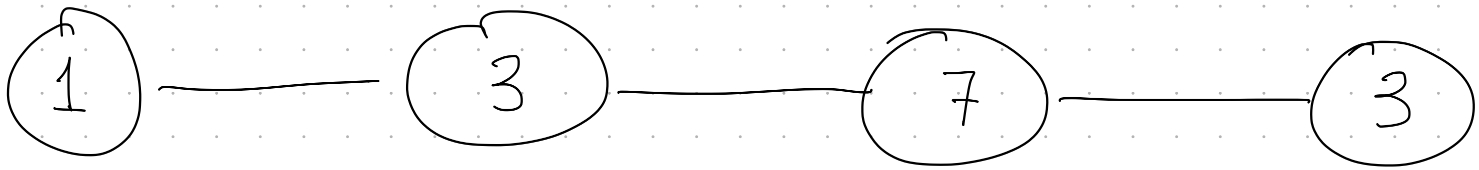
# 7 February 2024 — Dynamic Programming

## Plan

* Weighted Independent Set on a Path
  ↳ Greedy Attempts & Failures
* Announcements
* Dynamic Programming
  ↳ Recursive Formulation
  ↳ Memoization
  ↳ Iterative Reformulation

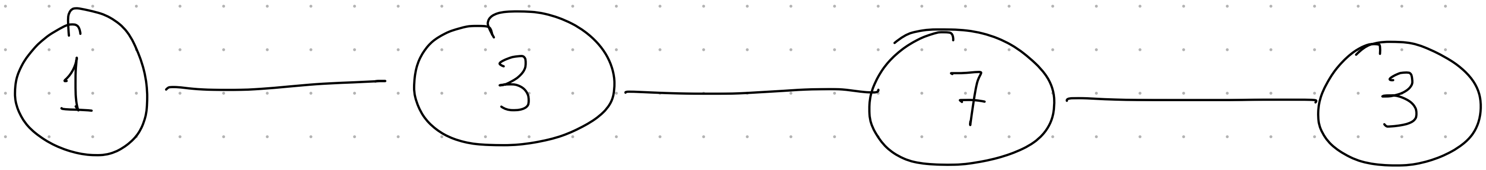
Given a path graph w/ vertex wts




Find "independent set" $S \subseteq V$ of vertices of maximum weight

$$W_S = \sum_{v \in S} w_v$$

Given a path graph w/ vertex wts




Find "independent set" $S \subseteq V$ of vertices of maximum weight

where no two vertices share an edge

$$W_S = \sum_{v \in S} w_v$$

Given a path graph w/ vertex wts

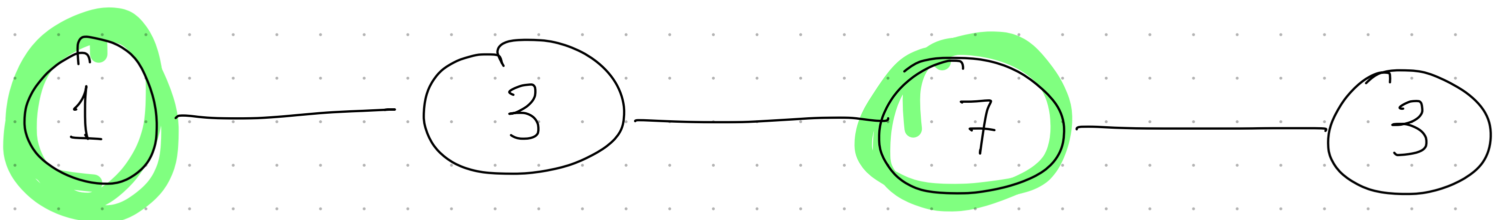

Find "independent set" $S \subseteq V$ of vertices of maximum weight

where no two vertices share an edge

$$W_S = \sum_{v \in S} w_v$$

$$S = \{v_2, v_4\} \qquad W_S = 6$$

Given a path graph w/ vertex wts



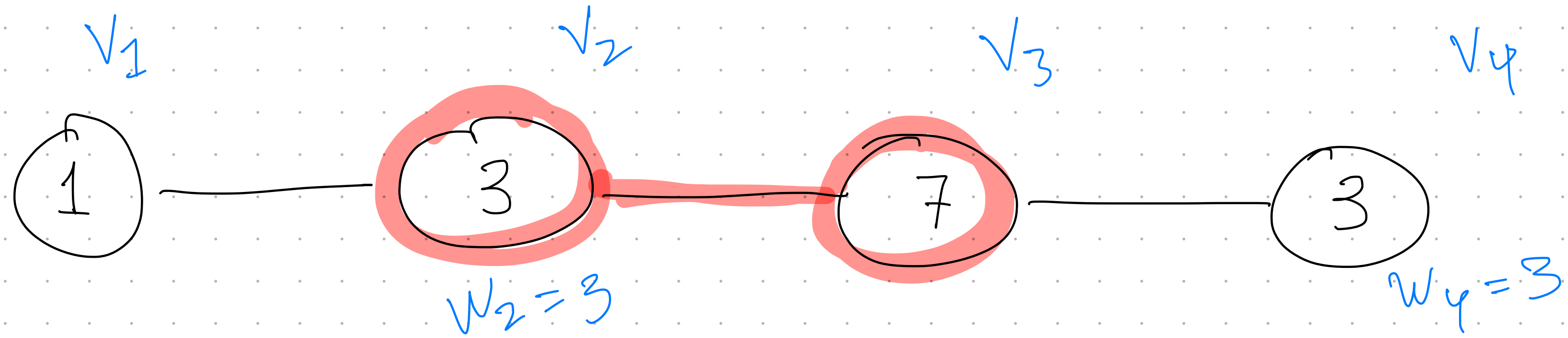

Find "independent set" $S \subseteq V$ of vertices of maximum weight

