Dynamic Programming

Recall the maximum common subsequence problem from last day

```
TARMAC
CATAMARAN
```

More sophisticated: count # changes from 1^{st} string to 2^{nd}

e.g. You: Pythagorus — Google: Pythagoras? — 1 change

You: recurance — Google: recurrence? — 2 changes

a change is:

- add a letter
- delete a letter

} gap

- replace a letter — mismatch

This is called <u>edit distance</u>.

This problem comes up in <u>bioinformatics</u> for DNA strings.
DNA is a sequence of chromosomes, i.e. string over
A, C, T, G

Two strings can be aligned in different ways

e.g.

```
A A C A T          A A C A T
A A   A A G        A A A A G
```

3 changes — 2 changes

delete C
change T to A
add G

Problem: Given 2 strings $x_1 \cdots x_m$ and $y_1 \cdots y_n$ compute their edit distance

i.e. find the alignment that gives min # changes.

Dynamic Programming Algorithm

Subproblem for $x_1 \cdots x_i$ and $y_1 \cdots y_j$

$M(i,j)$ = min # changes from x_i to y_j

choices: – match x_i to y_j, pay replacement cost if they differ

– match x_i to blank (delete x_i)

– match y_j to blank (add y_j)

$$M(i,j) = \min \begin{cases} M(i-1, j-1) & \text{if } x_i = y_j \\ r + M(i-1, j-1) & \text{if } x_i \neq y_j \end{cases} \text{match } x_i \text{ to } y_j$$

$$\qquad\qquad \begin{cases} d + M(i-1, j) & \text{match } x_i \text{ to blank} \\ a + M(i, j-1) & \text{match } y_j \text{ to blank} \end{cases}$$

where r = replacement cost

d = delete cost

a = add cost

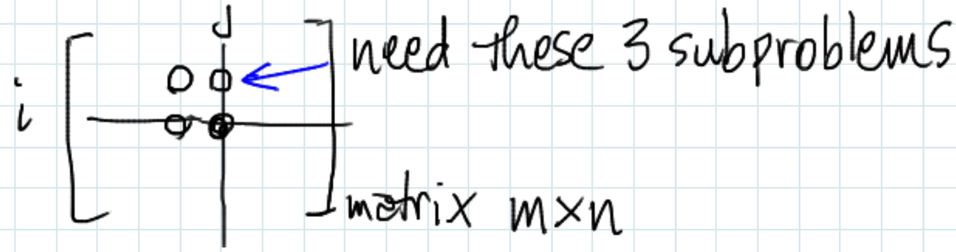
So far, we used $r = d = a = 1$

More sophisticated: $r(x_i, y_j)$ – replacement cost depends on the letters

e.g. $r(A,S) = 1$ because these keys are close on typewriter

$r(A,C) = 2$ --- not so close

In what order do we solve subproblems?
Same as last day.



$M[0 \ldots m, 0 \ldots n]$

for $i = 0 \ldots m$ $\quad M(i, 0) = i \cdot d$ $\quad$ – delete i letters

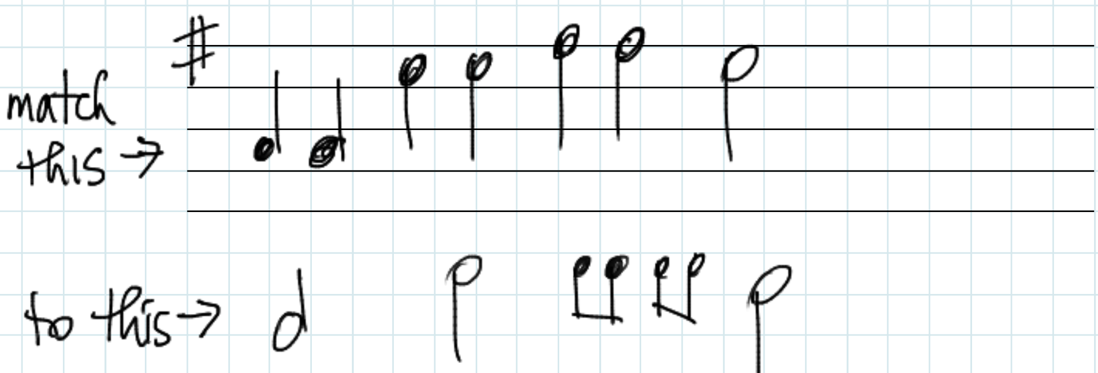
for $j = 0 \ldots n$ $\quad M(0, j) = j \cdot a$ $\quad$ – add j letters

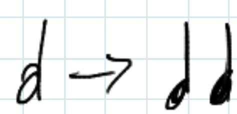
for $i = 1 \ldots m$
$\quad$ for $j = 1 \ldots n$
$\quad\quad M(i, j) = \ldots$

} fill matrix in order (or could do columns first)

Analysis $\quad O(n \cdot m)$ time $\quad$ ($n \cdot m$ subproblems, constant time each)
$\quad\quad O(n \cdot m)$ space

A different application [OPTIONAL]
Music pattern matching



Use replacement rules that allow 

Coin changing problem

Recall: given coin denominations

$c_1 \; c_2 \cdots c_k$ (e.g. penny, nickel etc)

and a value V

make change for V using min. # coins.

