DYNAMIC PROGRAMMING

1) Longest path in a DAG

Input: DAG $G=(V,E)$

Goal: Find length L of longest path in G

Step1 (Define Subproblems):

$L(v)$ = length of longeth path ending in v

Step2: (Recursion, determine dependencies)

$$L(v) = \begin{cases} \max\limits_{(u,v)\in G} L(u)+1 \\ 0 \quad \text{if no incom. edge} \end{cases}$$

$L(v)$ depends on all incoming edges $uv \in E$

Step 3: Pick order to compute

- topological sorted order of G

Pseudo Code:

```
• L = 0
• For all i, L[i] = 0
• Topologically sort G
• For i=1, ... n (in top. search order)
      L[i] = max   L[j]
             ji∈E
      If L[i] > L:  L = L[i]
```

```
L[i] = 0
For all ji ∈ E
   if L[j] + 1 > L[i]
      L[i] = L[j] + 1
```

Run Time: $O(|V|+|E|)$ (even if V is given in topological order)

2) Longest Increasing Subsequence

1 5 7 3 6 9 10

1 7 9 10 length 4

1 5 7 9 10 length 5

1 3 6 9 10 length 5

Reduce to previous problem:

sequence $a_1, a_2, \cdots, a_n$

Make it into DAG

$ij \in E$ if $i < j$ and $a_i < a_j$

Running time $O(n^2)$

3) Edit Distance

Input: two strings $x[1:n]$, $y[1:m]$

Goal: Calculate the $\boxed{\text{edit distance}}$

$E[x,y]$ = minimum # of keystrokes to edit x into y

[insert a char, delete a char, substitute a char]

Example:

$$\boxed{SUNNY} \xrightarrow[\text{delete } U]{} SNNY \xrightarrow[\text{subst. } N \to O]{} SNOY \xrightarrow[\text{insert } W]{} \boxed{SNOWY}$$

Dynamic Programming Steps

1) Define Subproblem

2) Write Recursion

3) Find Calculation order

1) Subproblems

Edit distance between prefixes

$E(x[1:i], y[1:j])$ $E(S,S)$, $E(SUNNY, SNO)$,...

2) Dependencies

$$E(x[1:i], y[1:j]) =$$

$$= \min \begin{cases} E(x[1:i], y[1:j-1]) + 1 & \text{Insert} \\ E(x[1:i-1], y[1:j]) + 1 & \text{Delete} \\ E(x[1:i-1], y[1:j-1]) + \underbrace{\text{dist}(x_i, x_j)}_{\mathbb{1}_{x_i \neq y_j}} & \text{Keep Subst.} \end{cases}$$

Ex: $E[SUNNY, SNOWY] =$

$$= \min \begin{cases} E[SUNNY, SNOW] + 1 & \text{Insert: } SNOW \to SNOWY \\ E[SUNN, SNOWY] + 1 & \text{Delete: } SUNNY \to SUNN \\ E[SUNN, SNOW] & \text{Keep } Y \end{cases}$$

Proof using Tile Representation:

$\begin{bmatrix} S \\ \downarrow \\ - \end{bmatrix}$ delete D $\begin{bmatrix} - \\ \downarrow \\ O \end{bmatrix}$ insert I $\begin{bmatrix} N \\ \downarrow \\ N \end{bmatrix}$ keep K $\begin{bmatrix} N \\ \downarrow \\ O \end{bmatrix}$ substitute S

Example:

SUNNY $\xrightarrow[\text{delete } U]{}$ S_NNY $\xrightarrow[\text{subst. } N \to O]{}$ SNOY $\xrightarrow[\text{Insert } W]{}$ SNOWY

$$\begin{array}{|c|c|c|c|c|}\hline S & U & N & N & Y \\ S & U & N & N & Y \\ \hline\end{array} \to \begin{array}{|c|c|c|c|c|}\hline S & U & N & N & Y \\ S & - & N & N & Y \\ \hline\end{array} \to \begin{array}{|c|c|c|c|c|}\hline S & U & N & N & Y \\ S & - & N & O & Y \\ \hline\end{array} \to \begin{array}{|c|c|c|c|c|c|}\hline S & U & N & N & - & Y \\ S & - & N & O & W & Y \\ \hline\end{array}$$

D S

$\underbrace{\text{D S I}}_{\text{cost } 3}$

Other Edit Sequences (cost 9)

SUNNY → Y → S → SNOWY

$$\begin{array}{|cccccccccc|}\hline S & U & N & N & Y & - & - & - & - & \\ - & - & - & - & - & S & N & O & W & Y \\ \hline\end{array}$$

SUNNY → SNOWSUNNY → SNOWYUNNY → SNOWY

$$\begin{array}{|cccccccccc|}\hline - & - & - & - & - & S & U & N & N & Y \\ S & N & O & W & Y & - & - & - & - & - \\ \hline\end{array}$$

Derivation of Recurrence:

Optimal edit $x[1:i] \to y[1:j]$

$$\begin{array}{|c|c|}\hline ?? & ? \\ & ? \\ \hline\end{array}$$

Possible last tile $\begin{array}{|c|}\hline - \\ y_j \\ \hline\end{array}$, $\begin{array}{|c|}\hline x_i \\ - \\ \hline\end{array}$, $\begin{array}{|c|}\hline x_i \\ y_j \\ \hline\end{array}$

What tiles are compatible for ???

