Dynamic Programming

Edit distance and its variants

Tyler Moore

CSE 3353, SMU, Dallas, TX

Lecture 17

Some slides created by or adapted from Dr. Kevin Wayne. For more information see `http://www.cs.princeton.edu/~wayne/kleinberg-tardos`. Some code reused from Python Algorithms by Magnus Lie Hetland.

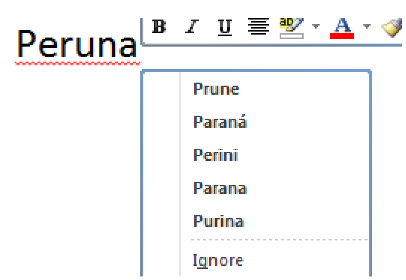
Edit distance

- Misspellings make `approximate pattern matching` an important problem
- If we are to deal with inexact string matching, we must first define a cost function telling us how far apart two strings are, i.e., a distance measure between pairs of strings.
- The edit distance is the minimum number of changes required to convert one string into another

String edit operations

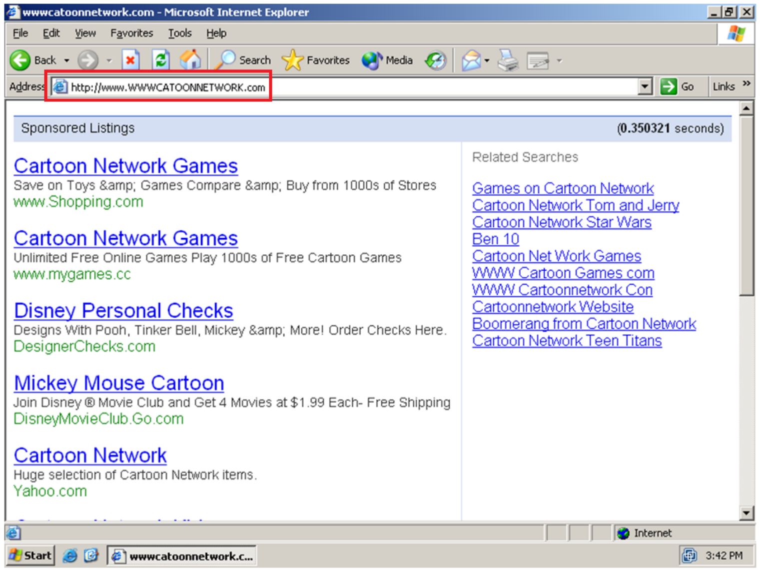
- We consider three types of changes to compute edit distance:
  1. Substitution: Change a single character from pattern s to a different character in text t, such as changing “shot” to “spot”
  2. Insertion: Insert a single character into pattern s to help it match text t, such as changing “ago” to “agog”.
  3. Deletion: Delete a single character from pattern s to help it match text t, such as changing “hour” to “our”
- This definition of edit distance is also called Levenshtein distance
- Can you think of any other natural changes that might capture a single misspelling?

Edit distance application #1



- Spell checkers identify words in a dictionary with close edit distance to the misspelled word
- But how do they order the list of suggestions?

Edit distance application #2



1,278 cartoonnetwork.com typos, including. . .

