CS314 - More dynamic programming

Note Title 9/12/2013

Announcements

- Turn in HW1
- HW2 - posted tonight

Edit distance

The minimum number of deletions, insertions, and substitutions of letters to transform between two strings.

Ex:

$$\text{FOOD} \xrightarrow{+1} \text{MOOD}$$

$$\text{MOOD} \xrightarrow{+1} \text{MOED}$$

$$\text{MONED} \xleftarrow{+1} \text{MOED}$$

$$\text{MONED} \xrightarrow{+1} \text{MONEY}$$

$$\leq 4$$

First: why do we care?

Spell checking

Auto correcting

Strings in gameshows

DNA - analysis

Second: any ideas?

2D matrix to math

Column format:

inserted letters

A	L	G	O	R		I		T	H	M
A	L		T	R	U	I	S	T	I	C

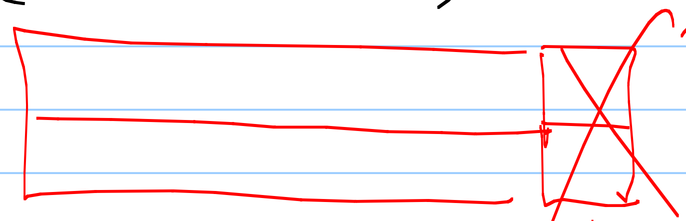
deleted letters

So edit distance is $\leq$ 6.

Nice property: If you delete the last column of the ~~previous~~ column representation, must still be optimal.

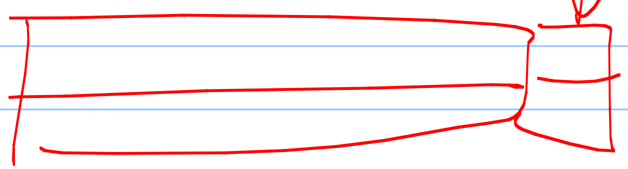
Pf: (contradiction) Assume not

edit dist k



If better one in first n-1, use it

k-1



contradiction - above wasn't correct edit distance.

So - recursive formulation:

Consider $A[1..m]$ and $B[1..n]$.

What are the 3 possibilities for just the last character?

(1) $A[m] \downarrow$ _

(2) _ $\downarrow B[n]$

(3) $A[m] \downarrow B[n]$

So – more formally:

Let Edit(A, B) = the edit distance between A[1..m] and B[1..n].

deletion

$$Edit(A[1..m], B[1..n]) = \min \left\{ \begin{array}{l} Edit(A[1..m-1], B[1..n]) + \underline{1} \\ Edit(A[1..m], B[1..n-1]) + \underline{1} \quad \text{Insertion} \\ Edit(A[1..m-1], B[1..n-1]) + [A[m] \neq B[n]] \end{array} \right\}$$

A

A[m]

↓

n

A[m]

↓

B[n]

G G+1

↓ ↓

G H

Base cases?

empty

n insertions

n

m

m deletions

(empty)

empty string

$$\text{Edit}(\varepsilon, B[1 \ldots n]) = n$$

$$\text{Edit}(A[1 \ldots m], \varepsilon) = m$$

✓

Now again, we have something like LIS — the recursive calls are always on prefixes!

So if we can somehow start with the "front" & move towards "end", we could get the value for Edit$(A[1 \ldots n], B[1, \ldots, n])$.

Let's try to simplify & formalize this idea...

First, Edit(A[1...i], B[1..j]) is too long.

Shorten: Edit(i, j). (since always a prefix)!

Recurrence (rewritten):

$$
Edit(i,j) = \begin{cases} i & \text{if } j = 0 \\ j & \text{if } i = 0 \\ \min \left\{ \begin{array}{l} Edit(i-1, j) + 1, \\ Edit(i, j-1) + 1, \\ Edit(i-1, j-1) + [A[i] \neq B[j]] \end{array} \right\} & \text{otherwise} \end{cases}
$$

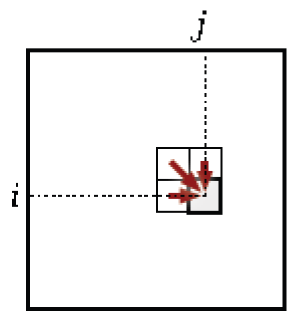
This does give a recursive algorithm.

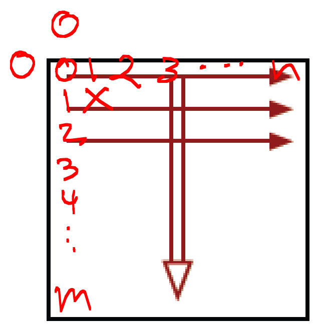
Running time - probably ugly.

<u>But</u> - dynamic programming seems like an option!

The table :

$$
Edit(i,j) = \begin{cases} i & \text{if } j = 0 \\ j & \text{if } i = 0 \\ \min \left\{ \begin{array}{l} Edit(i-1, j) + 1, \\ Edit(i, j-1) + 1, \\ Edit(i-1, j-1) + [A[i] \neq B[j]] \end{array} \right\} & \text{otherwise} \end{cases}
$$





Example:

Edit distance between algorithm and altruistic.

