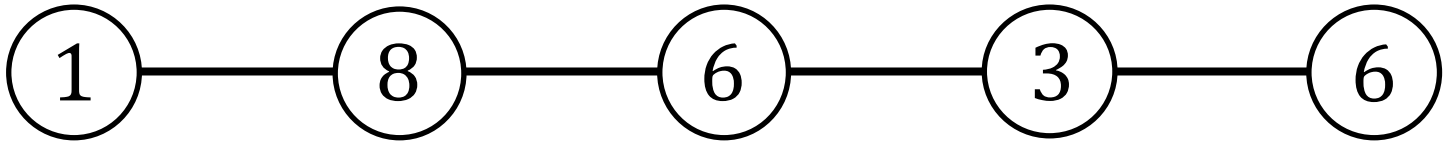


Goal: Find a subset of vertices such that they do not share any edges with each other, and the sum of their weights (w_i) is maximized.

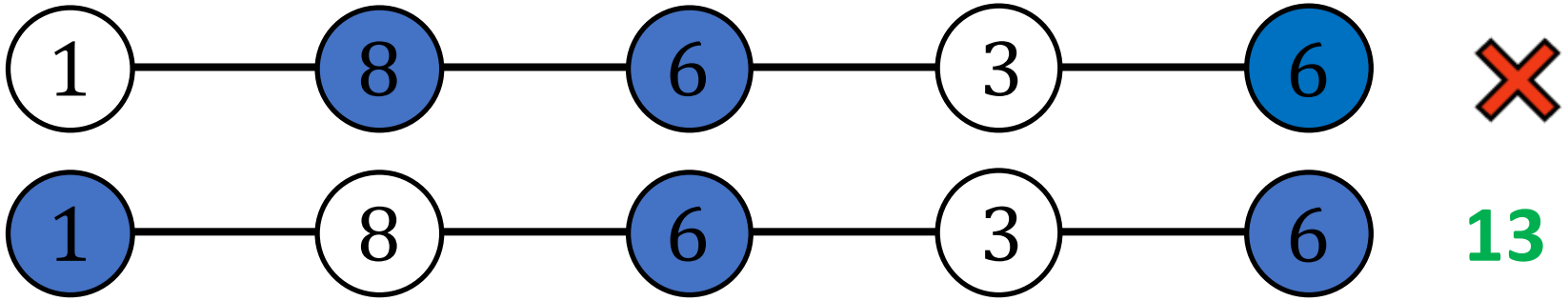


Problem Statement

Suppose we have a weighted graph of this form:

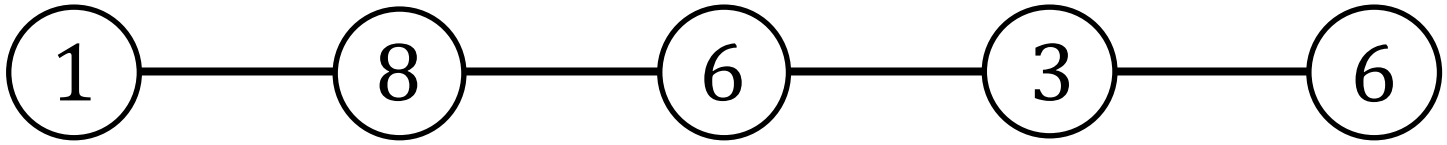


Goal: Find a subset of vertices such that they do not share any edges with each other, and the sum of their weights (w_i) is maximized.

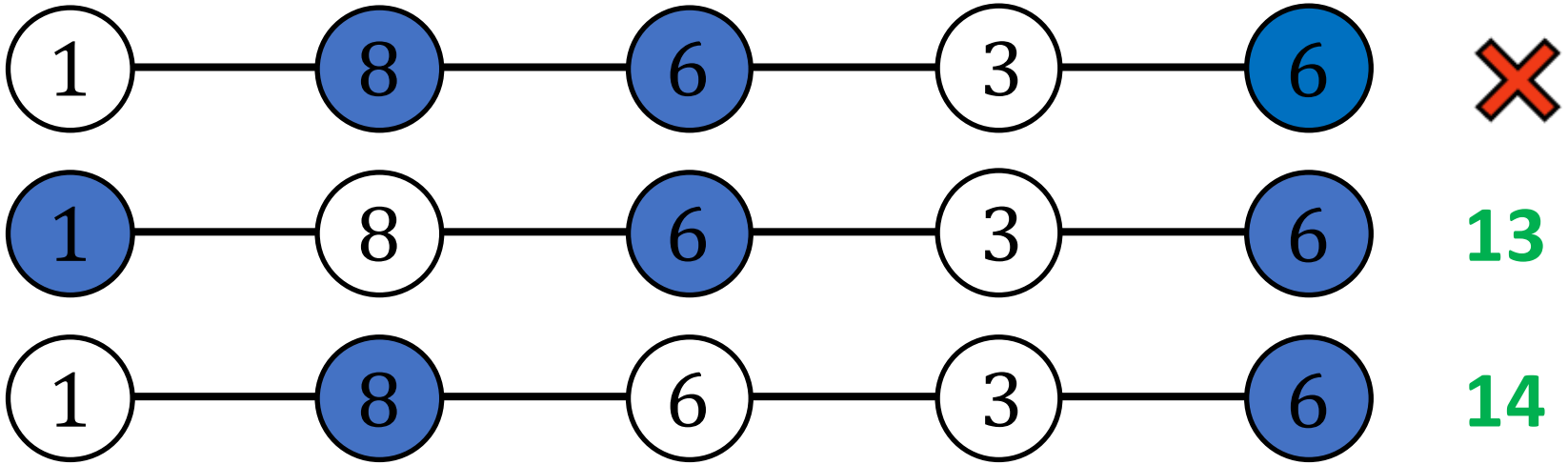


Problem Statement

Suppose we have a weighted graph of this form:



Goal: Find a subset of vertices such that they do not share any edges with each other, and the sum of their weights (w_i) is maximized.



Algorithm (attempts)

Goal: Find a subset of vertices such that they do not share any edges with each other, and the sum of their weights (w_i) is maximized.

Ideas?

Algorithm (attempts)

Goal: Find a subset of vertices such that they do not share any edges with each other, and the sum of their weights (w_i) is maximized.

Algorithm 1: Greedily select the node with the largest weight that does not share an edge with previously selected nodes. Repeat.

Algorithm 2: Select the best between either (a) all of the odd-indexed vertices, or (b) all of the even-indexed vertices.

Algorithm (attempts)

Goal: Find a subset of vertices such that they do not share any edges with each other, and the sum of their weights (w_i) is maximized.

Algorithm 1: Greedily select the node with the largest weight that does not share an edge with previously selected nodes. Repeat.

Algorithm 2: Select the best between either (a) all of the odd-indexed vertices, or (b) all of the even-indexed vertices.

Optimality

Counterexamples?

Algorithm (attempts)

Goal: Find a subset of vertices such that they do not share any edges with each other, and the sum of their weights (w_i) is maximized.

Algorithm 1: Greedily select the node with the largest weight that does not share an edge with previously selected nodes. Repeat.



Algorithm 2: Select the best between either (a) all of the odd-indexed vertices, or (b) all of the even-indexed vertices.