$\begin{array}{|ccc|}\hline x_1 & \cdots & x_i \\ y_1 & \cdots & y_{j-1} \\ \hline\end{array}$ $\begin{array}{|c|}\hline - \\ y_j \\ \hline\end{array}$ insert

$\begin{array}{|ccc|}\hline x_1 & \cdots & x_{i-1} \\ y_1 & \cdots & y_j \\ \hline\end{array}$ $\begin{array}{|c|}\hline x_i \\ - \\ \hline\end{array}$ delete $\Rightarrow$

$\begin{array}{|ccc|}\hline x_1 & \cdots & x_{j-1} \\ y_1 & \cdots & y_{j-1} \\ \hline\end{array}$ $\begin{array}{|c|}\hline x_i \\ y_j \\ \hline\end{array}$ keep subst.

$$E(x[1:i], y[1:j]) = \min \begin{cases} E(x[1:i], y[1:j-1]) + 1 \\ E(x[1:i-1], y[1:j]) + 1 \\ E(x[1:i-1], y[1:j-1]) + \mathbb{1}_{x_i \neq y_j} \end{cases}$$

End Review

Step 3: Pick an order to calculate

Matrix of cost $E[i,j] = E(X[1:i], Y[1:j])$

	$\emptyset$	S	N	O	W	Y
$\emptyset$	0	1	2	3	4	5
S	1					
U	2					
N	3					
N	4					
Y	5					

Dependencies

$E[i,j]$ depends on

$E[i,j-1]$, $E[i-1,j]$, $E[i-1,j-1]$

Base Case: $E[0:i] = E[i:0] = i$

Algorithm:

> **Init:** For $i = 1, \dots n$ $\quad E[i,0] = i$
> For $j = 1, \dots m$ $\quad E[0,j] = j$
>
> For $i = 1, \dots, n$
> For $j = 1, \dots m$
> $$E[i,j] = \min \begin{cases} E[i,j-1] + 1 \\ E[i-1,j] + 1 \\ E[i-1,j-1] + \mathrm{Diff}(x_i, y_j) \end{cases}$$
>
> return $E[n,m]$

Running Time $O(nm)$

Space $O(nm)$

To find sequence: store which edit

$S[i,j] \in \{ \searrow^{K}, \searrow^{S}, \downarrow^{D}, \xrightarrow[I]{} \}$

	$\emptyset$	S	N	O	W	Y
$\emptyset$	0	1	2	3	4	5
S	1	0	1	2	3	4
U	2	1	1	2	3	4
N	3	2	1	2	3	4
N	4	3	2	2	3	4
Y	5	4	3	3	4	3

SUNNY
SNOWY

SUNN_Y
S_NOWY

SUN_NY
S_NOWY

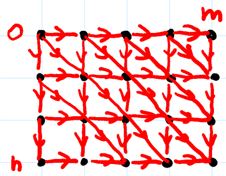
Remarks:

- For any dynamic program, there is an (implicit) DAG. What is it?

  The dependency graph on the set of subproblems.

  Where is the DAG for the EDIT distance?

  $V = \{(i,j) : \begin{matrix} 0 \le i \le n \\ 0 \le j \le m \end{matrix}\}$



- Strategies for Finding subproblems:

Input	Subproblem	
sequence $a[1:n]$	$Cost(a[1:i])$	$Cost(a[i:j])$
weighted graph $G=(V,E,w)$	$Cost(v) \quad v \in V$	
two sequences $a[1:n]$, $b[1:m]$	$Cost(a[1:i], b[1:j])$	
Tree	$Cost(Subtree)$	

3) Knapsack

Input: Total Weight W

List of n items with weights $w_1, \cdots, w_n$ (integers)

values $v_1, \cdots, v_n$ (")

Goal: Find set of item with

max total value, with total weight $\leq W$

2 Variants: with repitition / without repitition

Example:

$W = 10$

Item	Weight	Value
1	6	\$ 30
2	3	\$ 14
3	4	\$ 16
4	2	\$ 9

Without replac. Items 1, 3 $\rightarrow$ \$ 36

With replac. Items 1, 4, 4 $\rightarrow$ \$ 38

Knapsack with Replacement

1) Subproblem

Look at optimum

Items $i_1, \cdots i_k$

Value: $\underbrace{v_{i_1} + \cdots + v_{i_k}}_{\text{optimum}} = \underbrace{(v_{i_1} + \cdots \; v_{i_{k-1}})}_{\text{optimum}} + v_{i_k}$

Constraint: total weight $\leq W - v_{i_k}$

Subproblems:

$K(C) =$ max total value with total weight $\leq C$

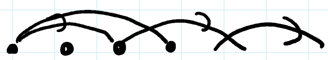
$C = 0, 1, \cdots, W$

2) Recurance:

$$K(C) = \max_{i:\, v_i \leq C} \left(v_i + K(C - w_i) \right)$$

3) Order

$C=0 : K(0)=0$



$C = 0 \quad 1 \quad 2 \cdots$

Algorithm:

Input: $W, v[1:n], w[1:n]$

$K(0) = 0$

For $C = 1$ to W

$$K(C) = \max_{i:\, w_i \leq C} v_i + K(C - w_i)$$

Output: $K(W)$

Runtime

$O(nW)$

exponential in

$\log W$