If the Seahawks win the Super Bowl, everyone gets a free extension on next weeks problems (11-13 and 55) till Mon Feb 16th at 6:00pm so that you can celebrate.

(If so, no late extensions)

Lecture 13

Dynamic Programming II: The Knapsack problem

Chinmay Nirkhe | CSE 421 Winter 2026

Previously in CSE 421…

General dynamic programming algorithm

- **Iterate through subproblems**: Starting from the "smallest" and building up to the "biggest." For each one:
  - Find the optimal value, using the previously-computed optimal values to smaller subproblems.
  - Record the choices made to obtain this optimal value. (If many smaller subproblems were considered as candidates, record which one was chosen.)
- **Compute the solution**: We have the value of the optimal solution to this optimization problem but we don't have the actual solution itself. Use the recorded information to actually reconstruct the optimal solution.

General dynamic programming runtime

$$\text{Runtime} = (\text{Total number of subproblems}) \times \begin{pmatrix} \text{Time it takes to solve problems} \\ \text{given solutions to subproblems} \end{pmatrix}$$

Today

Edit distance walkthrough

Edit distance walkthrough

TASTE v TREAT

$X = \text{T A S T E}$

$Y = \text{T R E A T}$

Recall def of subproblem: $X_k = k$ prefix of X

$Y_\ell = \ell$ prefix of Y

$$d(k, \ell) = \text{dist between } X_k \text{ \& } Y_\ell$$

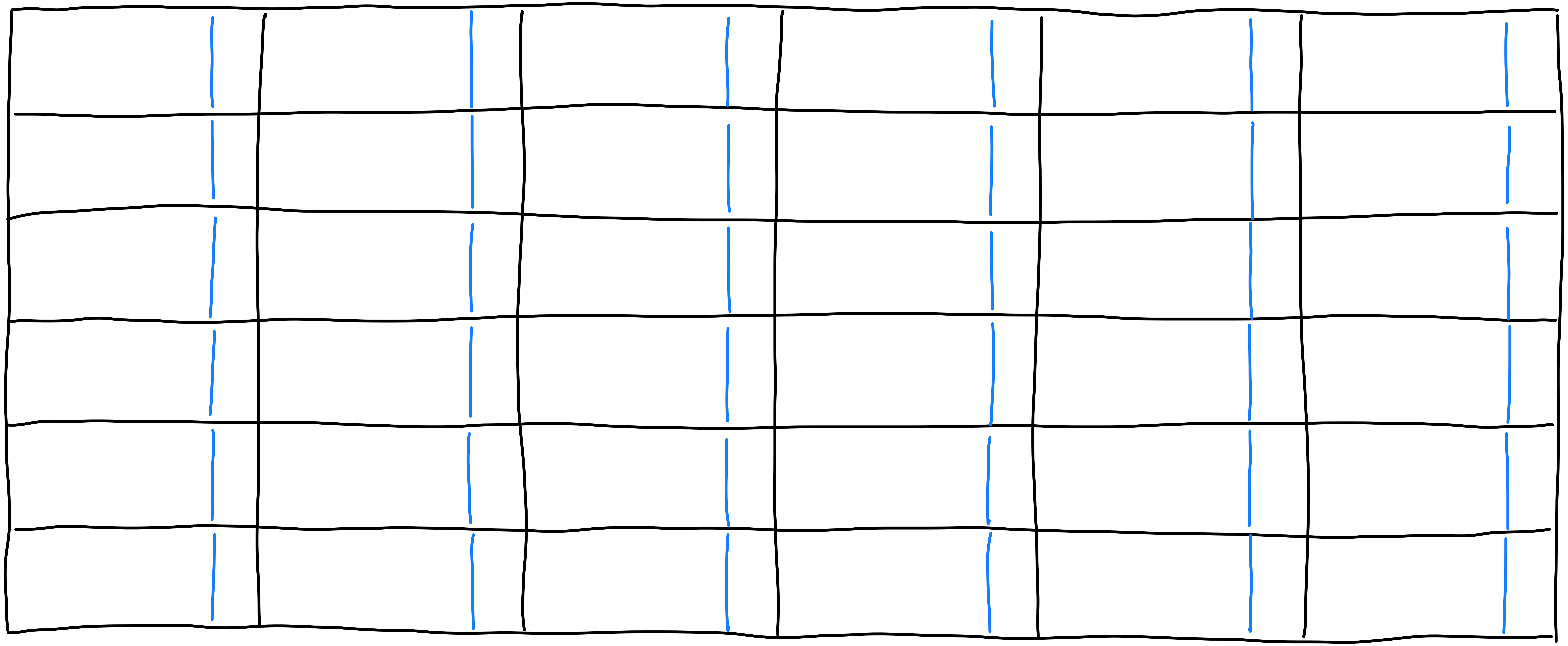
$$= \begin{cases} \text{if } x_k = y_\ell \text{, then } d(k-1, \ell-1) & \text{"last char agree"} \\ \text{else } 1 + \min \begin{cases} d(k, \ell-1) & \text{"insert last char } y_\ell\text{"} \\ d(k-1, \ell) & \text{"delete last char } x_k\text{"} \\ d(k-1, \ell-1) & \text{"substitute } x_k \text{ for } y_\ell\text{"} \end{cases} \end{cases}$$

Edit distance walkthrough

TASTE v TREAT

X = TASTE

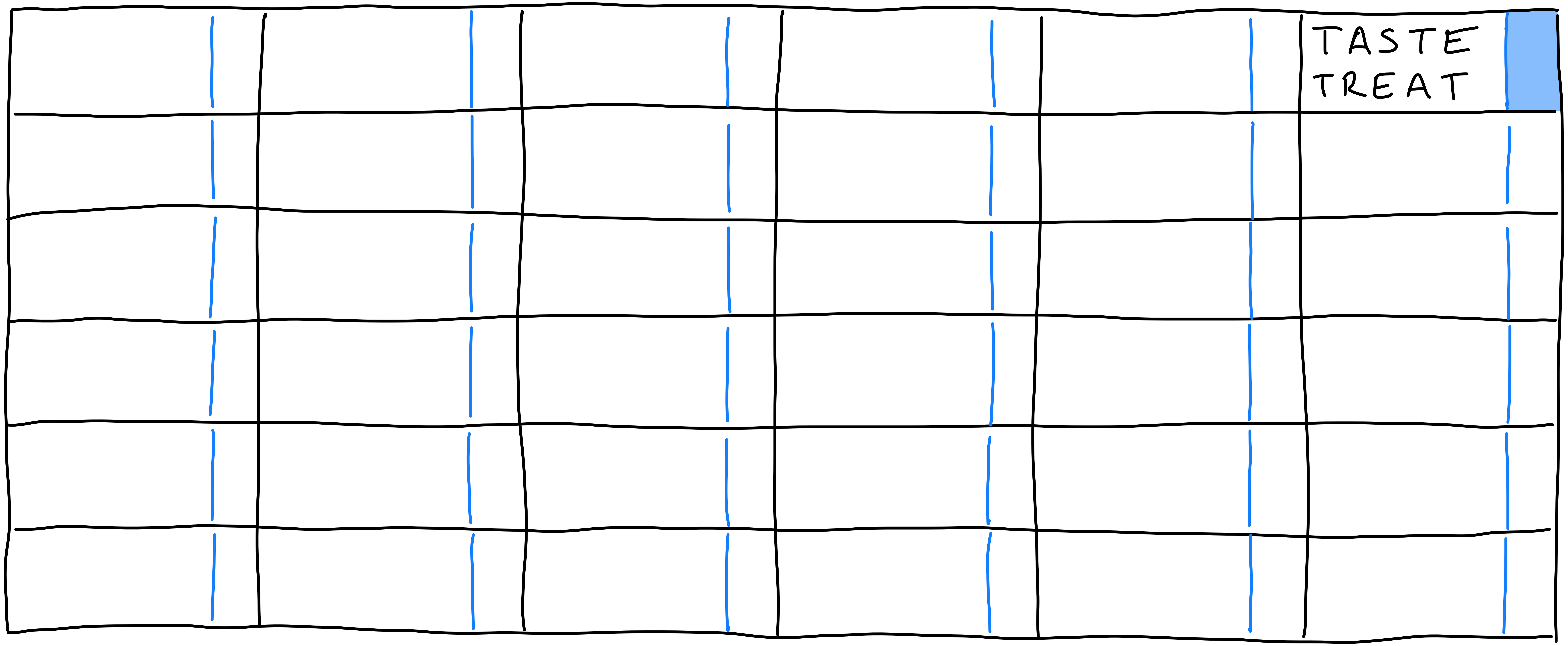
Y = TREAT



Edit distance walkthrough

TASTE v TREAT

X = TASTE
Y = TREAT



Edit distance walkthrough

TASTE v TREAT

X = TASTE
Y = TREAT

∅ TREAT					TASTE TREAT
∅ TREA					
∅ TRE					
∅ TR					
∅ T					
∅ ∅	T ∅	TA ∅	TAS ∅	TAST ∅	TASTE ∅

Edit distance walkthrough

TASTE v TREAT

X = TASTE
Y = TREAT

∅ TREAT	5	T TREAT		TA TREAT		TAS TREAT		TAST TREAT		TASTE TREAT	
∅ TREA	4	T TREA		TA TREA		TAS TREA		TAST TREA		TASTE TREA	
∅ TRE	3	T TRE		TA TRE		TAS TRE		TAST TRE		TASTE TRE	
∅ TR	2	T TR		TA TR		TAS TR		TAST TR		TASTE TR	
∅ T	1	T T		TA T		TAS T		TAST T		TASTE T	
∅ ∅	0	T ∅	1	TA ∅	2	TAS ∅	3	TAST ∅	4	TASTE ∅	5