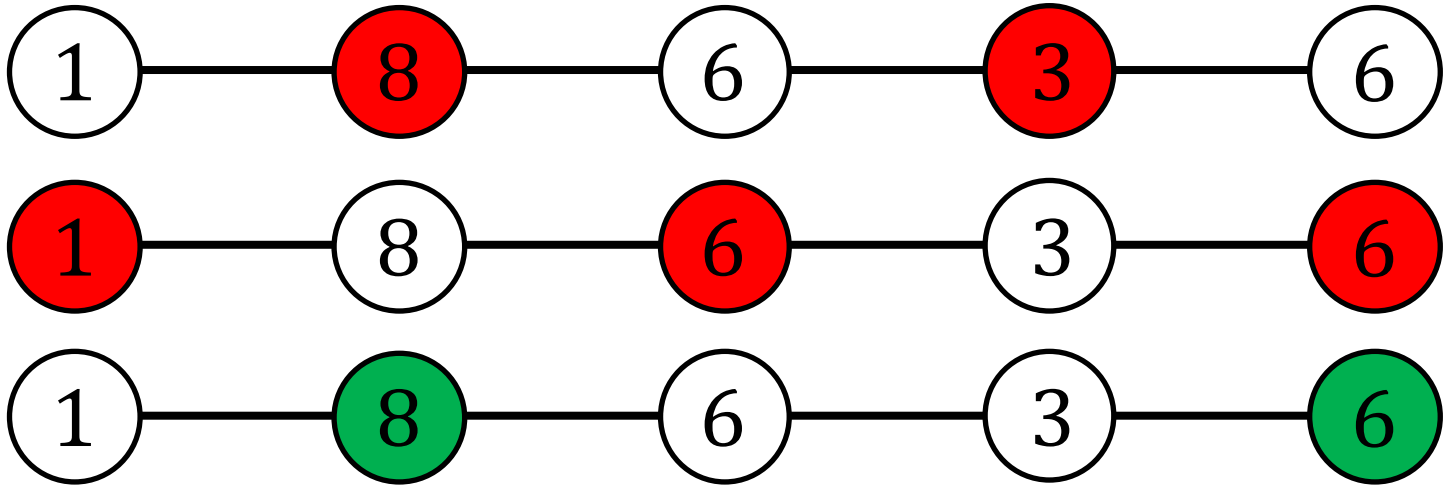
Algorithm (attempts)

Goal: Find a subset of vertices such that they do not share any edges with each other, and the sum of their weights (w_i) is maximized.

Algorithm 1: Greedily select the node with the largest weight that does not share an edge with previously selected nodes. Repeat.



Algorithm 2: Select the best between either (a) all of the odd-indexed vertices, or (b) all of the even-indexed vertices.



Dynamic Programming Algorithm

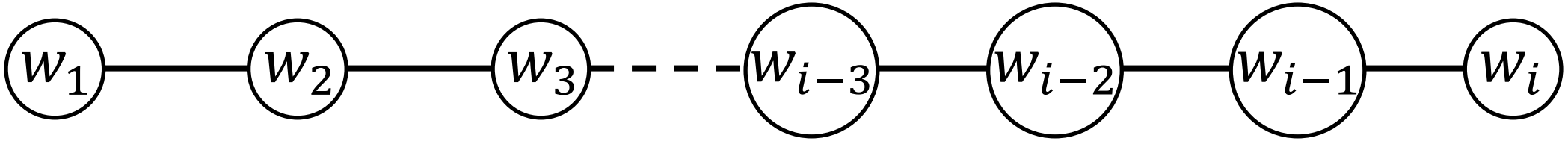
Input: n node graph, $G = (V, E)$, with vertex weights.

Output: Subset of vertices such that they do not share any edges with each other, and the sum of their weights (w_i) is maximized.

Dynamic Programming Algorithm

Input: n node graph, $G = (V, E)$, with vertex weights.

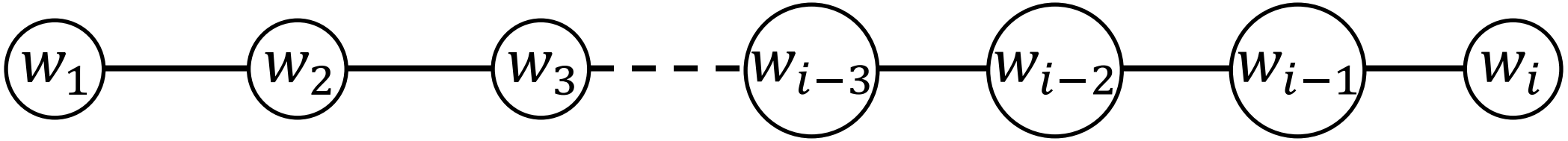
Output: Subset of vertices such that they do not share any edges with each other, and the sum of their weights (w_i) is maximized.



Dynamic Programming Algorithm

Input: n node graph, $G = (V, E)$, with vertex weights.

Output: Subset of vertices such that they do not share any edges with each other, and the sum of their weights (w_i) is maximized.



Is there an optimal substructure? Does an optimal solution to $V_i = \{v_1, \ldots, v_i\}$ imply something about an optimal solution to a sub problem?

Making Change

In general, suppose a country has denominations:

$$1 = d_1 < d_2 < \cdots < d_k$$

(US coins: $d_1 = 1, d_2 = 5, d_3 = 10, d_4 = 25$)

$C(p)$ – minimum number of coins to make p cents.

x – value (e.g. \$0.25) of a coin used in the optimal solution.

So, we'll check all possibilities.

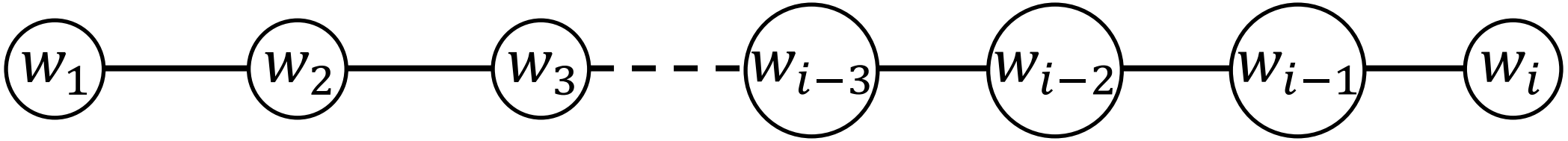
But we don't know what actual coin is in the optimal solution.

$$C(p) = \min_{d_i \leq p} C(p - d_i) + 1$$

Dynamic Programming Algorithm

Input: n node graph, $G = (V, E)$, with vertex weights.

Output: Subset of vertices such that they do not share any edges with each other, and the sum of their weights (w_i) is maximized.

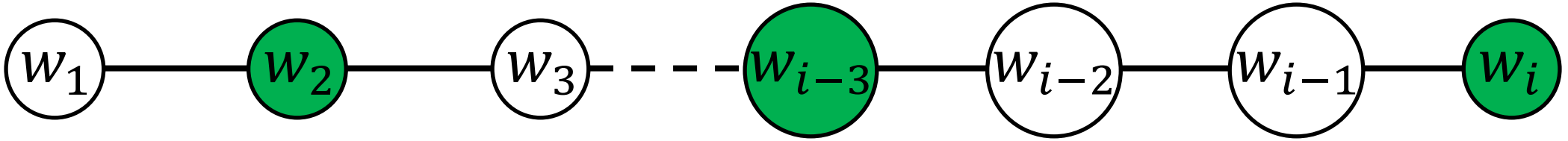


Is there an optimal substructure? Does an optimal solution to $V_i = \{v_1, \ldots, v_i\}$ imply something about an optimal solution to a sub problem?