Greedy does not always work.

A dynamic programming algorithm:

Given V, we might try each coin $c_i \le V$

Then must make change for $V - c_i$

Subproblems $C(v)$ for each $v = 0 \,..\, V$

$C(0) = 0$

for $v = 1 \,..\, V$

(*) $C(v) = \min \{ 1 + M(v - c_i) : \underset{c_i \le V}{i = 1 \,..\, k} \}$

end

(*) in more detail:

$C(v) \leftarrow \infty$

for $i = 1 \,..\, k$

if $c_i \le v$ and $(1 + C(v - c_i) < C(v))$

then $C(v) \leftarrow 1 + C(v - c_i)$

end

Run Time $O(\underbrace{V}_{\text{\# subproblems}} \cdot \underbrace{k}_{\text{time for each}})$

An algorithm runs in polynomial time if run-time is $O(n^c)$, c a constant on input of size n.

The above algorithm is NOT polynomial time. because size of V is $\log V$ but run time depends on V (not $\log V$).

This is called a pseudo-polynomial time algorithm.

More on these ideas later in the course

Constructing optimum binary search trees

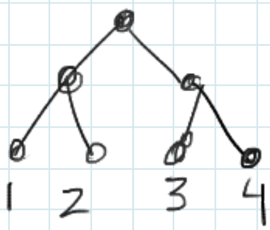
Given items $1 \cdots n$

probabilities $p_1 \cdots p_n$

Construct a binary search tree (items in leaves)

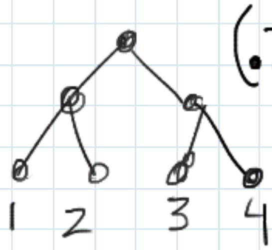
to minimize search cost $\sum_i p_i \underbrace{\text{depth}(i)}_{\text{\# probes into tree to find item } i.}$

e.g. $p_1 = \cdots = p_4 = \frac{1}{4}$

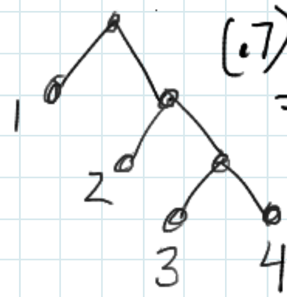


search cost $= 4 \cdot \frac{1}{4} \cdot 3 = 3$

$p_1 = .7 \quad p_2 = p_3 = p_4 = .1$



$(.7)3 + 3(.1)3$
$= 2.1 + .9$
$= 3$



$(.7)2 + (.1)3 + 2(.1)4$
$= 1.4 + .3 + .8$
$= 2.5$

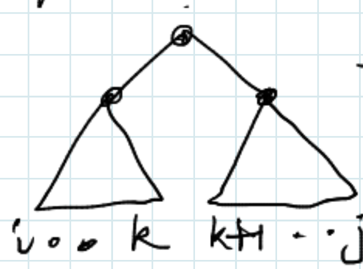
[In case you've seen optimum Huffman trees, this is different in that leaf ordering is fixed].

To apply dynamic programming:

subproblems: opt. binary search tree for items $i \cdots j$

order subproblems by # items, i.e. by $j - i$

to solve $i \cdots j$



try all choices for k

Details

$$M[i,j] = \min_{k=i..j-1} \{ M[i,k] + M[k+1,j] \} + \underbrace{\sum_{t=i}^{j} p_t}_{\text{because every node gets 1 deeper}}$$

ind. of choice of k

How to compute $\sum_{t=i}^{j} p_t$

First compute $P[i] = \sum_{j=1}^{i} p_j$

then we can get $\sum_{t=i}^{j} p_t$ as $P[j] - P[i-1]$

```
for i = 1 .. n    M[i,i] ← p_i
for r = 1 .. n-1
   for i = 1 .. n-r
      /* solve for M[i, i+r]
      best ← M[i,i] + M[i+1, i+r]
      for k = i+1 .. i+r-1          ← this was wrong in class, sorry.
         temp ← M[i,k] + M[k+1, i+r]
         if temp < best then best ← temp
      end
      M[i, i+r] ← best + P[i+r] - P[i-1]
   end
end.
```

Run time $O(\underbrace{n^2}_{\#\text{subproblems}} \cdot \underbrace{n}_{\text{time per subproblem}}) = O(n^2)$