Dynamic Programming

Faculty Development Programme
Design and Analysis of Algorithms
SSN College of Engg, 13 Dec 2014

Madhavan Mukund
Chennai Mathematical Institute,
http://www.cmi.ac.in/~madhavan

Outline

* Introduction to dynamic programming
  * Recursion with memoization (memory tables)
  * Filling the table iteratively
* Examples
  * Floyd/Warshall algorithms
  * Optimal search trees
  * Others

Inductive definitions

- Factorial
  - $f(n) = n \times f(n-1)$
  - $f(0) = 1$
- Insertion sort
  - isort([ ]) = [ ]
  - isort($[x_1,x_2,..,x_n]$) = insert(x_1,isort($[x_2,\ldots,x_n]$))

... Recursive programs

```
int factorial(n):
  if (n <= 0)
    return(1)
  else
    return(n*factorial(n-1))
```

Sub problems

- factorial(n-1) is a subproblem of factorial(n)
  - So is factorial(n-2), factorial(n-2), …, factorial(0)
- isort([x_2,…,x_n]) is a subproblem of isort([x_1,x_2,…,x_n])
  - So is isort([x_i,…,x_j]) for any $1 \leq i \leq j \leq n$
- Solution of f(y) can be derived by combining solutions to subproblems

Evaluating subproblems

- fib(0) = 0
- fib(1) = 1
- fib(n) = fib(n-1) + fib(n-2)
- Compute fib(7)?

0 1 2 3 4 5 6 7

0 1

1

2

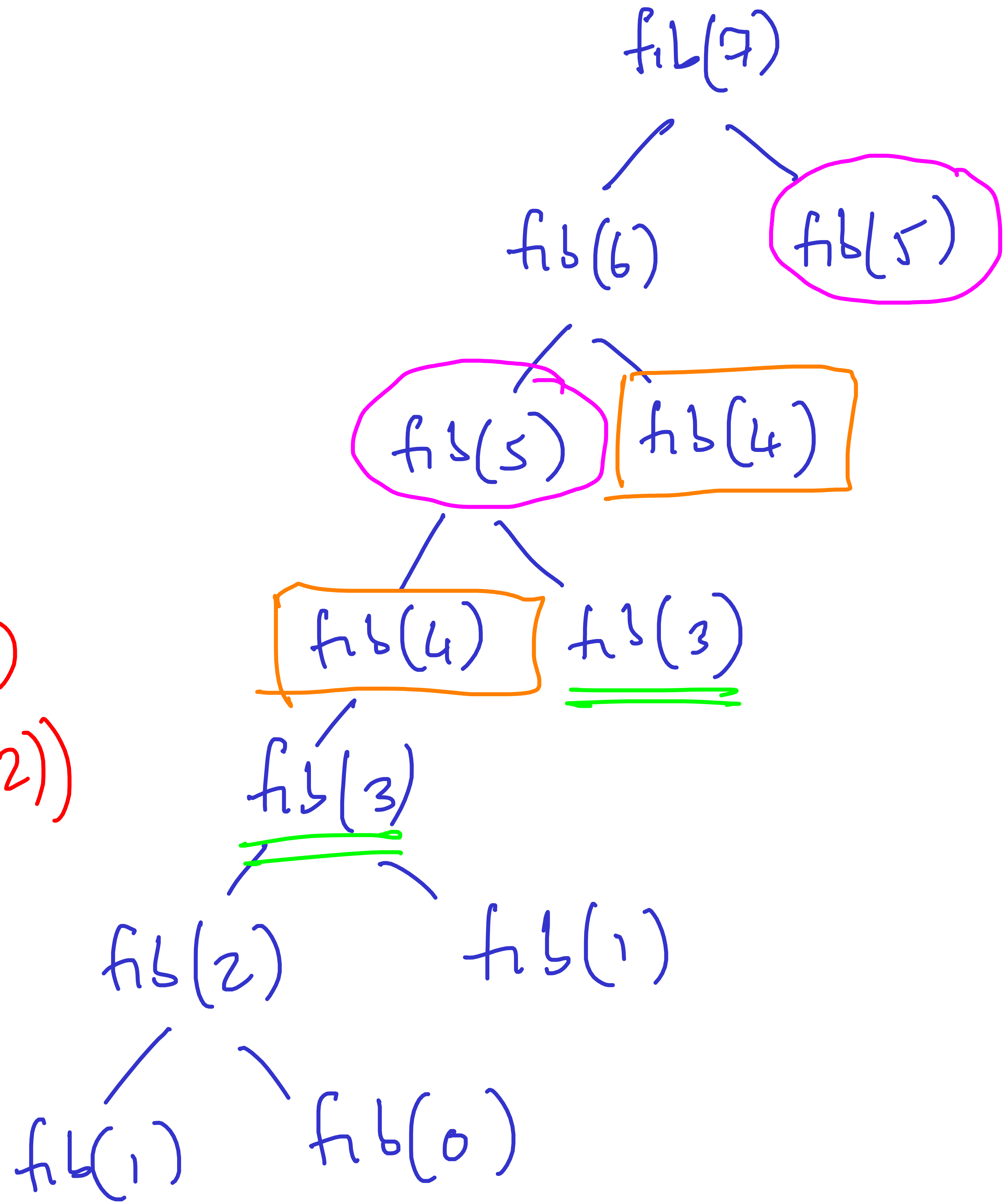
3

5

8

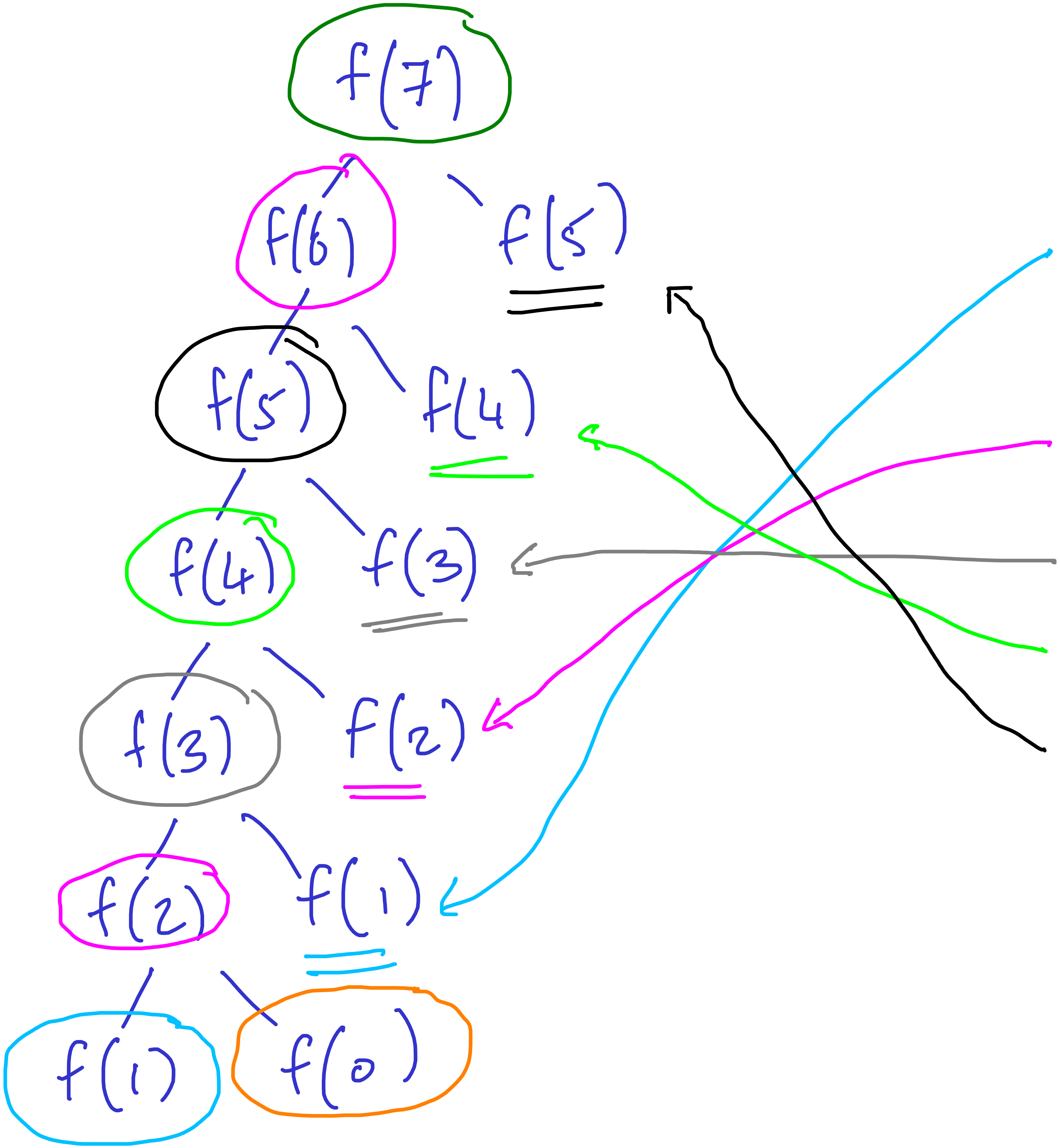
13

```
def fib(n):
    if n==0
        return(0)
    if n==1
        return(1)
    return(fib(n-1)
            + fib(n-2))
```