Dynamic Programming Algorithm

Input: n node graph, $G = (V, E)$, with vertex weights.

Output: Subset of vertices such that they do not share any edges with each other, and the sum of their weights (w_i) is maximized.



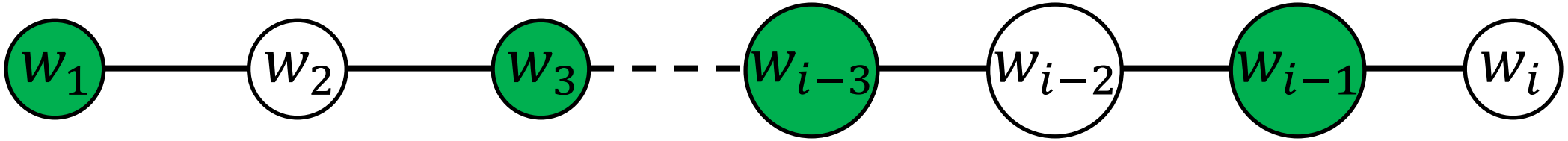
Is there an optimal substructure? Does an optimal solution to $V_i = \{v_1, \ldots, v_i\}$ imply something about an optimal solution to a sub problem?

Vertex v_i is in the optimal solution to V_i or it's not.

Dynamic Programming Algorithm

Input: n node graph, $G = (V, E)$, with vertex weights.

Output: Subset of vertices such that they do not share any edges with each other, and the sum of their weights (w_i) is maximized.



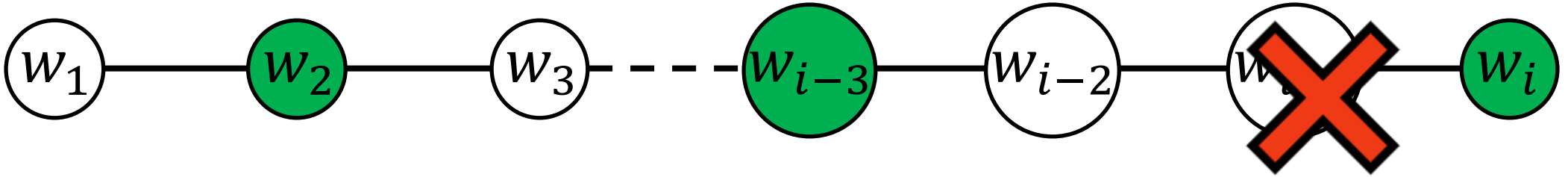
Is there an optimal substructure? Does an optimal solution to $V_i = \{v_1, \ldots, v_i\}$ imply something about an optimal solution to a sub problem?

Vertex v_i is in the optimal solution to V_i or it's not.

Dynamic Programming Algorithm

Input: n node graph, $G = (V, E)$, with vertex weights.

Output: Subset of vertices such that they do not share any edges with each other, and the sum of their weights (w_i) is maximized.



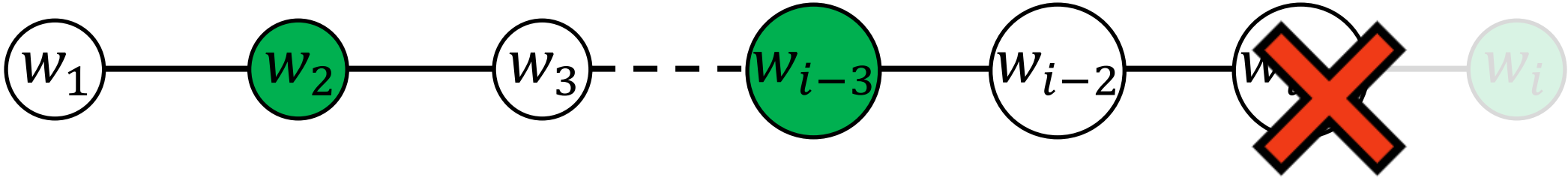
Is there an optimal substructure? Does an optimal solution to $V_i = \{v_1, \ldots, v_i\}$ imply something about an optimal solution to a sub problem?

Vertex v_i is in the optimal solution to V_i or it's not. If it is, …

Dynamic Programming Algorithm

Input: n node graph, $G = (V, E)$, with vertex weights.

Output: Subset of vertices such that they do not share any edges with each other, and the sum of their weights (w_i) is maximized.



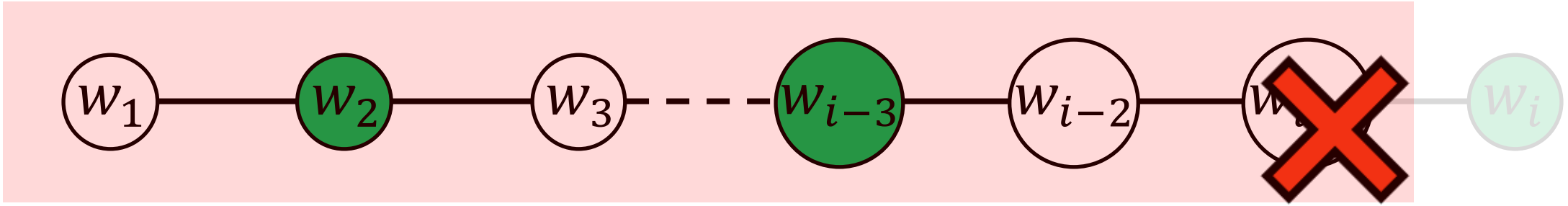
Is there an optimal substructure? Does an optimal solution to $V_i = \{v_1, \ldots, v_i\}$ imply something about an optimal solution to a sub problem?

Vertex v_i is in the optimal solution to V_i or it's not. If it is, then removing it must provide an optimal solution to ??

Dynamic Programming Algorithm

Input: n node graph, $G = (V, E)$, with vertex weights.

Output: Subset of vertices such that they do not share any edges with each other, and the sum of their weights (w_i) is maximized.



Is there an optimal substructure? Does an optimal solution to $V_i = \{v_1, \ldots, v_i\}$ imply something about an optimal solution to a sub problem?

Vertex v_i is in the optimal solution to V_i or it's not. If it is, then removing it must provide an optimal solution to ??

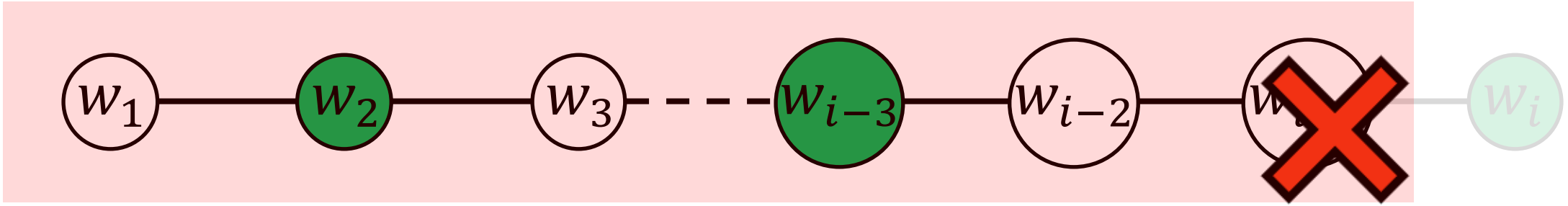
$$\boldsymbol{V_{i-1}?}$$

Dynamic Programming Algo

Input: n node graph, $G = (V, E)$, with ver

Output: Subset of vertices such that they d
with each other, and the sum of their weights (w_i) is maximized.

