Calculating minimum edit cost

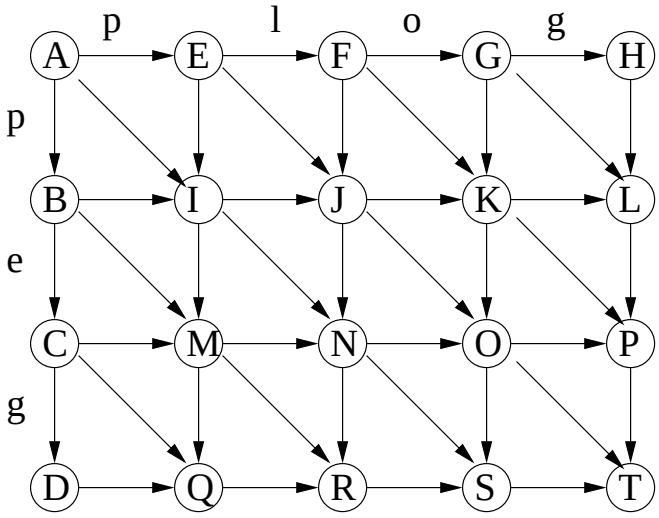
Adapted from chapter 8 of *English Spelling and the Computer* by Roger Mitton

1 Directed networks

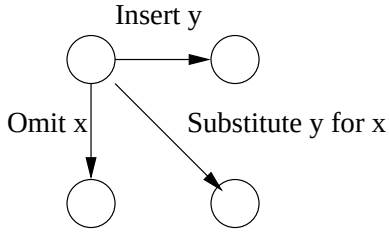
To calculate minimum edit distance, we set up a **directed network**, a set of nodes (circles) and arcs (arrows).

For instance, let's say the user types in *plog*, and we want to calculate how far away *peg* is (i.e. we want to calculate the minimum edit distance, or the minimum edit cost)

We set up the following directed graph:



First off, what do these arcs mean? As shown below, horizontal arcs correspond to insertions, vertical arcs correspond to deletions, and diagonal arcs correspond to substitutions (and a letter can be "substituted" for itself). [Note: we are not dealing here with transpositions.]



So, the route A-I-J-O-T corresponds to substituting p for p (A-I), inserting l (I-J), substituting o for e (J-O), and substituting g for g.

Questions:

1. What does A-B-M-R-S-T correspond to?
2. What route corresponds to deleting everything followed by inserting everything?

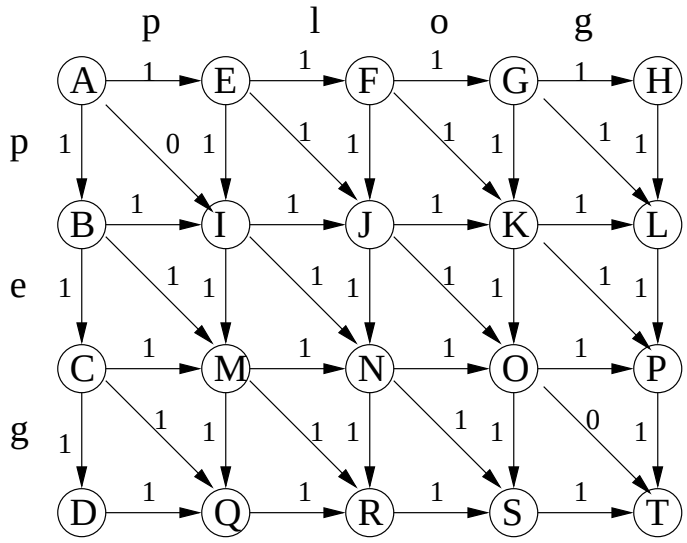
2 Graph properties

Note two things about this graph:

- It is **acyclic** = for any given node (circle), it is impossible to return to that node by following the arcs (arrows)
- It is **topologically ordered** = it can be ordered in some way. Here, it is alphabetically ordered.

  So, e.g. node I comes after nodes A, B, and E, so order is maintained. If D was there instead, order would not be maintained because D cannot follow E.