Never re-evaluate a subproblem

- Build a table of values already computed
  - Memory table
- Memoization
  - Remind yourself that this value has already been seen before



n	f(n)
1	1
0	0
2	1
3	2
4	3
5	5
6	8
7	13

Memoized fibonacci

```
def fib(n):
  if fibtable[n]
    return(fibtable[n])
  if n == 0 or n == 1
    newvalue = n
  else
    newvalue = fib(n-1) + fib(n-2)
  fibtable[n] = newvalue
  return(newvalue)
```

In general

```
def f(x,y,z):
  if ftable[x][y][z]
    return(ftable[x][y][z])
  newvalue = expression in terms of
             subproblems
  ftable[x][y][z] = newvalue
  return(newvalue)
```

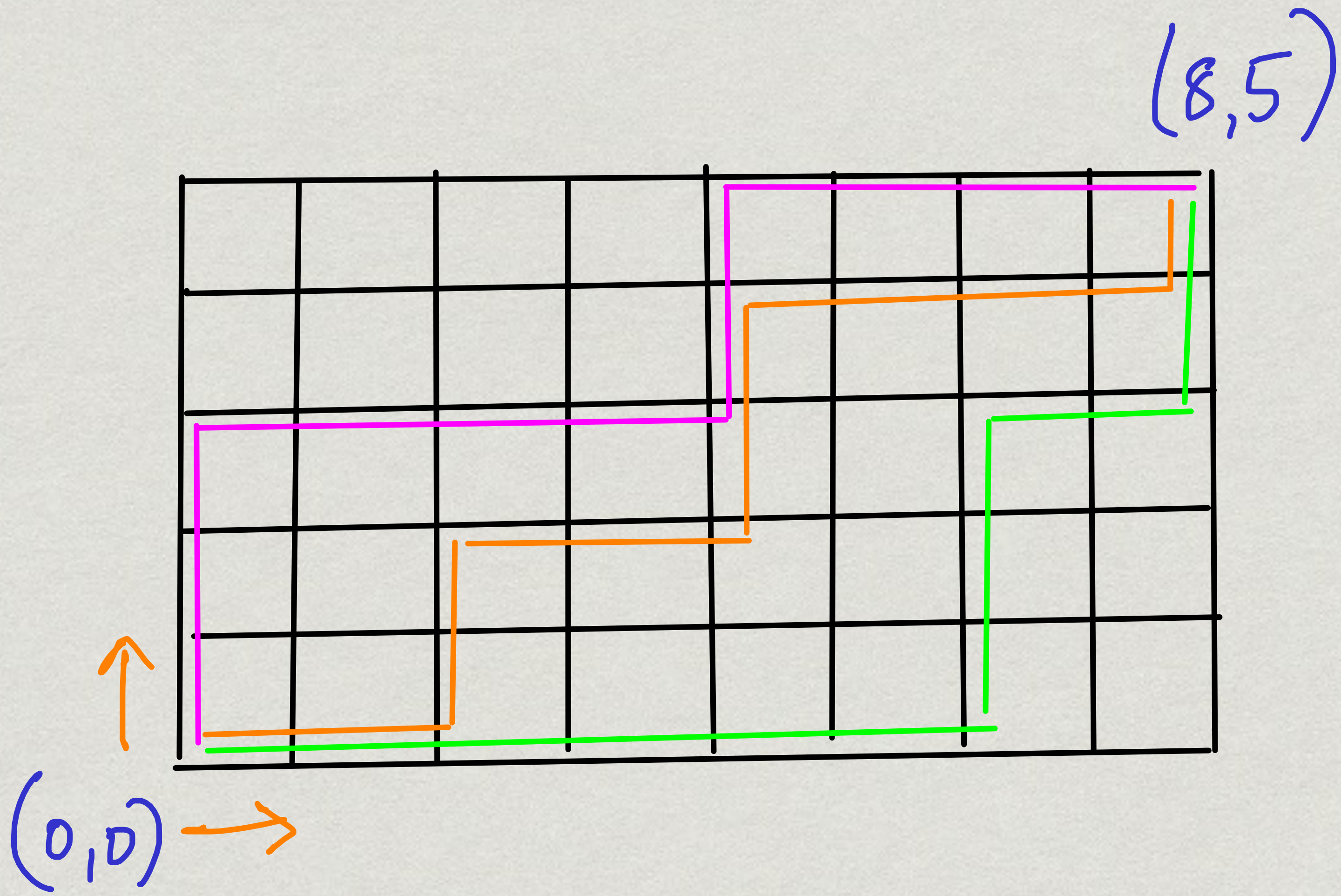
Dynamic programming

- Anticipate what the memory table looks like
- Fill it up iteratively
  - For each new value, subproblems already filled in table

n	0	1	2	3	4	5	6	7
fib(n)								

Grid Paths

- Roads arranged in a rectangular grid
- Can only go up or right
- How many different routes from (0,0) to (m,n)?



Analytic Solution

Every path takes exactly 8 steps $\rightarrow$, 5 steps $\uparrow$

All paths have exactly 13 steps

$$\underset{1}{\rightarrow}\ \underset{2}{\rightarrow}\ \underset{3}{\rightarrow}\ \underset{4}{\uparrow}\ \underset{5}{\rightarrow}\ \underset{6}{\rightarrow}\ \underset{7}{\uparrow}\ \underset{8}{\rightarrow}\ \underset{9}{\uparrow}\ \underset{10}{\rightarrow}\ \underset{11}{\uparrow}\ \underset{12}{\uparrow}\ \underset{13}{\rightarrow}$$