We can't guarantee what remains is an optimal solution to V_{i-1}, because vertex v_{i-1} might be in the optimal solution to V_{i-1}.



Is there an optimal substructure? Does an optimal solution to $V_i = \{v_1, \ldots, v_i\}$ imply something about an optimal solution to a sub problem?

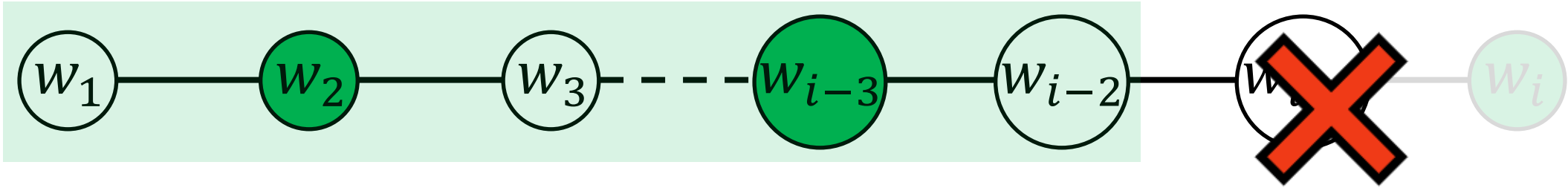
Vertex v_i is in the optimal solution to V_i or it's not. If it is, then removing it must provide an optimal solution to ??

$$\boldsymbol{V_{i-1}?}$$

Dynamic Programming Algorithm

Input: n node graph, $G = (V, E)$, with vertex weights.

Output: Subset of vertices such that they do not share any edges with each other, and the sum of their weights (w_i) is maximized.



Is there an optimal substructure? Does an optimal solution to $V_i = \{v_1, \ldots, v_i\}$ imply something about an optimal solution to a sub problem?

Vertex v_i is in the optimal solution to V_i or it's not. If it is, then removing it must provide an optimal solution to ??

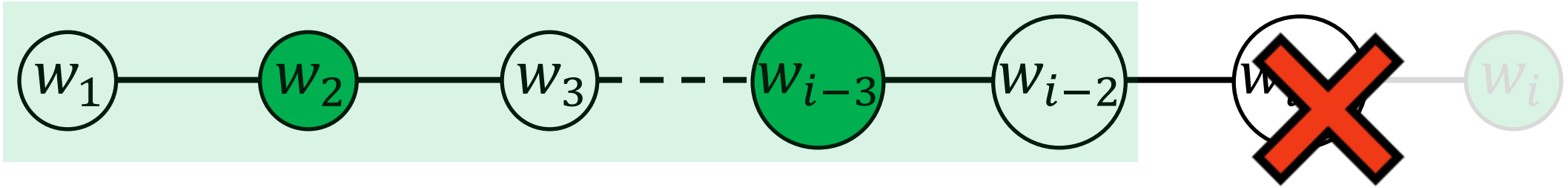
$\boldsymbol{V_{i-2}}$**?**

Dynamic Programming Algo

Input: n node graph, $G = (V, E)$, with ver

Output: Subset of vertices such that they d
with each other, and the sum of their weights (w_i) is maximized.

If what remains is not an optimal solution to V_{i-2}, then the optimal solution to V_{i-2} plus vertex v_i would provide a better solution to V_i.



Is there an optimal substructure? Does an optimal solution to $V_i = \{v_1, \ldots, v_i\}$ imply something about an optimal solution to a sub problem?

Vertex v_i is in the optimal solution to V_i or it's not. If it is, then removing it must provide an optimal solution to ??

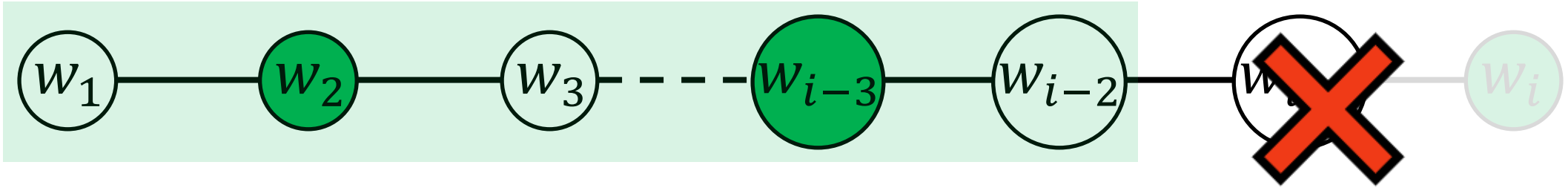
$$\boldsymbol{V_{i-2}?}$$

Dynamic Programming Algo

Input: n node graph, $G = (V, E)$, with ver

Output: Subset of vertices such that they d
with each other, and the sum of their weights (w_i) is maximized.

If what remains is not an optimal solution to V_{i-2}, then the optimal solution to V_{i-2} plus vertex v_i would provide a better solution to V_i.



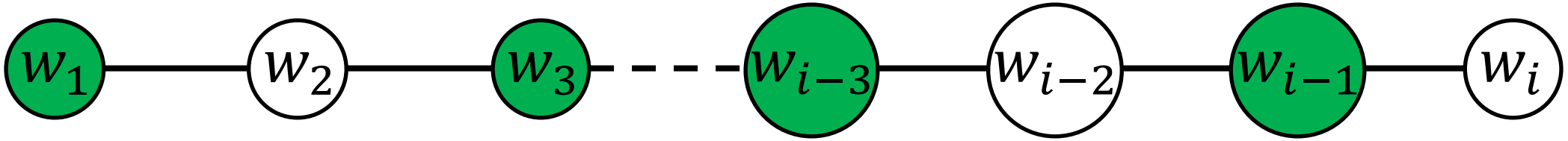
Is there an optimal substructure? Does an optimal solution to $V_i = \{v_1, \ldots, v_i\}$ imply something about an optimal solution to a sub problem?

Vertex v_i is in the optimal solution to V_i or it's not. If it is, then removing it must provide an optimal solution to V_{i-2}.

Dynamic Programming Algorithm

Input: n node graph, $G = (V, E)$, with vertex weights.

Output: Subset of vertices such that they do not share any edges with each other, and the sum of their weights (w_i) is maximized.



Is there an optimal substructure? Does an optimal solution to $V_i = \{v_1, \ldots, v_i\}$ imply something about an optimal solution to a sub problem?