Edit distance walkthrough

TASTE v TREAT

X = TASTE
Y = TREAT

∅ TREAT 5	T TREAT	TA TREAT	TAS TREAT	TAST TREAT	TASTE TREAT
∅ TREA 4	T TREA	TA TREA	TAS TREA	TAST TREA	TASTE TREA
∅ TRE 3	T TRE	TA TRE	TAS TRE	TAST TRE	TASTE TRE
∅ TR 2	T TR	TA TR	TAS TR	TAST TR	TASTE TR
∅ T 1	T T	TA T	TAS T	TAST T	TASTE T
∅ ∅ 0	T ∅ 1	TA ∅ 2	TAS ∅ 3	TAST ∅ 4	TASTE ∅ 5

Edit distance walkthrough

TASTE v TREAT

X = TASTE
Y = TREAT

∅ TREAT 5	T TREAT	TA TREAT	TAS TREAT	TAST TREAT	TASTE TREAT
∅ TREA 4	T TREA	TA TREA	TAS TREA	TAST TREA	TASTE TREA
∅ TRE 3	T TRE	TA TRE	TAS TRE	TAST TRE	TASTE TRE
∅ TR 2	T TR	TA TR	TAS TR	TAST TR	TASTE TR
∅ T 1	T T	TA T	TAS T	TAST T	TASTE T
∅ ∅ 0	T ∅ 1	TA ∅ 2	TAS ∅ 3	TAST ∅ 4	TASTE ∅ 5

Edit distance walkthrough

TASTE v TREAT

X = TASTE
Y = TREAT

∅ TREAT 5	T TREAT	TA TREAT	TAS TREAT	TAST TREAT	TASTE TREAT
∅ TREA 4	T TREA	TA TREA	TAS TREA	TAST TREA	TASTE TREA
∅ TRE 3	T TRE	TA TRE	TAS TRE	TAST TRE	TASTE TRE
∅ TR 2	T TR	TA TR	TAS TR	TAST TR	TASTE TR
∅ T 1	T T 0	TA T	TAS T	TAST T	TASTE T
∅ ∅ 0	T ∅ 1	TA ∅ 2	TAS ∅ 3	TAST ∅ 4	TASTE ∅ 5

Edit distance walkthrough

TASTE v TREAT

X = TASTE
Y = TREAT

∅ TREAT 5	T TREAT	TA TREAT	TAS TREAT	TAST TREAT	TASTE TREAT
∅ TREA 4	T TREA	TA TREA	TAS TREA	TAST TREA	TASTE TREA
∅ TRE 3	T TRE	TA TRE	TAS TRE	TAST TRE	TASTE TRE
∅ TR 2	T TR	TA TR	TAS TR	TAST TR	TASTE TR
∅ T 1	T T 0	TA T	TAS T	TAST T	TASTE T
∅ ∅ 0	T ∅ 1	TA ∅ 2	TAS ∅ 3	TAST ∅ 4	TASTE ∅ 5

Edit distance walkthrough

TASTE v TREAT

X = TASTE
Y = TREAT

∅ TREAT 5	T TREAT	TA TREAT	TAS TREAT	TAST TREAT	TASTE TREAT
∅ TREA 4	T TREA	TA TREA	TAS TREA	TAST TREA	TASTE TREA
∅ TRE 3	T TRE	TA TRE	TAS TRE	TAST TRE	TASTE TRE
∅ TR 2	T TR 1	TA TR	TAS TR	TAST TR	TASTE TR
∅ T 1	T T 0	TA T	TAS T	TAST T	TASTE T
∅ ∅ 0	T ∅ 1	TA ∅ 2	TAS ∅ 3	TAST ∅ 4	TASTE ∅ 5

Edit distance walkthrough

TASTE v TREAT

X = TASTE
Y = TREAT

∅ TREAT 5	T TREAT	TA TREAT	TAS TREAT	TAST TREAT	TASTE TREAT
∅ TREA 4	T TREA	TA TREA	TAS TREA	TAST TREA	TASTE TREA
∅ TRE 3	T TRE	TA TRE	TAS TRE	TAST TRE	TASTE TRE
∅ TR 2	T TR 1	TA TR	TAS TR	TAST TR	TASTE TR
∅ T 1	T T 0	TA T	TAS T	TAST T	TASTE T
∅ ∅ 0	T ∅ 1	TA ∅ 2	TAS ∅ 3	TAST ∅ 4	TASTE ∅ 5

Edit distance walkthrough

TASTE v TREAT

X = TASTE
Y = TREAT

∅ TREAT 5	T TREAT	TA TREAT	TAS TREAT	TAST TREAT	TASTE TREAT
∅ TREA 4	T TREA	TA TREA	TAS TREA	TAST TREA	TASTE TREA
∅ TRE 3	T TRE 2	TA TRE	TAS TRE	TAST TRE	TASTE TRE
∅ TR 2	T TR 1	TA TR	TAS TR	TAST TR	TASTE TR
∅ T 1	T T 0	TA T	TAS T	TAST T	TASTE T
∅ ∅ 0	T ∅ 1	TA ∅ 2	TAS ∅ 3	TAST ∅ 4	TASTE ∅ 5

Edit distance walkthrough

TASTE v TREAT

X = TASTE
Y = TREAT

∅ / TREAT: 5	T / TREAT	TA / TREAT	TAS / TREAT	TAST / TREAT	TASTE / TREAT
∅ / TREA: 4	T / TREA: 3	TA / TREA	TAS / TREA	TAST / TREA	TASTE / TREA
∅ / TRE: 3	T / TRE: 2	TA / TRE	TAS / TRE	TAST / TRE	TASTE / TRE
∅ / TR: 2	T / TR: 1	TA / TR	TAS / TR	TAST / TR	TASTE / TR
∅ / T: 1	T / T: 0	TA / T	TAS / T	TAST / T	TASTE / T
∅ / ∅: 0	T / ∅: 1	TA / ∅: 2	TAS / ∅: 3	TAST / ∅: 4	TASTE / ∅: 5

Edit distance walkthrough

TASTE v TREAT

X = TASTE
Y = TREAT

∅ TREAT 5	T TREAT 4	TA TREAT	TAS TREAT	TAST TREAT	TASTE TREAT
∅ TREA 4	T TREA 3	TA TREA	TAS TREA	TAST TREA	TASTE TREA
∅ TRE 3	T TRE 2	TA TRE	TAS TRE	TAST TRE	TASTE TRE
∅ TR 2	T TR 1	TA TR	TAS TR	TAST TR	TASTE TR
∅ T 1	T T 0	TA T	TAS T	TAST T	TASTE T
∅ ∅ 0	T ∅ 1	TA ∅ 2	TAS ∅ 3	TAST ∅ 4	TASTE ∅ 5

Edit distance walkthrough

TASTE v TREAT

X = TASTE

Y = TREAT

∅ TREAT 5	T TREAT 4	TA TREAT	TAS TREAT	TAST TREAT	TASTE TREAT
∅ TREA 4	T TREA 3	TA TREA	TAS TREA	TAST TREA	TASTE TREA
∅ TRE 3	T TRE 2	TA TRE	TAS TRE	TAST TRE	TASTE TRE
∅ TR 2	T TR 1	TA TR	TAS TR	TAST TR	TASTE TR
∅ T 1	T T 0	TA T 1	TAS T	TAST T	TASTE T
∅ ∅ 0	T ∅ 1	TA ∅ 2	TAS ∅ 3	TAST ∅ 4	TASTE ∅ 5

Edit distance walkthrough

TASTE v TREAT

X = TASTE
Y = TREAT

∅ TREAT	5	T TREAT	4	TA TREAT		TAS TREAT		TAST TREAT		TASTE TREAT	
∅ TREA	4	T TREA	3	TA TREA		TAS TREA		TAST TREA		TASTE TREA	
∅ TRE	3	T TRE	2	TA TRE		TAS TRE		TAST TRE		TASTE TRE	
∅ TR	2	T TR	1	TA TR	1	TAS TR		TAST TR		TASTE TR	
∅ T	1	T T	0	TA T	1	TAS T		TAST T		TASTE T	
∅ ∅	0	T ∅	1	TA ∅	2	TAS ∅	3	TAST ∅	4	TASTE ∅	5

