Algorithms so Far

Divide & Conquer:

Integer Multiplication $O(n^{\log_2 3}) = O(n^{1.58})$
Matrix Multiplication $O(n^{\log_2 7}) = O(n^{2.81})$
Merge Sort $O(n \log n)$
FFT $O(n \log n)$

Simple Graph Algorithms

DFS, connected components
topological search, SCC $O(n+m)$ $n=|V|, m=|E|$

Single Source shortest Paths

DFS $O(n+m)$
Dijkstra $O((n+m)\log n)$
Bellman-Ford $O(nm)$
DAG-SSSP $O(n+m)$

Greedy

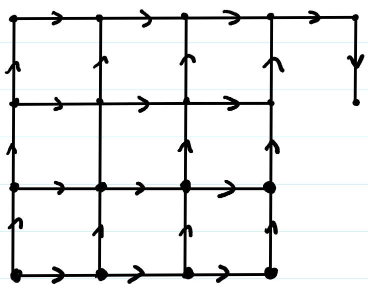
Scheduling $O(n)$
Huffman Coding $O(n \log n)$
Kruskal & Prim (MST) $O((n+m)\log n)$
Horn Formulae $O(|F|)$
Greedy set Cover (later, only finds approx. min)

- Important basic algorithms, fast
- Not a very general tool

Dynamic Programming

- A versatile, powerful algorithm design principle

1) Longest path in a DAG



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

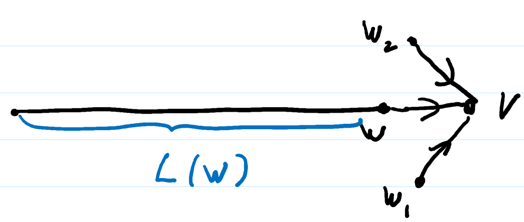
Goal: Find length of longest path in G

Recursive Algorithm

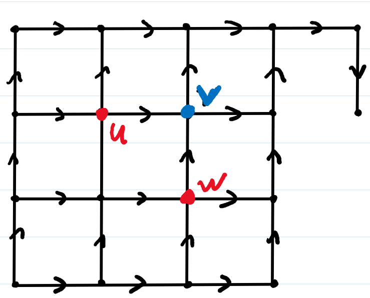
L = length of longest path in G:

$$\max_{v \in V} L(v)$$

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



$$L(v) = \max_{(w,v) \in E} L(w) + 1$$



$$L(v) = \max\{L(u)+1,\ L(w)+1\}$$

Recursive Relation

$$L(v) = \max_{(u,v)\in G} L(u) + 1$$

0 if no incom. edge

Algorithm

$\underline{L(v)}$

"Returns length of longest path end. in v"

If no incom. edge $L(v) = 0$

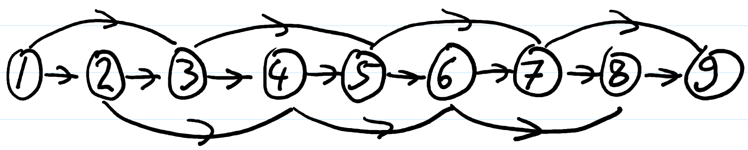
Else: $L(v) = \max_{wv \in E} L(w) + 1$

Implementation:

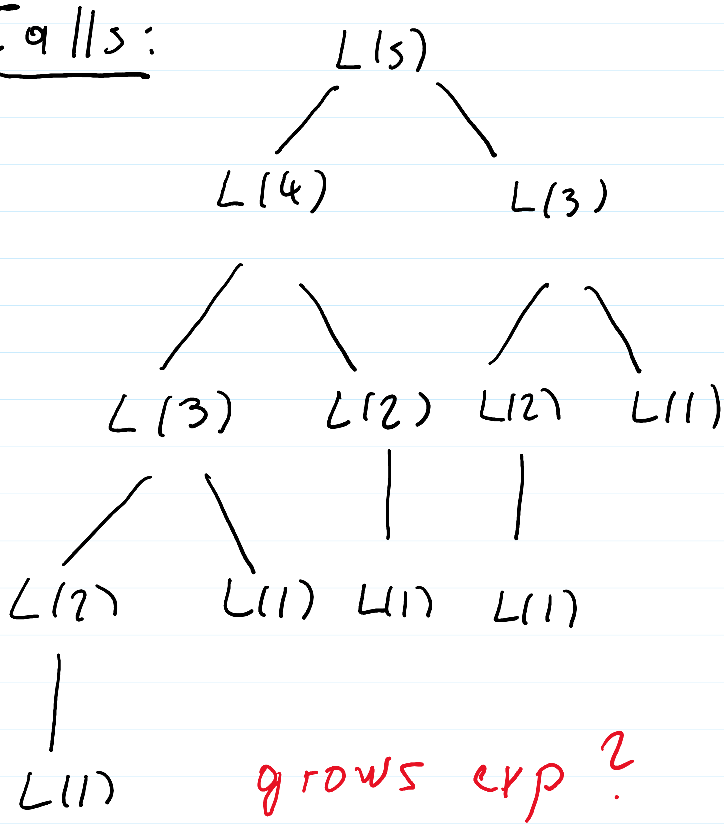
```
current = 0
For all wv ∈ E
    if L(w)+1 > current
        current = L(w)+1
Return current
```

Does it terminate: Yes. Each iterative call explores edges pointing backwards in the DAG, so eventully will end at the sources.