where no two vertices share an edge

$$W_S = \sum_{v \in S} w_v$$

$S = \{v_1, v_3\}$ $\quad W_S = 8$

Given a path graph w/ vertex wts



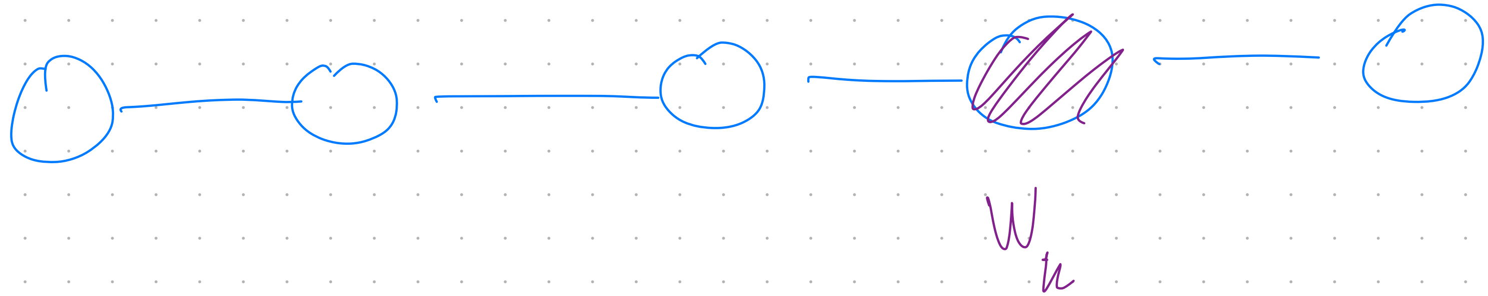

Find "independent set" $S \subseteq V$ of vertices of maximum weight

where no two vertices share an edge

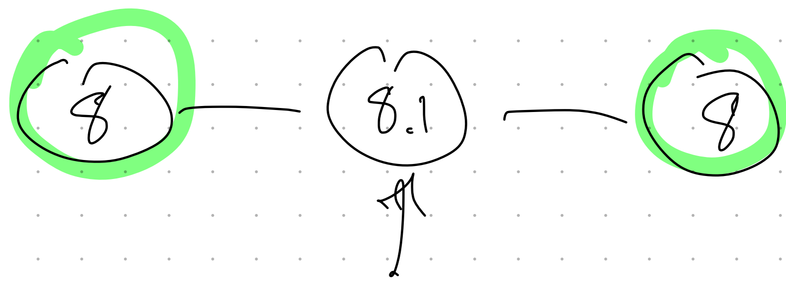
$$W_S = \sum_{v \in S} w_v$$

$S = \{v_2, v_3\}$ Not an independent set!

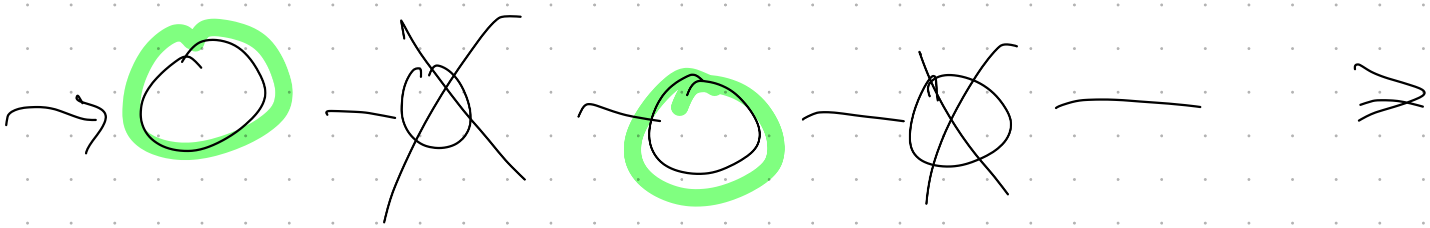
# Greedy Algorithms?




Maximum wt first.




4 — 9 — 4



# Greedy Algorithms?

- → Greedy attempts fail.
- → Weights mess things up.
- → Need a global understanding of solution

# Greedy Algorithms?

→ Greedy attempts fail.

→ Weights mess things up.

→ Need a global understanding of solution

$\boxed{\text{Dynamic Programming}}$

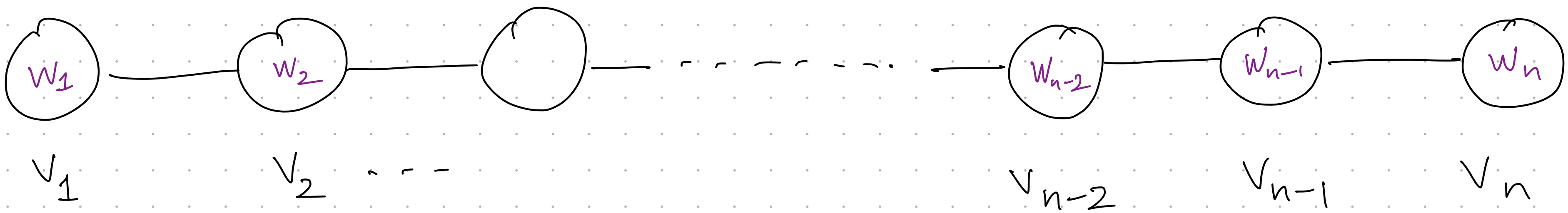
# Announcements

* HW1 Solutions posted to Canvas
  ↳ Graded assignment next week.

* HW2 due Tomorrow 11:59 pm

* Lecture Notes posted to site
  (This material Not in KT)