Edit distance walkthrough

TASTE v TREAT

X = TASTE
Y = TREAT

∅ TREAT	5	T TREAT	4	TA TREAT		TAS TREAT		TAST TREAT		TASTE TREAT	
∅ TREA	4	T TREA	3	TA TREA		TAS TREA		TAST TREA		TASTE TREA	
∅ TRE	3	T TRE	2	TA TRE	2	TAS TRE		TAST TRE		TASTE TRE	
∅ TR	2	T TR	1	TA TR	1	TAS TR		TAST TR		TASTE TR	
∅ T	1	T T	0	TA T	1	TAS T		TAST T		TASTE T	
∅ ∅	0	T ∅	1	TA ∅	2	TAS ∅	3	TAST ∅	4	TASTE ∅	5

Edit distance walkthrough

TASTE v TREAT

X = TASTE
Y = TREAT

∅ / TREAT: 5	T / TREAT: 4	TA / TREAT:	TAS / TREAT:	TAST / TREAT:	TASTE / TREAT:
∅ / TREA: 4	T / TREA: 3	TA / TREA: 2	TAS / TREA:	TAST / TREA:	TASTE / TREA:
∅ / TRE: 3	T / TRE: 2	TA / TRE: 2	TAS / TRE:	TAST / TRE:	TASTE / TRE:
∅ / TR: 2	T / TR: 1	TA / TR: 1	TAS / TR:	TAST / TR:	TASTE / TR:
∅ / T: 1	T / T: 0	TA / T: 1	TAS / T:	TAST / T:	TASTE / T:
∅ / ∅: 0	T / ∅: 1	TA / ∅: 2	TAS / ∅: 3	TAST / ∅: 4	TASTE / ∅: 5

Edit distance walkthrough

TASTE v TREAT

X = T A S T E

Y = T R E A T

∅ TREAT 5	T TREAT 4	TA TREAT 3	TAS TREAT 3	TAST TREAT 3	TASTE TREAT 4
∅ TREA 4	T TREA 3	TA TREA 2	TAS TREA 3	TAST TREA 3	TASTE TREA 4
∅ TRE 3	T TRE 2	TA TRE 2	TAS TRE 2	TAST TRE 3	TASTE TRE 3
∅ TR 2	T TR 1	TA TR 1	TAS TR 2	TAST TR 3	TASTE TR 4
∅ T 1	T T 0	TA T 1	TAS T 2	TAST T 3	TASTE T 4
∅ ∅ 0	T ∅ 1	TA ∅ 2	TAS ∅ 3	TAST ∅ 4	TASTE ∅ 5

Edit distance walkthrough

TASTE v TREAT

X = TASTE
Y = TREAT

← = Delete, ↓ = Insert, ↙ = Equal or Substitute

∅ TREAT	5	T TREAT	4	TA TREAT	3	TAS TREAT	3	TAST TREAT	3	TASTE TREAT	4
∅ TREA	4	T TREA	3	TA TREA	2	TAS TREA	3	TAST TREA	3	TASTE TREA	4
∅ TRE	3	T TRE	2	TA TRE	2	TAS TRE	2	TAST TRE	3	TASTE TRE	3
∅ TR	2	T TR	1	TA TR	1	TAS TR	2	TAST TR	3	TASTE TR	4
∅ T	1	T T	0	TA T	1	TAS T	2	TAST T	3	TASTE T	4
∅ ∅	0	T ∅	1	TA ∅	2	TAS ∅	3	TAST ∅	4	TASTE ∅	5

Edit distance walkthrough

TASTE v TREAT

X = TASTE
Y = TREAT

← = Delete, ↓ = Insert, ↙ = Equal or Substitute

∅ / TREAT: 5	T / TREAT: 4	TA / TREAT: 3	TAS / TREAT: 3	TAST / TREAT: 3	TASTE / TREAT: 4
∅ / TREA: 4	T / TREA: 3	TA / TREA: 2	TAS / TREA: 3	TAST / TREA: 3	TASTE / TREA: 4
∅ / TRE: 3	T / TRE: 2	TA / TRE: 2	TAS / TRE: 2	TAST / TRE: 3	TASTE / TRE: 3
∅ / TR: 2	T / TR: 1	TA / TR: 1	TAS / TR: 2	TAST / TR: 3	TASTE / TR: 4
∅ / T: 1	T / T: 0	TA / T: 1	TAS / T: 2	TAST / T: 3	TASTE / T: 4
∅ / ∅: 0	T / ∅: 1	TA / ∅: 2	TAS / ∅: 3	TAST / ∅: 4	TASTE / ∅: 5

Edit distance walkthrough

TASTE v TREAT

X = T A S T E

Y = T R E A T

← = Delete, ↓ = Insert, ↙ = Equal or Substitute

∅ / TREAT: 5	T / TREAT: 4	TA / TREAT: 3	TAS / TREAT: 3	TAST / TREAT: 3	TASTE / TREAT: 4
∅ / TREA: 4	T / TREA: 3	TA / TREA: 2	TAS / TREA: 3	TAST / TREA: 3	TASTE / TREA: 4
∅ / TRE: 3	T / TRE: 2	TA / TRE: 2	TAS / TRE: 2	TAST / TRE: 3	TASTE / TRE: 3
∅ / TR: 2	T / TR: 1	TA / TR: 1	TAS / TR: 2	TAST / TR: 3	TASTE / TR: 4
∅ / T: 1	T / T: 0	TA / T: 1	TAS / T: 2	TAST / T: 3	TASTE / T: 4
∅ / ∅: 0	T / ∅: 1	TA / ∅: 2	TAS / ∅: 3	TAST / ∅: 4	TASTE / ∅: 5

Edit distance walkthrough

TASTE v TREAT

X = T A S T E

Y = T R E A T

← = Delete, ↓ = Insert, ↙ = Equal or Substitute

∅ TREAT	5	T TREAT	4	TA TREAT	3	TAS TREAT	3	TAST TREAT	3	TASTE TREAT	4
∅ TREA	4	T TREA	3	TA TREA	2	TAS TREA	3	TAST TREA	3	TASTE TREA	4
∅ TRE	3	T TRE	2	TA TRE	2	TAS TRE	2	TAST TRE	3	TASTE TRE	3
∅ TR	2	T TR	1	TA TR	1	TAS TR	2	TAST TR	3	TASTE TR	4
∅ T	1	T T	0	TA T	1	TAS T	2	TAST T	3	TASTE T	4
∅ ∅	0	T ∅	1	TA ∅	2	TAS ∅	3	TAST ∅	4	TASTE ∅	5

Edit distance walkthrough

TASTE v TREAT

X = TASTE
Y = TREAT

← = Delete, ↓ = Insert, ↙ = Equal or Substitute

break ties arbitrarily

Ø / TREAT: 5	T / TREAT: 4	TA / TREAT: 3	TAS / TREAT: 3	TAST / TREAT: 3	TASTE / TREAT: 4
Ø / TREA: 4	T / TREA: 3	TA / TREA: 2	TAS / TREA: 3	TAST / TREA: 3	TASTE / TREA: 4
Ø / TRE: 3	T / TRE: 2	TA / TRE: 2	TAS / TRE: 2	TAST / TRE: 3	TASTE / TRE: 3
Ø / TR: 2	T / TR: 1	TA / TR: 1	TAS / TR: 2	TAST / TR: 3	TASTE / TR: 4
Ø / T: 1	T / T: 0	TA / T: 1	TAS / T: 2	TAST / T: 3	TASTE / T: 4
Ø / Ø: 0	T / Ø: 1	TA / Ø: 2	TAS / Ø: 3	TAST / Ø: 4	TASTE / Ø: 5

Edit distance walkthrough

TASTE v TREAT

X = T A S T E

Y = T R E A T

← = Delete, ↓ = Insert, ↙ = Equal or Substitute

∅ TREAT	5	T TREAT	4	TA TREAT	3	TAS TREAT	3	TAST TREAT	3	TASTE TREAT	4
∅ TREA	4	T TREA	3	TA TREA	2	TAS TREA	3	TAST TREA	3	TASTE TREA	4
∅ TRE	3	T TRE	2	TA TRE	2	TAS TRE	2	TAST TRE	3	TASTE TRE	3
∅ TR	2	T TR	1	TA TR	1	TAS TR	2	TAST TR	3	TASTE TR	4
∅ T	1	T T	0	TA T	1	TAS T	2	TAST T	3	TASTE T	4
∅ ∅	0	T ∅	1	TA ∅	2	TAS ∅	3	TAST ∅	4	TASTE ∅	5

Edit distance walkthrough

TASTE v TREAT

X = T A S T E

Y = T R E A T

← = Delete, ↓ = Insert, ↙ = Equal or Substitute

Ø TREAT 5	T TREAT 4	TA TREAT 3	TAS TREAT 3	TAST TREAT 3	TASTE TREAT 4
Ø TREA 4	T TREA 3	TA TREA 2	TAS TREA 3	TAST TREA 3	TASTE TREA 4
Ø TRE 3	T TRE 2	TA TRE 2	TAS TRE 2	TAST TRE 3	TASTE TRE 3
Ø TR 2	T TR 1	TA TR 1	TAS TR 2	TAST TR 3	TASTE TR 4
Ø T 1	T T 0	TA T 1	TAS T 2	TAST T 3	TASTE T 4
Ø Ø 0	T Ø 1	TA Ø 2	TAS Ø 3	TAST Ø 4	TASTE Ø 5