fartoonnetwork.com	cagtoonnetwork.com	cartlonnetwork.com	cartoonnestwork.com	cartoonnewotk.com
cartoonnetsork.com	cartoinnetwork.com	cartolnnetwork.com	cartoonnftwork.com	cartoonneywork.com
cartoonntewrk.com	cartoonnetlork.com	cartoonnetowok.com	crtonnetwork.com	cartoonnegwork.com
cargoonnetwork.com	carttoonnetwork.com	cartoonnwetwork.com	cartoonetrwork.com	cartoonnetwodrk.com
cartoonnetwkor.com	catoonnnetwork.com	cartoooonnetwork.com	caoonnetwork.com	cartonbetwork.com
cartoonetgork.com	cartoonnetqork.com	cartoonneetwort.com	cartoonneetwork.com	catoonnetwrok.com
cartoomnetwoork.com	caryoonetwork.com	cartooonetwork.com	cantoonnetwork.com	cargoonnetworm.com
caretoonetwork.com	cartoonetwoork.com	cartoonnetwoer.com	carttoonnetwerk.com	chartoonnetwork.com
cartoonnetwokr.com	cartoonnetwokl.com	cartoonnetwoke.com	cartoownnetwork.com	cartoobetwork.com
cartoonnetworkcom.com	nartoonnetwork.com	cartoonnstwork.com	cartoounnetwork.com	cartoonework.com
carfoonnetwork.com	cartoonnotwork.com	cartoonnnetwok.com	cartoonnnetwor.com	cartonnetwortk.com
cartoopnetwork.com	cartoonnetwogk.com	cartoonetwaork.com	cartoonntewrok.com	cartoonetwoek.com
caqrtoonetwork.com	cartoonneework.com	cartppnnetwork.com	cartoonnetmwork.com	cartooonework.com
cartoonntwoork.com	catoonneetwork.com	crattoonnetwork.com	cartoonnetweark.com	carttooonetwork.com
cartoonetwoirk.com	cartoonznetwork.com	cartoobnetwork.com	catoonnework.com	cartiinnetwork.com
cartoonnnetwrk.com	cartoommetwork.com	cartoonnetwart.com	wwwcartonetwork.com	cartoonnttwork.com
cartoonhetwork.com	fcartoonnetwork.com	catoonnetwerk.com	artoonnetwor.com	cartoonnetwock.com
cartoonnetook.com	cartoonnetkwork.com	cartonnetwokr.com	carltonnetwork.com	cartoonetowrk.com
catoonnettwork.com	cartoo0nnetwork.com	cacrtoonnetwork.com	cartoonnetwoorkl.com	cartoonedtwork.com
cartoonnetwcrk.com	cartoonetwrk.com	cartoonnewark.com	cartoonnetwoirk.com	cartoknnetwork.com
cartooonnetwrk.com	cartoonnetbwork.com	caetooonetwork.com	cartoonknetwork.com	catoomnetwork.com
cartoonnexwork.com	carooonnetwork.com	dartoonnetwork.com	certoonnetwork.com	cartoonetword.com
cartoonetworg.com	cartoonetworl.com	cartoonetworj.com	cartoonetwork.com	cartoonetwort.com
crattonnetwork.com	cartoonnewtokr.com	carntoonnetwork.com	caretoonnetwork.com	cartooonnetwoork.com
cartoonnerwort.com	cartoonnerwork.com	cartoonnerworl.com	cartoonnetfork.com	cartoonnetttwork.com
cartoonnetwar.com	cartoonnetwak.com	cartoonnekwork.com	cartooknetwork.com	cartoonegwork.com
cattoonnetwok.com	cartoonnetwwork.com	cartoonnetgor.com	cartoonnetwowk.com	wwwcatoonetwork.com
cartoolnnetwork.com	cartoonetworkcom.com	casrtoonetwork.com	cartoonnetswork.com	cartoonnedwort.com
cartoonnedword.com	cartoonnedwork.com	wwwcarttoonnetwork.com	cartoonerwork.com	cattoonnetwark.com
carttoonnetwook.com	cartoonnetwowrk.com	cartoonetwqork.com	crartoonnetwork.com	czrtoonnetwork.com
cartomnetwork.com	cartoonnetwrak.com	cartoonnetorg.com	cratonnetwork.com	crtoonnework.com
cartioonnetwork.com	cartoonnetvork.com	catoonnetwort.com	cartoonnetwold.com	cartoonnetwolk.com
cartoonsnetwork.com	wwwcartoonetwerk.com	carttoonntwork.com	cartownnetwork.com	carthonnetwork.com
wwwcartoonnnetwork.com	caatoonnetwork.com	caetonnetwork.com	cartcoonnetwork.com	cartooanetwork.com
caartoonnetwor.com	cartoonnntwork.com	cartoonnetw2ork.com	cartoonnaetwork.com	cartoonne6work.com
dcartoonnetwork.com	cartoonnerwok.com	cartonneywork.com	hcartoonnetwork.com	artoonetwork.com
cartoonnetwoyk.com	cartoonnetworek.com	cartoonnetwo5k.com	carttonnetwoork.com	cartoonnettwork.com
caqrtoonnetwork.com	cartoonvetwork.com	cartoometwork.com	cartooetwork.com	cartoonnetwwwork.com
cartoonnetwokrk.com	cartoonnektwork.com	cartoonetwiork.com	cartoonetwirk.com	carttoonetwork.com
wwwcaroonnetwork.com	cartoonnetwood.com	cartoonnetwook.com	cartoonnetwoot.com	cartoonnetwoor.com