Note: Any path from top right to bottom left is an optimal set of substitutions.

		A	L	G	O	R	I	T	H	M
	0	1	2	3	4	5	6	7	8	9
A	1	**0**	1	2	3	4	5	6	7	8
L	2	1	**0**	1	2	3	4	5	6	7
T	3	2	1	1	2	3	4	**4**	5	6
R	4	3	2	2	2	**2**	3	4	5	6
U	5	4	3	3	3	3	3	4	5	6
I	6	5	4	4	4	4	**3**	4	5	6
S	7	6	5	5	5	5	4	4	5	6
T	8	7	6	6	6	6	5	**4**	5	6
I	9	8	7	7	7	7	**6**	5	5	6
C	10	9	8	8	8	8	7	6	6	6

Pseudo code:

$$\sum_{i=1}^{m} \sum_{j=1}^{n} 1$$

$\underline{\textsc{EditDistance}(A[1..m], B[1..n]):}$

for $j \leftarrow 1$ to n
    $Edit[0,j] \leftarrow j$

for $i \leftarrow 1$ to m
    $Edit[i,0] \leftarrow i$
    for $j \leftarrow 1$ to n
        if $A[i] = B[j]$
            $Edit[i,j] \leftarrow \min\{Edit[i-1,j]+1,\ Edit[i,j-1]+1,\ Edit[i-1,j-1]\}$
        else
            $Edit[i,j] \leftarrow \min\{Edit[i-1,j]+1,\ Edit[i,j-1]+1,\ Edit[i-1,j-1]+1\}$

$O(1)$

return $Edit[m,n]$

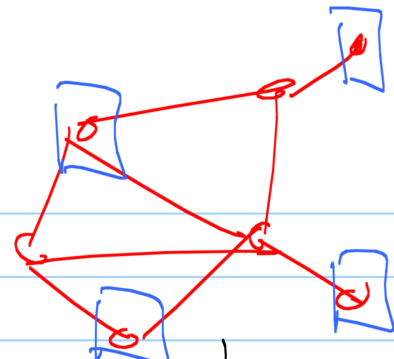
Running time: (+ space)

$O(n^2)$ space

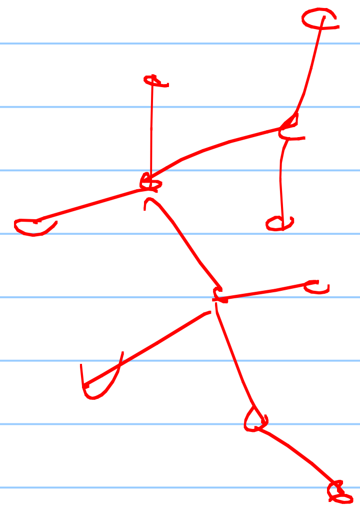
$\hookrightarrow O(nm)$

same time

Dynamic Programming on Trees



Dfn: An independent set in a graph is a subset that have no edges between them.



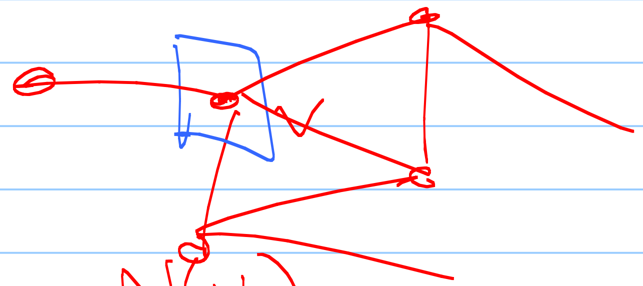
First - why?

Graphs model everything.

Recursion

Goal: Compute largest indep. set.

Consider a node v.
The options?



- include v

  recurse on $G - N(v)$

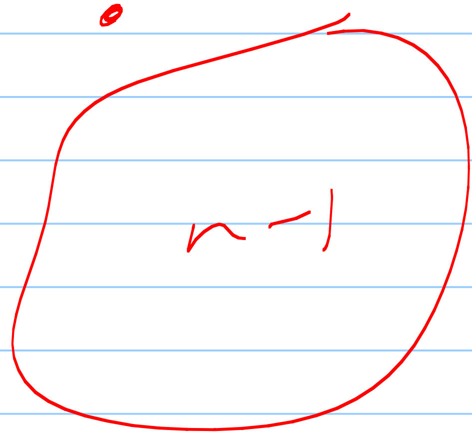
- not include v

  recurse on $G - v$

MaximumIndSetSize(G):
  if $G = \varnothing$
    return 0
  else
    $v \leftarrow$ any node in G
    $withv \leftarrow 1 +$ MaximumIndSetSize($G \setminus N(v)$)
    $withoutv \leftarrow$ MaximumIndSetSize($G \setminus \{v\}$)
    return $\max\{withv, withoutv\}$.

$O(1)$

$O(1)$



Runtime:

$$T(n) = 2T(n-1) + \text{poly}(n)$$

depends on d.s.

$$= O(\text{poly}(n)\, 2^n)$$

Aside: Can actually do a bit better.

How big will those recursive calls be?

(Continued next time...)