January 31, 2025 — Dynamic Programming

Plan

* Weighted Independent Set on a Path
* 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

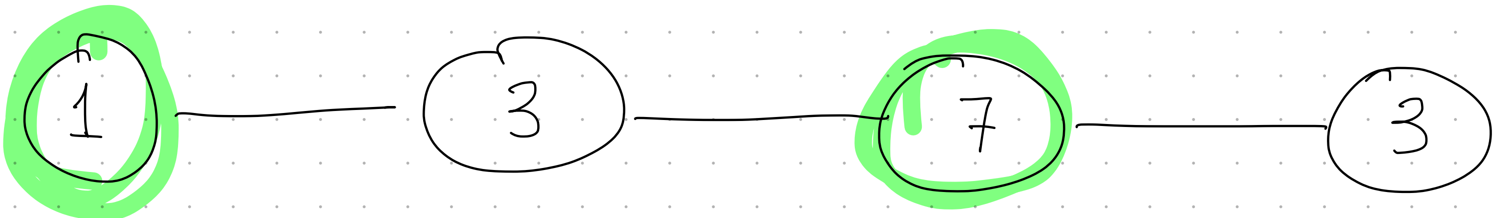
Find "independent set" $S \subseteq V$ of vertices of maximum weight

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



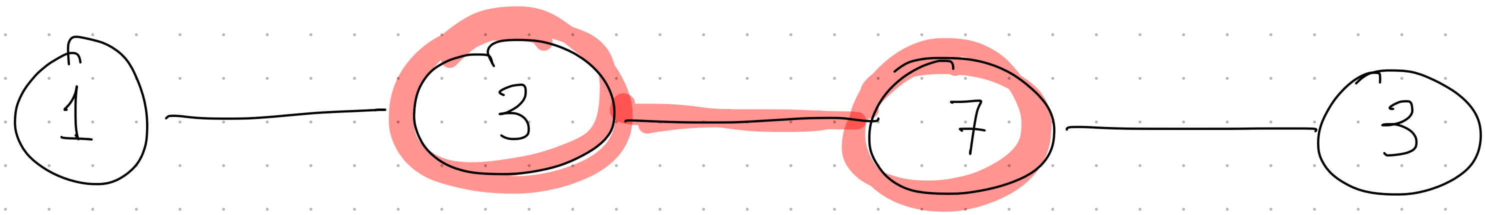
Find "independent set" $S \subseteq V$ of vertices of maximum weight

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?

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 <u>global</u> understanding of solution

↓

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

Announcements

* HW0 Solutions posted to Canvas

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

$\underbrace{\text{KT §6.1}}$ Recommended

Weighted Interval Scheduling

Announcements

* HW0 Solutions posted to Canvas
* Lecture Notes posted to site
  (This material Not in KT)

$\underbrace{\text{KT §6.1}}$ Recommended

Weighted Interval Scheduling

Weekend support

* Saturday 1–3p: Gates G01

  Recitation practice problems led by TAs

* Sunday 1p.

  LaTeX workshop (See Ed)

Greedy Algorithms?

→ Greedy attempts fail.

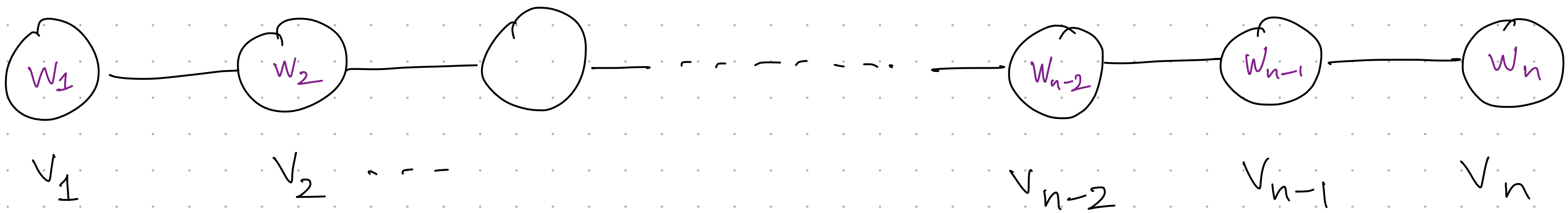
→ Weights mess things up.