Edit distance: recursive algorithm design

- Match: no substitutions

$$\overbrace{\text{shoe}}^{s_{i-1}}\ \text{s}$$
$$\underbrace{\text{show}}_{t_{j-1}}\ \underbrace{\text{s}}_{0}$$
$$(d(s_{i-1}, t_{j-1}) = 1) + 0$$
$$d(s_i, t_j) = 1$$

- Match: substitution

$$\overbrace{\text{shoe}}^{s_{i-1}}\ \text{s}$$
$$\underbrace{\text{show}}_{t_{j-1}}\ \underbrace{\text{n}}_{1}$$
$$(d(s_{i-1}, t_{j-1}) = 1) + 1$$
$$d(s_i, t_j) = 2$$

- Insertion

$$\overbrace{\text{show}}^{s_i}$$
$$\underbrace{\text{show}}_{t_{j-1}}\ \underbrace{\text{n}}_{1}$$
$$(d(s_i, t_{j-1}) = 0) + 1$$
$$d(s_i, t_j) = 1$$

- Deletion

$$\overbrace{\text{shoo}}^{s_{i-1}}\ \text{k}$$
$$\underbrace{\text{show}}_{t_j}\ \underbrace{}_{1}$$
$$(d(s_{i-1}, t_j) = 1) + 1$$
$$d(s_i, t_j) = 2$$

Recursive edit distance code

```
def string_compare(s,t):
    #start by prepending empty character to check 1st char
    s=" "+s
    t=" "+t
    P={}
    @memo
    def edit_dist(i,j):
        if i==0: return j
        if j==0: return i
        #case 1: check for match at i and j
        if s[i]==t[j]: c_match = edit_dist(i-1,j-1)
        else: c_match = edit_dist(i-1,j-1)+1
        #case 2: there is an extra character to insert
        c_ins = edit_dist(i,j-1)+1
        #case 3: there is an extra character to remove
        c_del = edit_dist(i-1,j)+1
        return min(c_match,c_ins,c_del)
    return edit_dist(len(s)-1,len(t)-1)
```

Towards a dynamic programming alternative

- We note that there are only $|s|$ possible values for `i` and $|t|$ possible values for `j` when invoking `edit_dist(i,j)` recursively
- This means there are at most $|s| \cdot |t|$ recursive function calls to cache in an iterative version
- The table is a two-dimensional matrix `C` where each of the $|s| \cdot |t|$ cells contains the cost of the optimal solution of this subproblem
- We just need a clever way to calculate the cost for each entry based on only a small subset of already-computed values.

Evaluation order

- To determine the value of cell (i, j) we need three values to already be computed: the cells $(i-1, j-1)$, $(i, j-1)$, and $(i-1, j)$.
- Any evaluation order with this property will do, including the row-major order used in the upcoming code
- But there are plenty of other valid orderings
- Think of the cells as vertices, where there is an edge (i, j) if cell is value is needed to compute cell j. Any topological sort of this DAG provides a proper evaluation order.

Edit distance: dynamic programming code

```
def iter_string_compare_lists(s,t):
    C,s,t =[]," "+s," "+t #prepend empty character for edge case
    C.append(range(len(t)+1)) #initialize cost data structure
    for i in range(len(s)):
        C.append([i+1])
    for i in range(1,len(s)): #go through all characters of s
        for j in range(1,len(t)):
            #case 1: check for match at i and j
            if s[i]==t[j]: c_match = C[i-1][j-1]
            else: c_match = C[i-1][j-1]+1
            #case 2: there is an extra character to insert
            c_ins = C[i][j-1]+1
            #case 3: there is an extra character to remove
            c_del = C[i-1][j]+1
            c_min=min(c_match,c_ins,c_del)
            C[i].append(c_min)
    return C[i][j]
```