Because of these properties, we can calculate the minimum edit distance. We assign a cost, or weight, of zero (0) for every time we see a letter substituted as itself; otherwise, we give an arc a cost of one (1).



Question:

1. Draw a directed graph for the following situation:

   The user types in *hon* and you want to compare it with *hint*.

3 Calculating minimum edit distance

Now, we want to find the path from the start (A) to the end (T) with the least cost.

3.1 The slow way

- Follow every path from start (A) to finish (T) and see how many changes we have to make.
- But this is very inefficient! There are 131 different paths to check.

3.2 The faster, better way (dynamic programming)

- We follow the topological ordering, i.e. we go in alphabetical order.
- As we go in order, we calculate the least cost for that node.

  That is, of all the arcs coming in to the node, we take the least-cost one.
- We store every result, so that we know the cost of all incoming arcs already.

  $\Rightarrow$ This is the key point: we are storing partial results along the way, instead of recalculating every time we encounter a new path.

So, for example, we have the following:

- Node A: cost = 0
- Node B: cost = 1 (cost of the arc from A to B)
- Node C: cost = 1 + cost of B = 2
- Node D: cost = 1 + cost of C = 3
- Node E: cost = 1 (cost of the arc from A to E)
- Node F: cost = 1 + cost of E = 2
- Node G: cost = 1 + cost of F = 3
- Node H: cost = 1 + cost of G = 4
- Node I: cost = lowest cost of the following:

  1. cost = 0 + cost of A = 0
  2. cost = 1 + cost of B = 2
  3. cost = 1 + cost of E = 2

  So, the cost of I = 0
- Node J: cost = lowest cost of the following:

  1. cost = 1 + cost of E = 2

2. cost = 1 + cost of F = 3
3. cost = 1 + cost of I = 1

So, the cost of J = 1

- ...
- And when you get to node T (the end), you'll find that the least cost there is 2. (from either of two routes [A-I-J-O-T and A-I-N-O-T], but it doesn't really matter which, since our goal is to compare costs).

So, now we only have 20 calculations to make.

Questions:

1. What would happen if we didn't go in order?
2. What is the pattern used here for ordering nodes?
3. Work through the graph you drew for *hon*/*hint* and find the least-cost path.

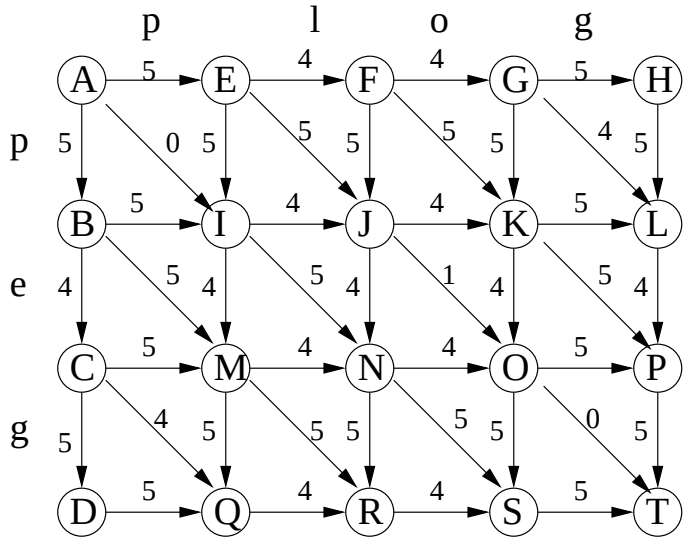
4 Adding weights

But there might be lots of words which are 2 units away from *plog*. How do we know which is the best choice?

⇒ Assign weights based on previously-seen data (perhaps a confusion matrix).

e.g., substituting *o* for *e* (J-O) is probably a better substitution than *l* for *e* (I-N).

The following graph reflects that fact by assigning weights up to 5:



Now, A-I-J-O-T will be the best path. We call this the **minimum edit cost** instead of the minimum edit distance because two words might be the same "distance" away from a typed-in string, but might have different costs.