# Properties of an optimal solution

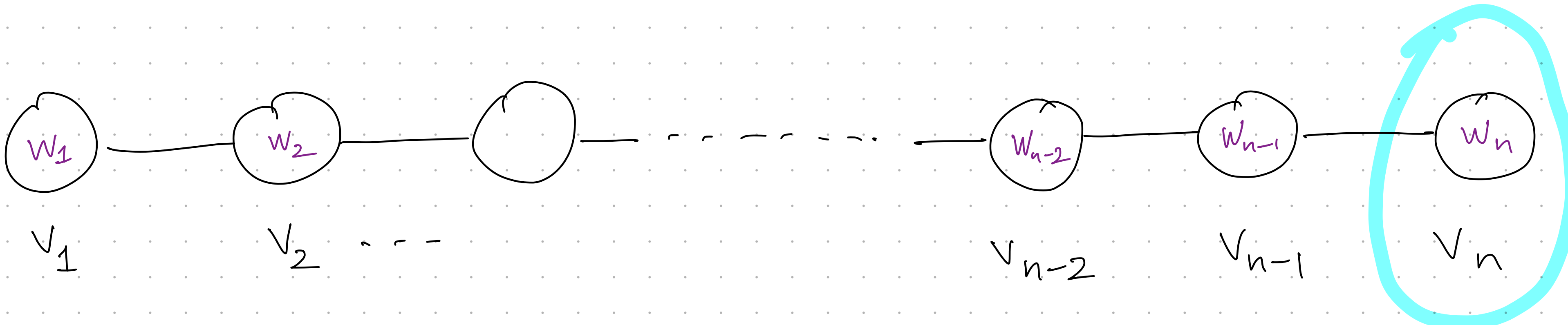



$S \subseteq V$ s.t. $\underbrace{\sum_{v \in S} w_v}_{\text{maximized}}$

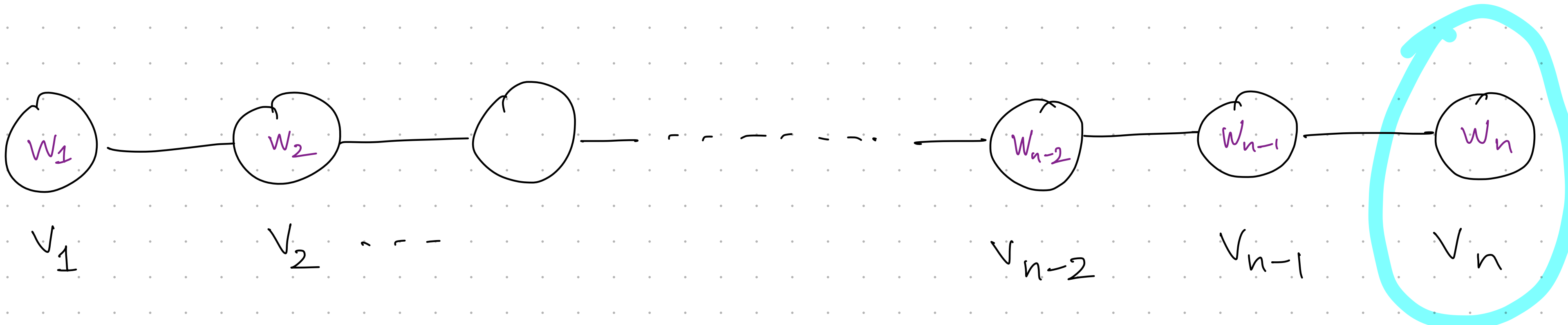
No $u, v \in S$

$(u, v) \in E$

# Properties of an optimal solution



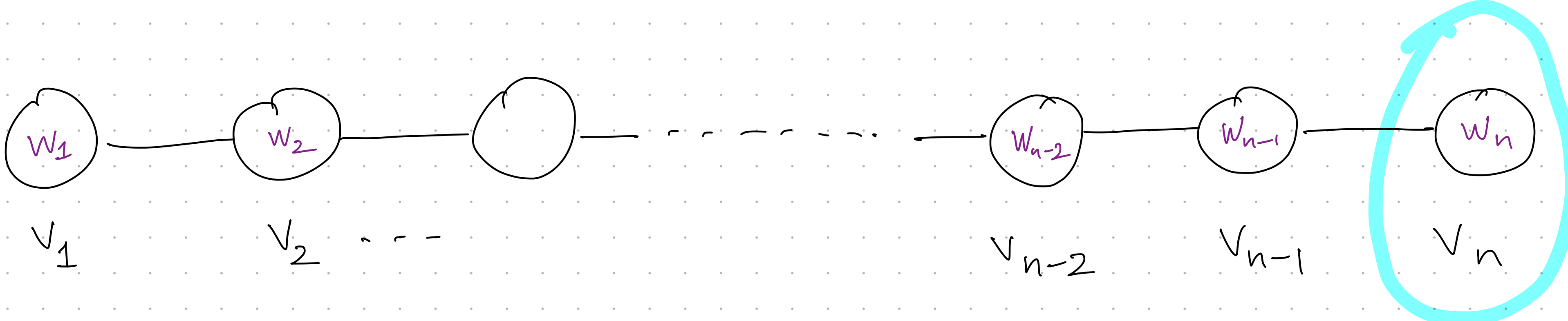
# Properties of an optimal solution




**Fact.** Fix any max wt. Independent Set $S \subseteq V$.

$$v_n \in S \quad \text{OR} \quad v_n \notin S.$$

# Properties of an optimal solution

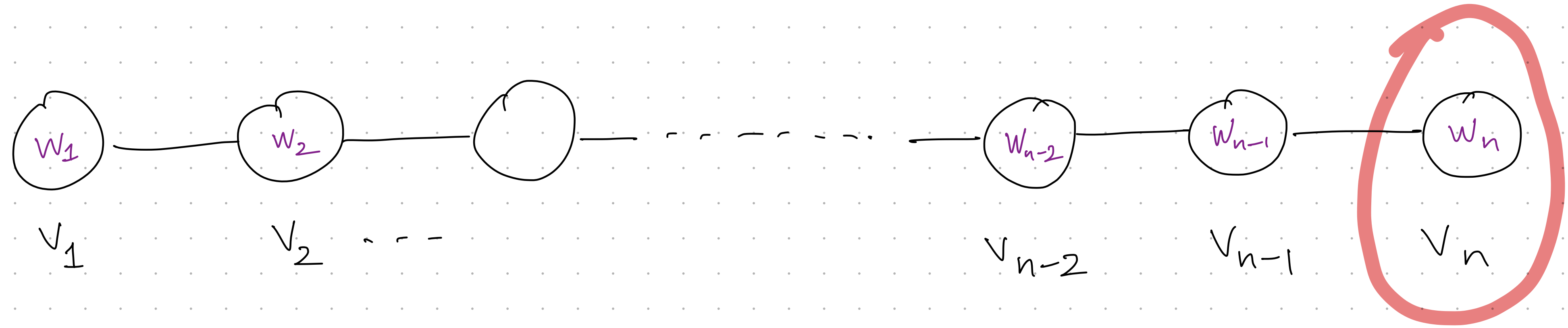



Fact. Fix any max wt. Independent Set $S \subseteq V$.

$v_n \in S$ OR $v_n \notin S$.

"Obvious" observation will get us surprisingly far!

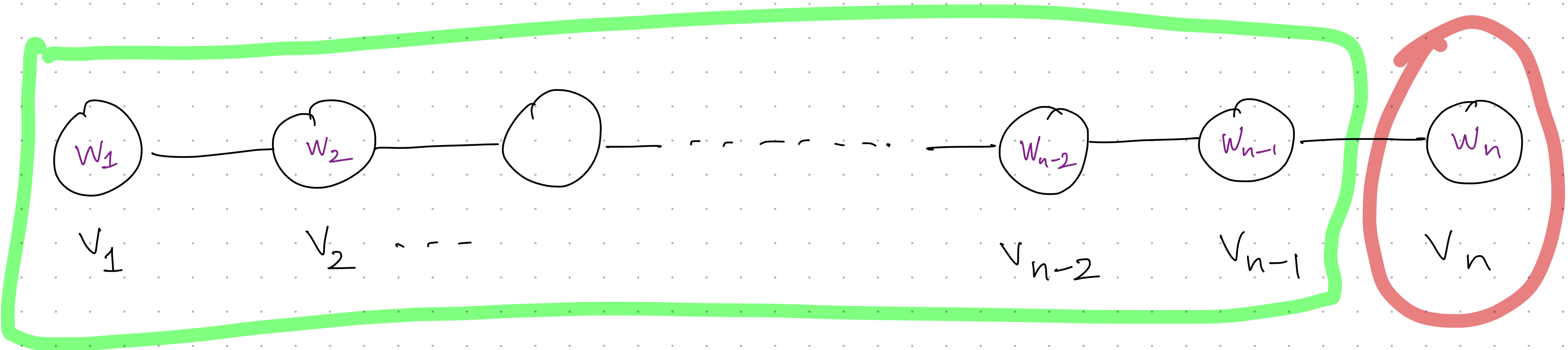
# Properties of an optimal solution $S \subseteq V$.




## Case Analysis

Suppose $v_n \notin S$. What is weight $W_S$?