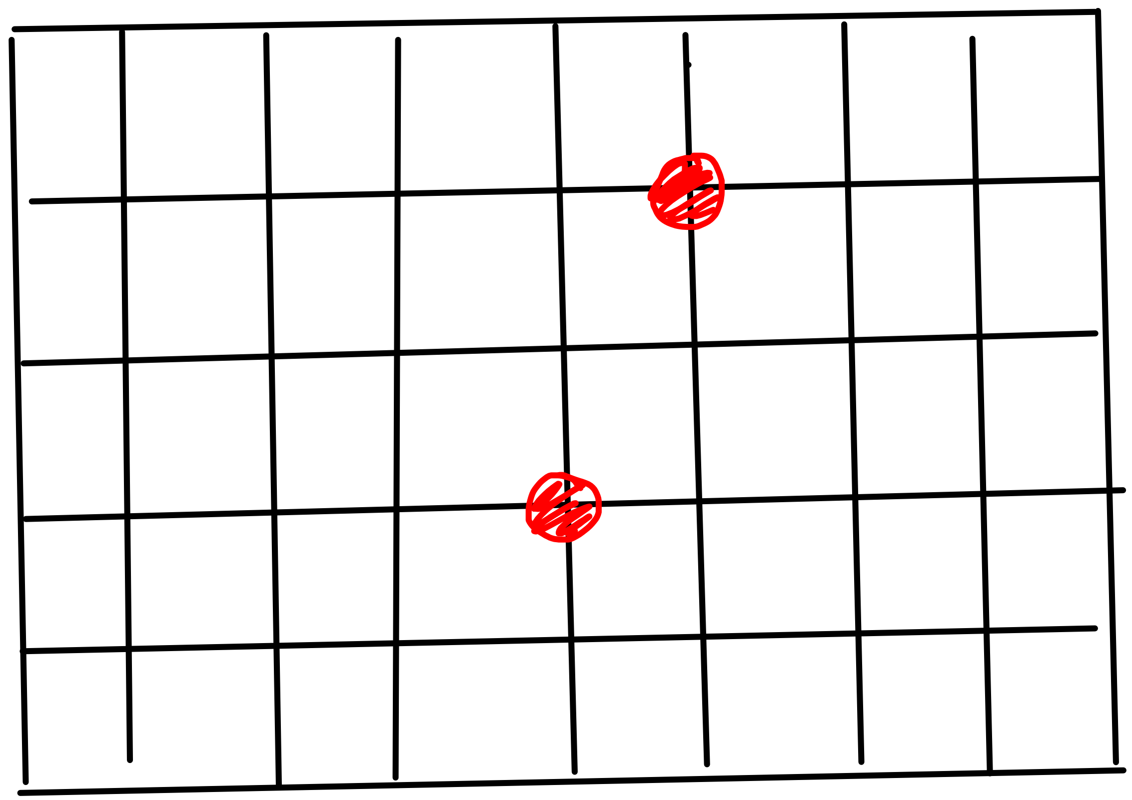
Identify 5 $\uparrow$ positions, rest must be $\rightarrow$

8 $\rightarrow$ positions, rest must be $\uparrow$

$$\binom{13}{8},\ {}^{13}C_8 \qquad \binom{n}{k} = \frac{n!}{k!(n-k)!} = \binom{n}{n-k} \qquad \binom{13}{8} = \binom{13}{5} = \frac{13!}{5!\,8!}$$

In general, $(0,0)$ to $(m,n) \rightsquigarrow \binom{m+n}{m} = \binom{m+n}{n}$

What if some intersections are blocked?



(4,2), (5,4) blocked

Exclude paths passing through these

Paths through (4,2) =

Paths (0,0) to (4,2)

× Paths (4,2) to (8,5)

Paths through (4,2) and (5,4) excluded twice!

Add these back

Complicated "inclusion-exclusion" counting

Analytic solution does not scale well

Inductive formulation

$$\begin{tikzcd} (i-1,j) \arrow[r] & (i,j) \\ & (i,j-1) \arrow[u] \end{tikzcd}$$

- Paths to (i,j) must go through left or bottom neighbour
- No path can go via both neighbours
- Each path to neighbour has unique extension to (i,j)

$\text{Paths}(i,j)$ = no. of paths from $(0,0)$ to (i,j)

$$\text{Paths}(i,j) = \text{Paths}(i-1,j) + \text{Paths}(i,j-1)$$

$$= 0 \text{ if } (i,j) \text{ is a blocked intersection}$$

Base Case:

$$\text{Paths}(0,0) = 1$$

$$\text{Paths}(0,j) = \text{Paths}(0,j-1)$$

$$\text{Paths}(i,0) = \text{Paths}(i-1,0)$$

What does our table look like?

To compute Paths(m,n), need Paths(i,j), $0 \le i \le n$, $0 \le j \le m$

Base Cases

5	1	6	21	56	111	111	150	249	438
4	1	5	15	35	55	0	39	99	189
3	1	4	10	20	20	26	39	60	90
2	1	3	6	10	0	6	13	21	30
1	1	2	3	4	5	6	7	8	9
0	1	1	1	1	1	1	1	1	1
	0	1	2	3	4	5	6	7	8