Edit distance: DP with cost table as dictionary

```
def iter_string_compare(s,t):
    C,s,t ={}," "+s," "+t #prepend empty character for edge case
    for j in range(len(t)): #initialize cost data structure
        C[0,j]=j
    for i in range(1,len(s)):
        C[i,0]=i
    for i in range(1,len(s)): #go through all chars of s
        for j in range(1,len(t)):
            #case 1: check for match at i and j
            if s[i]==t[j]: c_match = C[i-1,j-1]
            else: c_match = C[i-1,j-1]+1
            #case 2: there is an extra character to insert
            c_ins = C[i,j-1]+1
            #case 3: there is an extra character to remove
            c_del = C[i-1,j]+1
            c_min=min(c_match,c_ins,c_del)
            C[i,j]=c_min
    return C[i,j]
```

Building edit distance cache

s: run
t: drain

C	_	d	r	a	i	n
_	0	← 1 ← 1	← 2	← 3	← 4	← 5
r	↑ 1	↖ 1	↖ 1 ↖ 1	← 2	← 3	← 4
u	↑ 2	↖ 2	↖ 2	↖ 2 ↖ 2	← 3 ← 3	← 4
n	↑ 3	↖ 3	↖ 3	↖ 3	↖ 3	↖ 3 ↖ 3

Steps to turn "run" into "drain"

1. Insert d
2. Keep r
3. Substitute a for u
4. Insert i
5. Keep n

Edit distance exercises

- Build cost table by hand following DP algorithm
  1. s: bear, t: pea
  2. s: farm, t: for
- Performance cost of DP edit distance
  - Operations: $\Theta(|s| \cdot |t|)$
  - Storage: $\Theta(|s| \cdot |t|)$

Variation of edit distance: approximate substring matching

- Suppose we want to find the best close match to a smaller word in a larger string (e.g., find the closest match to "Tulsa" in "SMU Tulda Rice")
- We need to modify our existing code in two ways
  1. Cost table initialization: all starting costs C[0,j] should be set to 0
  2. Return the finishing cell C[i,k] that minimizes the overall cost

Substring matching code

```
def iter_substring_match(s,t):
    C,s,t ={}," "+s," "+t #prepend empty character for edge case
    for j in range(len(t)): #initialize cost data structure
        C[0,j]=0 #changed: ignore cost of preceding unmatched text
    for i in range(1,len(s)):
        C[i,0]=i
    for i in range(1,len(s)): #go through all chars of s
        for j in range(1,len(t)):
            #case 1: check for match at i and j
            if s[i]==t[j]: c_match = C[i-1,j-1]
            else: c_match = C[i-1,j-1]+1
            #case 2: there is an extra character to insert
            c_ins = C[i,j-1]+1
            #case 3: there is an extra character to remove
            c_del = C[i-1,j]+1
            c_min=min(c_match,c_ins,c_del)
            C[i,j]=c_min
    finj = min([(C[i,k],k) for k in range(1,len(t)-1)])
    return "with edit dist %i, %s morphs into %s finishing at position %
```

Excercise: substring matching cache

s: Tulsa
t: SMU Tulda Rice

C	␣	S	M	U	␣	T	u	l	d	a	␣	R	i	c	e
␣	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
T	↑1	↑1	↑1	↑1	↑1	↑0	↑1	↑1	↑1	↑1	↑1	↑1	↑1	↑1	↑1
u	↑2	↑2	↑2	↑2	↑2	↑1	↖0	←1	↑2	↑2	↑2	↑2	↑2	↑2	↑2
l	↑3	↑3	↑3	↑3	↑3	↑2	↑1	↖0	←1	←2	←3	↑3	↑3	↑3	↑3
s	↑4	↑4	↑4	↑4	↑4	↑3	↑2	↑1	↖1	←2	←3	↑4	↑4	↑4	↑4
a	↑5	↑5	↑5	↑5	↑5	↑4	↑3	↑2	↖2	↖1	←2	←3	←4	↑5	↑5

Substring ending at position 9 (“Tulda”) is the closest substring to “Tulsa”

Variation of edit distance: longest common subsequence

- We might want to find the longest scattered sequence of characters within both strings
- For example, the longest common subsequence of “republican” and “democrat” is “eca”
- To get the longest subsequence, we can still allow insertions and deletions, but substitutions are forbidden
- We can change the edit distance code to behave as before on matches where the last characters are the same, but never select a substitution