# Properties of an optimal solution $S \subseteq V$.



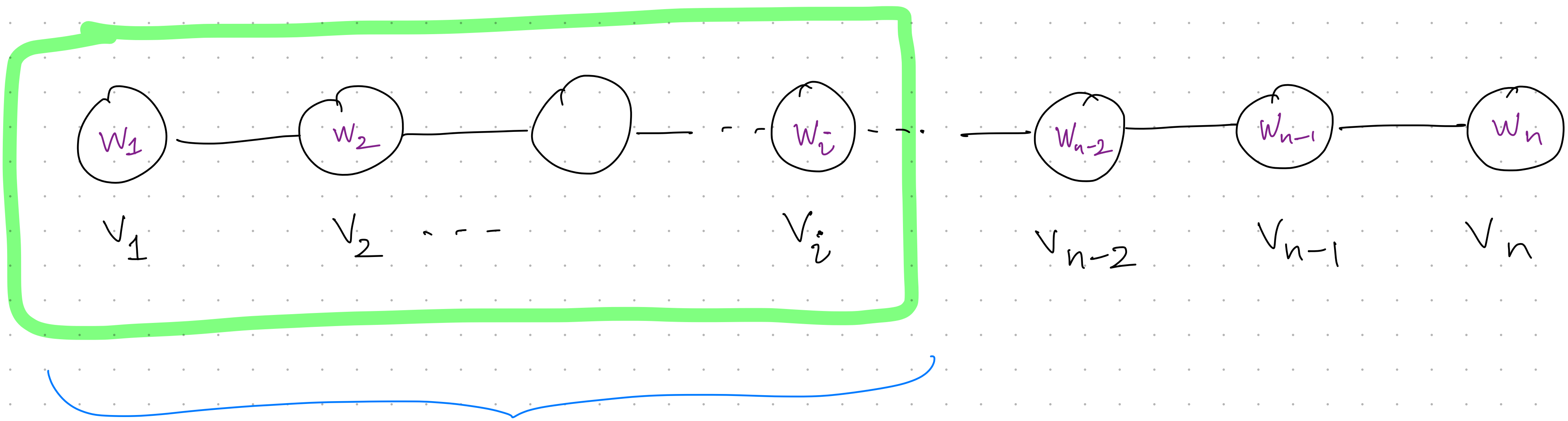

## Case Analysis

Suppose $v_n \notin S$.

What is weight $W_S$?

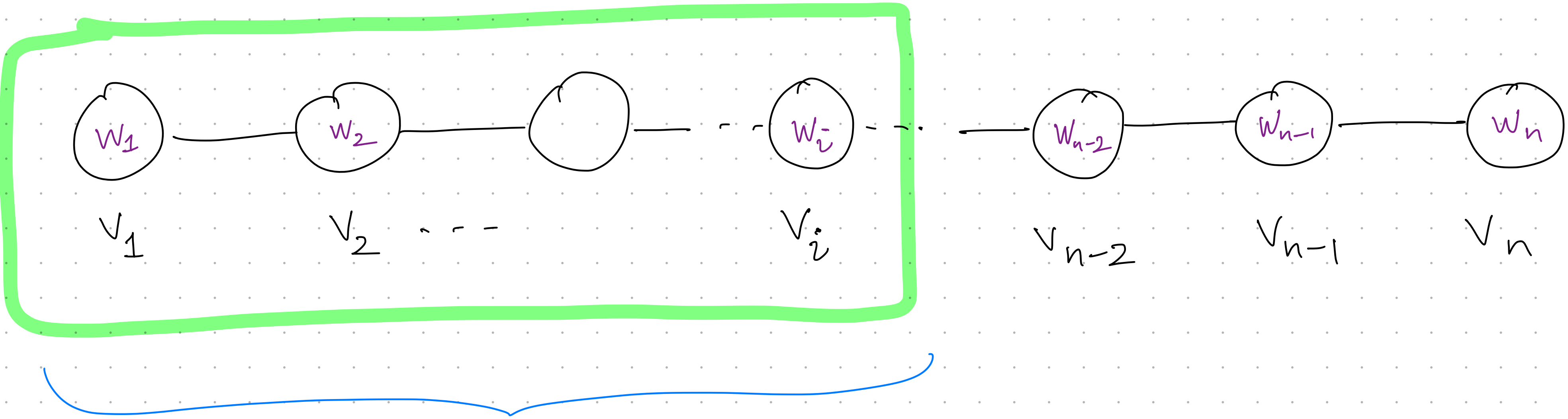
Same with $v_n$ removed.

# Properties of an optimal solution $S \subseteq V$.




$P_i$ (path up to $i$)

# Properties of an optimal solution $S \subseteq V$.

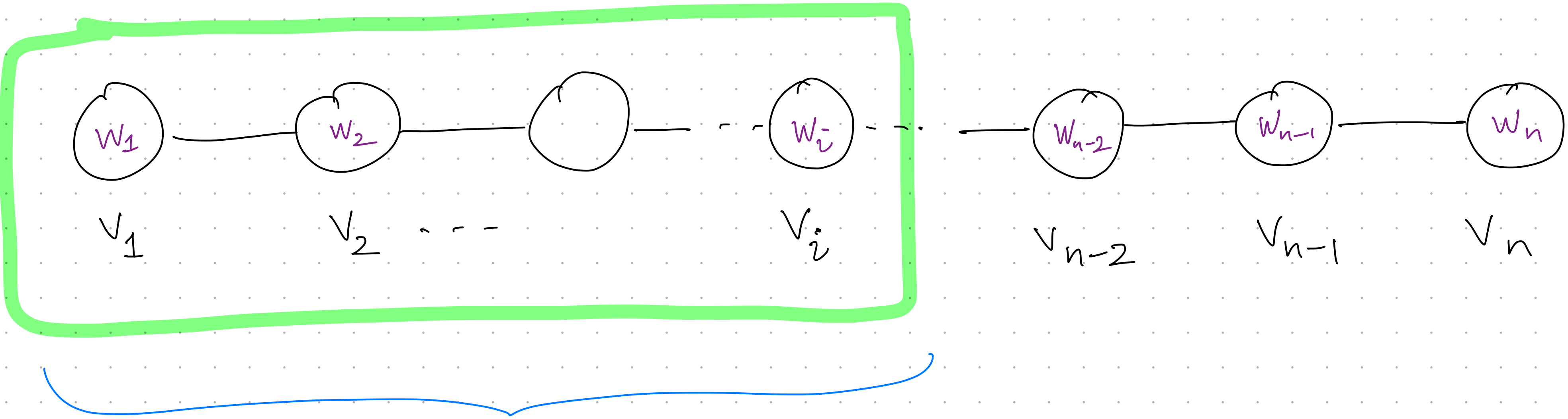



$P_i$ (path up to $i$)