→ Need a global understanding of solution

↓

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

Properties of an optimal solution

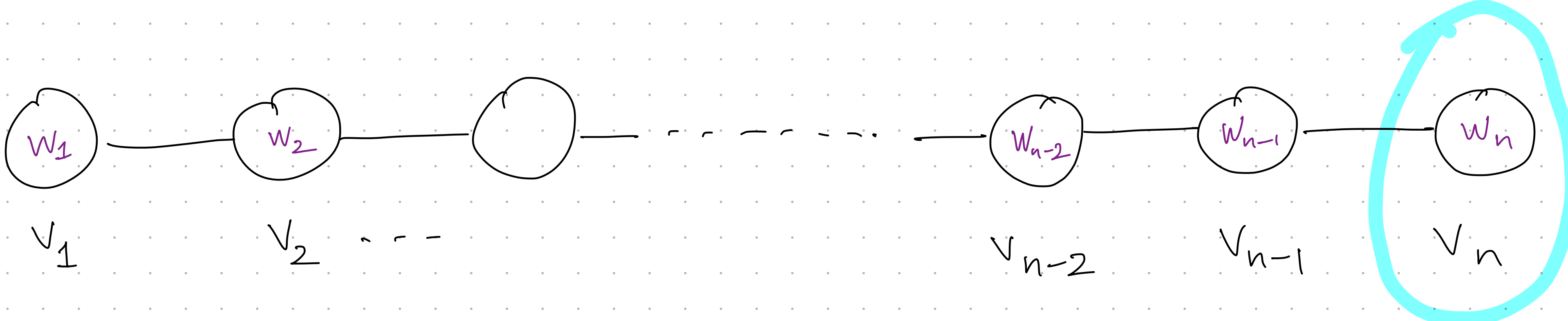


Find "independent set" $S \subseteq V$ of vertices of maximum weight

where no two vertices share an edge

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

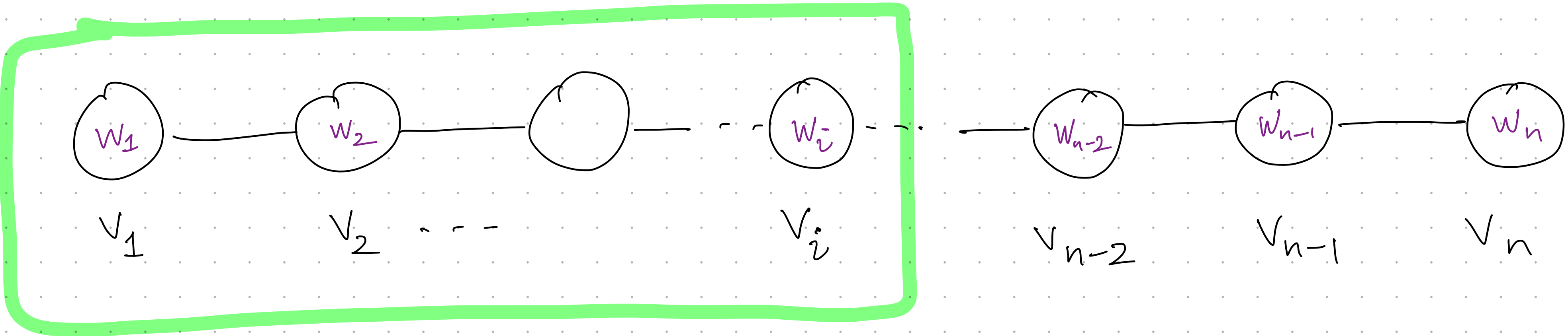
Properties of an optimal solution



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 $S \subseteq V$.

