Dynamic Programming Cont

And a very brief couple of slides on shortest path algorithms

The basic 4 steps

This looks like **divide and conquer!**

1. Characterize the *optimal* solution (i.e. how can we determine what is the optimal solution)
2. Recursively define the value of the optimal solution
3. Compute the optimal solution
4. Construct the optimal solution (i.e. what pieces went were combined to find the optimal value)

The basic 4 steps

This looks like **divide and conquer!**

1. Characterize the *optimal* solution (i.e. find the optimal substructure and use it to construct an optimal solution)
2. Recursively define the value of the optimal solution
3. Compute the optimal solution
4. Construct the optimal solution (i.e. what pieces went were combined to find the optimal value)

How could you write the unix program "diff"

The operation of `diff` is based on solving the longest common subsequence problem.[3]

In this problem, given two sequences of items:

```
a b c d f g h j q z
```

```
a b c d e f g i j k r x y z
```

and we want to find a longest sequence of items that is present in both original sequences in the same order. That is, we want to find a new sequence which can be obtained from the first original sequence by deleting some items, and from the second original sequence by deleting other items. We also want this sequence to be as long as possible. In this case it is

```
a b c d f g   j   z
```

https://en.wikipedia.org/wiki/Diff_utility

Longest Common Subsequence

```
AAB24882      TYHMCQFHCRYVNNHSGEKLYECNERSKAFSCPSHLQCHKRRQIGEKTHEHNQCGKAFPT 60
AAB24881      --------------------YECNQCGKAFAQHSSLKCHYRTHIGEKPYECNQCGKAFSK 40
                                  ****: .***:  * *:** * :****.:* *******..

AAB24882      PSHLQYHERTHTGEKPYECHQCGQAFKKCSLLQRHKRTHTGEKPYE-CNQCGKAFAQ- 116
AAB24881      HSHLQCHKRTHTGEKPYECNQCGKAFSQHGLLQRHKRTHTGEKPYMNVINMVKPLHNS 98
               **** *:***********:***:**.: .***************     :  *.: :
```

https://upload.wikimedia.org/wikipedia/commons/8/86/Zinc-finger-seq-alignment2.png

Problem: Given 2 sequences, $X = \langle x_1,\ldots, x_m\rangle$ and $Y = \langle y_1,\ldots, y_n\rangle$.

Find a subsequence common to both whose length is longest.

A subsequence doesn't have to be consecutive, but it has to be in *order*.

maelstrom	heroically	springtime	horseback
becalm	scholarly	pioneer	snowflake

https://upload.wikimedia.org/wikipedia/commons/8/89/Metal_tubes_stored_in_a_yard.jpg

Algorithm?

springtime	horseback
pioneer	snowflake
maelstrom	heroically
becalm	scholarly

brute force!

Try ever subsequence in X and see if it a subsequence in Y

Time? 2^m subsequences to check…

$\Theta(n)$ to see if a sequence was a subsequence

How can we find if a sequence is a subsequence?

Find the first letter in the sequence

Find the next letter in the sequence…

Given 2 sequences, $X = \langle x_1,\ldots, x_m\rangle$ and $Y = \langle y_1,\ldots, y_n\rangle$, find a subsequence, Z common to both whose length is longest. A subsequence doesn't have to be consecutive, but it has to be in order.

Can we use dynamic Programming?

i.e. Does it have an <u>optimal substructure?</u> <u>Yes!</u>

Notation

prefix: $X_i = \langle x_1,\ldots, x_i\rangle$ $\quad X_4 = \langle m,a,e,l\rangle$

$Y_i = \langle y_1,\ldots, y_i\rangle.$ $\quad Y_3 = \langle b,e,c\rangle$

Theorem: Let $Z = \langle x_1,\ldots, x_k\rangle$ be any LCS of X and Y

If $x_m = y_n$ ☛ $z_k = x_m = y_n$ and Z_{k-1} is a LCS of X_{m-1} and Y_{n-1}

If $x_m \neq y_n$ and $z_k \neq x_m$ ☛ Z is a LCS of X_{m-1} and Y

If $x_m \neq y_n$ and $z_k \neq y_n$ ☛ Z is a LCS of X and Y_{n-1}

Therefore, a LCS of two sequences contains as a prefix a LCS of prefixes of the sequences.

X = m,a,e,l,s,t,r,o,m

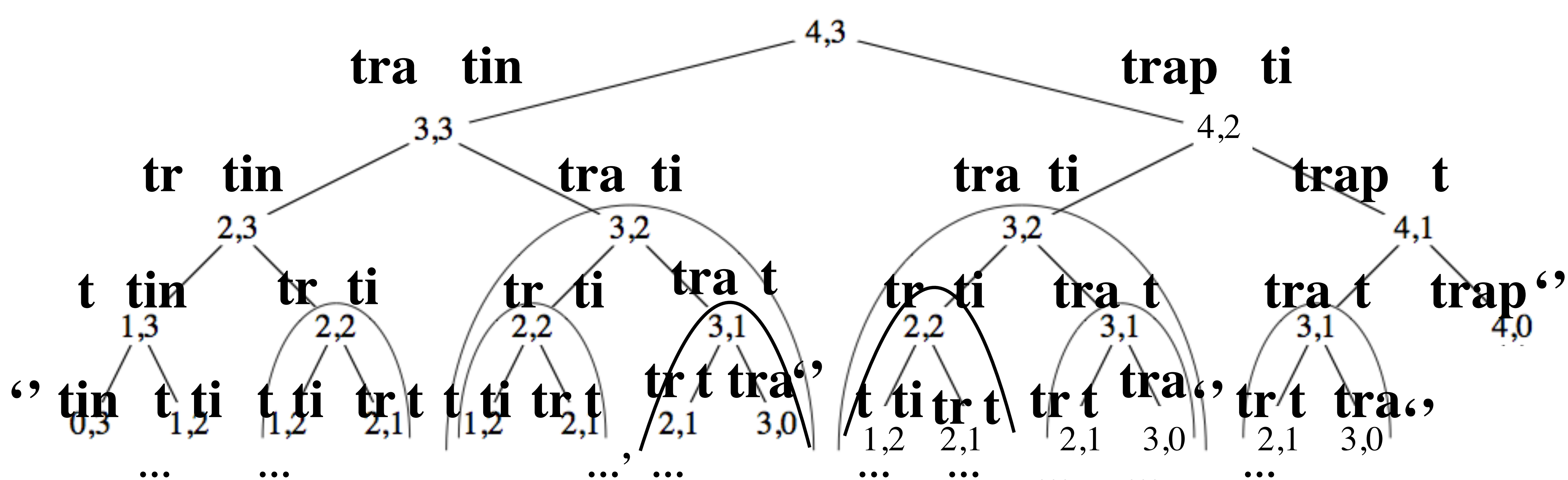
Y = b,e,c,a,l,m

Z = e, l, m $\quad z_1 = e \quad z_2 = l \quad z_3 = m$

$c[i,j] =$ length of LCS of X_i and Y_j

$$
c[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ c[i-1, j-1] + 1 & \text{if } i, j > 0 \text{ and } x_i = y_j \\ \max(c[i-1, j], c[i, j-1]) & \text{if } i, j > 0 \text{ and } x_i \neq y_j \end{cases}
$$