Edit distance walkthrough

TASTE v TREAT

X = T A S T E

Y = T R E A T

← = Delete, ↓ = Insert, ↙ = Equal or Substitute

	∅	T	TA	TAS	TAST	TASTE
TREAT	5	4	3	3	3	4
TREA	4	3	2	3	3	4
TRE	3	2	2	2	3	3
TR	2	1	1	2	3	4
T	1	0	1	2	3	4
∅	0	1	2	3	4	5

Edit distance walkthrough

TASTE v TREAT

X = T A S T E

← = Delete, ↓ = Insert, ↙ = Equal or Substitute

	∅	T	TA	TAS	TAST	TASTE
TREAT	5	4	3	3	3	4
TREA	4	3	2	3	3	4
TRE	3	2	2	2	3	3
TR	2	1	1	2	3	4
T	1	0	1	2	3	4
∅	0	1	2	3	4	5

Edit distance walkthrough

TASTE v TREAT

X = T A S T ~~E~~

← = Delete, ↓ = Insert, ↙ = Equal or Substitute

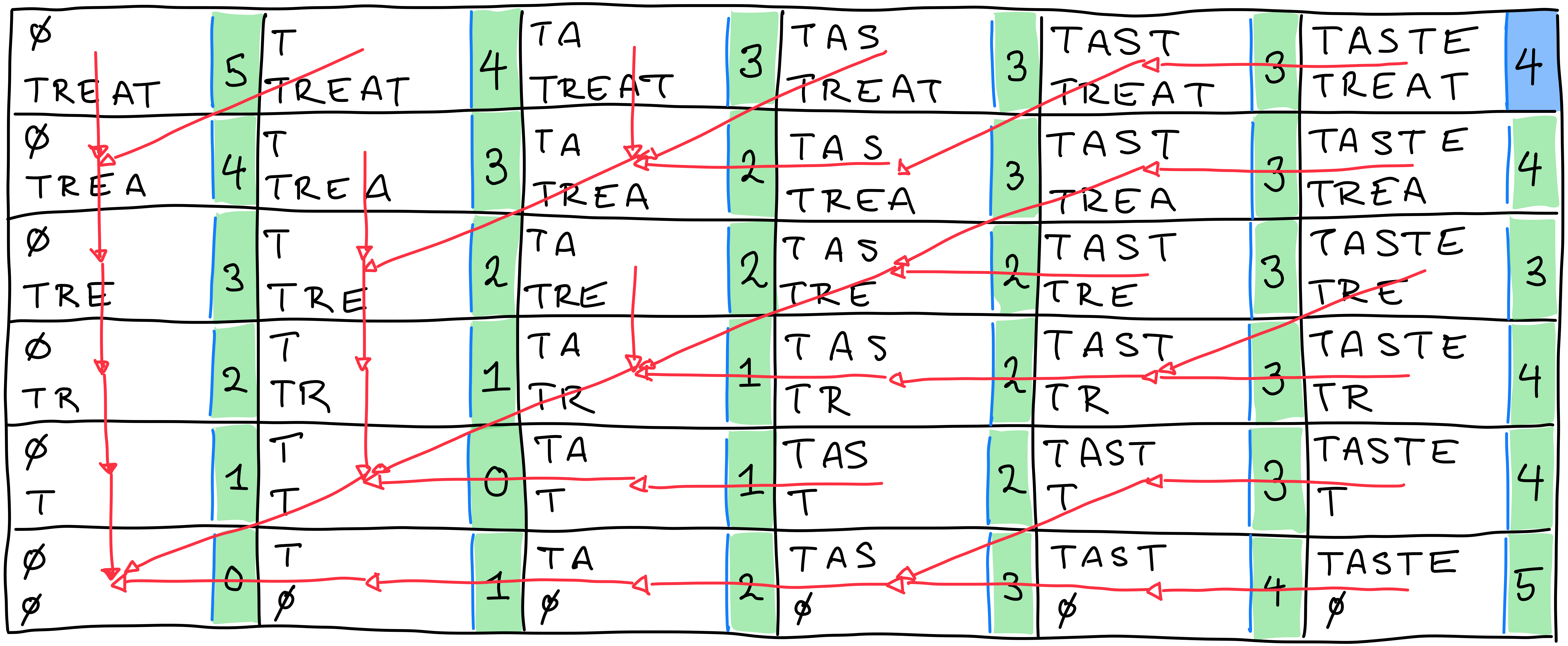
	∅	T	TA	TAS	TAST	TASTE
TREAT	5	4	3	3	3	4
TREA	4	3	2	3	3	4
TRE	3	2	2	2	3	3
TR	2	1	1	2	3	4
T	1	0	1	2	3	4
∅	0	1	2	3	4	5

X = TASTE

Edit distance walkthrough

TASTE v TREAT

← = Delete, ↓ = Insert, ↙ = Equal or Substitute



Edit distance walkthrough

TASTE v TREAT

X = T A ~~S~~ T ~~E~~

← = Delete, ↓ = Insert, ↙ = Equal or Substitute

∅ / TREAT: 5	T / TREAT: 4	TA / TREAT: 3	TAS / TREAT: 3	TAST / TREAT: 3	TASTE / TREAT: 4
∅ / TREA: 4	T / TREA: 3	TA / TREA: 2	TAS / TREA: 3	TAST / TREA: 3	TASTE / TREA: 4
∅ / TRE: 3	T / TRE: 2	TA / TRE: 2	TAS / TRE: 2	TAST / TRE: 3	TASTE / TRE: 3
∅ / TR: 2	T / TR: 1	TA / TR: 1	TAS / TR: 2	TAST / TR: 3	TASTE / TR: 4
∅ / T: 1	T / T: 0	TA / T: 1	TAS / T: 2	TAST / T: 3	TASTE / T: 4
∅ / ∅: 0	T / ∅: 1	TA / ∅: 2	TAS / ∅: 3	TAST / ∅: 4	TASTE / ∅: 5

Edit distance walkthrough

TASTE v TREAT

X = T A ~~S~~ T ~~E~~

← = Delete, ↓ = Insert, ↙ = Equal or Substitute

	∅	T	TA	TAS	TAST	TASTE
TREAT	5	4	3	3	3	4
TREA	4	3	2	3	3	4
TRE	3	2	2	2	3	3
TR	2	1	1	2	3	4
T	1	0	1	2	3	4
∅	0	1	2	3	4	5

Edit distance walkthrough

TASTE v TREAT

X = T A S T E

← = Delete, ↓ = Insert, ↙ = Equal or Substitute

	∅	T	TA	TAS	TAST	TASTE
TREAT	5	4	3	3	3	4
TREA	4	3	2	3	3	4
TRE	3	2	2	2	3	3
TR	2	1	1	2	3	4
T	1	0	1	2	3	4
∅	0	1	2	3	4	5

Edit distance walkthrough

TASTE v TREAT

X = T A S T E

RE

← = Delete, ↓ = Insert, ↙ = Equal or Substitute

∅ / TREAT: 5	T / TREAT: 4	TA / TREAT: 3	TAS / TREAT: 3	TAST / TREAT: 3	TASTE / TREAT: 4
∅ / TREA: 4	T / TREA: 3	TA / TREA: 2	TAS / TREA: 3	TAST / TREA: 3	TASTE / TREA: 4
∅ / TRE: 3	T / TRE: 2	TA / TRE: 2	TAS / TRE: 2	TAST / TRE: 3	TASTE / TRE: 3
∅ / TR: 2	T / TR: 1	TA / TR: 1	TAS / TR: 2	TAST / TR: 3	TASTE / TR: 4
∅ / T: 1	T / T: 0	TA / T: 1	TAS / T: 2	TAST / T: 3	TASTE / T: 4
∅ / ∅: 0	T / ∅: 1	TA / ∅: 2	TAS / ∅: 3	TAST / ∅: 4	TASTE / ∅: 5