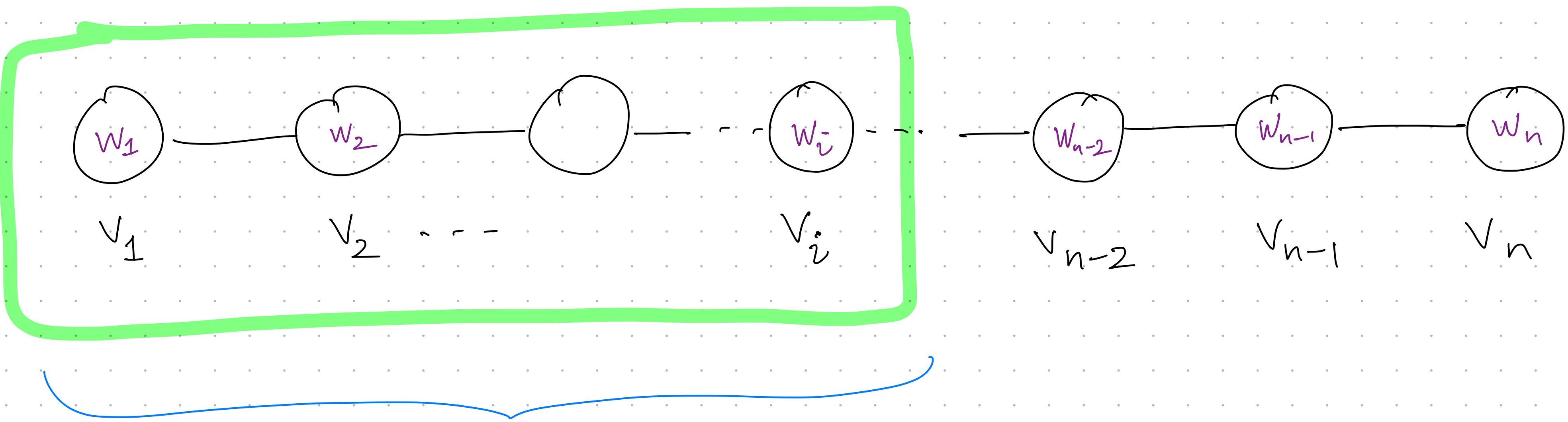


P_i (path up to i)

P_n (entire path graph)

P_1 (singleton path)

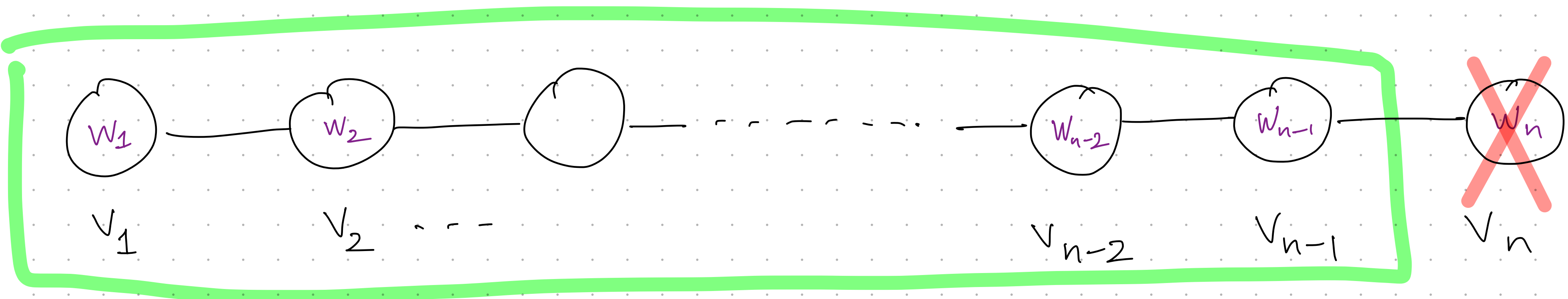
Properties of an optimal solution $S \subseteq V$.



P_i (path up to i)

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

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

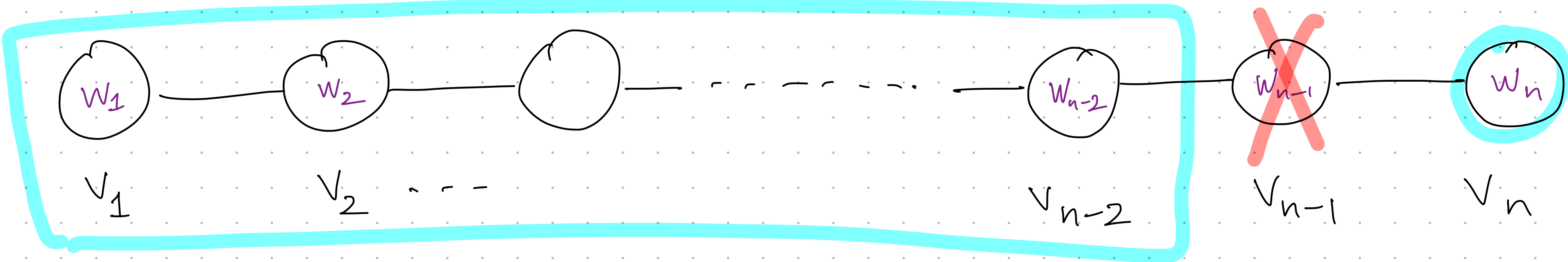


Case Analysis

* $v_n \notin S$ $\implies WIS(P_n) = WIS(P_{n-1})$

  ↳ Every vertex in P_{n-1} is feasible

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



Case Analysis

* $v_n \notin S$ $\Rightarrow WIS(P_n) = WIS(P_{n-1})$
  ↳ Every vertex in P_{n-1} is feasible

* $v_n \in S$ $\Rightarrow WIS(P_n) = w_n + WIS(P_{n-2})$
  ↳ $v_{n-1} \notin S$
  ↳ Every vertex in P_{n-2} is feasible

Combining Cases

Theorem. (WIS Recurrence)

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

Combining Cases

Theorem (WIS Recurrence)

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

Pf. By case analysis on previous slides.