Largest common subsequence of: **trap tin**



Only Θ(mn) distinct subproblems

Largest common subsequence of: **spanking** **amputation**

$$c[i,j]=\begin{cases} 0 & \text{if } i=0 \text{ or } j=0 \\ c[i-1,j-1]+1 & \text{if } i,j>0 \text{ and } x_i = y_j \\ \max(c[i-1,j],c[i,j-1]) & \text{if } i,j>0 \text{ and } x_i \neq y_j \end{cases}$$

		a	m	p	u	t	a	t	i	o	n
	0	0	0	0	0	0	0	0	0	0	0
s	0	0	0	0	0	0	0	0	0	0	0
p	0	0	0	1	1	1	1	1	1	1	1
a	0	1	1	1	1	1	2	2	2	2	2
n	0	1	1	1	1	1	2	2	2	2	3
k	0	1	1	1	1	1	2	2	2	2	3
i	0	1	1	1	1	1	2	2	3	3	3
n	0	1	1	1	1	1	2	2	3	3	4
g	0	1	1	1	1	1	2	2	3	3	4
				p			a		i		n

LCS-LENGTH(X, Y, m, n)

let b[1..m, 1..n] and c[0..m, o..n] be new tables

for i =1 **to** m

c[i, 0] = 0

for j = 0 **to** n

c[0, j] = 0

for i = 1 **to** m

for j = 1 **to** n

if x_i ==y_j

c[i, j] = c[i -1, j -1] + 1

b[i,j] = “↖”

else if c[i - 1, j] ≥ c[i, j -1]

c[i, j] = c[i -1, j]

b[i,j] = “↑”

else c[i, j] = c[i, j -1]

b[i,j] = “←”

return c and b

$$c[i,j] = \begin{cases} 0 & \text{if } i=0 \text{ or } j=0 \\ c[i-1,j-1]+1 & \text{if } i,j>0 \text{ and } x_i = y_j \\ \max(c[i-1,j],\, c[i,j-1]) & \text{if } i,j>0 \text{ and } x_i \neq y_j \end{cases}$$

Largest common subsequence of: **spanking** **amputation**

Running time?

		a	m	p	u	t	a	t	i	o	n
	0	0	0	0	0	0	0	0	0	0	0
s	0	0	0	0	0	0	0	0	0	0	0
p	0	0	0	1	1	1	1	1	1	1	1
a	0	1	1	1	1	1	2	2	2	2	2
n	0	1	1	1	1	1	2	2	2	2	3
k	0	1	1	1	1	1	2	2	2	2	3
i	0	1	1	1	1	1	2	2	3	3	3
n	0	1	1	1	1	1	2	2	3	3	4
g	0	1	1	1	1	1	2	2	3	3	4
				p			**a**		**i**		**n**

Optimal substructure varies across problem domains:

1. *How many subproblems* are used in an optimal solution.

2. *How many choices* in determining which subproblem(s) to use.