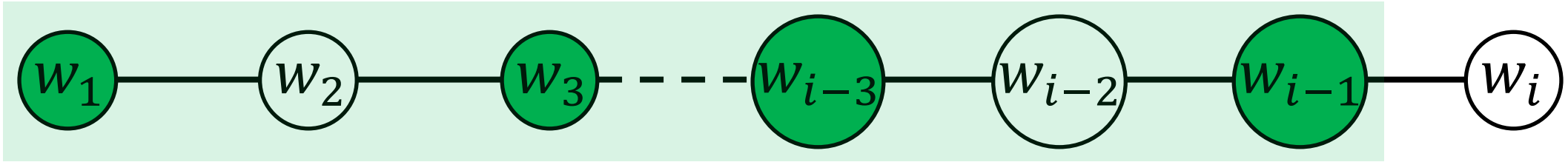
Vertex v_i is in the optimal solution to V_i or it's not. If it is, then removing it must provide an optimal solution to V_{i-2}. If it is not,

Dynamic Programming Algorithm

Input: n node graph, $G = (V, E)$, with

Output: Subset of vertices such that th
with each other, and the sum of their weights (w_i) is maximized.

If what remains is not an optimal solution to V_{i-1}, then optimal the solution to V_{i-1} would provide a better solution to V_i, while still ignoring vertex v_i.



Is there an optimal substructure? Does an optimal solution to $V_i = \{v_1, \ldots, v_i\}$ imply something about an optimal solution to a sub problem?

Vertex v_i is in the optimal solution to V_i or it's not. If it is, then removing it must provide an optimal solution to V_{i-2}. If it is not, then the optimal solution to V_i must be the same as the optimal solution to V_{i-1}.

Dynamic Programming Algorithm

Input: n node graph, $G = (V, E)$, with vertex weights.

Output: Subset of vertices such that they do not share any edges with each other, and the sum of their weights (w_i) is maximized.

Optimal solution to V_i is either the optimal for V_{i-1} (with vertex v_i excluded), or the optimal for V_{i-2} (with vertex v_i included).

Dynamic Programming Algorithm

Input: n node graph, $G = (V, E)$, with vertex weights.

Output: Subset of vertices such that they do not share any edges with each other, and the sum of their weights (w_i) is maximized.

Optimal solution to V_i is either the optimal for V_{i-1} (with vertex v_i excluded), or the optimal for V_{i-2} (with vertex v_i included).

Let $A[i]$ be the weight of the optimal solution for V_i.

$A[i] = ??$

Dynamic Programming Algorithm

Input: n node graph, $G = (V, E)$, with vertex weights.

Output: Subset of vertices such that they do not share any edges with each other, and the sum of their weights (w_i) is maximized.

Optimal solution to V_i is either the optimal for V_{i-1} (with vertex v_i excluded), or the optimal for V_{i-2} (with vertex v_i included).

Let $A[i]$ be the weight of the optimal solution for V_i.

$$A[i] = \max \begin{pmatrix} A[i-1] \\ A[i-2] + w_i \end{pmatrix}$$

Code

```
maxWeight(G = (V, E))

    ?
```

Input: n node graph, $G = (V, E)$, with vertex weights.

Output: Weight of the heaviest vertex subset that do not share edges.

$A[i]$ – weight of the optimal solution for V_i.

$$A[i] = \max \begin{pmatrix} A[i-1] \\ A[i-2] + w_i \end{pmatrix}$$

Code

Input: n node graph, $G = (V, E)$, with vertex weights.

Output: Weight of the heaviest vertex subset that do not share edges.

```
maxWeight(G = (V, E))
    A[0,...,n] = [0,0,0,...,0]
```

$A[i]$ – weight of the optimal solution for V_i.

$$A[i] = \max \begin{pmatrix} A[i-1] \\ A[i-2] + w_i \end{pmatrix}$$

Code

Input: n node graph, $G = (V, E)$, with vertex weights.

Output: Weight of the heaviest vertex subset that do not share edges.

```
maxWeight(G = (V, E))
    A[0,...,n] = [0,w_1,0,...,0]
```

$A[i]$ – weight of the optimal solution for V_i.

$$A[i] = \max \begin{pmatrix} A[i-1] \\ A[i-2] + w_i \end{pmatrix}$$

Code

Input: n node graph, $G = (V, E)$, with vertex weights.

Output: Weight of the heaviest vertex subset that do not share edges.

```
maxWeight(G = (V, E))
    A[0,...,n] = [0,w_1,0,...,0]
    for i = 2,...,n
```

$A[i]$ – weight of the optimal solution for V_i.

$$A[i] = \max\begin{pmatrix} A[i-1] \\ A[i-2] + w_i \end{pmatrix}$$

Code

Input: n node graph, $G = (V, E)$, with vertex weights.

Output: Weight of the heaviest vertex subset that do not share edges.

```
maxWeight(G = (V, E))
    A[0,...,n] = [0,w_1,0,...,0]
    for i = 2,...,n
      A[i] = max(A[i - 1], A[i - 2] + w_i)
```

$A[i]$ – weight of the optimal solution for V_i.

$$A[i] = \max\begin{pmatrix} A[i-1] \\ A[i-2] + w_i \end{pmatrix}$$

Code

Input: n node graph, $G = (V, E)$, with vertex weights.

Output: Weight of the heaviest vertex subset that do not share edges.

```
maxWeight(G = (V, E))
    A[0,...,n] = [0,w_1,0,...,0]
    for i = 2,...,n
      A[i] = max(A[i - 1], A[i - 2] + w_i)
    return A[n]
```

$A[i]$ – weight of the optimal solution for V_i.

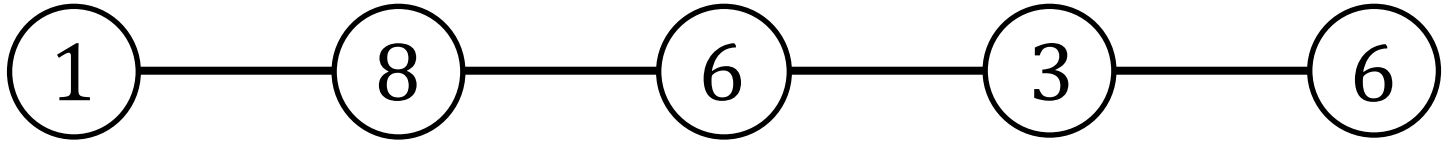
$$A[i] = \max\begin{pmatrix} A[i-1] \\ A[i-2] + w_i \end{pmatrix}$$

Code

Input: n node graph, $G = (V, E)$, with vertex weights.

Output: Weight of the heaviest vertex subset that do not share edges.

```
maxWeight(G = (V, E))
    A[0,...,n] = [0,w_1,0,...,0]
    for i = 2,...,n
      A[i] = max(A[i - 1], A[i - 2] + w_i)
    return A[n]
```



$A[i]$ – weight of the optimal solution for V_i.

$$A[i] = \max\begin{pmatrix} A[i-1] \\ A[i-2] + w_i \end{pmatrix}$$

A[n] =

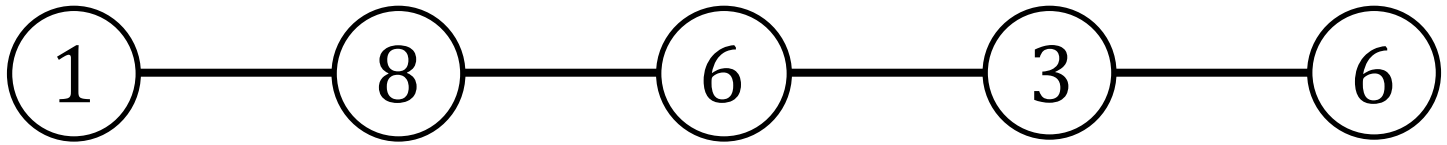
0	1	2	3	4	5
0	0	0	0	0	0

Code

Input: n node graph, $G = (V, E)$, with vertex weights.