How long does it take?



Calls:



grows exp ?

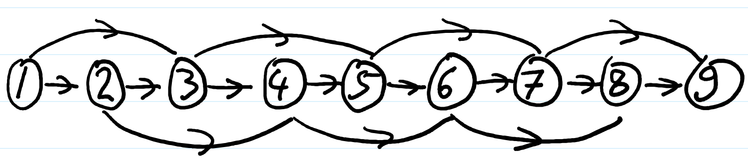
$$T(i) = T(i-1) + T(i-2)$$

$\rightarrow$ Fibonancy #s

Solution: Recursion with memorization

$\rightarrow$ Remember $L(i)$ if calculated once

Non recursive Implementation



Subproblems:

$L(1), \cdots, L(9)$

Dependence:

$L(i)$ depends on $L(j)$ $j<i$

Compute in order

$L(1), L(2), \cdots, L(9)$

General Graph:

Subproblem: $L(v)$ for all $v \in V$

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

Order to compute:

topological sorted order of G

Pseudo Code:

- Topologically sort G

Let i be the ith vertex in topol. sort order

- For all i, $L[i] = 0$
- For $i = 1, \cdots n$ $\quad L[i] = \max\limits_{ji \in E} L[j]$

```
current = 0
For all wv ∈ E
   if L(w)+1 > current
      current = L(w)+1
Return current
```

Run time:

$O(|V|+|E|)$ (both for topol. sort, and since algorith calls all edges)

2) Longest Increasing Subsequence

1 5 7 3 6 9 10

1 7 9 10

1 5 7 9 10

1 3 6 9 10

Reduce to previous problem:

sequence $a_1, a_2, \cdots, a_n$

Make it into DAG

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

running time $O(n^2)$ (we need to check $a_i < a_j \; \forall \binom{n}{2}$ pairs $i<j$

3) Edit Distance

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

Task: Find the minimum # of keystrokes to edit x into y

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

$x =$ SUNNY

$y =$ SNOWY

CAT $\rightarrow$ HAT cost = 1

ABABAB BABABA cost = 2 (delete first A, insert A at end)

Why is this interesting?

- spellchecker
- DNA

How can we represent a sequence of edits?

- Complicated deleting, inserting, shifting things
- Better visulization needed

S U N N Y

↓

S N O W Y

where do these letters come from?

S	—	N
↓ keep	↓ insert	↓ substitute
S	S	O

where do these letters go to?

S	U	N
↓ keep	↓ delete	↓ substitute
S	—	W

S	U	N	N	Y	—	—	—	—	—
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
—	—	—	—	—	S	N	O	W	Y

Tiles:

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

S	U	N	N	_	Y
K	D	K	S	I	K
S	_	N	O	W	Y

Can do this in arb. order

Dynamic Programming Strategy

3 Steps:

1) Define subproblem
2) Write down recurrence
3) Determine order of calculations

Step 1:

$E[i,j]$ = Edit dist. between $x[1,\cdots,i]$ and $y[1,\cdots,j]$ $\quad$ | prefix

x = SUNNY

y = SNOWY

e.g. $E[S, S]$ $\quad$ $E[SU, SNOW]$

...

$(m+1)(n+1)$ subproblems (include empty string)

Step 2: Recurrence relation

Write down tiles

• Look at optimal solution

Last tile

Cost last step

$$\begin{array}{c} x \\ y \end{array} \begin{bmatrix} - & S & U & N & N & - & \cdots & \\ S & - & N & O & - & W & \cdots & \end{bmatrix} \quad \boxed{}$$

$\begin{bmatrix} - & S & U & N & N & - \\ S & - & N & O & - & W \end{bmatrix} + \boxed{\begin{matrix} Y \\ Y \end{matrix}}$	keep	Case 1	0
$\begin{bmatrix} S & U & N & N & Y & - & - & - & - \\ - & - & - & - & - & S & N & O & W \end{bmatrix} + \boxed{\begin{matrix} - \\ Y \end{matrix}}$	insert	Case 2	1
$\begin{bmatrix} - & - & - & - & - & S & U & N & N \\ S & N & O & W & Y & - & - & - & - \end{bmatrix} \quad \boxed{\begin{matrix} Y \\ - \end{matrix}}$	delete	Case 3	1

These are the possibilities

Actual answer → minimum

⇒

$$E[SUNNY, SNOWY] = \min \begin{cases} E[SUNN, SNOW] \\ E[SUNNY, SNOW] + 1 \\ E[SUNN, SNOWY] + 1 \end{cases}$$

In general

$$E[i,j] = \min \begin{cases} E[i,j-1] + 1 & \text{Insert} \\ E[i-1,j] + 1 & \text{Delete} \\ E[i-1,j-1] + \underbrace{\mathrm{Diff}(x_i, y_j)}_{\mathbb{1}_{x_i \neq y_j}} & \end{cases}$$

$x_i = y_j$ → Keep

$x_i \neq y_j$ → Replace

<u>Step 3: Pick an order</u>

Next lecture!