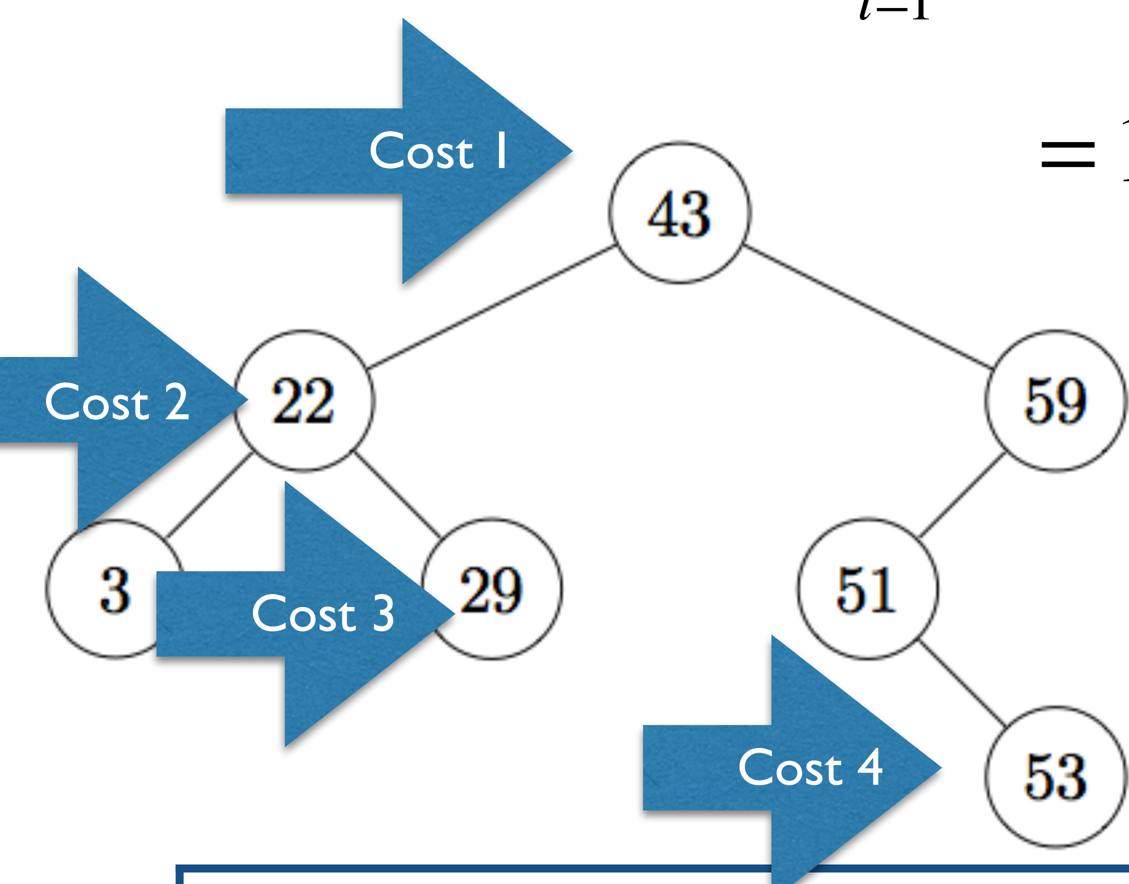
Optimal Binary Search Trees

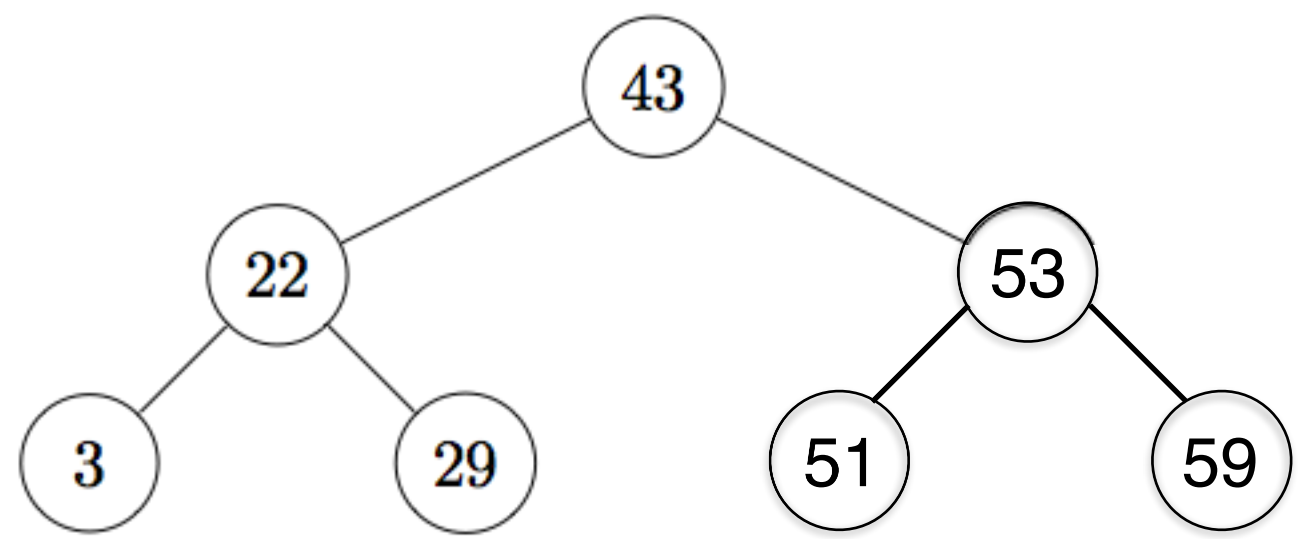
Problem: Given sequence $K = \langle k_1, k_2, \ldots, k_n \rangle$ of n sorted distinct keys ($k_1 < k_2 < \cdots < k_n$) where each key, k_i, we have a probability, p_i, that the search will be for k_i

Output: A binary search tree with minimum expected search cost

$$E[\text{search cost in } T] = \sum_{i=1}^{n} (\text{depth}_T(k_i) + 1) * p_i = \sum_{i=1}^{n} \text{depth}_T(k_i) p_i + \sum_{i=1}^{n} p_i$$

$$= 1 + \sum_{i=1}^{n} \text{depth}_T(k_i) * p_i$$

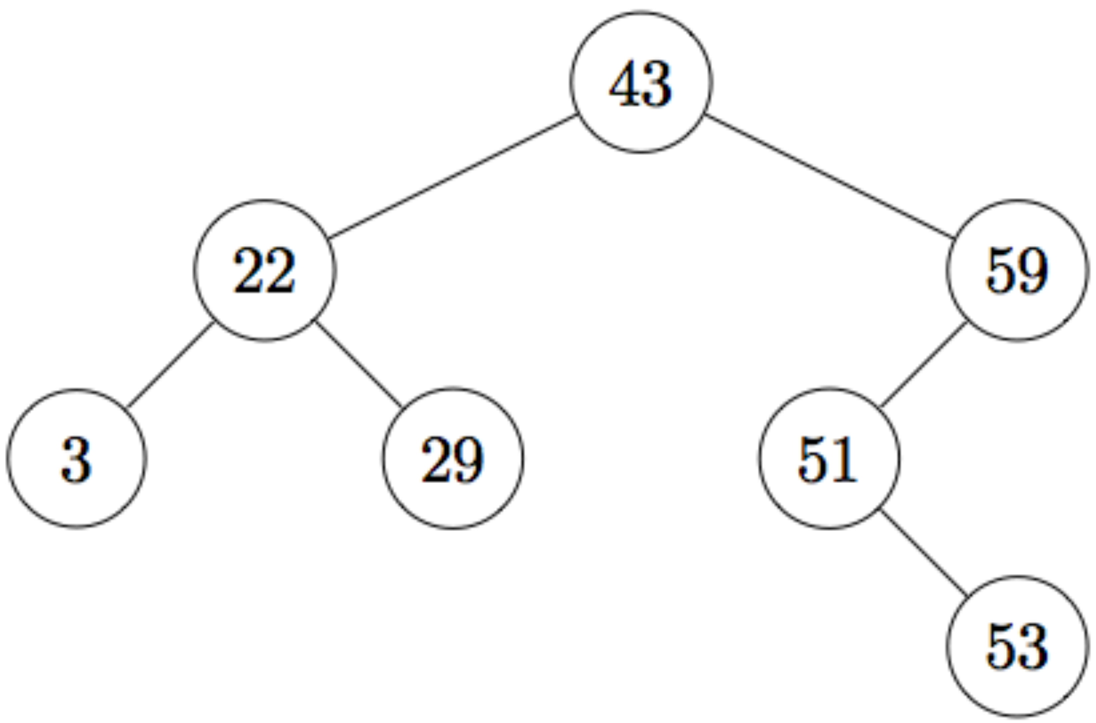




k_i	3	22	29	43	51	53	59
p_i	0.1	0.15	0.1	0.3	0.1	0.01	0.24

$$E[\text{search cost in } T] = 1 + \sum_{i=1}^{n} \text{depth}_T(k_i) * p_i$$

k_i	3	22	29	43	51	53	59
p_i	0.1	0.15	0.1	0.3	0.1	0.01	0.24



$$E[\text{search cost in } T] = 2.02$$

k_i	$\text{depth}_T(k_i)$	$\text{depth}_T(k_i)*p_i$
3	2	0.2
22	1	0.15
29	2	0.2
43	0	0
51	2	0.2
53	3	0.03
59	1	0.24