Output: Weight of the heaviest vertex subset that do not share edges.

```
maxWeight(G = (V, E))
    A[0,...,n] = [0,w_1,0,...,0]
    for i = 2,...,n
      A[i] = max(A[i - 1], A[i - 2] + w_i)
    return A[n]
```



$A[i]$ – weight of the optimal solution for V_i.

$$A[i] = \max\begin{pmatrix} A[i-1] \\ A[i-2] + w_i \end{pmatrix}$$

A[n] =

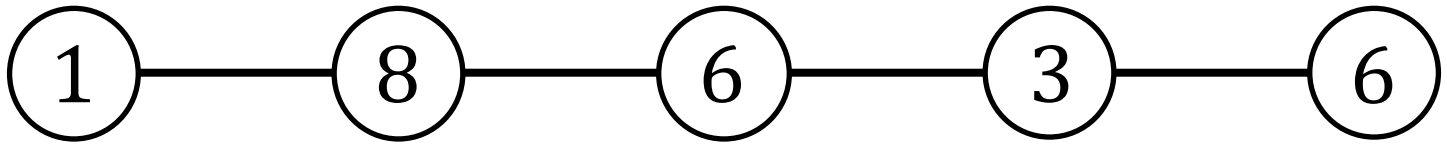
0	1	2	3	4	5
0	1	0	0	0	0

Code

Input: n node graph, $G = (V, E)$, with vertex weights.

Output: Weight of the heaviest vertex subset that do not share edges.

```
maxWeight(G = (V, E))
    A[0,...,n] = [0,w1,0,...,0]
    for i = 2,...,n
      A[i] = max(A[i - 1], A[i - 2] + wi)
    return A[n]
```



$A[i]$ – weight of the optimal solution for V_i.

	0	1	2	3	4	5
A[n] =	0	1	8	0	0	0

$$\max\begin{bmatrix}1\\0+8\end{bmatrix}$$

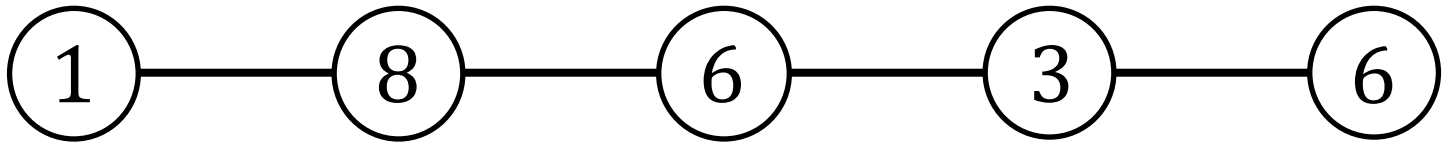
$$A[i] = \max\begin{pmatrix}A[i-1]\\A[i-2]+w_i\end{pmatrix}$$

Code

Input: n node graph, $G = (V, E)$, with vertex weights.

Output: Weight of the heaviest vertex subset that do not share edges.

```
maxWeight(G = (V, E))
    A[0,...,n] = [0,w_1,0,...,0]
    for i = 2,...,n
      A[i] = max(A[i - 1], A[i - 2] + w_i)
    return A[n]
```



$A[i]$ – weight of the optimal solution for V_i.

	0	1	2	3	4	5
A[n] =	0	1	8	8	0	0

$$\max\begin{bmatrix} 8 \\ 1+6 \end{bmatrix}$$

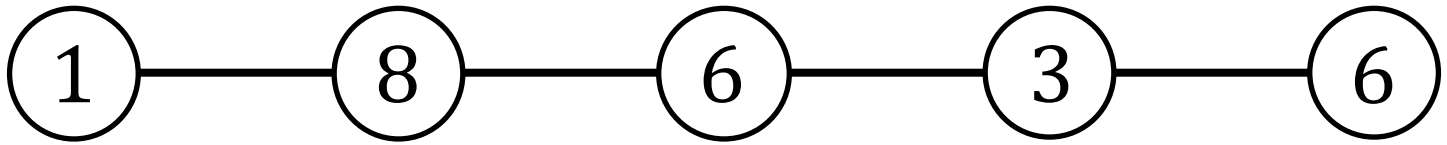
$$A[i] = \max\begin{pmatrix} A[i-1] \\ A[i-2] + w_i \end{pmatrix}$$

Code

Input: n node graph, $G = (V, E)$, with vertex weights.

Output: Weight of the heaviest vertex subset that do not share edges.

```
maxWeight(G = (V, E))
    A[0,...,n] = [0,w_1,0,...,0]
    for i = 2,...,n
      A[i] = max(A[i - 1], A[i - 2] + w_i)
    return A[n]
```



$A[i]$ – weight of the optimal solution for V_i.

	0	1	2	3	4	5
A[n] =	0	1	8	8	11	0

$$\max\begin{bmatrix} 8 \\ 8+3 \end{bmatrix}$$

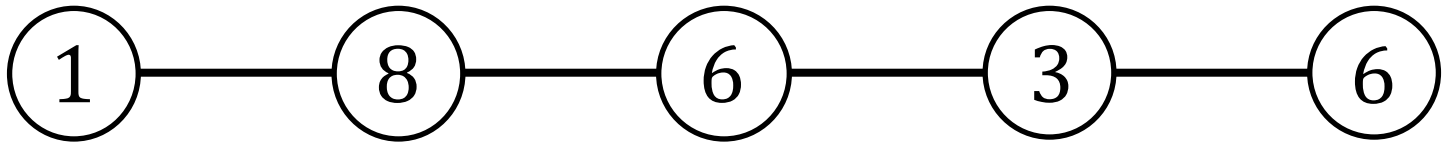
$$A[i] = \max\begin{pmatrix} A[i-1] \\ A[i-2] + w_i \end{pmatrix}$$

Code

Input: n node graph, $G = (V, E)$, with vertex weights.

Output: Weight of the heaviest vertex subset that do not share edges.

```
maxWeight(G = (V, E))
    A[0,...,n] = [0,w1,0,...,0]
    for i = 2,...,n
      A[i] = max(A[i - 1], A[i - 2] + wi)
    return A[n]
```



$A[i]$ – weight of the optimal solution for V_i.

	0	1	2	3	4	5
A[n] =	0	1	8	8	11	14

$$\max\begin{bmatrix}\mathbf{11}\\ \mathbf{8+6}\end{bmatrix}$$

$$A[i] = \max\begin{pmatrix}A[i-1]\\ A[i-2]+w_i\end{pmatrix}$$

Code

Input: n node graph, $G = (V, E)$, with vertex weights.

Output: Weight of the heaviest vertex subset that do not share edges.

```
maxWeight(G = (V, E))
    A[0,...,n] = [0,w_1,0,...,0]
    for i = 2,...,n
      A[i] = max(A[i - 1], A[i - 2] + w_i)
    return A[n]
```

$A[i]$ – weight of the optimal solution for V_i.

$$A[i] = \max\begin{pmatrix} A[i-1] \\ A[i-2] + w_i \end{pmatrix}$$

Running time $\in$??

Code

Input: n node graph, $G = (V, E)$, with vertex weights.

Output: Weight of the heaviest vertex subset that do not share edges.

```
maxWeight(G = (V, E))
    A[0,...,n] = [0,w_1,0,...,0]
    for i = 2,...,n
      A[i] = max(A[i - 1], A[i - 2] + w_i)
    return A[n]
```

$A[i]$ – weight of the optimal solution for V_i.

$$A[i] = \max\begin{pmatrix} A[i-1] \\ A[i-2] + w_i \end{pmatrix}$$

Running time $\in O(n)$

Backtracking

Input: n node graph, $G = (V, E)$, with vertex weights.

Output: Weight of the heaviest vertex subset that do not share edges.

```
maxWeight(G = (V, E))
    A[0,...,n] = [0,w1,0,...,0]
    for i = 2,...,n
      A[i] = max(A[i - 1], A[i - 2] + wi)
    return A[n]
```

Find the actual nodes in the heaviest vertex set?

Running time $\in O(n)$

$A[i]$ – weight of the optimal solution for V_i.

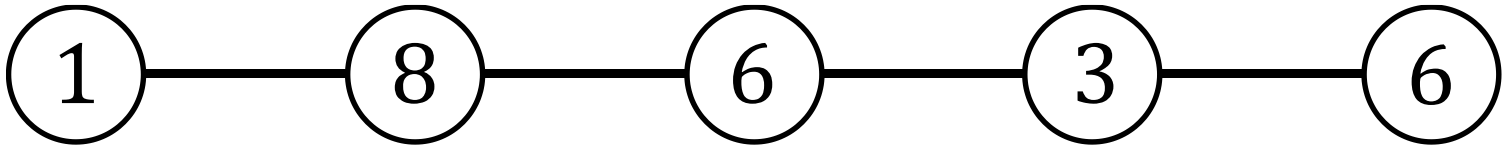
$$A[i] = \max \begin{pmatrix} A[i-1] \\ A[i-2] + w_i \end{pmatrix}$$

Backtracking

Input: n node graph, $G = (V, E)$, with vertex weights.

Output: Weight of the heaviest vertex subset that do not share edges.

```
maxWeight(G = (V, E))
    A[0,...,n] = [0,w1,0,...,0]
    for i = 2,...,n
      A[i] = max(A[i − 1], A[i − 2] + wi)
    return A[n]
```



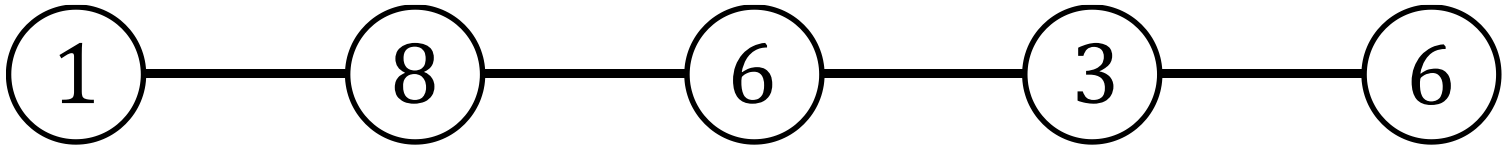
0	1	2	3	4	5
0	1	8	8	11	14

Backtracking

Input: n node graph, $G = (V, E)$, with vertex weights.

Output: Weight of the heaviest vertex subset that do not share edges.

```
maxWeight(G = (V, E))
    A[0,...,n] = [0,w1,0,...,0]
    for i = 2,...,n
      A[i] = max(A[i − 1], A[i − 2] + wi)
    return A[n]
```



0	1	2	3	4	5
0	1	8	8	11	14

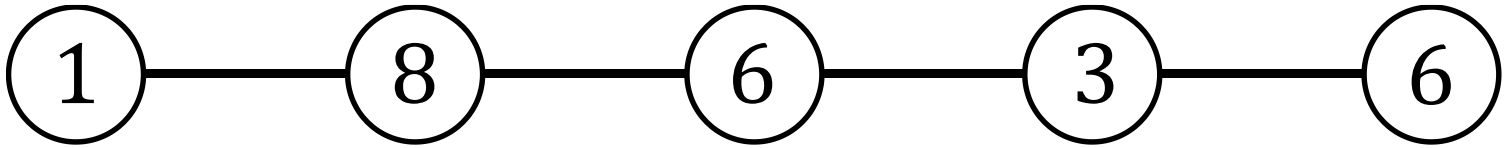
At A[i], how do you know if vertex i was added?

Backtracking

Input: n node graph, $G = (V, E)$, with vertex weights.

Output: Weight of the heaviest vertex subset that do not share edges.

```
maxWeight(G = (V, E))
    A[0,...,n] = [0,w1,0,...,0]
    for i = 2,...,n
      A[i] = max(A[i - 1], A[i - 2] + wi)
    return A[n]
```



0	1	2	3	4	5
0	1	8	8	11	14

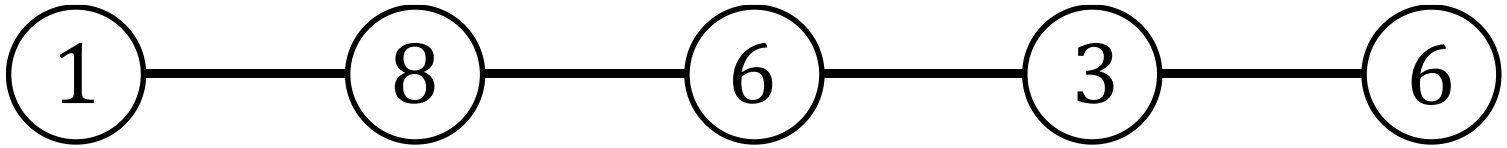
At A[i], how do you know if vertex i was added?

Backtracking

Input: n node graph, $G = (V, E)$, with vertex weights.

Output: Weight of the heaviest vertex subset that do not share edges.

```
maxWeight(G = (V, E))
    A[0,...,n] = [0,w1,0,...,0]
    for i = 2,...,n
      A[i] = max(A[i - 1], A[i - 2] + wi)
    return A[n]
```



0	1	2	3	4	5
0	1	8	8	11	14

At A[i], how do you know if vertex i was added?

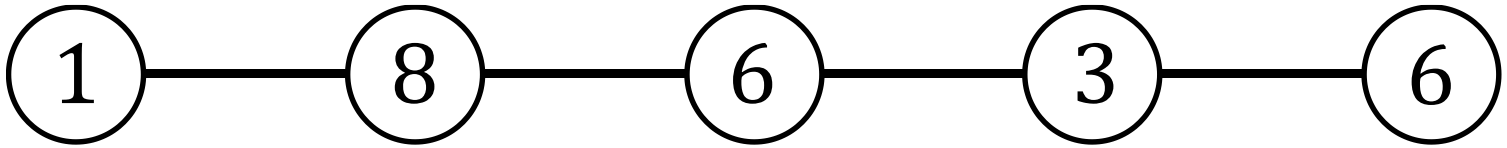
Backtracking

Input: n node graph, $G = (V, E)$, with vertex weights.