↳ $P_n$ = entire path graph

↳ $P_0$ = empty path

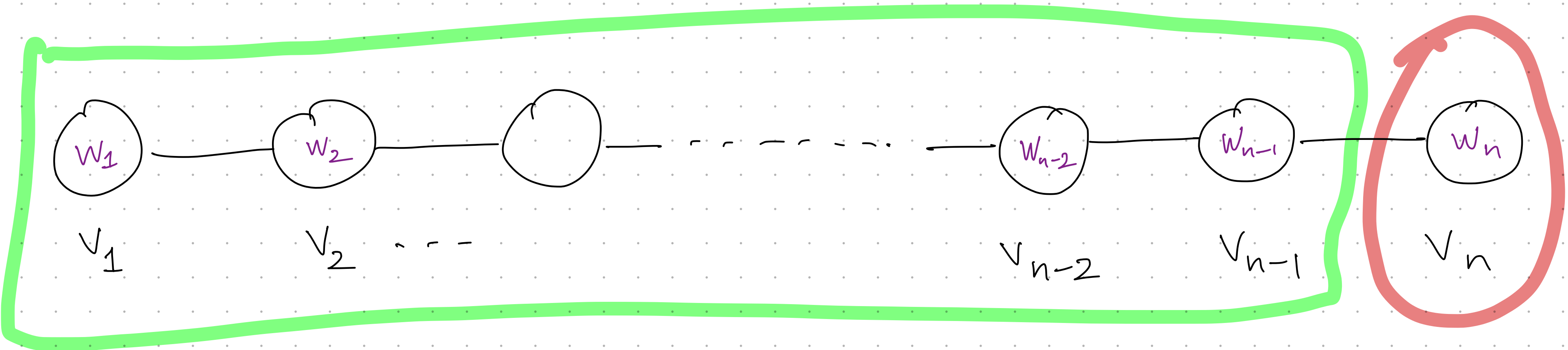
# Properties of an optimal solution $S \subseteq V$.




$P_i$ (path up to $i$)

$$\boxed{WIS(P_i) = \text{weight of max IS in } P_i}$$

# Properties of an optimal solution $S \subseteq V$.



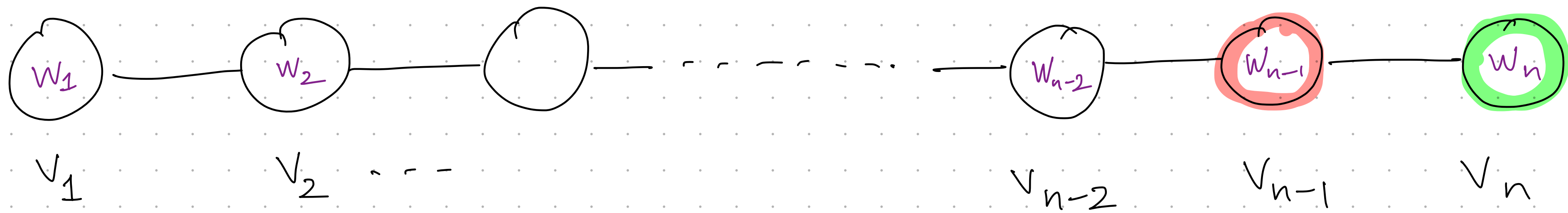

## Case Analysis

Suppose $v_n \notin S$. What is weight $W_S$?

Same with $v_n$ removed.

$$\Rightarrow WIS(P_n) = WIS(P_{n-1})$$

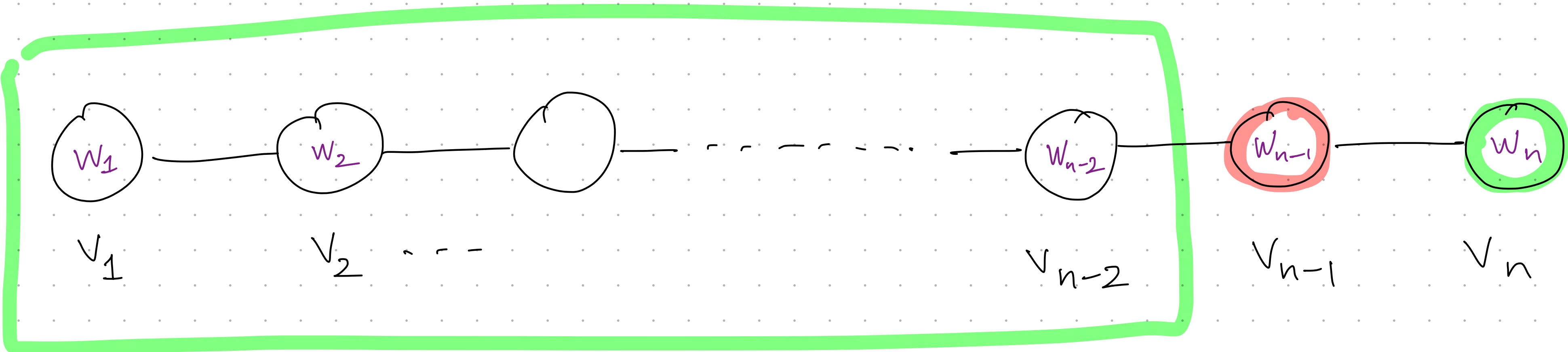
# Properties of an optimal solution $S \subseteq V$.




## Case Analysis

Suppose $v_n \in S$. What is weight $W_S$?

# Properties of an optimal solution $S \subseteq V$.

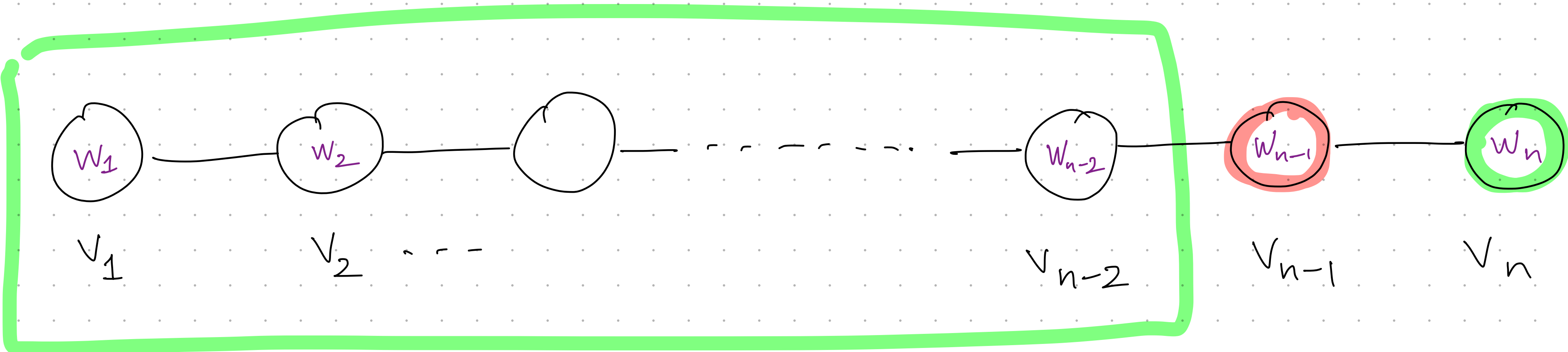



## Case Analysis

Suppose $v_n \in S$. What is weight $W_S$?

