Lindsey Bieda and Joe Frambach
Dynamic Programming Problems
9.20.2011

5. The input to this problem is a pair of strings $A = a_1 \dots a_m$ and $B = b_1 \dots b_n$. The goal is to convert A into B as cheaply as possible. The rules are as follows. For a cost of 3 you can delete any letter. For a cost of 4 you can insert a letter in any position. For a cost of 5 you can replace any letter by any other letter. For example, you can convert $A = abcabc$ to $B = abacab$ via the following sequence: *abcabc* at a cost of 5 can be converted to *abaabc*, which at cost of 3 can be converted to *ababc*, which at cost of 3 can be converted to *abac*, which at cost of 4 can be converted to *abacb*, which at cost of 4 can be converted to *abacab*. Thus the total cost for this conversion would be 19. This is almost surely not the cheapest possible conversion.

The following is a modified Levenshtein distance algorithm that takes into account the different weights of each of the operations. As such when the initalization takes place the value for comparing against the empty string needs to be multiplied by 4 to take into account all of the insert operations, or 3 to take into account deletes.

```
cost[][]

for i = 0 to len(A): cost[i,0] = i*4
for i = 0 to len(B): cost[0,i] = i*3

for i = 1 to len(A):
  for j = 1 to len(B):
    if A[i] = B[j]:
      cost[i,j] = cost[i-1,j-1]
    else:
      cost[i,j] = min( cost[i-1,j-1] + 5  # sub
                       cost[i-1, j]  + 3  # del
                       cost[i, j-1]  + 4  # ins
                     )

return cost[len(A), len(B)]
```

Using the above we determine that the minimal cost for converting $A = abcabc$ to $B = abacab$ is 7, the deletion of the final "C" and insertion of "A" after position 2. We build the following table:

	ϵ	a	b	c	a	b	c
ϵ	0	4	8	12	16	20	24
a	3	0	4	8	12	16	20
b	6	3	0	4	8	12	15
a	9	6	3	5	4	13	17
c	12	9	6	3	7	9	13
a	15	12	9	6	3	7	11
b	18	15	12	9	11	3	7

To trackback, start at the $i = len(A), j = len(B)$ position in the table. Procede to the minimum of $[i-1, j-1], [i-1, j], [i, j-1]$, if the minimum is to the left there is a delete operation at position i. If the minimum is above there is a insertion operation at position i. If the minimum is the diagonal (and not equal to i, j's value) there was a subsitution at position i. Otherwise, no operation was performed at this position in the string.

6. Find the optimal binary search tree for keys $K_1 < K_2 < K_3 < K_4 < K_5$ where the access probabilities/weights are .5, .05, .1, .2, .25 respectively using the algorithm discussed in class and in the notes. Construct one table showing the optimal expected access time for all subtrees considered in the algorithm, and another showing the roots of the optimal subtrees computed in the other table. Show how to use the table of roots to recompute the tree.

Optimal expected access time for all subtrees =

	1	2	3	4	5
1	0.05	0.2	0.55	1.05	2.15
2	0	0.1	0.4	0.9	1.95
3	0	0	0.2	0.65	1.6
4	0	0	0	0.25	1
5	0	0	0	0	0.5

Roots of the optimal subtrees =

	1	2	3	4	5
1	1	1	3	3	5
2		2	3	3	4
3			3	4	5
4				4	5
5					5

Beginning at point (1,5) in the table, where l=1 and r=5, in the roots table we determine the root to be k_5, where $k = 5$, the left child is recursively defined by repeating this procedure at (l, k-1) and the right child by (k+1, r). Using this method we get the following tree: