



*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

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

Runtime = (Total number of subproblems) \times (Time it takes to solve problems
given solutions to subproblems)

Today

Edit distance walkthrough

Edit distance walkthrough

TASTE v TREAT

$X = \overline{\text{T A S T E}}$
 $Y = \text{T R E A T}$

Recall def of subproblem: $X_k = k$ prefix of X
 $Y_\ell = \ell$ prefix of Y

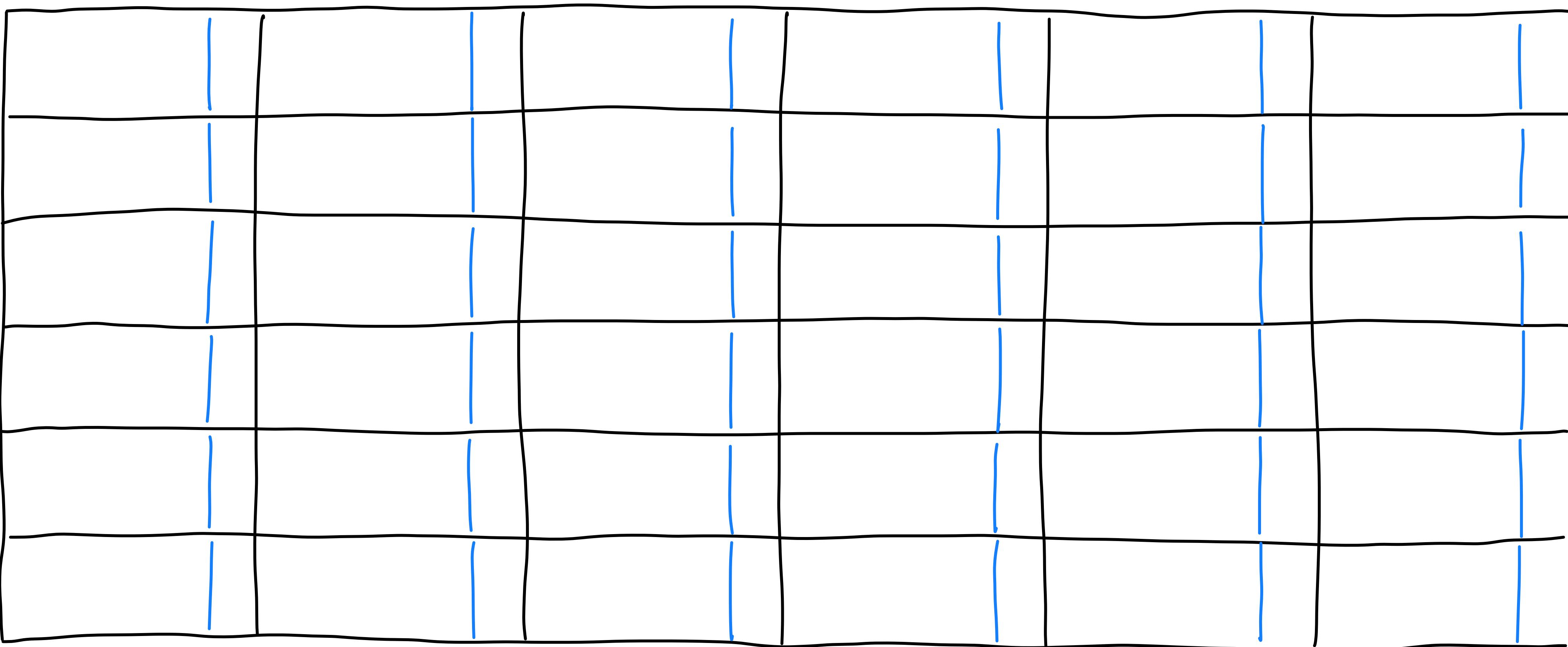
$d(k, \ell)$ = dist between X_k & Y_ℓ

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

Edit distance walkthrough

TASTE v TREAT

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



Edit distance walkthrough

TASTE v TREAT

$X = \text{TASTE}$
 $Y = \text{TREAT}$

								TASTE	TREAT

Edit distance walkthrough

TASTE v TREAT

$X = \text{TASTE}$
 $Y = \text{TREAT}$

\emptyset								TASTE	TREAT
TREAT									
\emptyset									
TREA									
\emptyset									
TRE									
\emptyset									
TR									
\emptyset									
T									
\emptyset	T		TA	TAS	TAST	TASTE			
\emptyset	\emptyset		\emptyset	\emptyset	\emptyset	\emptyset			

Edit distance walkthrough

TASTE v TREAT

X = T A S T E
Y = T R E A T

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

Edit distance walkthrough

TASTE v TREAT

X = T A S T E
Y = T R E A T

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

Edit distance walkthrough

TASTE v TREAT

X = T A S T E
Y = T R E A T

\emptyset	5	T TREAT	TA TREAT	TAS TREAT	TAST TREAT	TA TREAT
\emptyset	4	T TREA	TA TREA	TAS TREA	TAST TREA	TA TREA
\emptyset	3	T TRE	TA TRE	TAS TRE	TAST TRE	TA TRE
\emptyset	2	T TR	TA TR	TAS TR	TAST TR	TA TR
\emptyset	1	T T	TA T	TAS T	TAST T	TA T
\emptyset	0	T \emptyset	TA \emptyset	TAS \emptyset	TAST \emptyset	TA \emptyset

Edit distance walkthrough

TASTE v TREAT

X = T A S T E
Y = T R E A T

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

Edit distance walkthrough

TASTE v TREAT

X = T A S T E
Y = T R E A T

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

Edit distance walkthrough

TASTE v TREAT

X = T A S T E
Y = T R E A T

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

Edit distance walkthrough

TASTE v TREAT

X = T A S T E
Y = T R E A T

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

Edit distance walkthrough

TASTE v TREAT

X = T A S T E
Y = T R E A T

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

Edit distance walkthrough

TASTE v TREAT

X = T A S T E
Y = T R E A T

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

Edit distance walkthrough

TASTE v TREAT

X = T A S T E
Y = T R E A T

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

Edit distance walkthrough

TASTE v TREAT

X = T A S T E
Y = T R E A T

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

Edit distance walkthrough

TASTE v TREAT

X = T A S T E
Y = T R E A T

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

Edit distance walkthrough

TASTE v TREAT

X = T A S T E
Y = T R E A T

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

Edit distance walkthrough

TASTE v TREAT

X = T A S T E
Y = T R E A T

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

Edit distance walkthrough

TASTE v TREAT

X = T A S T E
Y = T R E A T

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

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

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

Edit distance walkthrough

TASTE v TREAT

$X = \text{TASTE}$
 $Y = \text{TREAT}$

\leftarrow = Delete, \downarrow = Insert, \simeq = Equal or Substitute

break ties
arbitrarily

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

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

The table illustrates the edit distance between 'TASTE' (X) and 'TREAT' (Y). The cost matrix shows the minimum edit distance for each character comparison. Red arrows indicate the path of edits: deletions (down), insertions (left), and substitutions/equalities (diagonal). The final edit distance is 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	TREAT	4	TREAT	TREAT	TREAT
∅	4	T	TA	TAS	TAST	TASTE
TREA	3	TREA	3	TREA	TREA	TREA
∅	3	T	TA	TAS	TAST	TASTE
TRE	2	TRE	2	TRE	TRE	TRE
∅	2	T	TA	TAS	TAST	TASTE
TR	1	TR	1	TR	TR	TR
∅	1	T	TA	TAS	TAST	TASTE
T	0	T	0	T	T	T
∅	0	T	TA	TAS	TAST	TASTE
∅	1	∅	1	∅	2	3

The table illustrates the edit distance between the words "TASTE" and "TREAT". The rows represent the sequence of characters in "TREAT" and the columns represent the sequence in "TASTE". The cost of each edit operation is shown in the cells: a red double-headed arrow indicates a deletion (Delete), a red downward arrow indicates an insertion (Insert), and a red diagonal arrow indicates an equal or substitutive edit (Equal or Substitute). The final edit distance is 5, with the path from "TREAT" to "TASTE" highlighted by red arrows.

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

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

Edit distance walkthrough

TASTE v TREAT

\leftarrow = Delete, \downarrow = Insert, \simeq = Equal or Substitute

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

~~X = T A S T E~~

Edit distance walkthrough

TASTE v TREAT

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

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

~~X = T A S T E~~

Edit distance walkthrough

TASTE v TREAT

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

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

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

Edit distance walkthrough

TASTE v TREAT

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

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

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

Edit distance walkthrough

TASTE v TREAT

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

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

~~X = T A S T E~~

~~A~~
~~E~~

Edit distance walkthrough

TASTE v TREAT

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

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

The table illustrates the edit distance between the words "TREAT" and "TASTE". The rows represent the source word "TREAT" and the columns represent the target word "TASTE". The cost of each edit operation is shown in the cells, with red arrows indicating the path of operations: deletions (down), insertions (left), and substitutions (diagonal). The final cost of 5 is marked with a red arrow at the bottom right.

~~X = T A S T E~~

^{TA}
^{RE}

Edit distance walkthrough

TASTE v TREAT

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

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

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

~~T~~
~~A~~
~~S~~
~~T~~
~~E~~

Edit distance walkthrough

TASTE v TREAT

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

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

The diagram shows a 7x10 grid of edit distance matrices for the words "TREAT" and "TASTE". The columns are labeled with the characters of "TREAT" and the rows with the characters of "TASTE". The matrices are color-coded: green for the first 6 rows and blue for the last row. Red arrows indicate the operations: deletions (downward), insertions (leftward), and substitutions/equalities (diagonal). The cost for each operation is indicated by the numbers in the cells. The final cost of 5 is highlighted in red at the bottom right.

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

~~T~~
~~A~~
~~S~~
~~T~~
~~E~~

Edit distance walkthrough

TASTE v TREAT

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

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

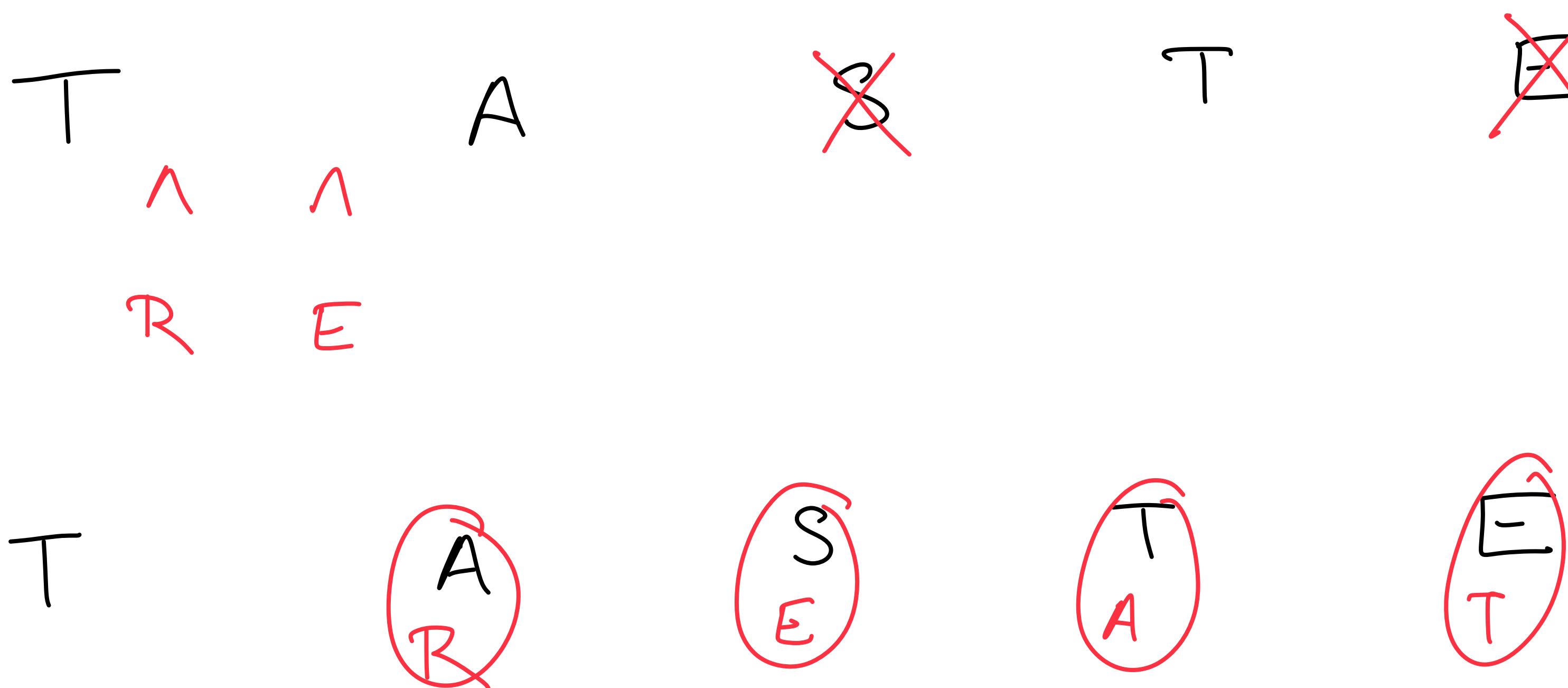
The diagram shows a 7x10 grid of edit distance matrices for the words "TREAT" and "TASTE". The columns are labeled with the characters of "TREAT" and the rows with the characters of "TASTE". The matrices are color-coded: green for the first 6 rows and blue for the last row. Red arrows indicate the operations: deletions (downward), insertions (leftward), and substitutions/equalities (diagonal). The final matrix shows a total edit distance of 5, with the word "TASTE" underlined.

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

~~TA~~
~~RE~~

Edit distance walkthrough

TASTE v TREAT



Answer not output
due to choices for
how we broke ties.

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

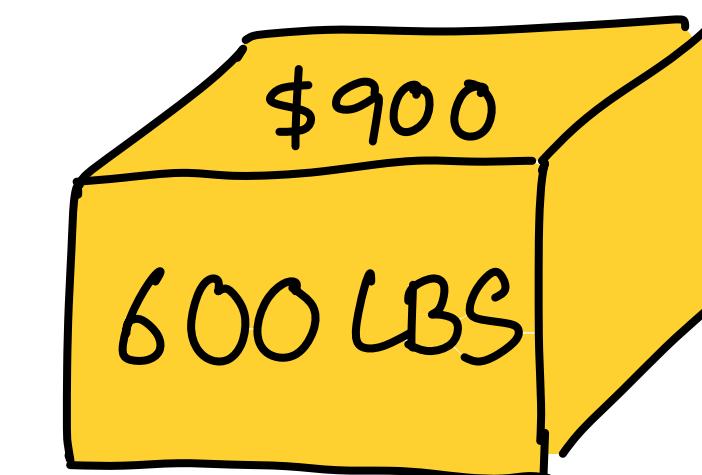
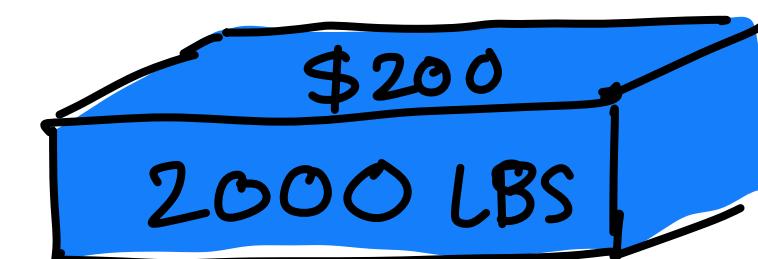
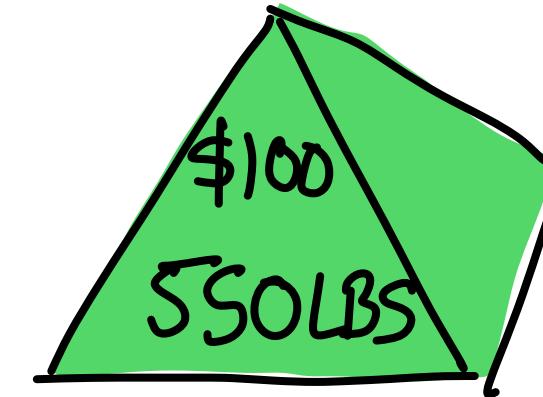
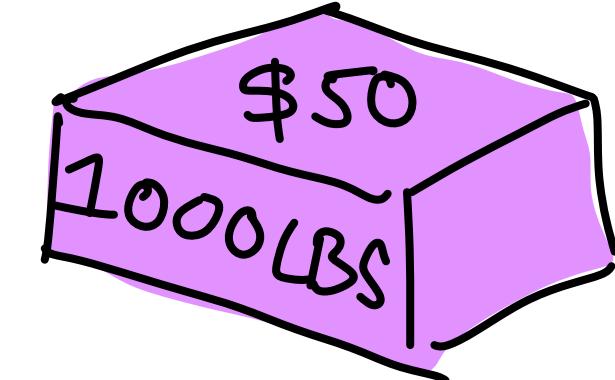
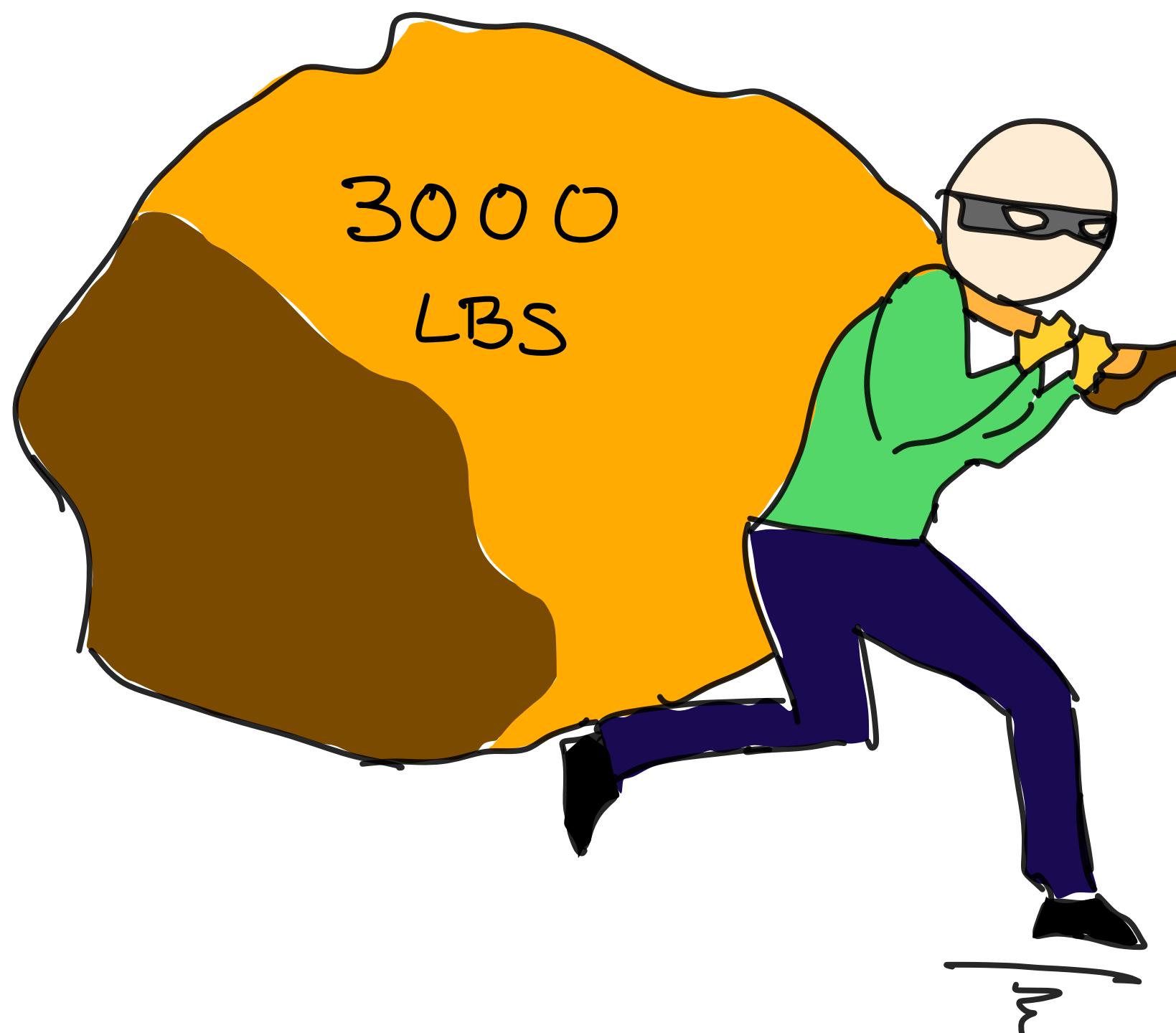
Proof of correctness

- 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

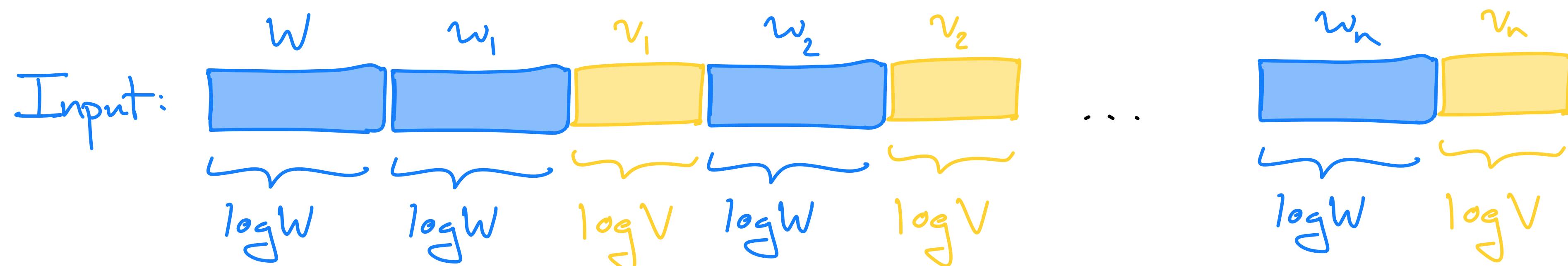
Maximize the items grabbed subject to the weight constraint of the bag.



The Knapsack problem

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

- **Input:** Items with **integer** weights $w_1, \dots, w_n \in \mathbb{N}$ and values $v_1, \dots, v_n \in \mathbb{N}$ 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 $w_1, \dots, w_n \in \mathbb{N}$ and values $v_1, \dots, v_n \in \mathbb{N}$ 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 $\sum_{i \in S} v_i$.
- **Brute force solution:** Check all 2^n possible S and choose the optimal S amongst those satisfying the weight constraint.

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

$\underbrace{2^n}_{= \log V + \log W}$ ← 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 $S \subseteq \{1, \dots, i\}$ such that S 's items have net weight $\leq W'$ and let $V(i, W)$ be their optimal value
- **Base cases:** $S(\cdot, 0) = S(0, \cdot) = \emptyset, V(\cdot, 0) = V(0, \cdot) = 0.$
- **Target problem:** $S(n, W)$ and $V(n, W)$

A better dynamic programming algorithm

- Let $S(i, W')$ be the optimal subset $S \subseteq \{1, \dots, i\}$ 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 $\{1, \dots, i - 1\}$
- Else
 - Bag still has remainder available weight W'
 - Need to recursively choose between items $\{1, \dots, i - 1\}$

A better dynamic programming algorithm

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

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

- 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 $V(i, w')$:

0	0	0	0	0	0
0					$V(i, w')$
0					
0					
0					
0	0	0	0	0	1

Diagram illustrating the memoization table for the Knapsack problem. The table has 6 rows and 6 columns. The columns are labeled with 'i' and the rows with 'w''. The cell $V(i, w')$ is highlighted in green. A blue arrow points upwards from the label 'i' to the top of the table, and another blue arrow points to the right from the label 'w'' to the right of the table.

Memoization for Knapsack

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

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

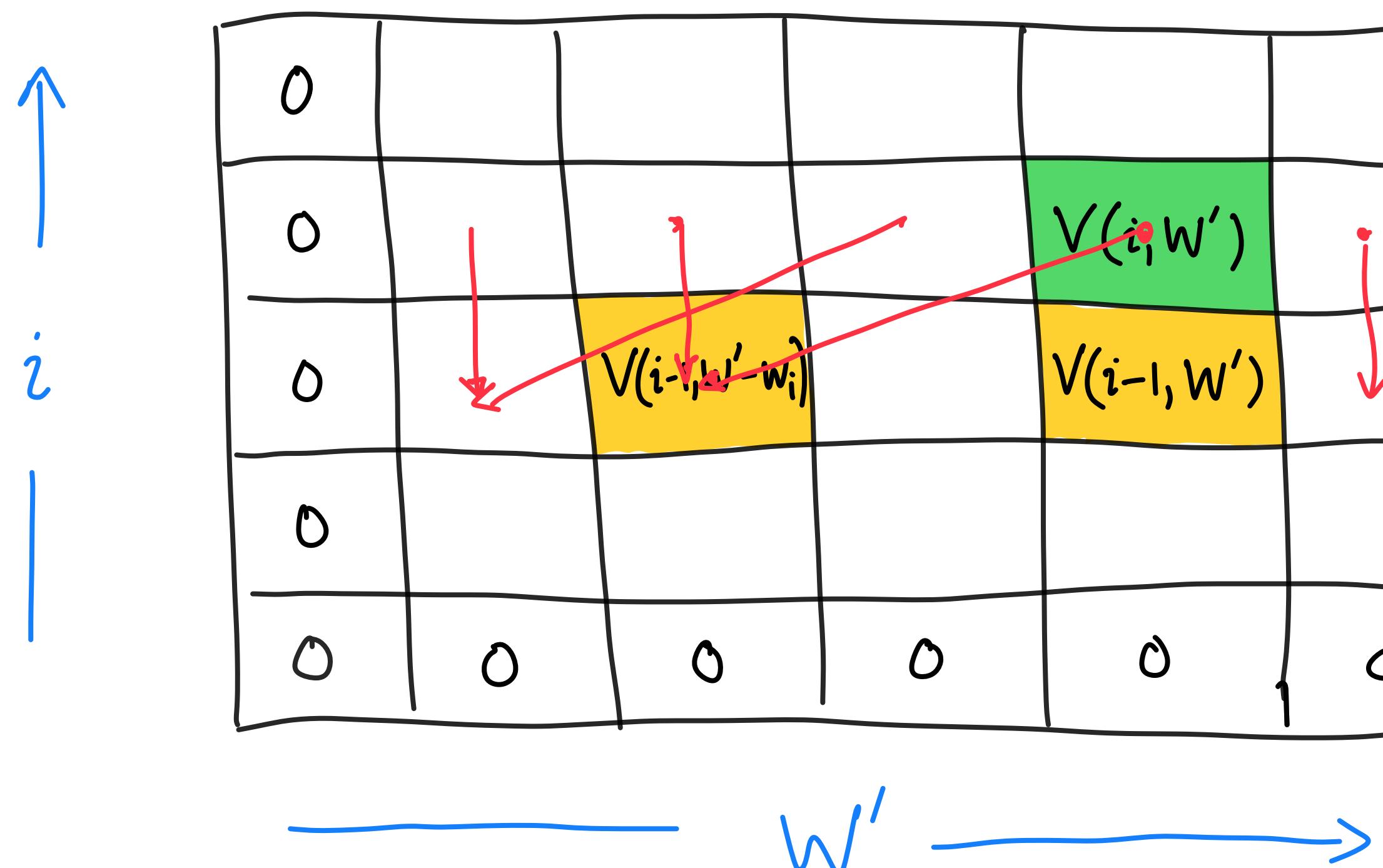
0	0	0	0	0	1
0	0	0	0	0	1
0	0	$V(i-1, W' - w_i)$	$V(i-1, W')$		
0	0				
0	0				
0	0				

i

W'

Memoization for Knapsack

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



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

Each edge \downarrow or \uparrow 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

- **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 $\text{Inc}(i, W') = \text{false}$
 - Else, set $V(i, W') \leftarrow V(i - 1, W' - w_i) + v_i$ and set $\text{Inc}(i, W') = \text{true}$

short for "Include"

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 $(v_1, \dots, v_n, w_1, \dots, w_n, W)$, 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 $(v_1, \dots, v_n, w_1, \dots, w_n, W)$, let OPT be the optimal value of subset of items weighing $\leq W$:

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

- An alg. \mathcal{A} is an **ϵ -approximation alg.** if \mathcal{A} always outputs a subset \tilde{S} such that (a) $\text{weight}(\tilde{S}) \leq W$ and (b) $\text{value}(\tilde{S}) \geq (1 - \epsilon) \cdot \text{OPT}$.
- 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 .