**Claim.** $WIS(P_n) = w_n + WIS(P_{n-2})$

# Properties of an optimal solution $S \subseteq V$.



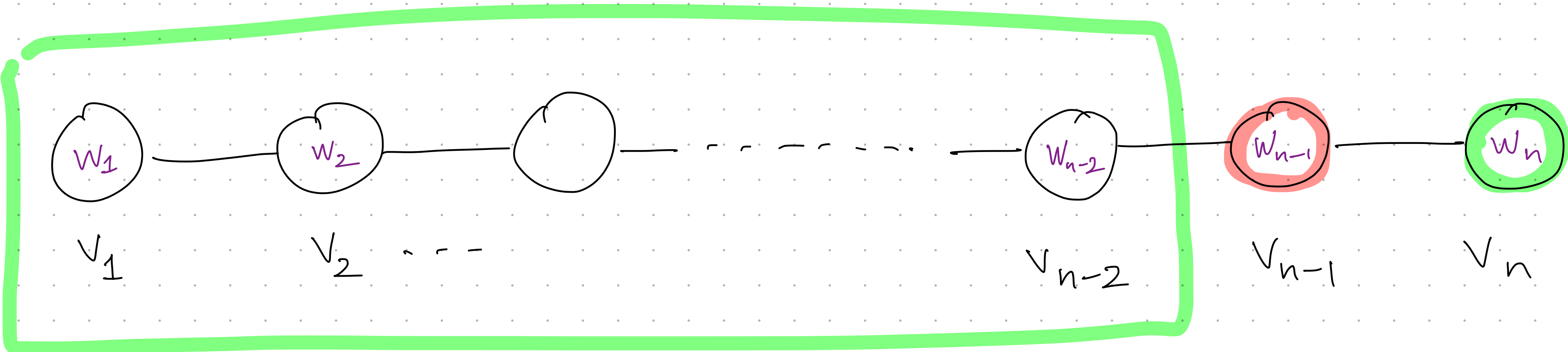

## Case Analysis

Suppose $v_n \in S$. What is weight $W_S$?

**Claim.** $WIS(P_n) = w_n + WIS(P_{n-2})$

Pf. – All vertices in $P_{n-2}$ are feasible for $S$ after removing $\{v_{n-1}, v_n\}$.

# Properties of an optimal solution $S \subseteq V$.




## Case Analysis

Suppose $v_n \in S$. What is weight $W_S$?

**Claim.** $WIS(P_n) = w_n + WIS(P_{n-2})$

Pf. – All vertices in $P_{n-2}$ are feasible for $S$ after removing $\{v_{n-1}, v_n\}$.

– If the $WIS(P_n)$ were larger contradicts optimality $WIS(P_{n-2})$

# Combining Cases

**Theorem.** (Recursive Formulation)

$$WIS(P_n) = \max \left\{ \begin{array}{l} WIS(P_{n-1}), \\ w_n + WIS(P_{n-2}) \end{array} \right\}$$

## Combining Cases

**Theorem.** (Recursive Formulation)

$$WIS(P_n) = \max \begin{cases} WIS(P_{n-1}), \\ w_n + WIS(P_{n-2}) \end{cases}$$

**Key Observation.** Global solution can be expressed in terms of solution to subproblems

# Dynamic Programming:

* Optimal solution requires searching all possible solutions

# Dynamic Programming:

* Optimal solution requires searching <u>all possible solutions</u>

* To avoid exponential RT, need to explore solutions <u>implicitly</u>

# Dynamic Programming.

* Optimal solution requires searching <u>all possible solutions</u>

* To avoid exponential RT, need to explore solutions <u>implicitly</u>

* Requires <u>identifying</u> <u>structure</u> in optimal solutions to avoid duplicating work.


Compute WIS ($P_k$). // Returns WIS ($P_k$)

if $k = 0$, return 0

if $k = 1$, return $w_1$

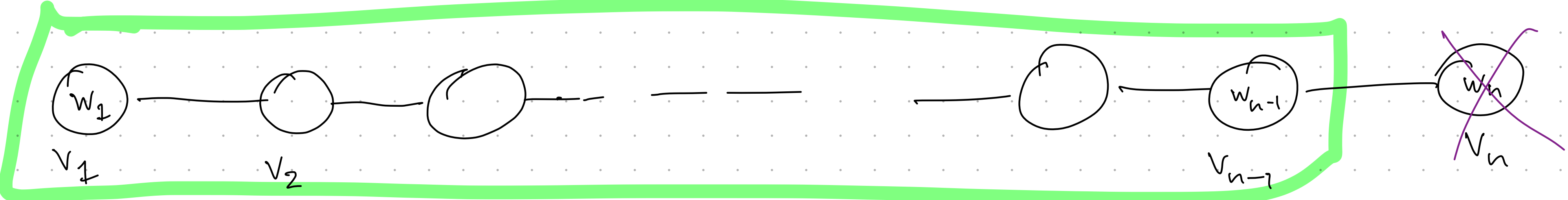
Let $W_{k-1} \leftarrow$ Compute WIS ($P_{k-1}$)

Let $W_{k-2} \leftarrow$ Compute WIS ($P_{k-2}$)

if $w_k + W_{k-2} > W_{k-1}$.

  return $w_k + W_{k-2}$

return $W_{k-1}$




**Compute WIS ($P_k$):** // Returns WIS($P_k$)

if $k=0$, return $0$

if $k=1$, return $\cancel{1}$ $w_1$

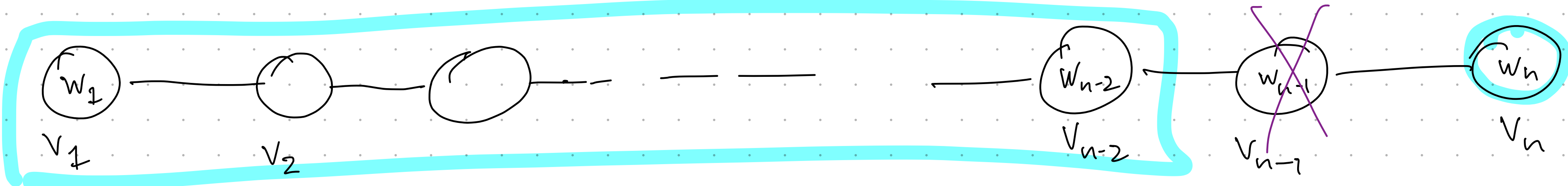
Let $\boxed{W_{k-1} \leftarrow \text{ComputeWIS}(P_{k-1})}$

Let $W_{k-2} \leftarrow \text{ComputeWIS}(P_{k-2})$

if $w_k + W_{k-2} > W_{k-1}$:

  return $w_k + W_{k-2}$

return $W_{k-1}$




**Compute WIS ($P_k$):** // Returns WIS ($P_k$)

if $k=0$, return 0

if $k=1$, return ~~1~~ $w_1$

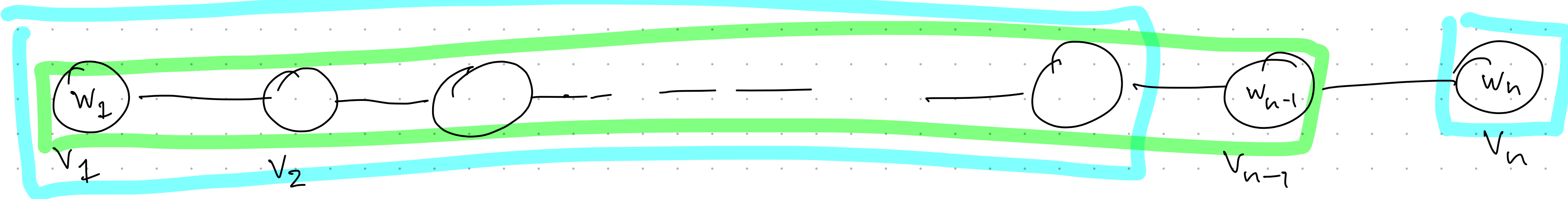
Let $W_{k-1} \leftarrow$ Compute WIS ($P_{k-1}$)