Blocked

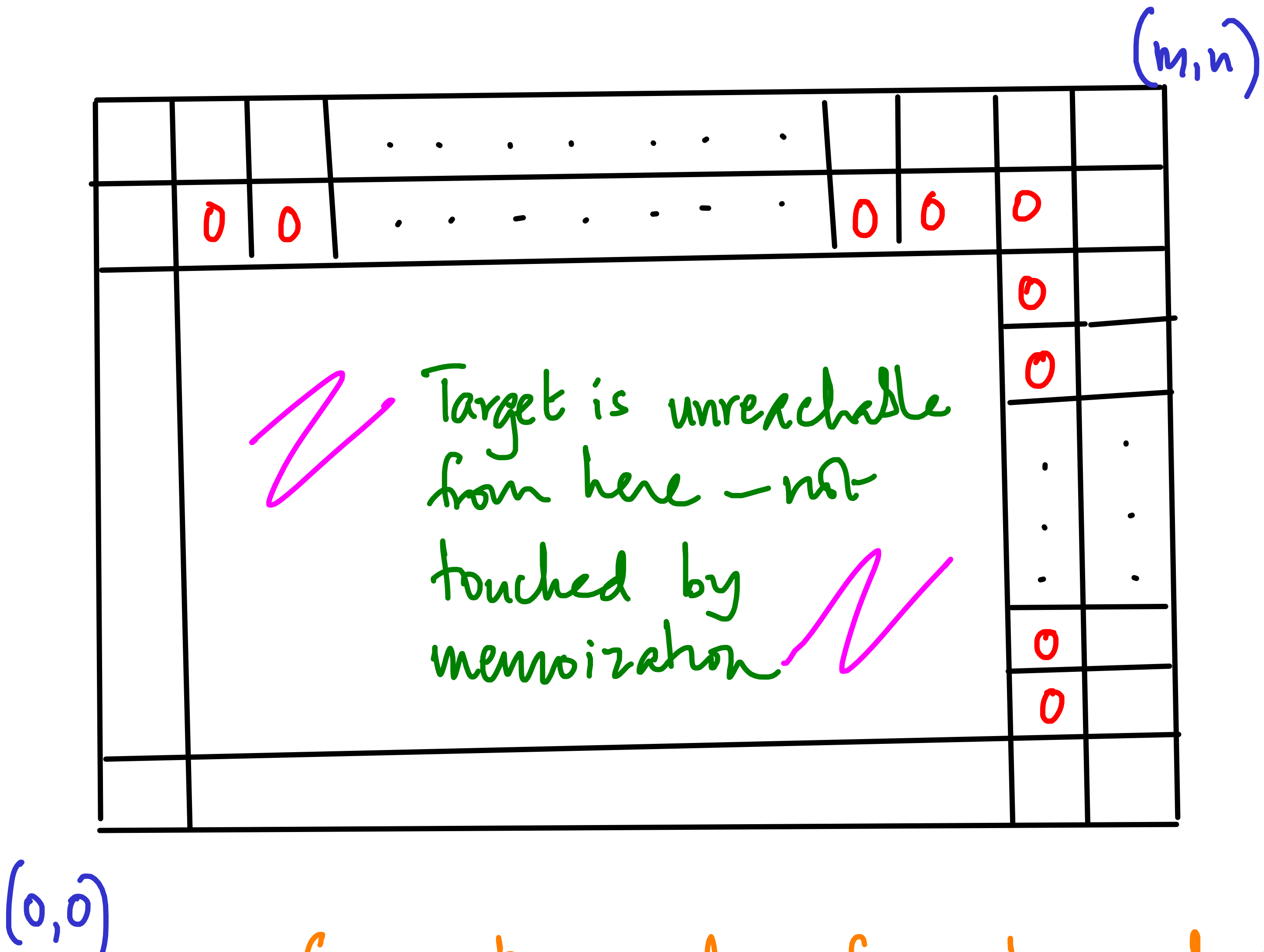
Base Cases

Start here

Table above filled row-wise

Can also fill column-wise or in diagonals

Memoization vs Dynamic Programming



Memoization fills table "on demand" - unreachable subproblems are not evaluated

D.P. blindly evaluates all subproblems

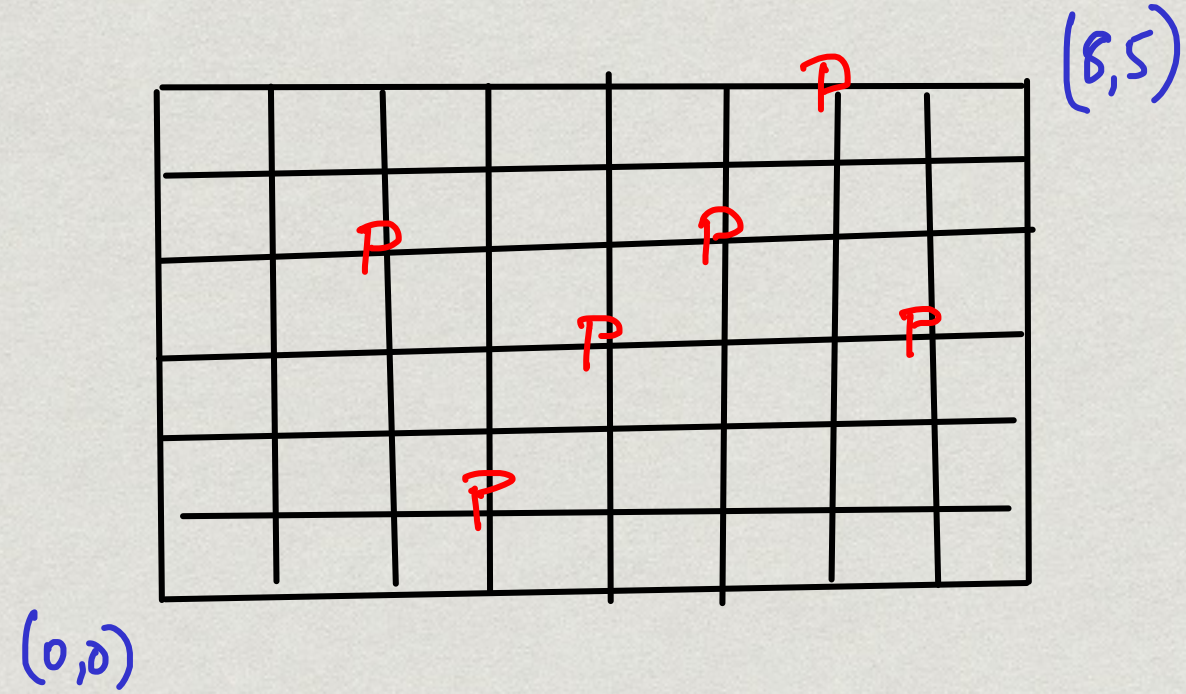
Given the grid configuration above

- Memoization explores $O(m+n)$ subproblems along boundary
- D.P. evaluates all $O(mn)$ subproblems

Nevertheless "wasteful" D.P. usually wins because of saving in iteration vs recursion

Picking prizes along a grid

- Similar layout of roads, but objective is different
- Some intersections have prizes
- Find the maximum number of prizes you can pick up



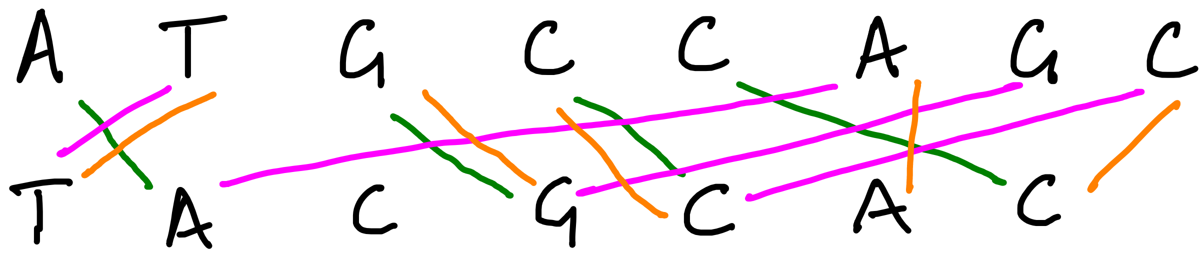
$P(i,j)$ = max prizes from $(0,0)$ to (i,j)

$= \max(P(i-1,j), P(i,j-1))$, if no prize at (i,j)

$= 1 + \max(P(i-1,j), P(i,j-1))$, if prize at (i,j)

Longest common subsequence

- Two sequences, drawn from the same set of symbols
- What is the maximum overlap?
- Applications
  - Comparing DNA sequences
  - Comparing versions of text files (Unix `diff`)



Subsequence – omit some letters

Find length of longest identical subsequence

AGCC TAGC TGCAC

4 4 5

$s \qquad s_1 \; s_2 \; s_3 \; \ldots \; s_n$

$t \qquad t_1 \; t_2 \; t_3 \; \ldots \; t_m$

$lcs(s_1 .. s_n, t_1 .. t_m)$

Subproblems?

In general, $lcs(s_i .. s_j, t_k .. t_l)$

Our solution will use $lcs(s_2 .. s_n, t_2 .. t_m)$

$lcs(s_1 .. s_n, t_2 .. t_m)$

$lcs(s_2 .. s_n, t_1 .. t_m)$

$s = \boxed{s_1} \; s_2 \; s_3 \; \ldots \; s_n$

$t = \boxed{t_1} \; t_2 \; t_3 \; \ldots \; t_m$

Why is this always a good choice?

$s_1 == t_1$? Match (s_1, t_1) and inductively solve rest

$$lcs(s_1 .. s_n, t_1 .. t_m) = 1 + lcs(s_2 .. s_n, t_2 .. t_m)$$

$s_1 \neq t_1$? Drop either s_1 or t_1 (not both! $\begin{matrix} \boxed{A} & T & C \\ \boxed{G} & A\ T & C \end{matrix}$)

$$lcs(s_1 .. s_n, t_1 .. t_m) =$$

$$\max\big(lcs(s_1 .. s_n, t_2 .. t_m), \quad \text{- drop } t_1$$

$$lcs(s_2 .. s_n, t_1 .. t_m) \big) \quad \text{- drop } s_1$$

In general

$lcs(i,j)$ is solution for $(s_i s_{i+1} \cdots s_n, t_j t_{j+1} \cdots t_m)$

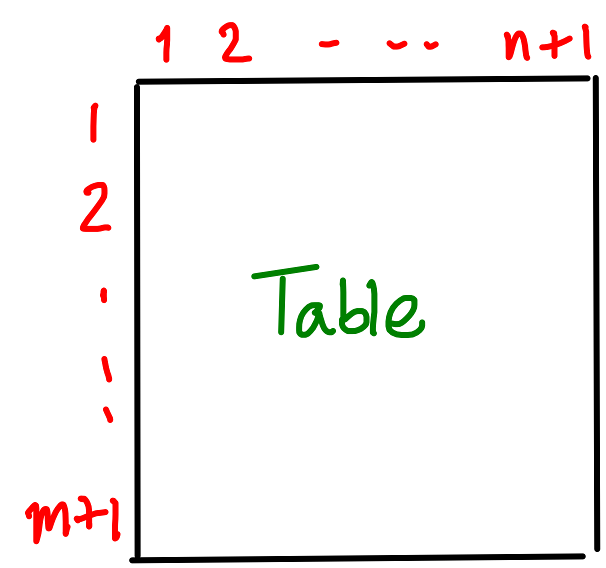
$$lcs(i,j) = \begin{cases} 1 + lcs(i+1, j+1), & \text{if } s_i = t_j \\ \max(lcs(i, j+1), lcs(i+1, j)), & \text{if } s_i \neq t \end{cases}$$