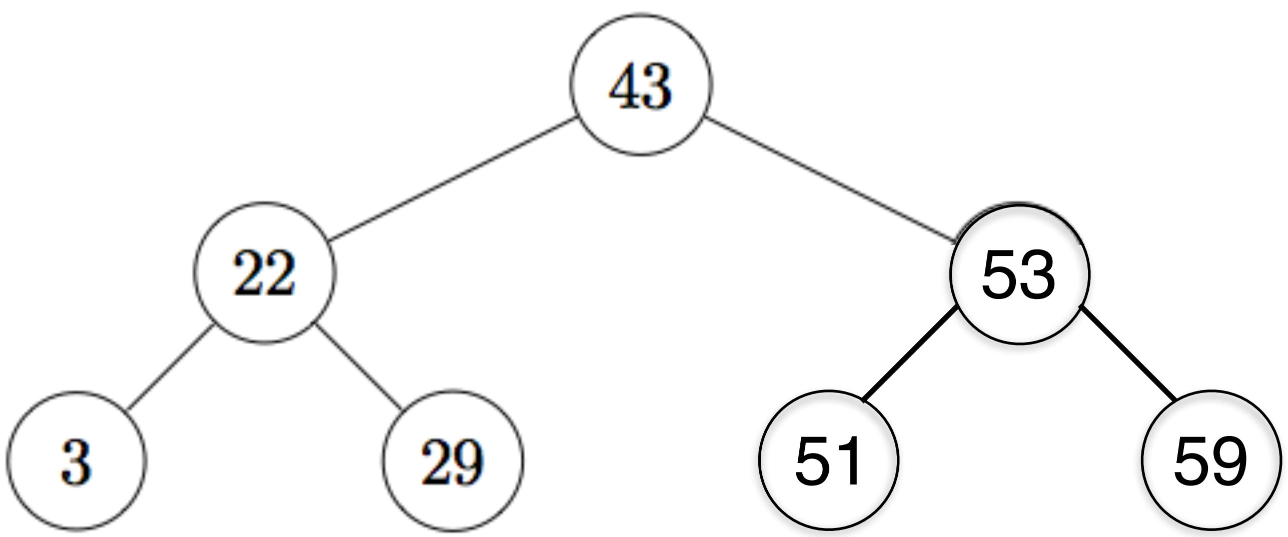
Total = 1.02

Optimal BST might not be the most balanced tree!

Optimal BST might not have the highest cost as the root!

$$E[\text{search cost in } T] = 1 + \sum_{i=1}^{n} \text{depth}_T(k_i) * p_i$$

k_i	3	22	29	43	51	53	59
p_i	0.1	0.15	0.1	0.3	0.1	0.01	0.24



$$E[\text{search cost in } T] = 2.24$$

k_i	$\text{depth}_T(k_i)$	$\text{depth}_T(k_i)*p_i$
3	2	0.2
22	1	0.15
29	2	0.2
43	0	0
51	2	0.2
53	1	0.01
59	2	0.48

Total = 1.24

Exhaustive Search?

- construct every *n* node BST
- put in keys
- compute expected cost

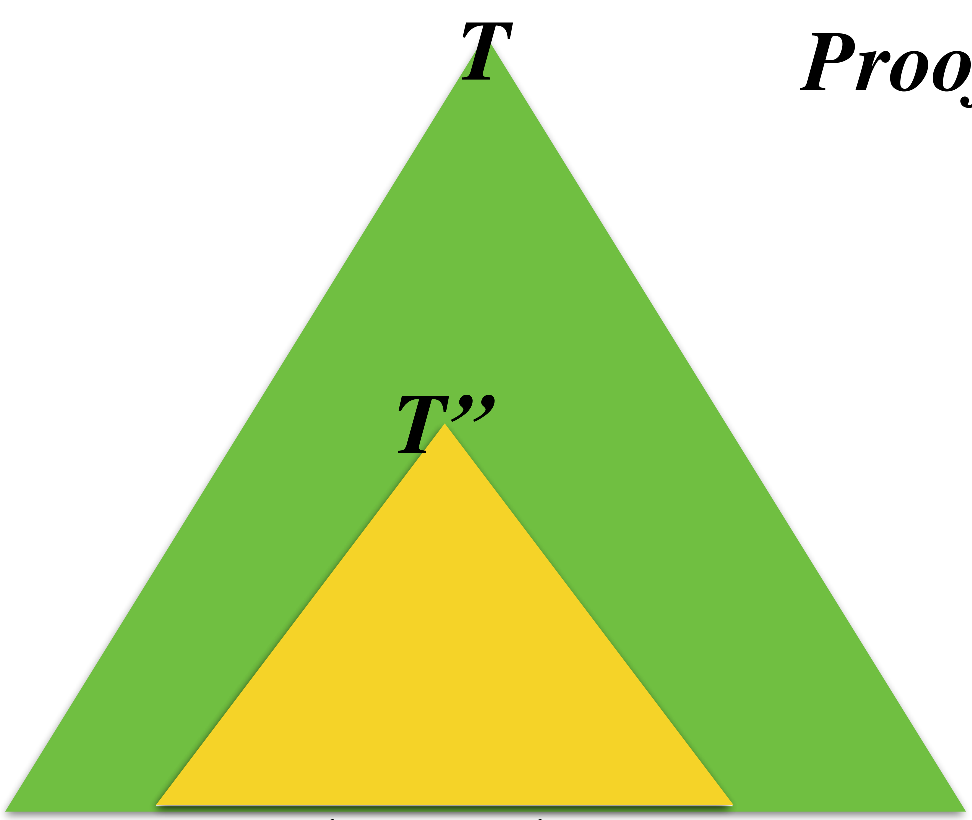
$\Omega(4^n / n^{3/2})$different BST

Optimal Substructure?

If T is an optimal BST and T' is a subtree,

then T' is an optimal BST for the keys it contains!

Proof: Cut and paste!



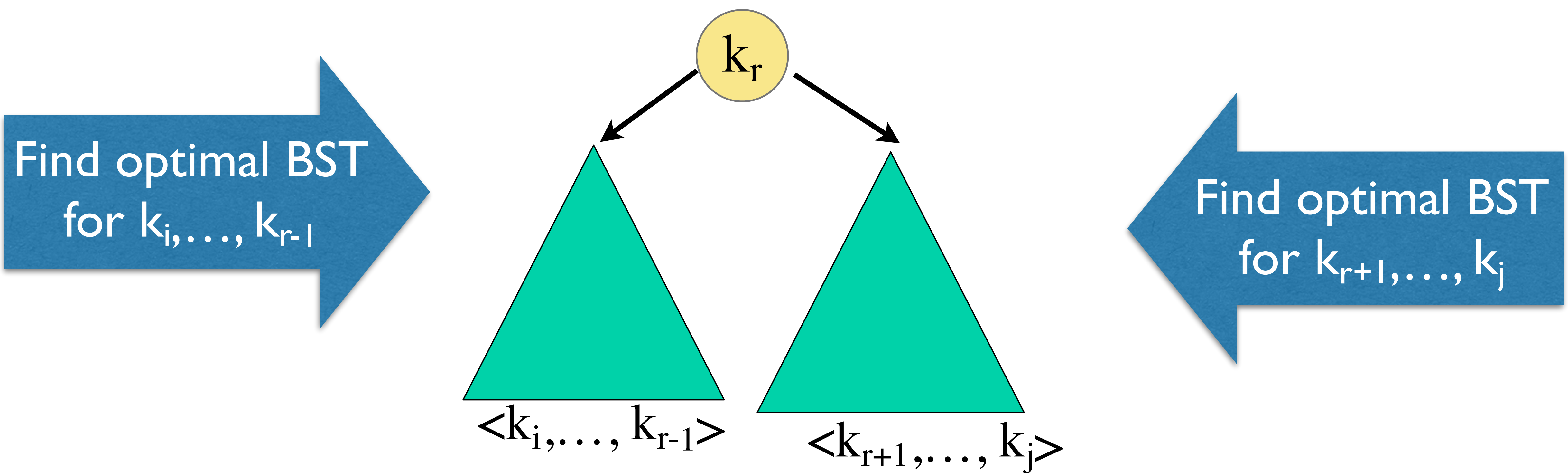
$<k_i, \ldots, k_j>$

Use optimal substructure to construct an optimal solution from optimal solutions to subproblems

Given $k_i, k_{i+1}, \ldots, k_j$

Observation: one key must be the root: $\boldsymbol{k_r}$ for some $i \le r \le j$

Solution: try all keys as the root! One must be the best.



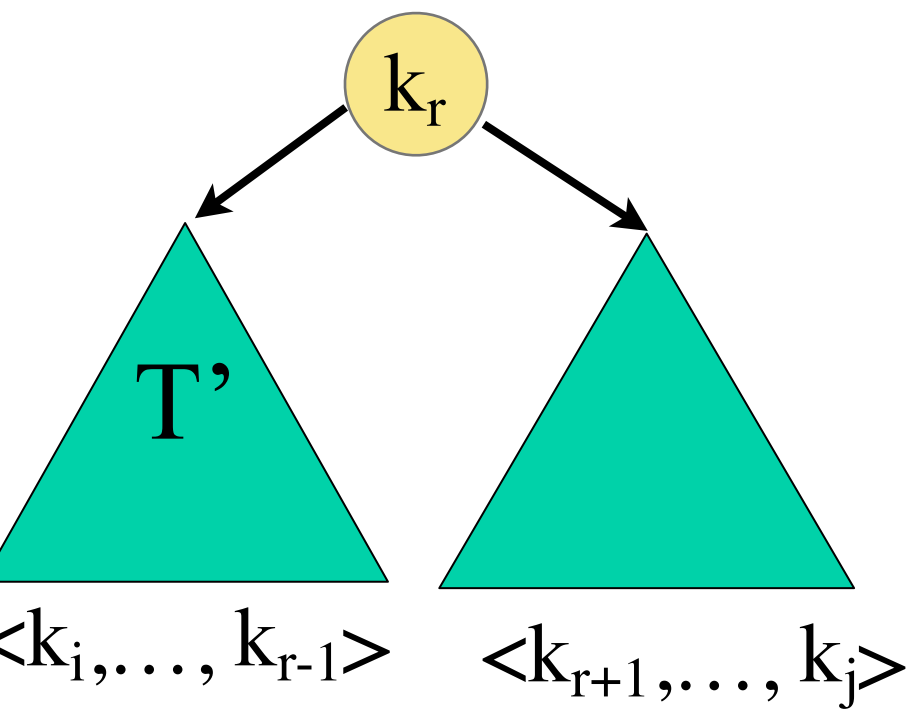
The best one we find is guaranteed to be an optimal BST for $k_i, \ldots, k_j$

For keys $k_i, \ldots, k_{r-1}$ ***if we knew the root was r - what is the optimal BST?***

$$w(i,j) = \sum_{l=i}^{j} p_l$$

$$e[i,j] = \text{expected search time for optimal BST for } k_i, \ldots, k_j$$

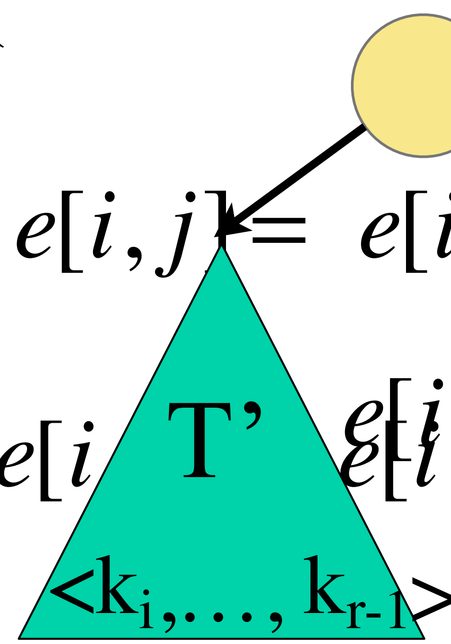
if *root = r* $$e[i,j] = p_r + \left(e[i,r-1] + \sum_{l=i}^{r-1} p_l \right) + \left(e[r+1,j] + \sum_{l=r+1}^{j} p_l \right)$$



$$e[i,j] = e[i,r-1] + e[r+1,j] + \sum_{l=i}^{j} p_l$$

$$e[i,r-1] + \sum_{l=i}^{r-1} p_l$$

$$e[i,j] = e[i,r-1] + e[r+1,j] + w(i,j)$$



Recursive solution:

$$e[i,j] = \begin{cases} 0 & \text{if } j = i-1 \\ \min\limits_{i \le r \le j} \{ e[i,r-1] + e[r+1,j] + w(i,j) \} & \text{if } i \le j \end{cases}$$

1	2	3	4	5
0.25	0.2	0.05	0.2	0.3

w	0	1	2	3	4	5
1	0	0.25	0.45	0.5	0.7	1.0
2		0	0.2	0.25	0.45	0.75
3			0	0.05	0.25	0.55
4				0	0.2	0.5
5					0	0.3
n+1						0

e	0	1	2	3	4	5
1	0	0.25	0.65	0.8	1.25	2.1
2		0	0.2	0.3	0.75	1.35
3			0	0.05	0.3	0.85
4				0	0.2	0.7
5					0	0.3
n+1						0

$$w(i,j) = \sum_{l=i}^{j} p_l$$

$$e[i,j] = \begin{cases} 0 & \text{if } j = i-1 \\ \min\limits_{i \le r \le j} \{ e[i,r-1] + e[r+1,j] + w(i,j) \} & \text{if } i \le j \end{cases}$$

Time $O(n^3)$: n^2 slots and each slot takes $O(n)$ time

Can also show Big-Omega(n^3). Therefore, Big-Theta(n^3)

OPTIMAL-BST(p, q, n)

```
let e[1.. n, 0 .. n], w[1 .. n, 0 .. n], and root[1 .. n, 1 .. n] be new tables
for i = 1 to n +1
    e [i, i -1] = 0
    w[i, i -1] = 0
for l = 1 to n
    for i = 1 to n - l + 1
        j = i + l -1
        e[i,j]  = ∞
        w[i,j] = w[i,j - 1] + pj
         for r = i to j// r = i to i + l -1
            t = e[i,r - 1] + e[r +1, j] + w[i, j]
            if t < e[i, j]
                e[i, j] =  t
                root[i, j] = r
return e and root
```

1	2	3	4	5
0.25	0.2	0.05	0.2	0.3

We are filling in positon e[i,j]. Optimal BST for $k_i,..,k_j$

Trying all different choices for the root in the optimal BST for $k_i,..,k_j$

w	0	1	2	3	4	5
1	0	0.25	0.45	0.5	0.7	1.0
2		0	0.2	0.25	0.45	0.75
3			0	0.05	0.25	0.55
4				0	0.2	0.5
5					0	0.3
						0

	0	1	2	3	4	5
1	0	0.25	0.65	0.8	1.25	2.1
2		0	0.2	0.3	0.75	1.35
3			0	0.05	0.3	0.85
4				0	0.2	0.7
5					0	0.3
n+1						0

Time $O(n^3)$: for loops nested 3 deep, each loop index takes on <= n values.