Let $W_{k-2} \leftarrow$ Compute WIS ($P_{k-2}$)

if $w_k + W_{k-2} > W_{k-1}$

return $w_k + W_{k-2}$

return $W_{k-1}$




Compute WIS $(P_k)$. // Returns WIS $(P_k)$

if $k=0$, return $0$

if $k=1$, return $\cancel{1}$ $w_1$

Let $W_{k-1} \leftarrow$ Compute WIS $(P_{k-1})$

Let $W_{k-2} \leftarrow$ Compute WIS $(P_{k-2})$

if $\boxed{w_k + W_{k-2}} > \boxed{W_{k-1}}$.

return $w_k + W_{k-2}$

return $W_{k-1}$

Return the max.

**ComputeWIS ($P_k$):**

if $k=0$, return $0$

if $k=1$, return $\cancel{1}$ $w_1$

Let $W_{k-1} \leftarrow$ ComputeWIS($P_{k-1}$)

Let $W_{k-2} \leftarrow$ ComputeWIS($P_{k-2}$)

if $w_k + W_{k-2} > W_{k-1}$:

  return $w_k + W_{k-2}$

return $W_{k-1}$

**Claim:**

ComputeWIS($P_n$) returns value of the max weight IS.

Run time

## ComputeWIS $(P_k)$:

if $k=0$, return $0$ $\qquad$ $T(k)$

if $k=1$, return $\cancel{1}$ $w_1$

Let $W_{k-1} \leftarrow$ ComputeWIS $(P_{k-1})$ $\qquad$ $T(k-1)$

Let $W_{k-2} \leftarrow$ ComputeWIS $(P_{k-2})$ $\qquad$ $T(k-2)$

if $w_k + W_{k-2} > W_{k-1}$:

return $w_k + W_{k-2}$

return $W_{k-1}$

$$T(k) \geq T(k-1) + T(k-2)$$

$$T(k) \geq T(k-1) + T(k-2)$$

$$\geq 2 \cdot T(k-2)$$

$$
\begin{aligned}
T(k) &\geq T(k-1) + T(k-2) \\
&\geq 2 \cdot T(k-2) \\
&\geq 2 \cdot \left(2 \cdot T(k-4)\right) \\
&\ \ \vdots
\end{aligned}
$$

$$T(k) \geq T(k-1) + T(k-2)$$

$$\geq 2 \cdot T(k-2)$$

$$\geq 2 \cdot (2 \cdot T(k-4))$$

$$\vdots$$

$$\geq 2^{k/2}$$

$T(k) \geq 2^{k/2}$ Exponential time!

$$
\begin{aligned}
T(k) &\geq T(k-1) + T(k-2) \\
&\geq 2 \cdot T(k-2) \\
&\geq 2 \cdot \left(2 \cdot T(k-4)\right) \\
&\vdots \\
&\geq 2^{k/2}
\end{aligned}
$$

$$T(k) \geq 2^{k/2}$$ <u>Exponential time!</u>

But the whole point was to avoid Exponential Time...

# The Dynamic Programming Table

* Many of the recursive calls we make are redundant!

# The Dynamic Programming Table

* Many of the recursive calls we make are <u>redundant</u>!

<u>Idea</u> Record (aka "<u>memoize</u>") the answers as we go.

# The Dynamic Programming Table

* Many of the recursive calls we make are <u>redundant</u>!

<u>Idea</u> Record (aka "memoize") the answers as we go.

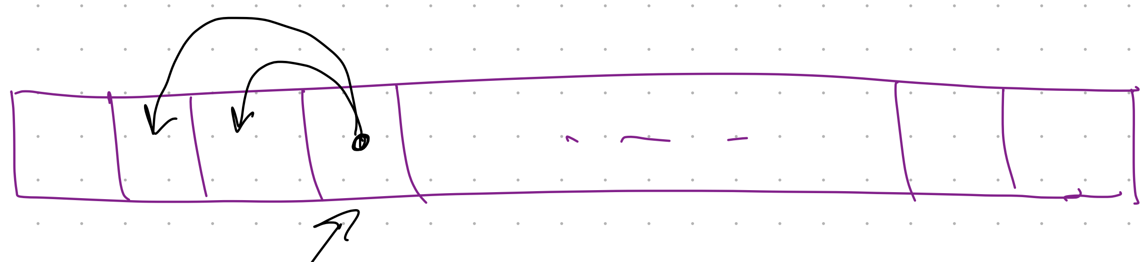
Fill in a Dynamic Programming Table

# The Dynamic Programming Table

* Many of the recursive calls we make are <u>redundant</u>!

<u>Idea</u> Record (aka "memoize") the answers as we go.

Fill in a Dynamic Programming Table

W = 

$$W[k] = \max\{W[k-1],\ w_k + W[k-2]\}$$

Global $W = [-1, -1, \ldots, -1]$

$WIS_n \leftarrow \text{Memoized WIS}(P_n)$

Memoized WIS $(P_k)$:

if $k=0$, return $0$
if $k=1$, return $w_1$.

if $W[k-1] = -1$: $W[k-1] \leftarrow \text{Memoized WIS}(P_{k-1})$

if $W[k-2] = -1$: $W[k-2] \leftarrow \text{Memoized WIS}(P_{k-2})$

if $w_k + W[k-2] > W[k-1]$.

return $w_k + W[k-2]$

return $W[k-1]$

Global $W = [-1, -1, \ldots, -1]$

$WIS_n \leftarrow$ Memoized WIS $(P_n)$

Memoized WIS $(P_k)$:

if $k=0$, return 0

if $k=1$, return $w_1$.

if $W[k-1] = -1$: $W[k-1] \leftarrow$ Memoized WIS $(\boxed{P_{k-1}})$

if $W[k-2] = -1$: $W[k-2] \leftarrow$ Memoized WIS $(\boxed{P_{k-2}})$

if $\boxed{w_k + W[k-2]} > \boxed{W[k-1]}$.

return $w_k + W[k-2]$

return $W[k-1]$

Global $W = [-1, -1, \ldots, -1]$

$WIS_n \leftarrow$ Memoized WIS $(P_n)$

## Running Time

How many times is W updated?