Edit distance walkthrough

TASTE v TREAT

X = T A S T E

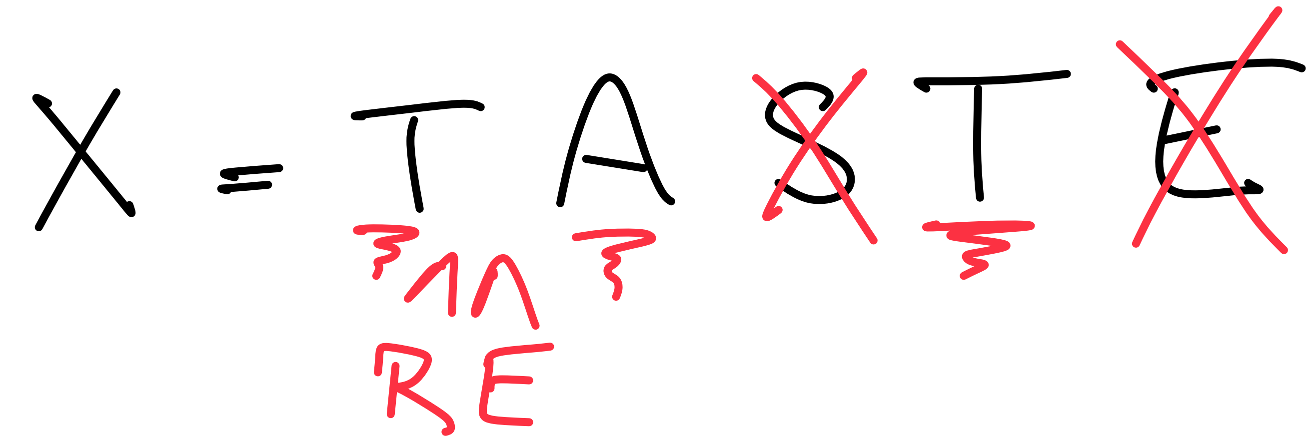
RE

← = Delete, ↓ = Insert, ↙ = Equal or Substitute

∅ / TREAT: 5	T / TREAT: 4	TA / TREAT: 3	TAS / TREAT: 3	TAST / TREAT: 3	TASTE / TREAT: 4
∅ / TREA: 4	T / TREA: 3	TA / TREA: 2	TAS / TREA: 3	TAST / TREA: 3	TASTE / TREA: 4
∅ / TRE: 3	T / TRE: 2	TA / TRE: 2	TAS / TRE: 2	TAST / TRE: 3	TASTE / TRE: 3
∅ / TR: 2	T / TR: 1	TA / TR: 1	TAS / TR: 2	TAST / TR: 3	TASTE / TR: 4
∅ / T: 1	T / T: 0	TA / T: 1	TAS / T: 2	TAST / T: 3	TASTE / T: 4
∅ / ∅: 0	T / ∅: 1	TA / ∅: 2	TAS / ∅: 3	TAST / ∅: 4	TASTE / ∅: 5

Edit distance walkthrough

TASTE v TREAT

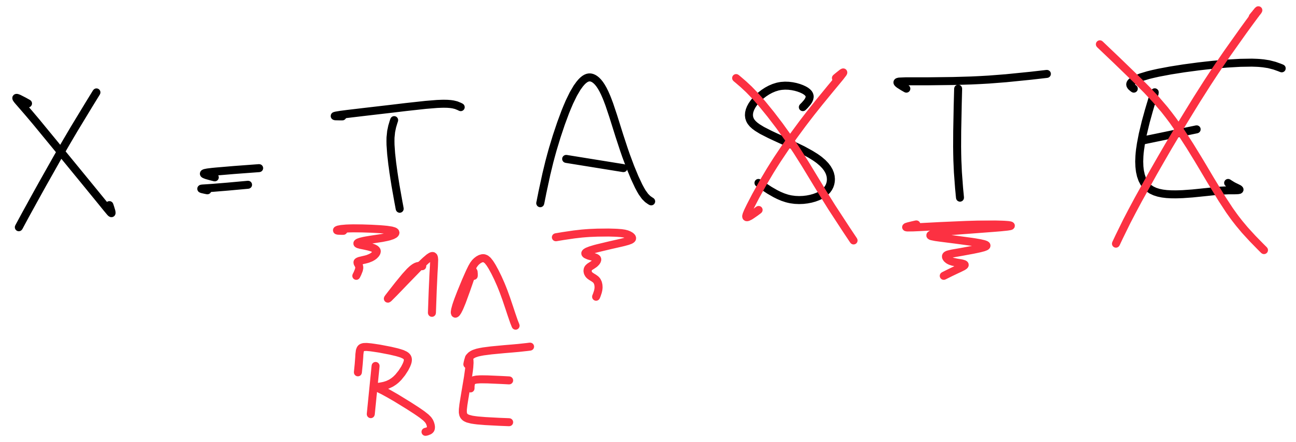


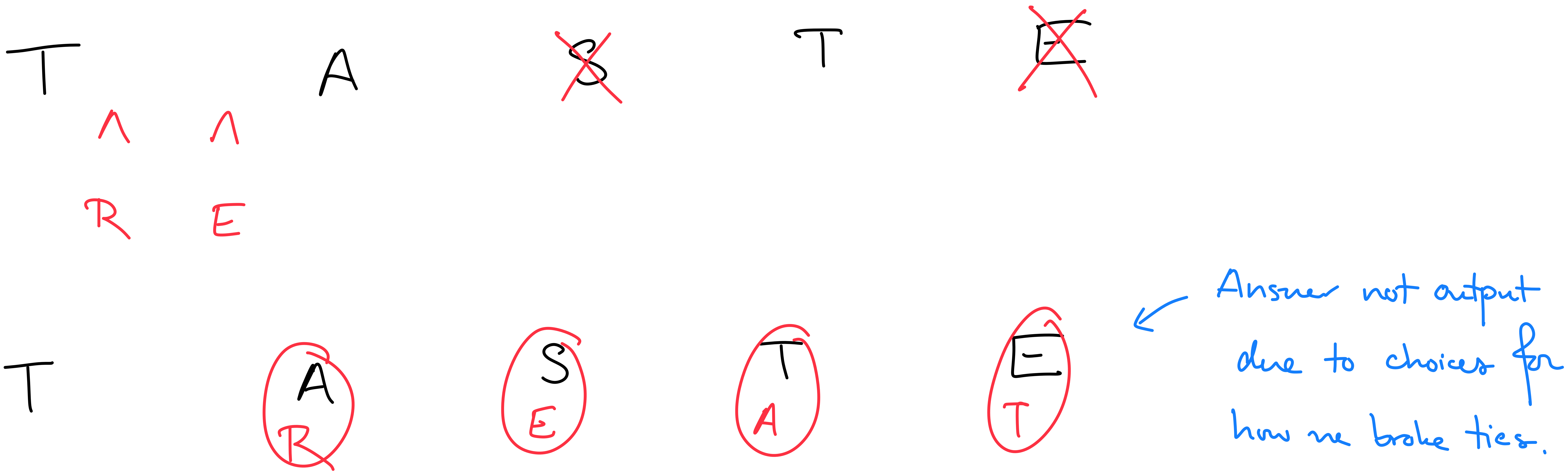
← = Delete, ↓ = Insert, ↙ = Equal or Substitute

	∅	T	TA	TAS	TAST	TASTE
TREAT	5	4	3	3	3	4
TREA	4	3	2	3	3	4
TRE	3	2	2	2	3	3
TR	2	1	1	2	3	4
T	1	0	1	2	3	4
∅	0	1	2	3	4	5

Edit distance walkthrough

TASTE v TREAT





Proof of correctness

- For dynamic programming, proof of correctness is often the easiest part of the proof!
- Because the problem is recursively defined, the proof should also be recursive — I.e., we prove the correctness inductively
- **Base cases**: $d(k, \ell)$ when $k = 0$ or $\ell = 0$
- **Induction:** Argue correctness of $d(k, \ell)$ from “smaller” problems
    - When computing $d(k, \ell)$, either the last chars agree or disagree
    - If they agree, then we can edit the $k - 1$ string to the $\ell - 1$ string
    - If they disagree, then we can either delete, insert, or substitute the last char
    - In all 4 cases, our problem simplifies to a subproblem
    - Since we consider **exhaustively** all possible choices for the last char, we are guaranteed that our optimization will be minimal over all edit distances