Output: Weight of the heaviest vertex subset that do not share edges.

```
maxWeight(G = (V, E))
    A[0,...,n] = [0,w1,0,...,0]
    for i = 2,...,n
      A[i] = max(A[i - 1], A[i - 2] + wi)
    return A[n]
```

5




```
while i > 0
  if A[i] > A[i-1]
    print(i)
```

0	1	2	3	4	5
0	1	8	8	11	14

At A[i], how do you know if vertex i was added?

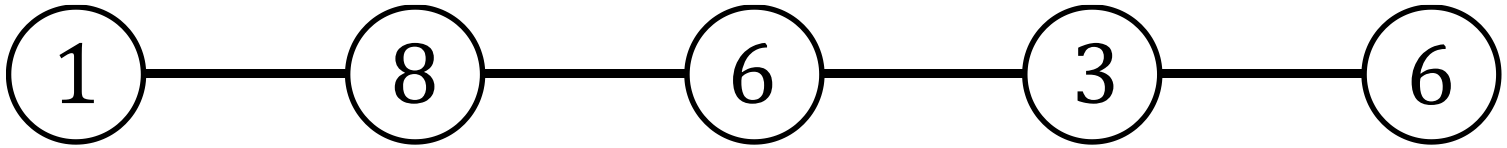
Backtracking

Input: n node graph, $G = (V, E)$, with vertex weights.

Output: Weight of the heaviest vertex subset that do not share edges.

```
maxWeight(G = (V, E))
    A[0,...,n] = [0,w1,0,...,0]
    for i = 2,...,n
      A[i] = max(A[i - 1], A[i - 2] + wi)
    return A[n]
```

5




```
while i > 0
  if A[i] > A[i-1]
    print(i)
    ??
```

0	1	2	3	4	5
0	1	8	8	11	14

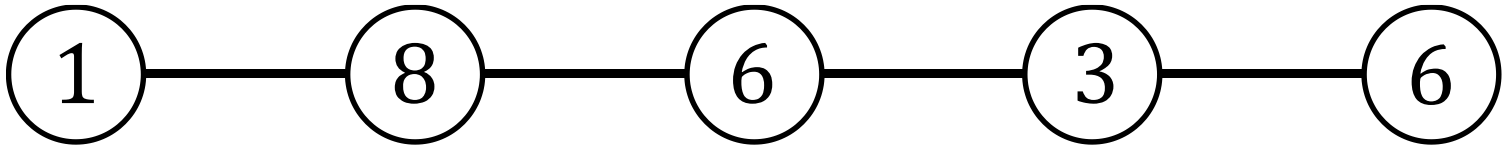
Backtracking

Input: n node graph, $G = (V, E)$, with vertex weights.

Output: Weight of the heaviest vertex subset that do not share edges.

```
maxWeight(G = (V, E))
    A[0,...,n] = [0,w1,0,...,0]
    for i = 2,...,n
      A[i] = max(A[i - 1], A[i - 2] + wi)
    return A[n]
```

5



0	1	2	3	4	5
0	1	8	8	11	14

```
while i > 0
  if A[i] > A[i-1]
    print(i)
    i = i - 2
```

We are not adding i−2, just considering it.

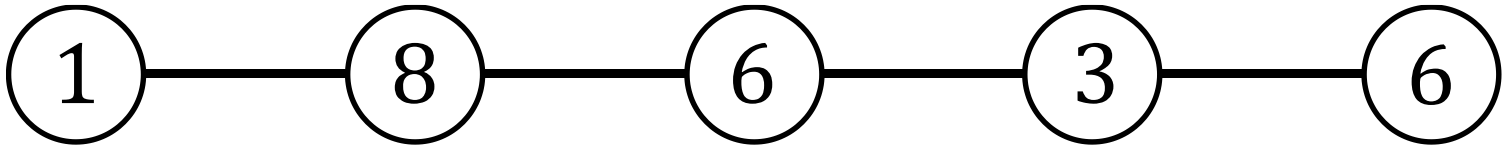
Backtracking

Input: n node graph, $G = (V, E)$, with vertex weights.

Output: Weight of the heaviest vertex subset that do not share edges.

```
maxWeight(G = (V, E))
    A[0,...,n] = [0,w1,0,...,0]
    for i = 2,...,n
      A[i] = max(A[i - 1], A[i - 2] + wi)
    return A[n]
```

5



0	1	2	3	4	5
0	1	8	8	11	14

```
while i > 0
  if A[i] > A[i-1]
    print(i)
  i = i - 2
```

We are not adding `i-2`, just considering it. Did we add it?

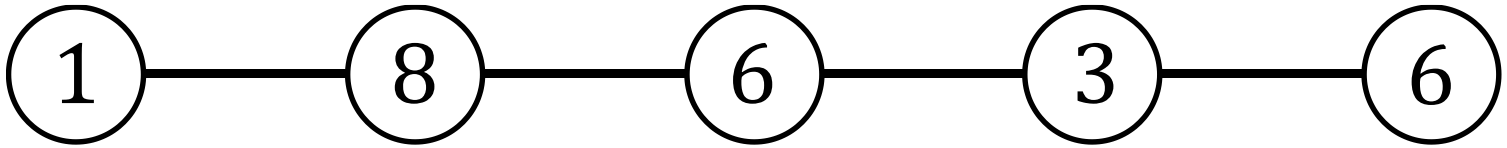
Backtracking

Input: n node graph, $G = (V, E)$, with vertex weights.

Output: Weight of the heaviest vertex subset that do not share edges.

```
maxWeight(G = (V, E))
    A[0,...,n] = [0,w1,0,...,0]
    for i = 2,...,n
      A[i] = max(A[i - 1], A[i - 2] + wi)
    return A[n]
```

5,2



0	1	2	3	4	5
0	1	8	8	11	14

```
while i > 0
  if A[i] > A[i-1]
    print(i)
    i = i - 2
  else
    i = i - 1
```

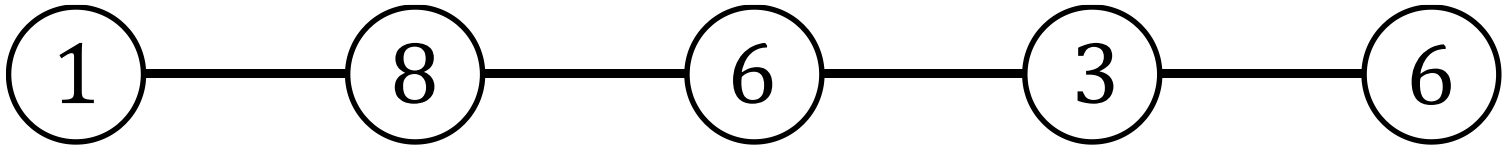
Backtracking

Input: n node graph, $G = (V, E)$, with vertex weights.

Output: Weight of the heaviest vertex subset that do not share edges.

```
maxWeight(G = (V, E))
    A[0,...,n] = [0,w1,0,...,0]
    for i = 2,...,n
      A[i] = max(A[i - 1], A[i - 2] + wi)
    return A[n]
```

5,2



0	1	2	3	4	5
0	1	8	8	11	14

```
while i > 0
  if A[i] > A[i-1]
    print(i)
    i = i - 2
  else
    i = i - 1
```