* Two exhaustive cases $v_n \in S$ AND $v_n \notin S$
* $v_n \notin S \implies WIS(P_n) = WIS(P_{n-1})$
* $v_n \in S \implies WIS(P_n) = WIS(P_{n-2}) + w_n$

Global Solution expressed in terms of Solutions to Subproblems

$WIS(P_n)$

$WIS(P_{n-1})$ $WIS(P_{n-2})$

Dynamic Programming

* Search over all possible solutions

  (in contrast to greedy)

* Exploit <u>structure</u> based on subproblems to avoid exponential blow-up

  (search implicitly)

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

Compute $WIS(P_k)$:

// Base Cases

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

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

return $\max \{ W_{k-1}, w_k + W_{k-2} \}$

<u>Compute WIS (P_k):</u> // Returns WIS (P_k)

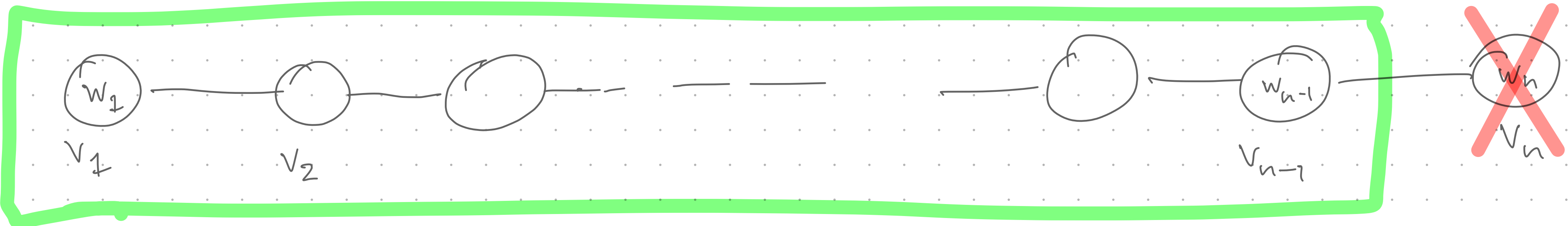
if $k=0$, return 0

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})

return $\max\{W_{k-1},\ w_k + W_{k-2}\}$



<u>Compute WIS (P_k)</u>: // Returns WIS (P_k)

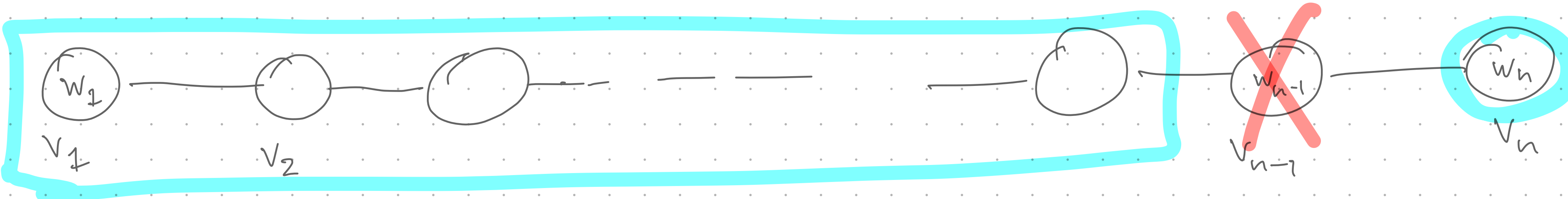
if $k=0$, return 0

if $k=1$, return w_1

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

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

return $\max\{\underline{W_{k-1}},\ w_k + W_{k-2}\}$



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

return $\max\{W_{k-1},\ \underline{w_k + W_{k-2}}\}$

Claim. ComputeWIS returns the maximum weight independent set on a path correctly.

$\hookrightarrow$ Follows from proof of Recurrence.

ComputeWIS (P_k): // Returns $WIS(P_k)$

if $k=0$, return 0

if $k=1$, return w_1

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

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

return $\max\{W_{k-1},\ w_k + W_{k-2}\}$

What about Running Time?

ComputeWIS(P_k). // Returns WIS(P_k)

if $k=0$, return 0

if $k=1$, return w_1

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

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

return $\max\{ W_{k-1},\ w_k + W_{k-2} \}$

What about Running Time?

$T(k) \equiv$ Running Time on paths of k vertices

Compute WIS (P_k). $\qquad T(k)$

if $k=0$, return 0

if $k=1$, return w_1

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

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

return $\max\{ W_{k-1},\ w_k + W_{k-2} \}$

What about Running Time?