Can also show Big-Omega(n^3). Therefore, Big-Theta(n^3)

Two key ingredients:

1. *Optimal Substructure*
   - optimal solution to the global problem uses optimal solutions of the subproblems

2. *Subproblem overlap*
   - optimal solutions to subproblems can contain optimal solutions to other subproblems.

Steps for optimal substructure

- Solution involves making a choice which typically leaves one or two subproblems to solve
- Determine which subproblems arise when given the correct choice. (And characterize this space of subproblems.)
- Show the solutions to the subproblems used within the optimal solution must be optimal by using a cut and paste argument

Characterize the space of subproblems

- Keep as simple as possible
- Expand if necessary
- Examples:
  - Rod cutting: optimal price of rods of length n-i for $1 \leq i \leq n$
  - Matrix chain mult: optimal order of mult. for $A_i \ldots A_j$
  - LCS: LCS of X_i, Y_j for $1 \leq i \leq j \leq n$
  - Optimal BST: subtrees with keys $k_i, \ldots, k_j$ $1 \leq i \leq j \leq n$

Running time

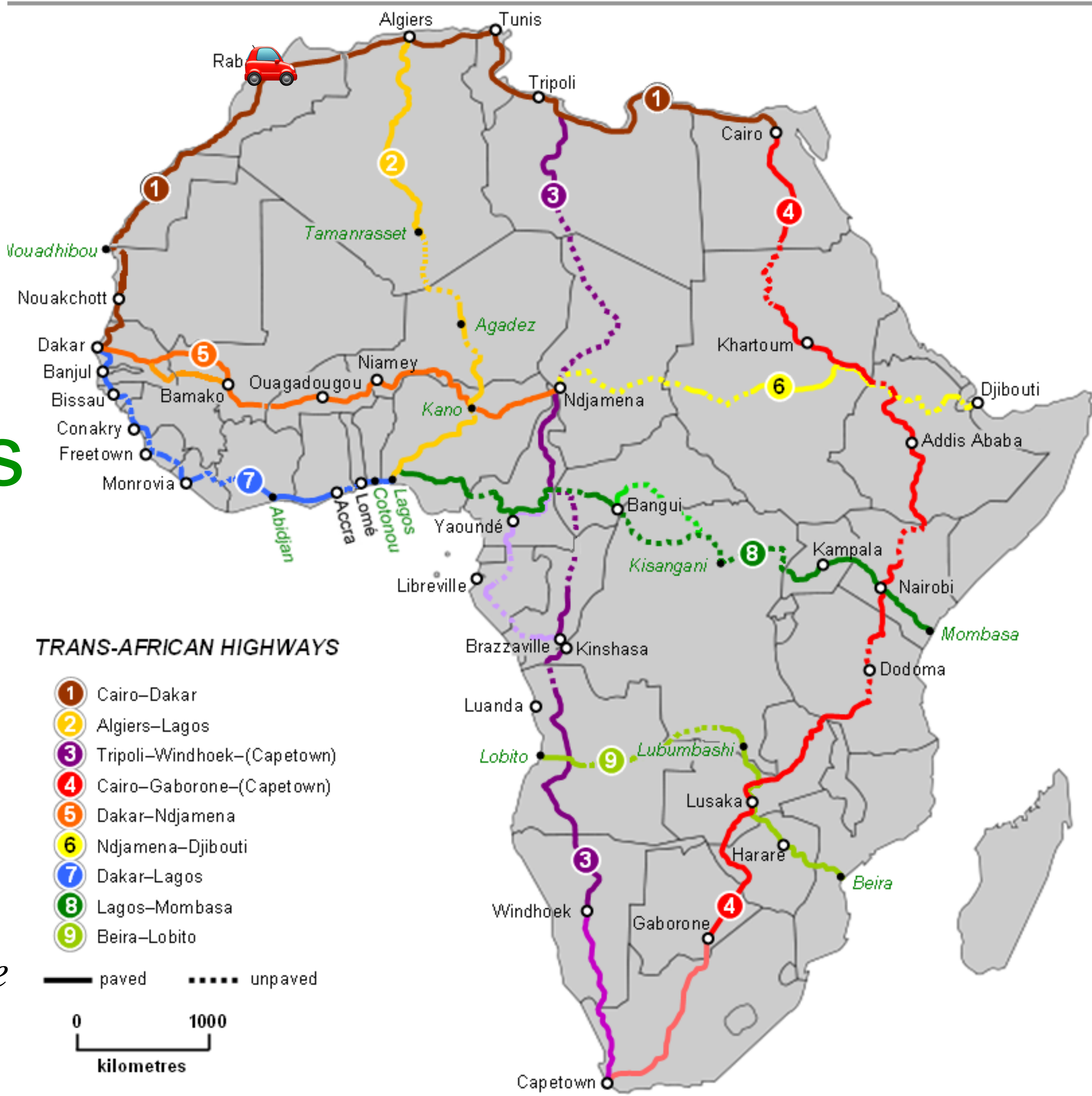
- informally - # subproblems × # choices
- Examples:
  - Rod cutting: $\Theta(n)$ subproblems, $\leq$ n choices
  - Matrix chain multiplication: $\Theta(n^2)$ subproblems, $\leq$ O(n) choices
  - LCS: $\Theta(m\ n)$ subproblems, $\leq$ 3 choices
  - Optimal BST: $\Theta(n^2)$ subproblems, $\leq$ O(n) choices
- Subproblem graph gets the same analysis

Shortest Path Algorithms

Single Source Shortest Path Algorithms

Find shortest distance between two points on the graph.

Edge weight can be negative (for some algorithms) but we *don't allow negative weight cycles*



By I, Rexparry sydney, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=2427121

Single Source Shortest Path Algorithms

$$w(p) = \text{weight of path } p = \langle v_0, v_1, \ldots, v_k \rangle = \sum_{i=1}^{k} w(v_{i-1}, v_i) = \text{sum of weights on path } p$$

shortest-path weight u to v:

$$\delta(u,v) = \begin{cases} \min\{w(p) : u \overset{p}{\sim>} v & \text{if there exists a path } u \sim> v \\ \infty & \text{otherwise} \end{cases}$$