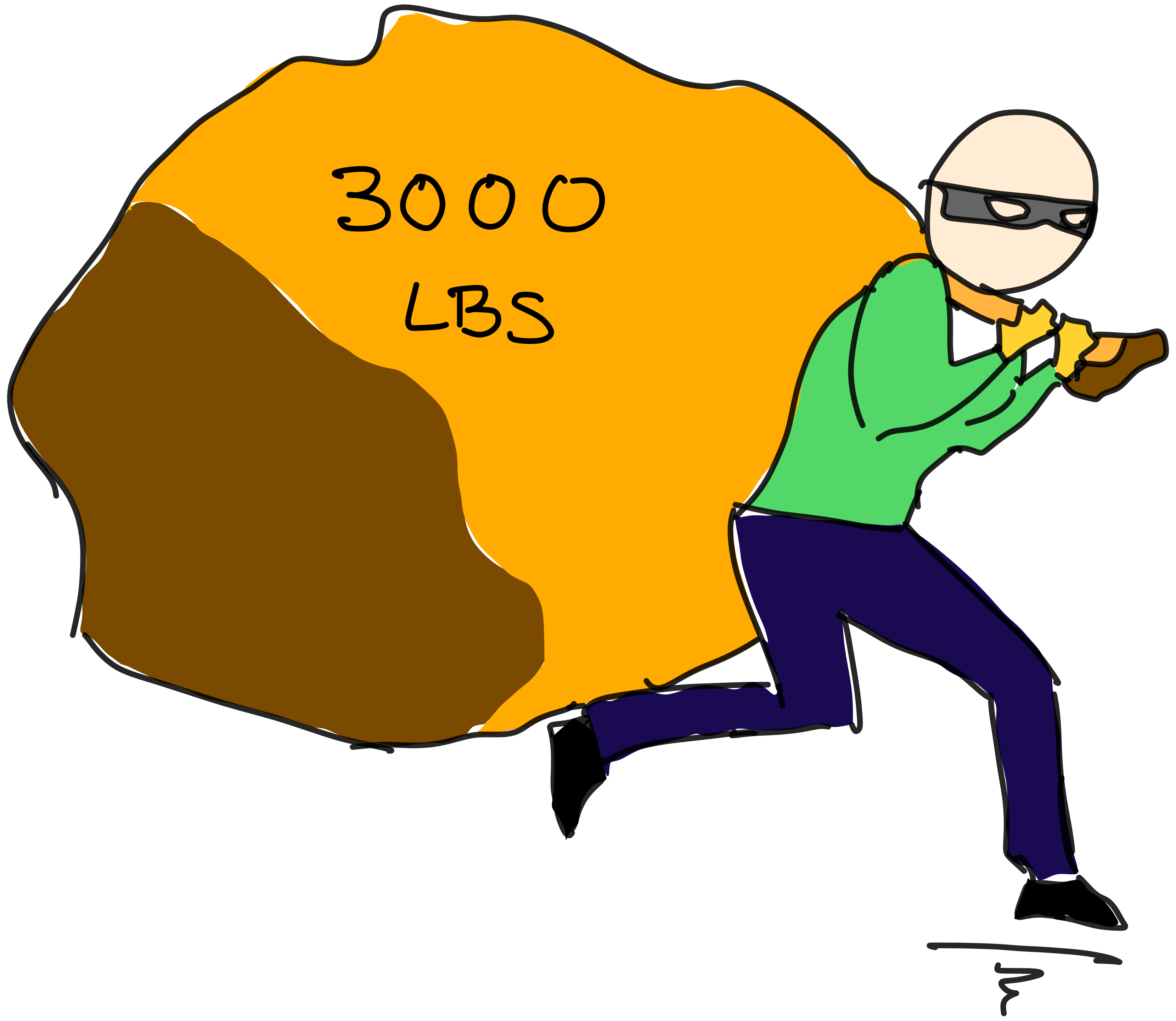
Proof of correctness

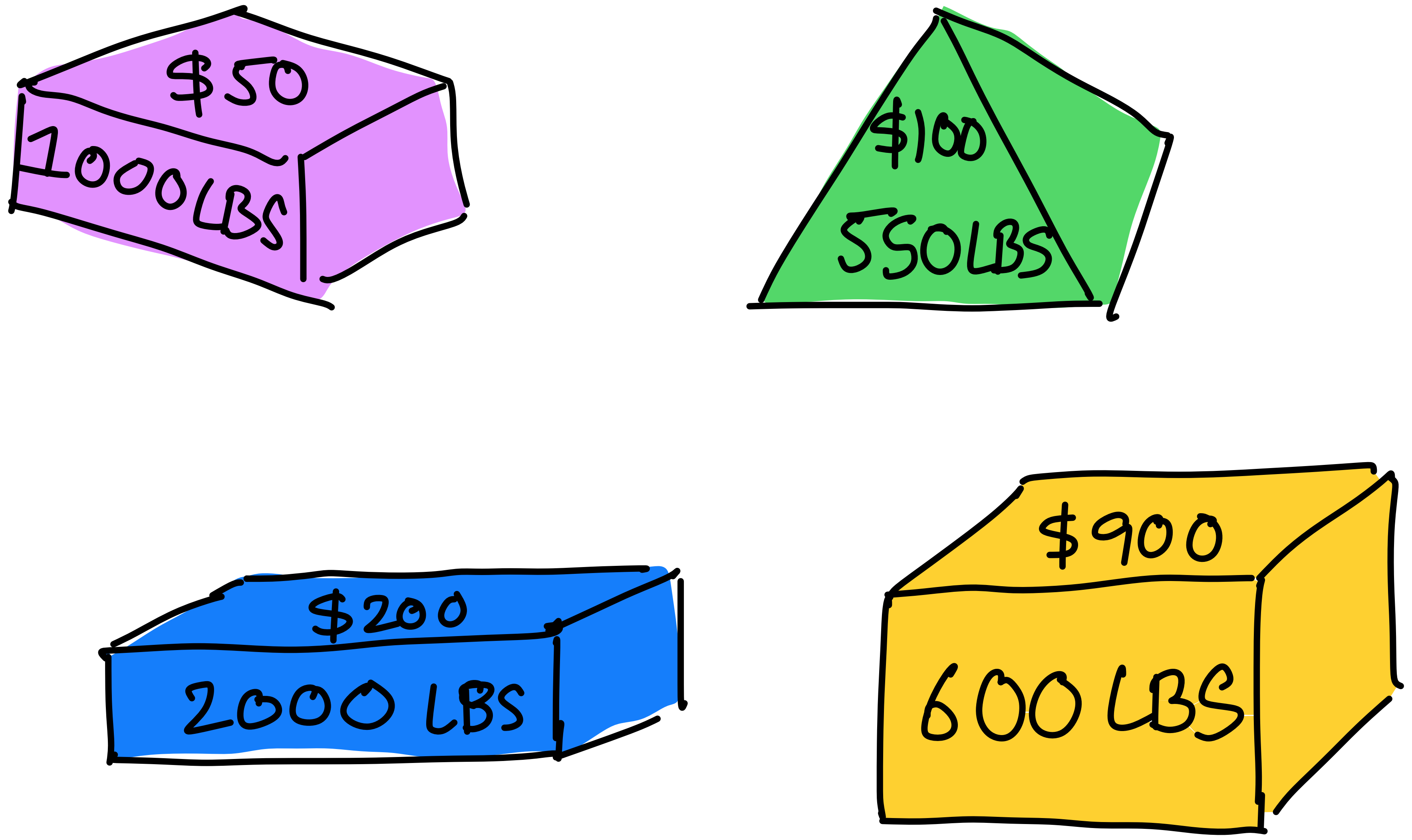
- This proves that the length of the edit distance is minimal.
- Proving that we find the correct sequence of edits follows next.
  - Having recorded which subproblem is minimal, we identify a path from the (k, ℓ) vertex to the root consisting of $d(k, \ell)$ edits as each constructed edge corresponds to an edit or preservation of the last char.
  - This finds a sequence of $d(k, \ell)$ edits. We proved this was the optimal length so we are done.

Knapsack problem

The Knapsack problem

Maximize the items grabbed subject to the weight constraint of the bag.

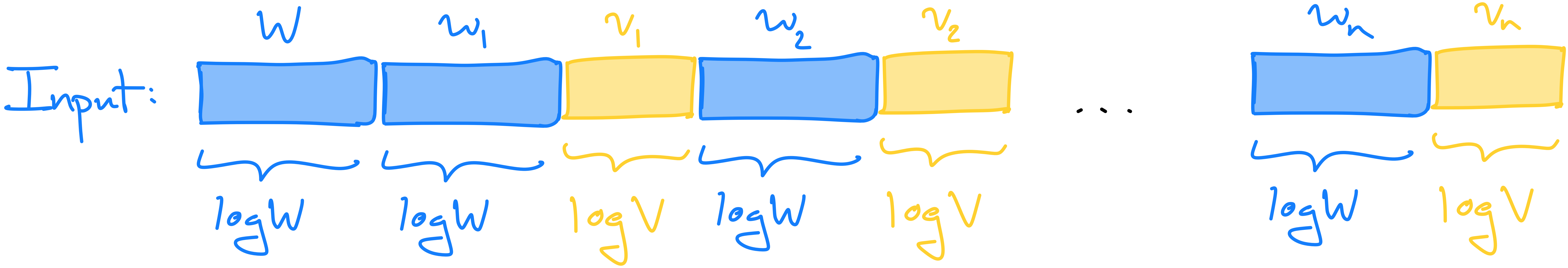




The Knapsack problem

Let $V = \sum_{i=1}^{n} v_i$, the max value of any set.

- **Input**: Items with **integer** weights $w_1, \ldots w_n \in \mathbb{N}$ and values $v_1, \ldots, v_n \in \mathbb{N}$ and a max weight $W \in \mathbb{N}$
- **Output**: Subset $S \subseteq [n]$ such that $\sum_{i \in S} w_i \leq W$ and maximizing $\sum_{i \in S} v_i$.



Input length: $(n+1)\log W + n \log V = \Theta(n \log VW)$

The Knapsack problem

- **Input:** Items with **integer** weights $w_1, \ldots w_n \in \mathbb{N}$ and values $v_1, \ldots, v_n \in \mathbb{N}$ and a max weight $W \in \mathbb{N}$

- **Output:** Subset $S \subseteq [n]$ such that $\sum_{i \in S} w_i \leq W$ and maximizing $\sum_{i \in S} v_i$.

Let $V = \sum_{i=1}^{n} v_i$, the max value of any set.