Memoized WIS $(P_k)$:

if $k=0$, return 0

if $k=1$, return $w_1$.

if $W[k-1] = -1$: $W[k-1] \leftarrow$ Memoized WIS $(\boxed{P_{k-1}})$

if $W[k-2] = -1$: $W[k-2] \leftarrow$ Memoized WIS $(\boxed{P_{k-2}})$

if $\boxed{w_k + W[k-2]} > \boxed{W[k-1]}$.

return $w_k + W[k-2]$

return $W[k-1]$

Global $W = [-1, -1, \ldots, -1]$

$WIS_n \leftarrow$ Memoized WIS $(P_n)$

Running Time

How many times is W updated?

Memoized WIS $(P_k)$:

if $k=0$, return 0

if $k=1$, return $w_1$.

if $W[k-1] = -1$: $W[k-1] \leftarrow$ Memoized WIS $(\boxed{P_{k-1}})$

if $W[k-2] = -1$: $W[k-2] \leftarrow$ Memoized WIS $(\boxed{P_{k-2}})$

if $\boxed{w_k + W[k-2]} > \boxed{W[k-1]}$.

return $w_k + W[k-2]$

return $W[k-1]$

n updates

$O(1)$ work per update

$\Rightarrow O(n)$ RT.

# Max Weight Independent Set on a Path.

→ Greedy Fails

→ Naive Recursive Algo $T(n) \geq 2^{n/2}$

→ Dynamic Programming $T(n) \leq O(n)$

(Recursion + Memoization)

# Iterative Solution

* Recursive formulation conceptually nice,
* Dynamic Programs always have an equivalent iterative formulation

  ↳ fills in the DP Table directly

## Iterative Solution

* Recursive formulation conceptually nice,
* Dynamic Programs always have an equivalent iterative formulation

↳ fills in the DP Table directly

---

Iterative WIS $(P_n)$.

Let $W = [-1, -1, \ldots, -1]$

$W[0] \leftarrow 0$, $W[1] = w_1$.

for $k = 2, \ldots, n$:

$W[k] \leftarrow \max\{ W[k-1], w_k + W[k-2] \}$

Return $W[n]$

Iterative WIS ($P_n$).

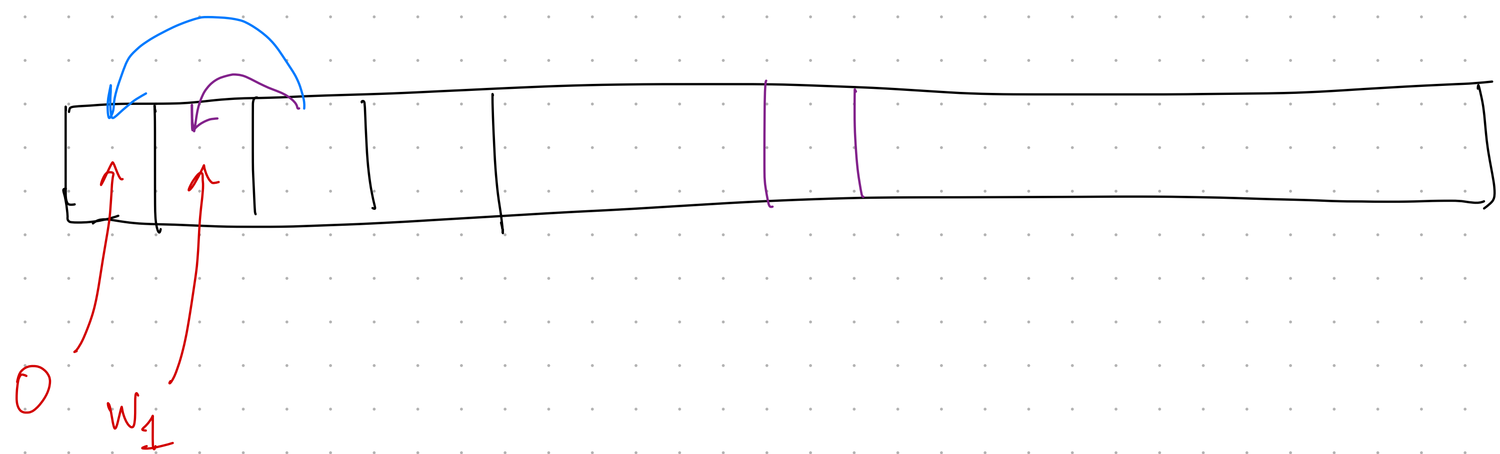
Let $W = [-1, -1, \ldots, -1]$

$W[0] \leftarrow 0$, $W[1] = w_1$.

for $k = 2, \ldots, n$:

$W[k] \leftarrow \max\{W[k-1], w_k + W[k-2]\}$

Return $W[n]$



Iterative WIS $(P_n)$.

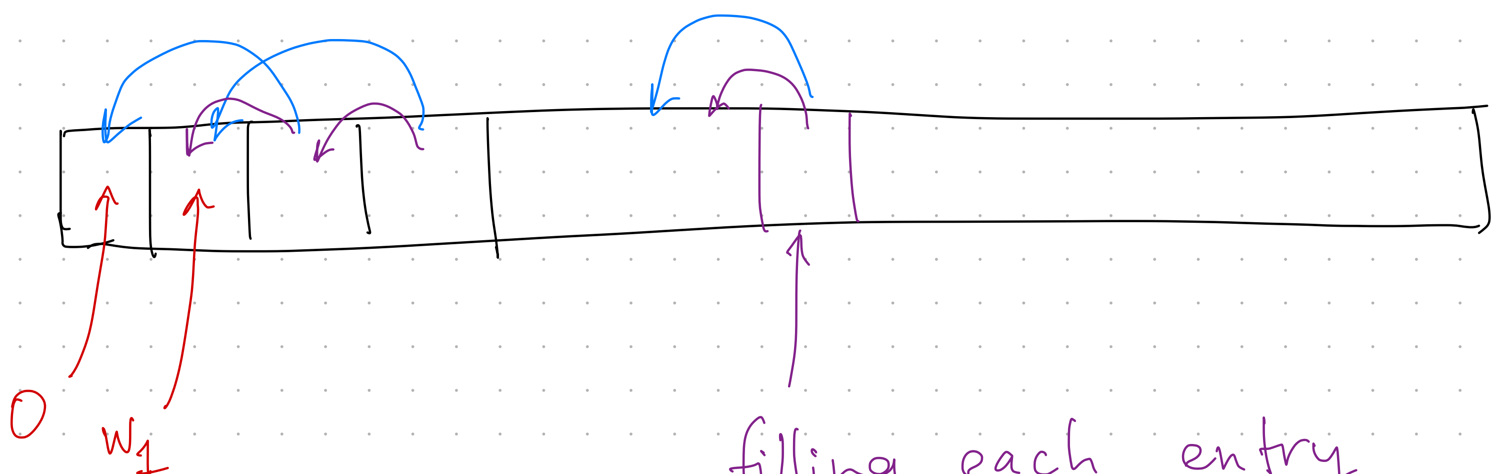
Let $W = [-1, -1, \ldots, -1]$

$W[0] \leftarrow 0$, $W[1] = w_1$.

for $k = 2, \ldots, n$:

$W[k] \leftarrow \max\{W[k-1],\ w_k + W[k-2]\}$

Return $W[n]$




filling each entry probes 2 prior entries.