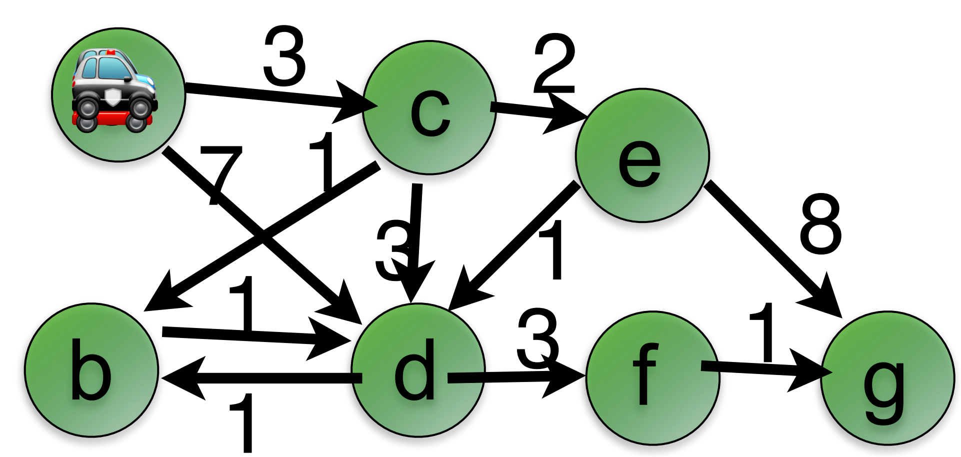
Travel from *a* to *g*

p = a, d, f, g   w(p) = 11

p’ = a, c, e, g   w(p’) = 13

p’’ = a, c, e, d, g   w(p’’) = 10

p’’’ = a, c, b, d, g   w(p’’’) = 9



$\delta(a,g) = 9$

Observation: A shortest* path doesn't have a cycle

reasons:

negative weight cycles are not well defined

$p_{a\,j}$ = a, h, i, a h i, ~~a~~ h, i, …., j $\quad$ $w(p_{a\,j}) = -\infty$

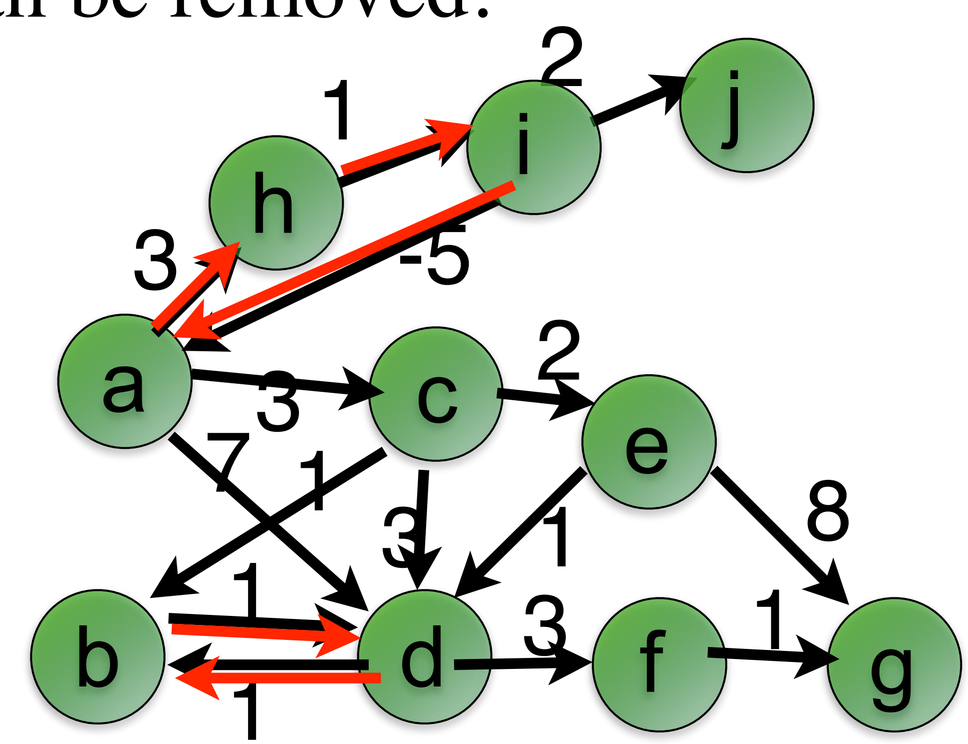
A positive weight cycle can be removed

$p_{a\,g}$ = a, c, b, d, b, d, f, g $\quad$ $w(p_{a\,g}) = 11$

$p'_{a\,g}$ = a, c, b, d, f, g $\quad$ $w(p'_{a\,g}) = 9$

*A zero weight cycle doesn't matter and can be removed.

So we assume they don't have them…



Optimal Substructure

Dynamic Programming might apply

Greedy Algorithm might apply

Lemma: Any subpath of a shortest path is a shortest path

Proof: Cut-and-paste.

Proof by contradiction, **Suppose not**

$$u \xrightarrow{p_{ux}} x \xrightarrow{p_{xy}} y \xrightarrow{p_{yv}} v$$

Suppose this path, p, is a shortest path from u to v

$$\delta(u,v) = w(p) = w(p_{ux}) + w(p_{xy}) + w(p_{yv})$$

Suppose there is a shorter path from x to y, $x \xrightarrow{p'_{xy}} y$

then $\quad w(p'_{xy}) < w(p_{xy})$

construct p': $\quad u \xrightarrow{p_{ux}} x \xrightarrow{p'_{xy}} y \xrightarrow{p_{yv}} v$

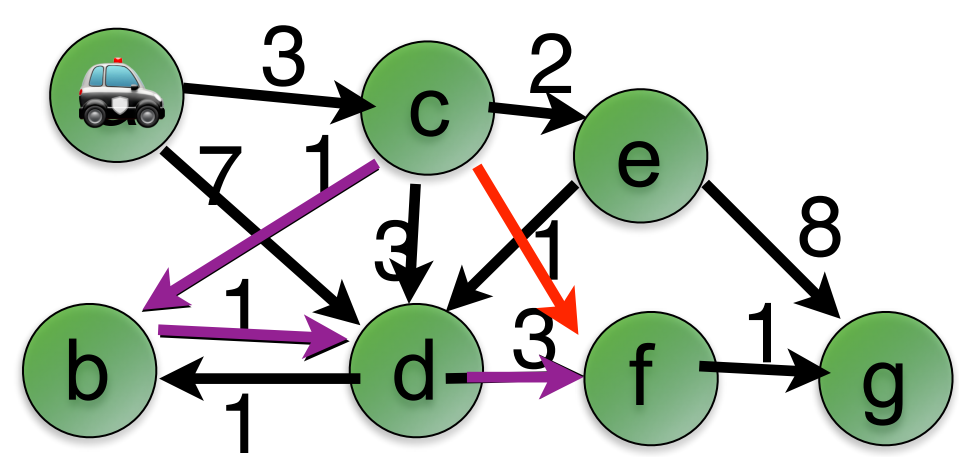
Then $\quad w(p') = w(p_{ux}) + w(p'_{xy}) + w(p_{yv})$

$$< w(p_{ux}) + w(p_{xy}) + w(p_{yv})$$

$$= w(p) \quad ◗◖$$

p''' = a, <u>c, b, d, f</u>, g $\quad$ w(p''') = 9

p''' = a, c, f, g $\quad$ w(p''') = 5



Output of single source shortest path algorithms

For every vertex, v:

$v.d$ shortest path estimate - initially ∞

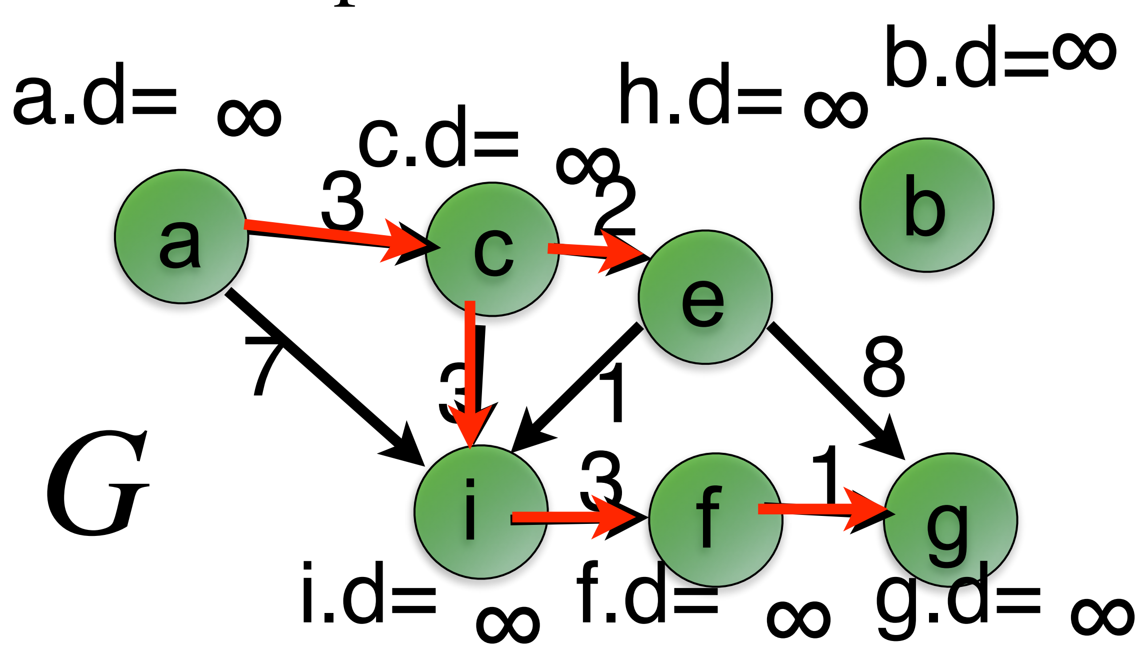
Predecssor subgraph $G_\pi = (V_\pi, E_\pi)$

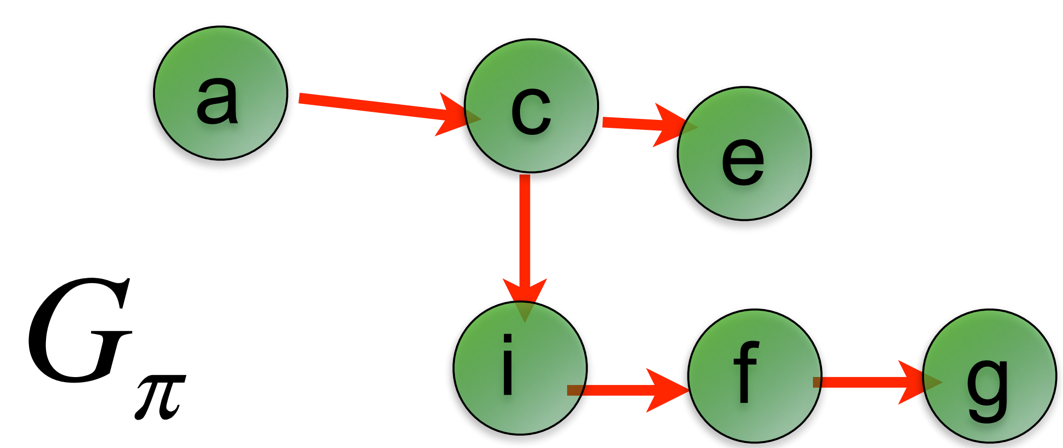
$V_\pi = \{v \in V \mid v.\pi \neq NIL\} \cup \{s\}$ $\quad E_\pi = \{(v.\pi, v) \in E \mid v \in V_\pi - \{s\}\}$

$v.\pi$ predecessor on path can change during the algorithm

Creates a tree (shortest-paths tree)

If no predecssor $v.\pi$ = NIL



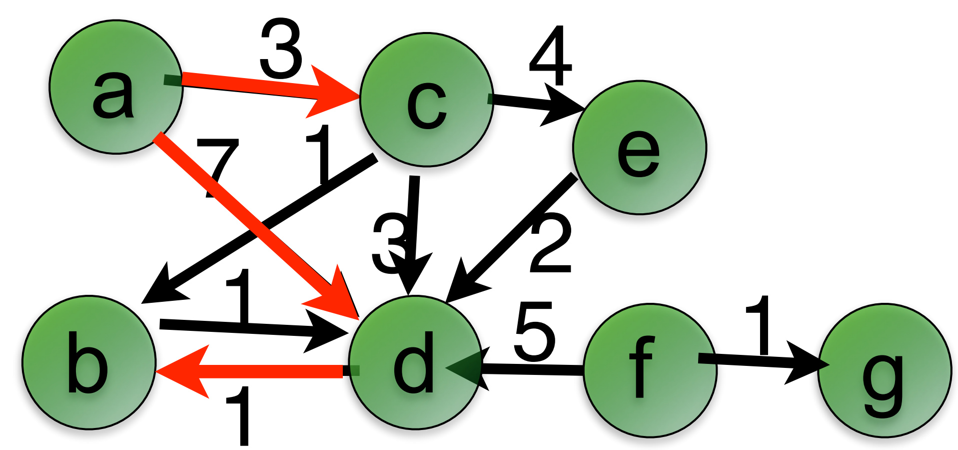


a	b	c	i	e	f	g
0	∞	3	6	5	9	10

$$E_\pi = \{(a,c),(c,e),(c,i),(e,f),(f,g)\}$$
$$V_\pi = \{a,c,i,e,f,g\}$$

Initialization

INIT-SINGLE-SOURCE(G, s)
for each $v \in G.V$
$v.d = \infty$
$v.\pi = \text{NIL}$
$s.d = 0$



a	b	c	d	e	f	g
0	7	3	6	∞	∞	∞

Relaxation of an edge

RELAX(u, v, w)
if $v.d > u.d + w(u, v)$
$v.d = u.d + w(u, v)$
$v.\pi = u$

if you have found a shorter path to v by going from s to u and u to v,, **update the distance from s to v**

RELAX(a,c,w)
RELAX(a,d,w)
RELAX(d,b,w)
RELAX(c,d,w)
RELAX(d,b,w)