Gorilla Report

afin, alol, amand, anok, jave, jehj & thss

September 26, 2020

Results

Our solution produces the expected results. By printing the resulting array after finding costs, we can verify the results of the cost matrix by doing examples by hand, which we have done. By looking at the output of our pathfinding algorithm for finding a optimal alignment, we can manually verify that the pathfinding chooses the optimal route through the matrix. Attached is the resulting cost matrix for the strings KQRK and KAK.

```
Matrix after cost:
[[ 0 -4 -8 -12] [-4 5 1 -3] [-8 1 4 2] [-12 -3 0 6]
[-16 -7 -4 5]]

Path chosen:
5 -> 6 -> 4 -> 5
Sum: 20
```

This path gives us a gap of 1 and a mismatch of one. The resulting string is not identical to the out-file, but has the same amount of gaps and mismatches, and is therefore equally good.

Because of this we have not been able to verify the results using a diff script, but when manually comparing our output to the expected outcome in `HbB_FASTAs-out` the results appear to be in order.

We compared the species in `HbB_FASTAs-in.txt` with the common rat, given by

```
MVHLTDAEKA AVNALWGKVN PDDVGGEALG RLLVVYPWTQ RYFDSFGDLS
SASAIMGNPK VKAHGKKVIN AFNDGLKHLD NLKGTFAHLS ELHCDKLHVD
PENFRLLGNM IVIVLGHHLG KEFTPCAQAA FQKVVAGVAS ALAHKYH
```

The closest species to *Rattus rattus* is [. . .], with the following optimal alignment:[1]

[1] Replace with actual alignment between the two species.

```
MVHLTDAEKAAVNALWGKVNPDDV-YGGEAL----VVYPWTQRYFDSFGDLS
MVHLTDAEKAAVNALWGKVNPDDVX-GGEALGRLLVVYPWTIRYFDSFGDLS

SASAIMGNPKVKAHGKKVINAFND---KHLDNLKGTFAHLSELHTDKLHVDPENFRLLGN
SASAIMGNPKVKAHGKKVINAFNDDVHKHLDNLKGTFAHLSELHSDKLHVDPENFRLLGN

MIVIVLGHHLGKEFTPC----------AQAAFQKVVAGVASALAHKYH
MIVIVLGHHLGKEFTPCVOLKSWAGENAQAAFQKVVAGVASALAHKYH
```

Implementation details

We chose a recursive implementation. The running time of our implementation is $O(n^2)$, since we recursively run through both strings and then trace back to find the optimal alignment. n is the length of the longest string. We find the optimal alignment by building the matrix and then performing pathfinding afterwards to find the alignment. This still gives us a running time of $O(n^2)$.

Distance penalties are used by reading `BLOSUM62.txt` and building the penalty matrix from this file. We read the values as a 2D array and look up penalties based on character indexes in the file.