Convenient to let i, j go upto $n+1, m+1$

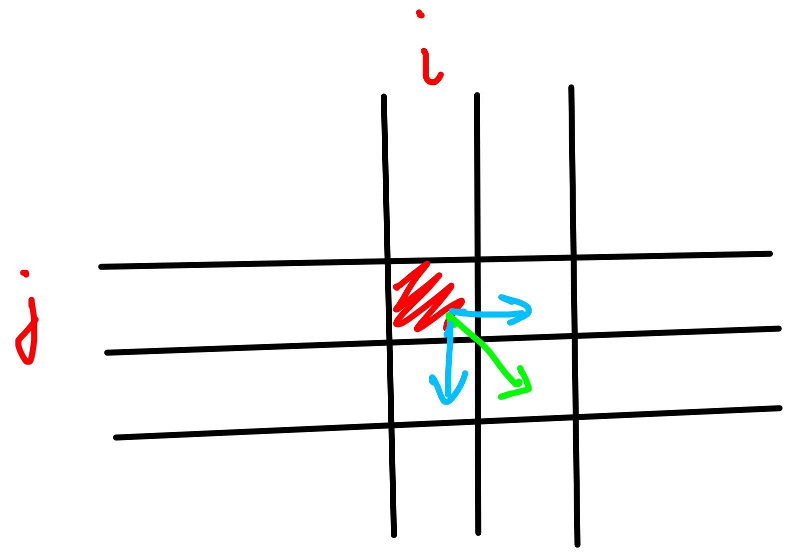
$lcs(n+1, j) = 0$ for all j

$lcs(i, m+1) = 0$ for all i

Comparing empty sequence on one side



What does table entry (i,j) depend on?

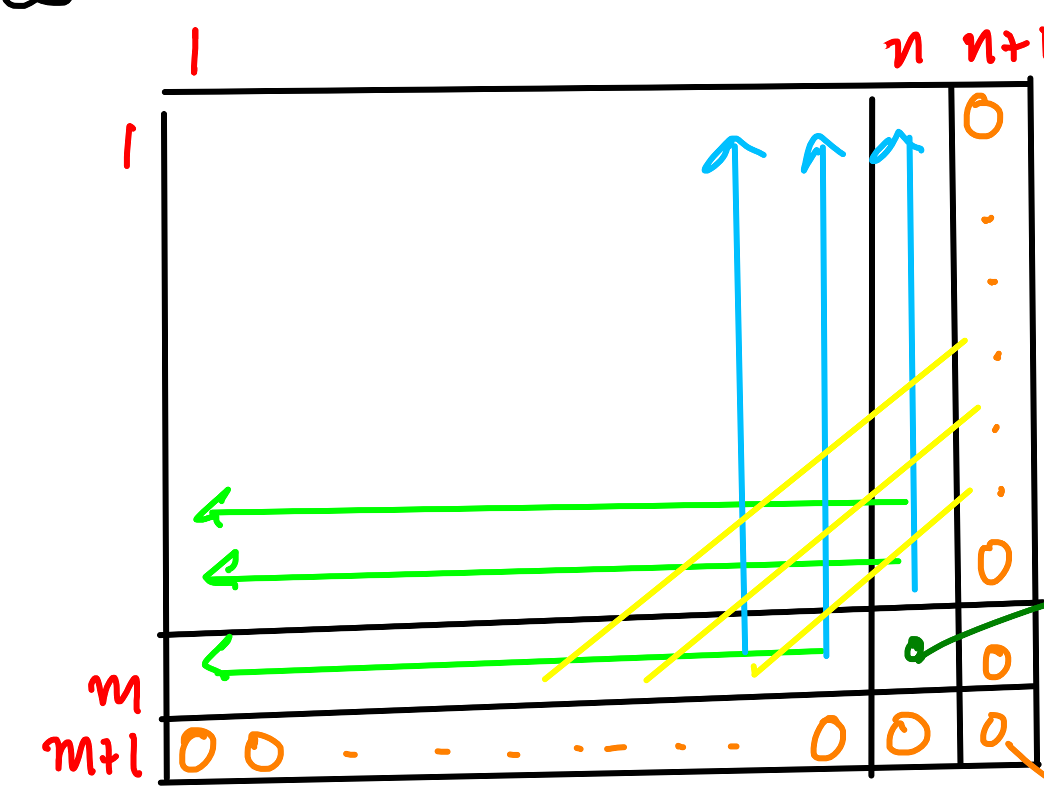


$$lcs(i,j) = 1 + lcs(i+1, j+1)$$

OR

$$\max\left(lcs(i, j+1),\ lcs(i+1, j)\right)$$

Filling up table



By row,
column,
or diagonal

dependencies known

Start here, base cases

1 2 3 4 5 6 7

A T A C G A C

T G C A C B

1 2 3 4 5 6

	A	T	A	C	G	A	C	
T	4	4	3	3	3	2	1	0
G	3	3	3	3	3	2	1	0
C	3	3	3	3	2	2	1	0
A	2	2	2	2	2	2	1	0
C	1	1	1	1	1	1	1	0
B	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0

Recovering the actual subsequence

- trace back how each table entry arose

- each time $1 + lcs(i+1, j+1)$ is chosen, $s_i = t_j$ added to common subsequence

- when $\max(lcs(i, j+1), lcs(i+1, j))$ is chosen, have to make a choice if both are equal