- **Brute force solution:** Check all 2^n possible S and choose the optimal S amongst those satisfying the weight constraint.

    - **Runtime:** $O(n \cdot 2^n \log VW)$

$= \log V + \log W$ ← arithmetic complexity of adding numbers $\leq W$ or $\leq V$.

A better dynamic programming algorithm

- **Observation:** Either item i is included in S or it is not
- Defining an appropriate subproblem
- Let $S(i, W')$ be the optimal subset $S \subseteq \{1,\ldots,i\}$ such that S's items have net weight $\leq W'$ and let $V(i, W)$ be their optimal value
- **Base cases:** $S(\cdot,0) = S(0,\cdot) = \varnothing, V(\cdot,0) = V(0,\cdot) = 0.$
- **Target problem:** $S(n, W)$ and $V(n, W)$

A better dynamic programming algorithm

- Let $S(i, W')$ be the optimal subset $S \subseteq \{1, \ldots, i\}$ such that S's items have net weight $\leq W'$ and let $V(i, W)$ be their optimal value
- To calculate $S(i, W')$, if we include item i
  - Value of bag is at least v_i and bag now has remainder available weight $W' - w_i$
  - Need to recursively choose between items $\{1, \ldots, i-1\}$
- Else
  - Bag still has remainder available weight W'
  - Need to recursively choose between items $\{1, \ldots, i-1\}$

A better dynamic programming algorithm

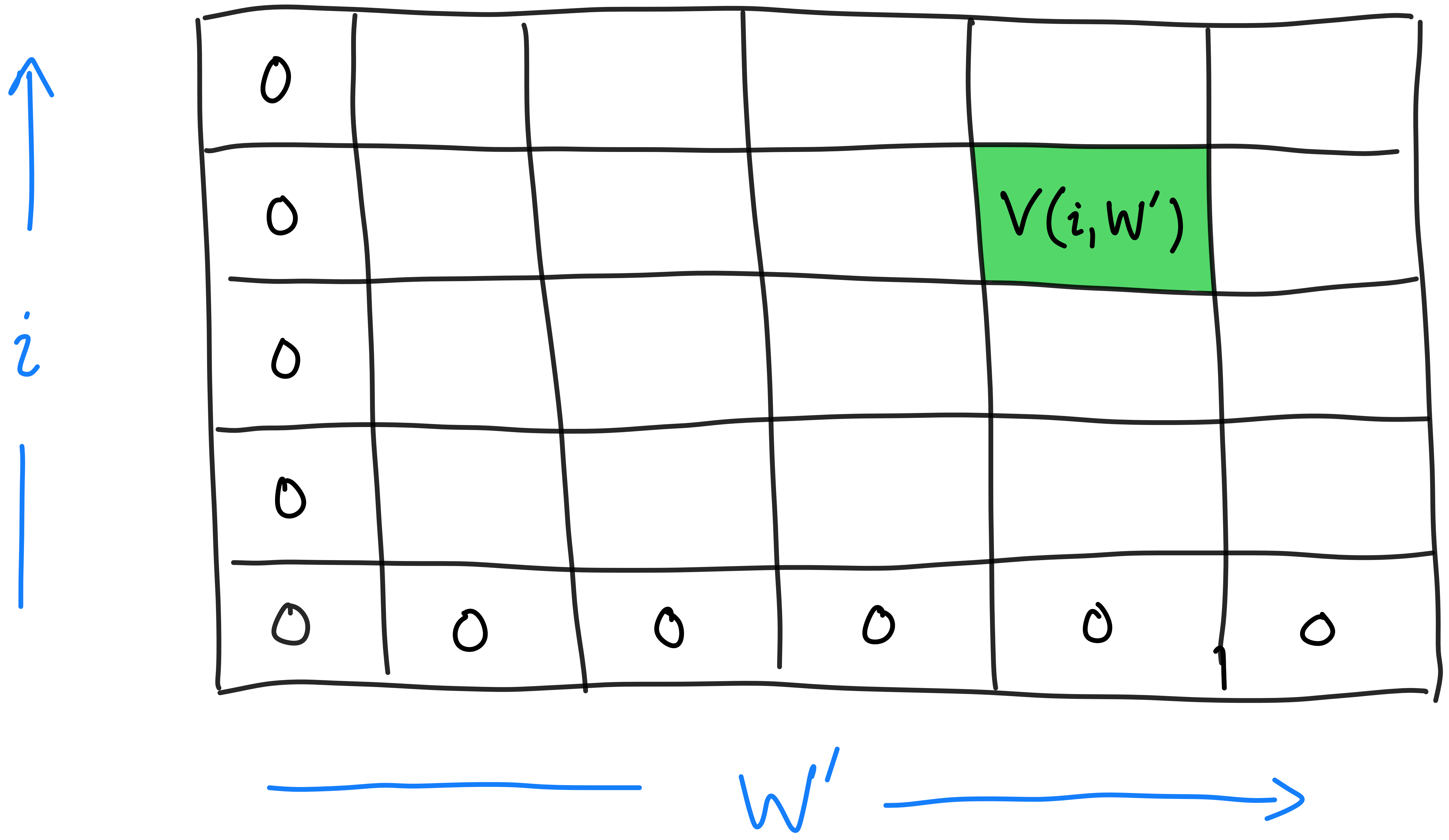
- Let $S(i, W')$ be the optimal subset $S \subseteq \{1,\ldots,i\}$ such that S's items have net weight $\leq W'$ and let $V(i, W')$ be their optimal value

$$V(i, W') = \max \left\{ \begin{array}{c} V(i-1, W'-w_i) + v_i, \\ V(i-1, W') \end{array} \right\}$$

- Depending on maximization, $S(i, W') = S(i-1, W'-w_i) \cup \{i\}$ or $S(i, W') = S(i-1, W')$ respectively.

Memoization for Knapsack

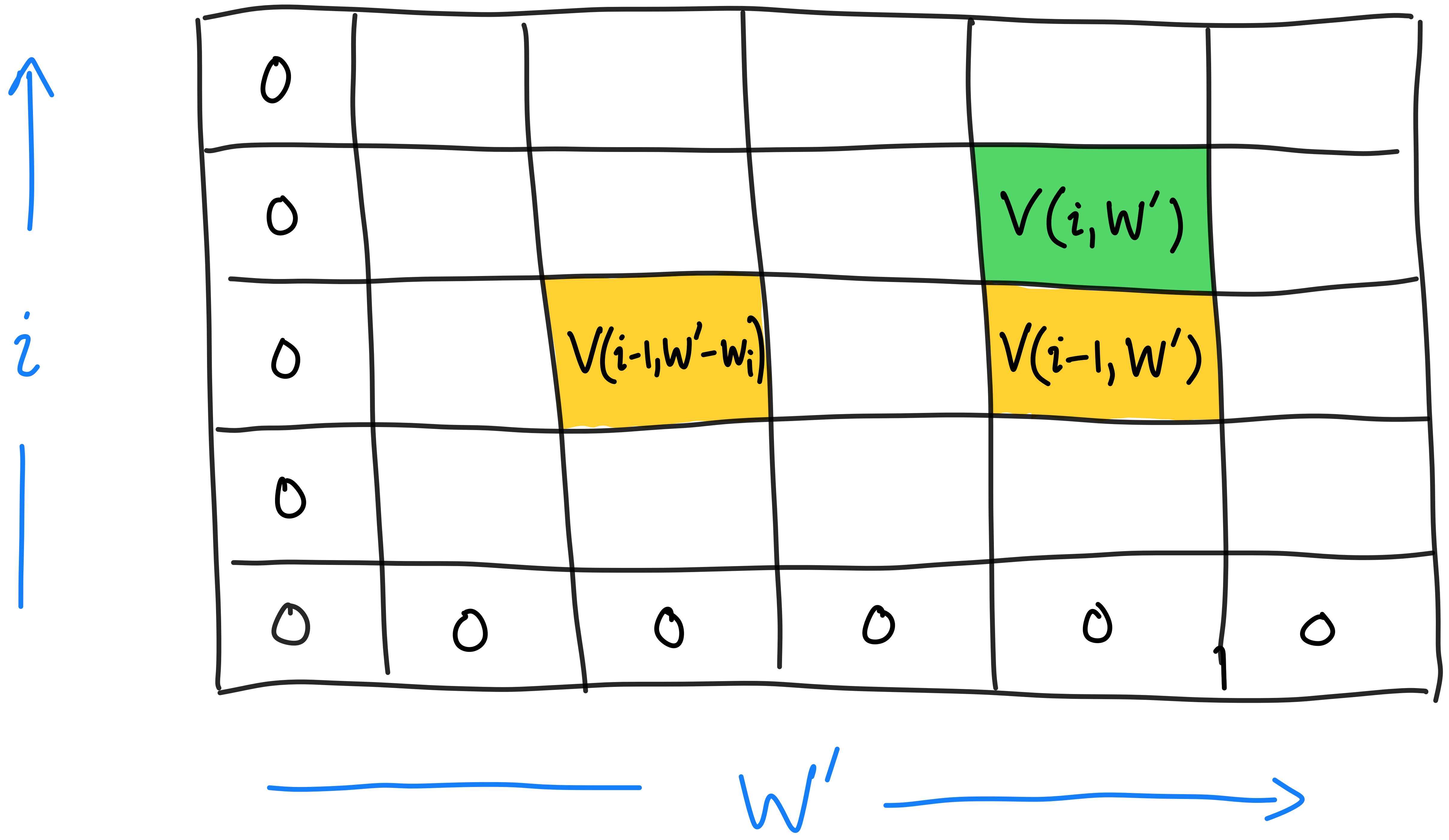
Table of $V(i, W')$:



Memoization for Knapsack

$$V(i, W') = \max \begin{Bmatrix} V(i-1, W'-w_i) + v_i, \\ V(i-1, W') \end{Bmatrix}$$

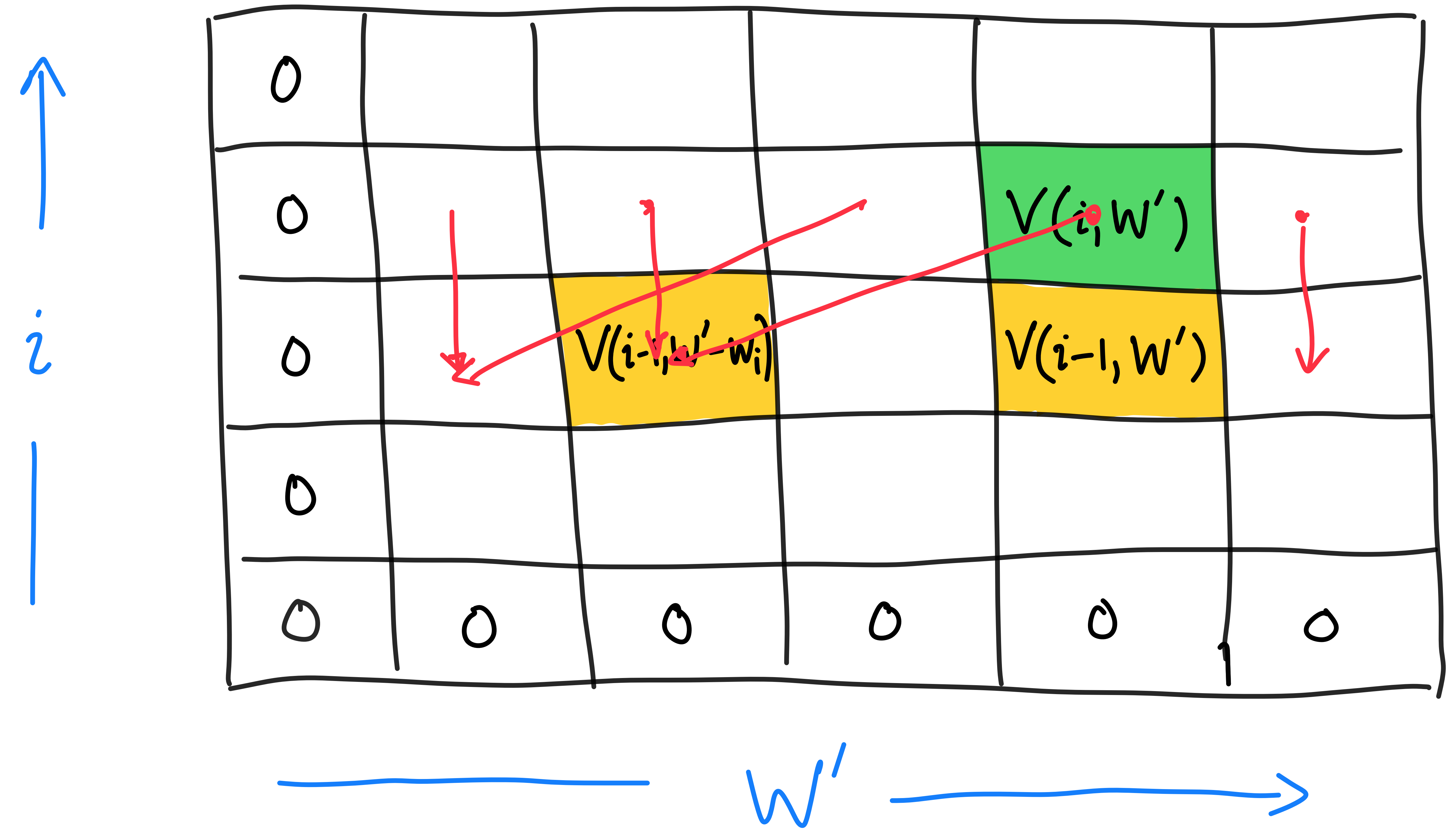
Table of V(i, W'):



Memoization for Knapsack

$$V(i, W') = \max \begin{Bmatrix} V(i-1, W'-w_i) + v_i, \\ V(i-1, W') \end{Bmatrix}$$

Table of $V(i, W')$:



Each edge ↓ or ↙ goes from $(i, \cdot)$ to $(i-1, \cdot)$.

Record for every (i, W') if we are going to include or exclude item i.

Knapsack dynamic programming algorithm

short for "Include"

- **Generate tables:**
  - Let V, Inc be $(n+1) \times (W+1)$ sized tables and set $V(0,\cdot) = V(\cdot,0) \leftarrow 0$.
  - For i from 1 to n, W' from 1 to W
    - If $V(i-1,W') > V(i-1,W'-w_i) + v_i$
      - Then, set $V(i,W') \leftarrow V(i-1,W')$ and set $\mathrm{Inc}(i,W') = \mathrm{false}$
      - Else, set $V(i,W') \leftarrow V(i-1,W'-w_i) + v_i$ and set $\mathrm{Inc}(i,W') = \mathrm{true}$

Knapsack dynamic programming algorithm

- Using precomputed Inc terms, walk from (n, W) to $(0, \cdot)$ finding the items to include.
- **Find optimal Knapsack:**
  - Set $(i, W') \leftarrow (n, W)$. Set $S \leftarrow \emptyset$.
  - While $i \neq 0$,
    - If $\text{Inc}(i, W') = \text{true}$,
      - Then, $S \leftarrow S \cup \{i\}$ and $(i, W') \leftarrow (i - 1, W' - w_i)$.
      - Else, $(i, W') \leftarrow (i - 1, W')$.
  - Return S.

Knapsack dynamic programming algorithm

Runtime analysis

- Tables are of size $O(nW)$ and computing each entry takes $O(\log VW)$ time given past entries
- Total compute time of tables is $O(nW \log VW)$
- To find the set S, path walks from $(i, \cdot\,)$ to $(i-1, \cdot)$ each step. The path has length $\leq n$.
- Computing S takes time $O(n)$.
- **Total computation time**: $O(nW \log VW)$.

Knapsack runtime

- The input for Knapsack is usually written in **binary** with each item weight w_i expressed with $O(\log W)$ bit numbers and value with $O(\log V)$ bit numbers
- Total input length is $\Theta(n \log V + n \log W) = \Theta(n \log VW)$
- Runtime of Knapsack brute-force alg is $O(n2^n \log VW)$, exp in input length
- Runtime of Knapsack DP alg is $O(nW \log VW)$ also exp in the input length
- This is expected. The decision version of Knapsack is a NP-complete problem. We do not expect an efficient algorithm for Knapsack.

polynomial time in the input length

Approximation algorithms

- We've only alluded to NP-completeness so far, but the NP-completeness of the Knapsack problem means that we strongly believe that there is *no* algorithm for optimizing Knapsack that runs in time

$$O(\text{poly}(n \log VW)) = O(n^c \text{polylog} VW)$$

- Instead we will have to turn to **approximation algorithms**

- Given a Knapsack problem $(v_1, \ldots, v_n, w_1, \ldots, w_n, W)$, let OPT be the optimal value of subset of items weighing $\leq W$:

$$\text{OPT} = V(n, W)$$

Approximation algorithms

- Instead we will have to turn to **approximation algorithms**
- Given a Knapsack problem $(v_1, \ldots, v_n, w_1, \ldots, w_n, W)$, let OPT be the optimal value of subset of items weighing $\leq W$:

$$\mathrm{OPT} = V(n, W)$$

- An alg. $\mathscr{A}$ is an **ϵ-approximation alg.** if $\mathscr{A}$ always outputs a subset $\tilde{S}$ such that (a) $\mathrm{weight}(\tilde{S}) \leq W$ and (b) $\mathrm{value}(\tilde{S}) \geq (1 - \epsilon) \cdot \mathrm{OPT}$.
- Our target today: Come up with an efficient algorithm for constant ϵ (like 0.01)

Knapsack approximation algorithm

- **Theorem:** For every $\epsilon > 0$, there exists an ϵ-approximation alg. for n-item Knapsack that runs in time $O\left(\frac{n^3 \log(VW)}{\epsilon}\right)$.
- The construction will be another dynamic programming algorithm.
- However, we will have to make adjustments to not depend on W.