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

Compute WIS (P_k). $\qquad T(k)$

if $k=0$, return 0

if $k=1$, return w_1

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

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

return $\max\{ W_{k-1},\ w_k + W_{k-2} \}$

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

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

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

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

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

$$\vdots$$

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

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

Exponential time!

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

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

Exponential time!

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

The Dynamic Programming Table

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

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.

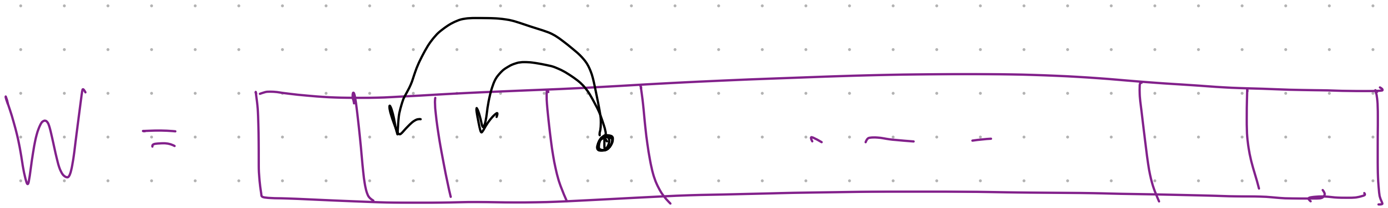
Fill in a Dynamic Programming Table

The Dynamic Programming Table

* Many of the recursive calls we make are __redundant!__

__Idea__ Record (aka "memoize") the answers as we go.

Fill in a Dynamic Programming Table



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

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

$WIS_n \leftarrow$ Memoized WIS (P_n)

<u>Memoized WIS (P_k)</u>:

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

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

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

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

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

Running Time?

Memoized WIS (P_k):

* How many times is W updated?

if $k=0$, return 0

if $k=1$, return W_1.

if $W[k-1] = -1$:

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

if $W[k-2] = -1$:

$\quad W[k-2] \leftarrow \text{Memoized WIS}(P_{k-2})$

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

Claim. Memoized WIS runs in $O(n)$ time.

Intuitive Argument: W updated at most n times

Formal: Induction on length of path

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).

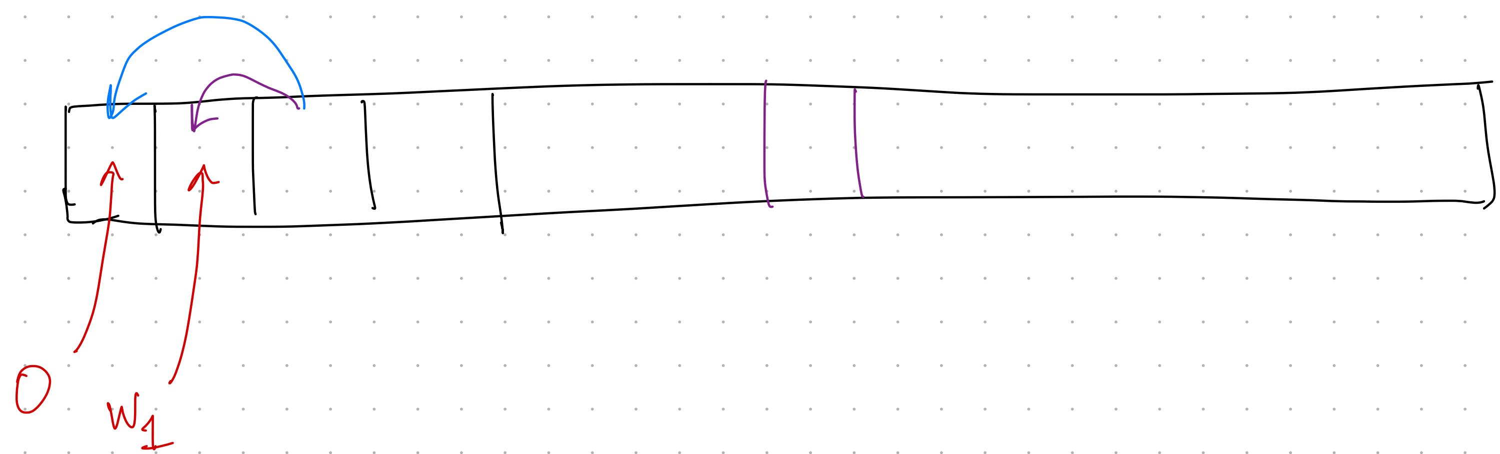
Let $W = [-1, -1, \dots, -1]$

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

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

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

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 $W[n]$