T C A C

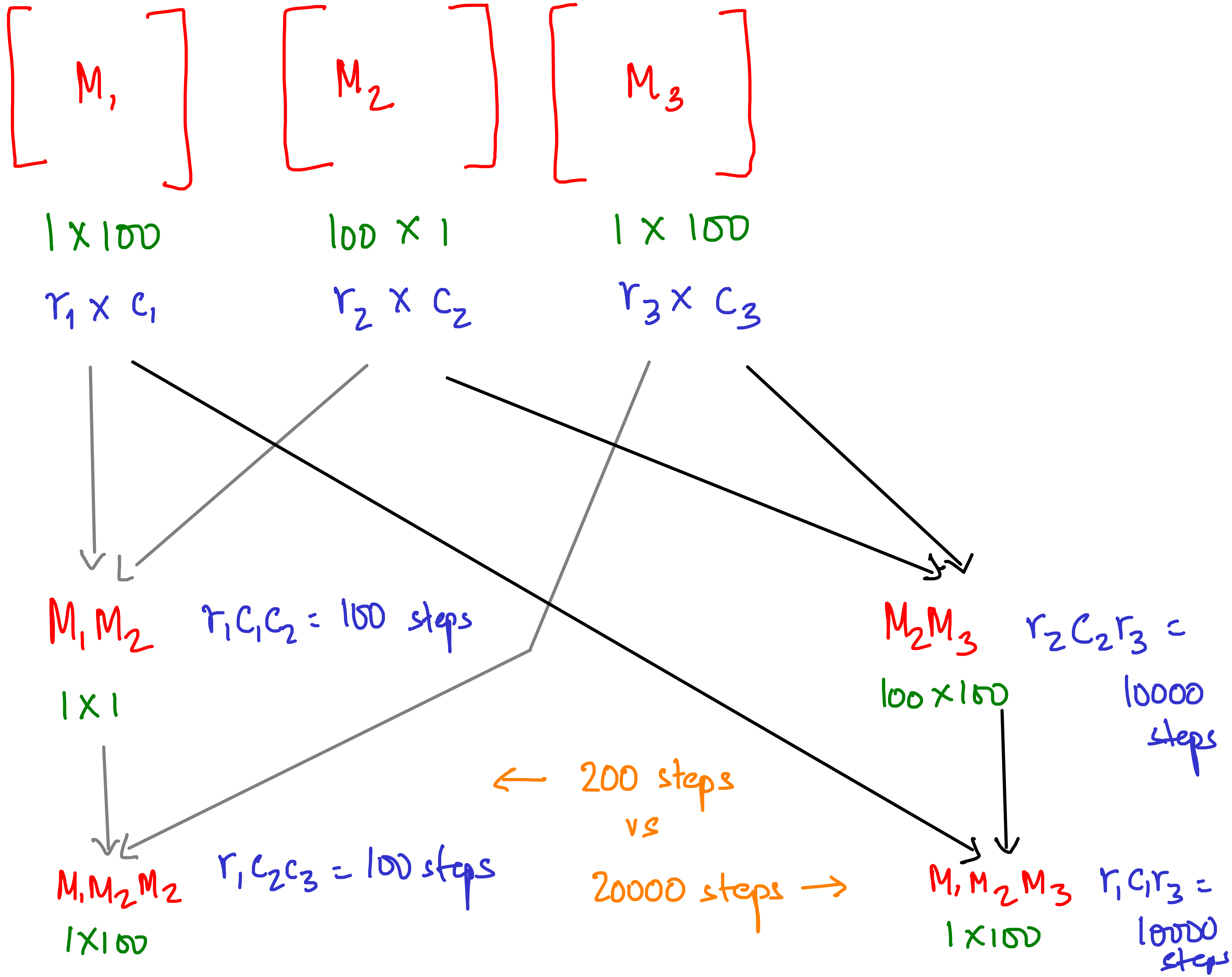
T G A C

Two "witnesses"

Not unique, in general

Matrix multiplication

- Multiplying matrices of dimensions M x N and N x P requires O(MNP) arithmetic operations
    - Product has size M x P
    - Each entry takes time O(N) to compute
- Multiplying $(M_1M_2)M_3$ and $M_1(M_2M_3)$
    - Same answer, but different complexity, in general



M1
M2
M3
1 x 100
100 x 1
1 x 100
r1 x c1
r2 x c2
r3 x c3
M1M2
r1c1c2 = 100 steps
1 x 1
M2M3
r2c2r3 =
10000 steps
100 x 100
← 200 steps
vs
20000 steps →
M1M2M2
1x100
r1c2c3 = 100 steps
M1M2M3
1 x 100
r1c1r3 =
10000 steps

Given $M_1 \quad M_2 \quad M_3 \quad \cdots \quad M_n$

$(r_1, c_1) \quad (r_2, c_2) \quad (r_3, c_3) \quad \quad (r_n, c_n)$

Dimensions match $c_i = r_{i+1}$, $1 \leq i < n$

Find an optimal order to evaluate product

Final multiplication is of form

$$(M_1 M_2 \cdots M_i)(M_{i+1} \cdots M_n)$$

$$r_1 \times c_i \qquad\qquad r_{i+1} \times c_n$$

Final step costs $r_1 \cdot c_i \cdot c_n$

Add cost of subproblems $(M_1 \cdot M_2 \cdots M_i)$, $(M_{i+1} \cdots M_n)$

Which M_i to choose?

- No idea! Try them all and choose best

$$Cost(1,n) = \min_{1 \le i < n} \left(r_1 \cdot c_i c_n + Cost(1,i) + Cost(i+1,n) \right)$$

In general

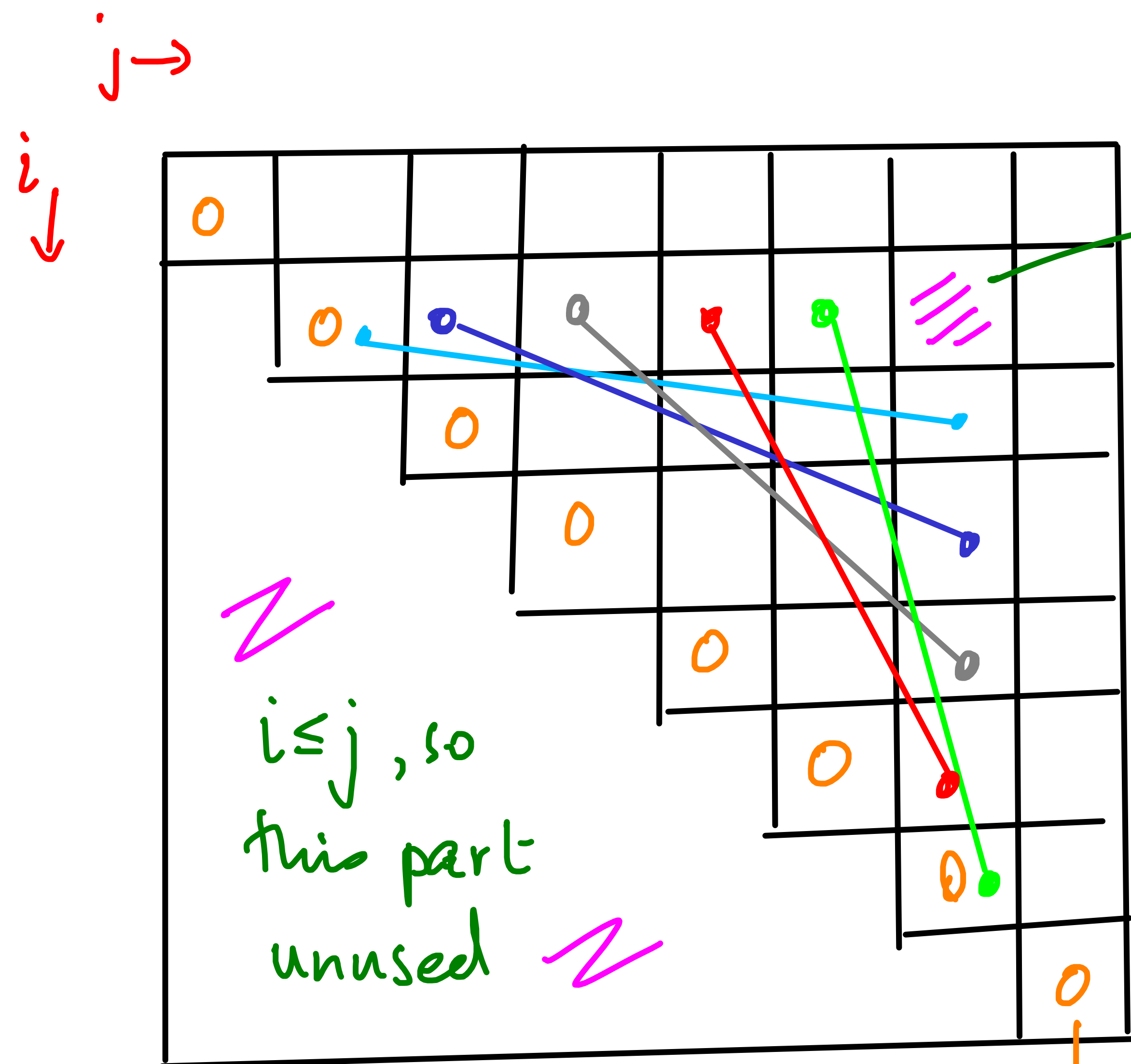
$$Cost(i,j) = \min_{i \le k < n} \left(r_i \cdot c_k c_j + Cost(i,k) + Cost(k+1,j) \right)$$

Multiplying $M_i M_{i+1} \cdots M_j$

Base Case: $Cost(i,i) = 0$

Cost of "multiplying" single matrix M_i

Filling the table



Diagonal, base case

lcs(2,7) = min

$$\left\{\begin{array}{ll} k=2 & r_2 c_2 c_7 + (2,2) + (3,7), \\ k=3 & r_2 c_3 c_7 + (2,3) + (4,7), \\ k=4 & r_2 c_4 c_7 + (2,4) + (5,7), \\ k=5 & r_2 c_5 c_7 + (2,5) + (6,7), \\ k=6 & r_2 c_6 c_7 + (2,6) + (7,7) \end{array}\right\}$$

Need $O(j-i)$ steps to compute entry (i,j)

Overall, $O(n^3)$ steps to fill $O(n^2)$ size table

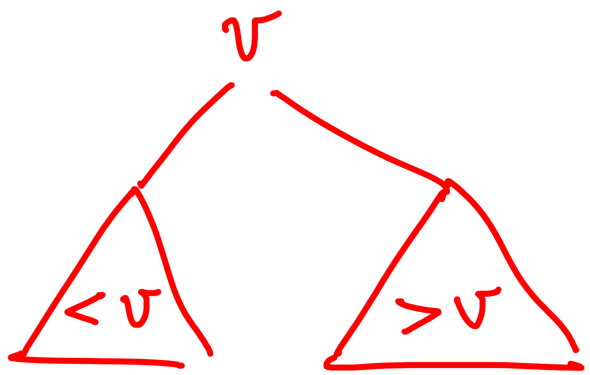
Fill table diagonally

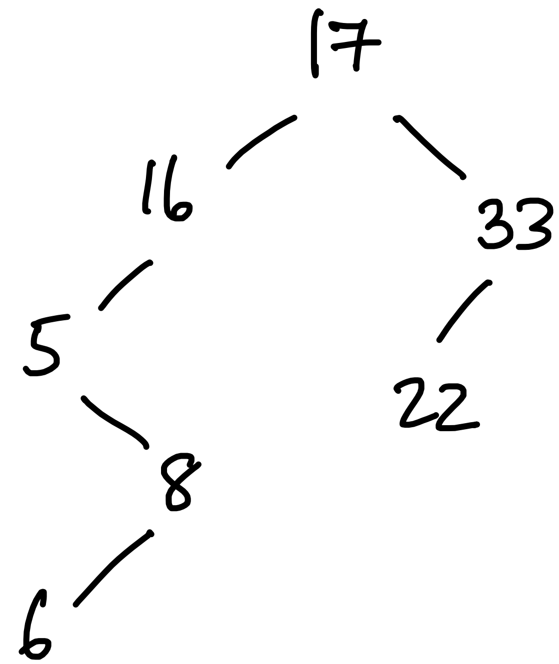
Need values to left and below

Optimal search tree

- Search tree
  - No duplicate values
  - At each node with value v
    - Left subtree has values less than v
    - Right subtree has values greater than v
- Search for a value like binary search

Search tree





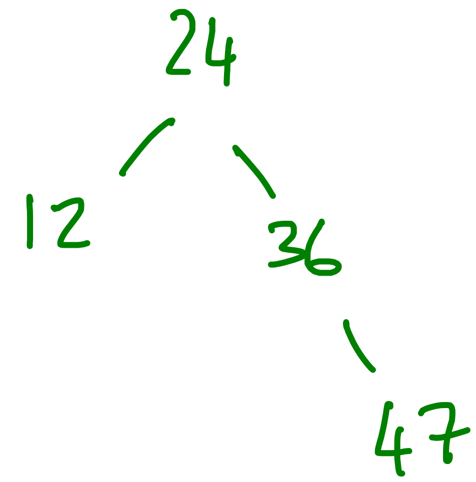
Different notions of optimality

- Search time proportional to tree height

- All values equally likely – "balance" the tree

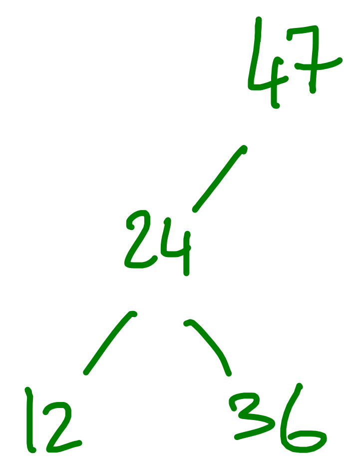
- Suppose different values are searched with different probabilities

Values	Probabilities
12	0.2
24	0.3
36	0.1
47	0.4



Expected search time

$$= 0.3 \times 1 \quad (24)$$
$$+ 0.2 \times 2 \quad (12)$$
$$+ 0.1 \times 2 \quad (36)$$
$$+ 0.4 \times 3 \quad (47)$$
$$= 2.1$$

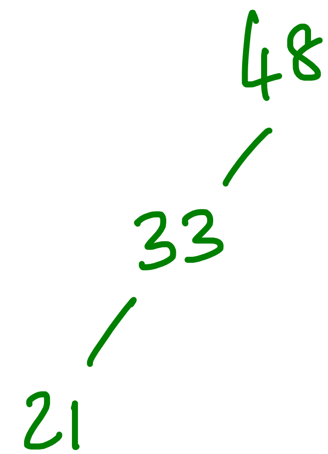


$$0.4 \times 1 \quad (47)$$
$$+ 0.3 \times 2 \quad (24)$$
$$+ 0.2 \times 3 \quad (12)$$
$$+ 0.1 \times 3 \quad (36)$$
$$= 1.9$$

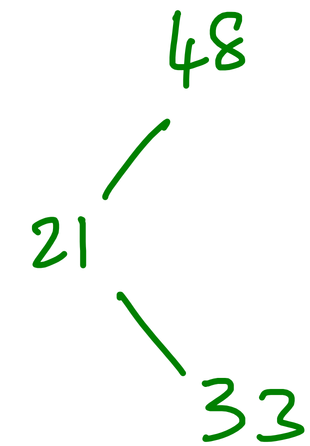
Second tree is better than first

Always make highest probability value root?

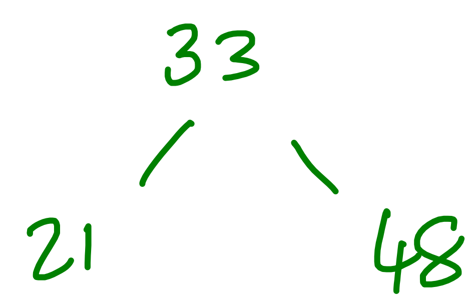
21	0.3
33	0.3
48	0.4



OR



Both have cost $0.4 + 0.6 + 0.9 = 1.9$



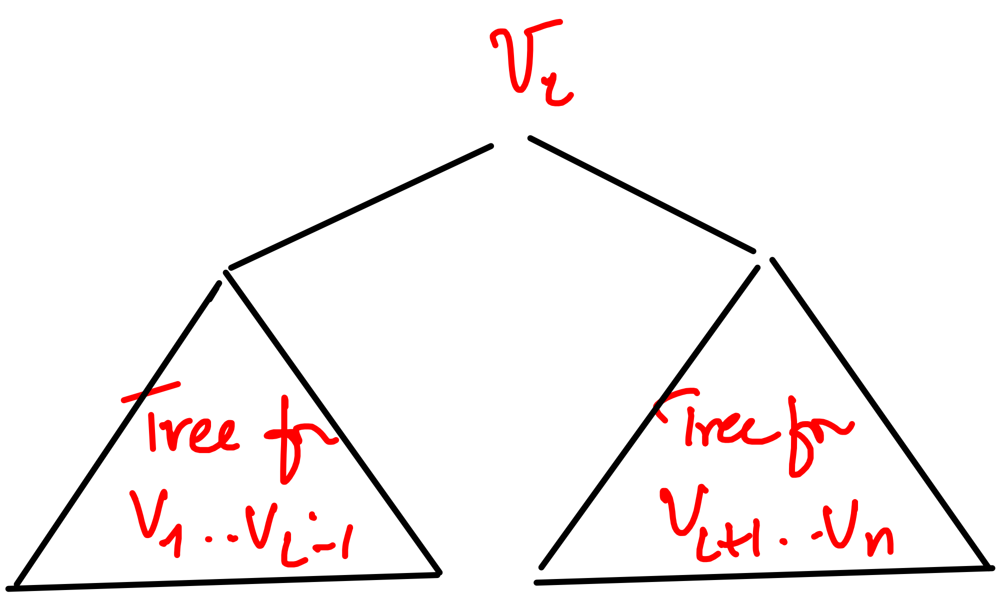
Cost is $0.3 + 0.6 + 0.8 = 1.7$

Given n values $\quad v_1 \quad v_2 \quad \ldots \quad v_n$

probabilities $\quad p_1 \quad p_2 \quad \ldots \quad p_n$

Assume w.l.o.g. values are sorted $\quad v_1 < v_2 < \ldots < v_n$

If we pick some v_i as root, the tree will look like



Expected cost =

p_i — v_i

$+ \sum_{j=1}^{i-1} p_j \left(\text{depth of } v_j \text{ in } T_{1..i-1} \underline{\underline{+1}} \right)$ — v_j in left subtree

+1 because down one level

$+ \sum_{j=i+1}^{n} p_j \left(\text{depth of } v_j \text{ in } T_{i+1..n} \underline{\underline{+1}} \right)$ — v_j in right subtree

In general, subproblem is $\quad v_i \quad v_{i+1} \quad \ldots \quad v_j$

$\quad p_i \quad p_{i+1} \quad \ldots \quad p_j$

$$\mathrm{Cost}(i,j) = \text{minimize cost over all choices for the root}$$

$$= \min_{i \le k \le j} \; p_k + \sum_{\ell=i}^{k-1} p_\ell \big(\underset{\text{in } T_{i..\ell-1}}{\mathrm{depth}(v_\ell)} + 1\big) + \sum_{\ell=i+1}^{j} p_\ell \big(\underset{\text{in } T_{\ell+1..j}}{\mathrm{depth}(v_\ell)} + 1\big)$$

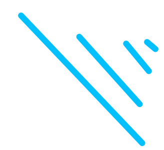
Rearranging

$$= \min_{i \le k \le j} \sum_{\ell=1}^{k-1} p_\ell \big(\mathrm{depth}(v_\ell)\big) + \sum_{\ell=l+1}^{j} p_\ell \big(\mathrm{depth}(v_\ell)\big) + p_k + \sum_{\ell=1}^{k-1} p_\ell + \sum_{\ell=k+1}^{j} p_\ell$$

$$= \min_{i \le k \le j} \mathrm{Cost}(i,k-1) + \mathrm{Cost}(k+1,j) + \sum_{\ell=i}^{j} p_\ell$$

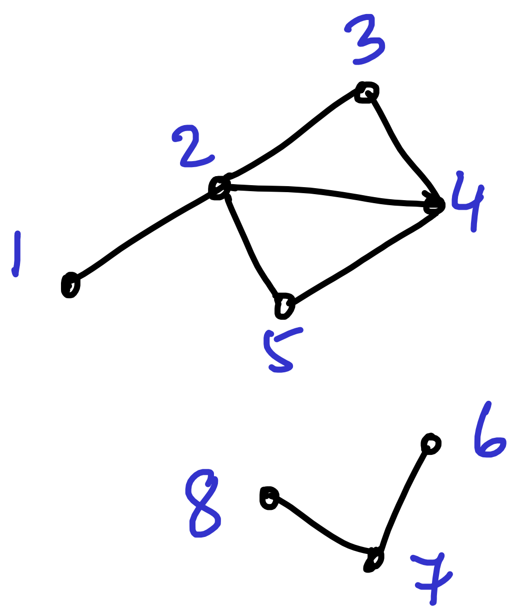
Assume $\mathrm{Cost}(\ell+1,\ell) = 0$ (when $k=i$ or $k=j$)

Table filling is similar to matrix multiplication problem

- Cost(i,j) needs value to left on same row and below on same column
- Fill diagonally  Only need values above diagonal, $i \leq j$
- Time taken is $O(j-i)$ for entry (i,j)
- Overall $O(n^3)$ steps to fill $O(n^2)$ table

Warshall, transitive closure

- Given an undirected graph …
  - Edges represented by adjacency matrix A
- … compute which pairs of vertices are connected by a path (sequence of edges)
  - Connectivity = transitive closure of edge relation



Adjacency matrix

$A[i][j] = 1$ iff (i,j) is an edge

	1	2	3	4	5	6	7	8
1	0	1	0	0	0	0	0	0
2	1	0	1	1	1	0	0	0
3	0	1	0	1	0	0	0	0
4	0	1	1	0	1	0	0	0
5	0	1	0	1	0	0	0	0
6	0	0	0	0	0	0	1	0
7	0	0	0	0	0	1	0	1
8	0	0	0	0	0	0	1	0

Want to compute a new matrix P (for paths)

$P[i][j] = 1$ iff there is a path from i to j

Warshall's algorithm

- Vertices are numbered $1, 2, \ldots, k$
- Compute a sequence of matrices $A^0, A^1, \ldots, A^n$

  $A^k[i][j] = 1$ iff there is a path from v_i to v_j where intermediate vertices are in $\{v_1, \ldots v_k\}$

  Note that i, j can be bigger than k

- Clearly

  $A^0 = A$ — no intermediate vertices — all paths are direct edges

  $A^n = P$ — any vertex can appear on path

Computing A^k from A^{k-1}

$A^k[i][j] = 1$ if $A^{k-1}[i][j] = 1$ Path exists already, v_k not needed

OR

$A^{k-1}[i][k] = 1$, $A^{k-1}[k][j] = 1$

Paths from v_i to v_k, v_k to v_j only use vertices in $\{v_1, \ldots v_{k-1}\}$

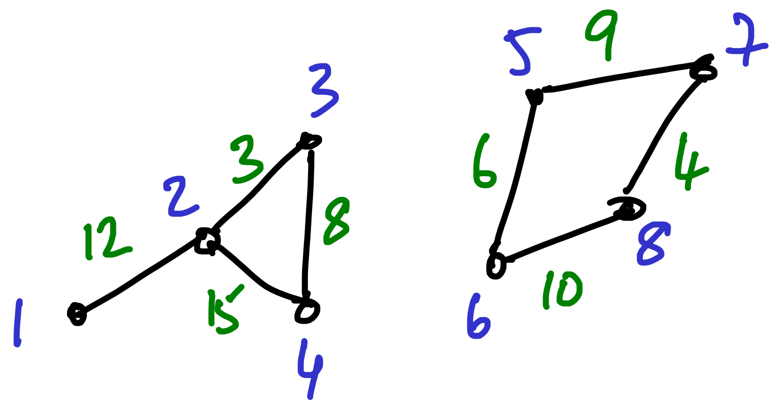
Updating A^k from A^{k-1} takes $O(n^2)$ time

Overall, n iterations, so $O(n^3)$

Simple nested loop implementation - see Levitin

Floyd, all pairs shortest path

- Given an undirected graph with edge weights …
  - Weight represents cost—price, distance, time …
  - No edge—infinite cost
- … find the smallest weight path between any pairs of vertices
- "All pairs shortest path"



- Find smallest weight path between v_i, v_j
- No path $\Rightarrow$ weight is ∞
  (to implement, choose a value bigger than sum of all weights)

Initial adjacency matrix has weight information

$A[i][j]$ = weight of edge (i,j), ∞ if no edge

Like Warshall's algorithm, define a sequence of matrices

$A^0, A^1, \dots, A^n$

$A^k[i][j]$ = min weight of paths from i to j using intermediate vertices in $\{v_1, \dots, v_k\}$

Computing A^k from A^{k-1}

$$A^k[i][j] = \min\Big(A^{k-1}[i][j], \quad A^{k-1}[i][k] + A^{k-1}[k][j] \Big)$$

existing path, without using v_k

best new path via v_k

Again, each iteration $A^{k-1} \to A^k$ takes $O(n^2)$ time

Overall n iterations $\to O(n^3)$