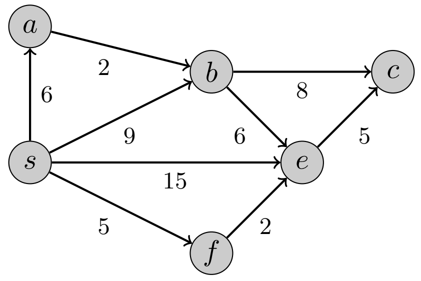
Written Problems. due Mar 3 (Wed), 6.30 pm.

1. **(Dijkstra's Algorithm)** Run Dijkstra's Algorithm on the following graph starting from the source node s. Show the intermediate computations of the list values `d`, `d'` and `p`. In case of a tie, pick the node that comes first in alphabetical order.



2. **(Paths with Changing Colors)** Consider a (weighted) directed graph G with n vertices and m edges where each edge e has a positive weight $w(e)$ and is colored either red or blue. The cost of a path is equal to the sum of the weights of edges on the path plus 5 for each pair of adjcent edges that are not the same color. That is, when traversing a path, it costs an additional 5 units to switch from one edge to another of a different color. (In the context of planning the shortest commute on the New York MTA, this could correspond to the delay from switching amongst subway lines.)

   Give an efficient algorithm to find a lowest-cost path between two vertices s and t in time $O(m \log n)$.

   (HINT. Run Dijkstra's Algorithm on a graph G' that's related to the graph G. Roughly speaking, the graph G' will contain edges of G and some additional edges with appropriate weights to capture the switching cost.)

Programming Problems. due Mar 5 (Fri), 11.59 pm.

1. **(Matching Subsequences)** Textbook Chapter 4, Exercise 4

   Download `ps4_subseq.zip`. Run `test_subseq.py` to help determine if your solutions work. Complete the implementation of the subsequence matching algorithm in `subsequence.py` and submit `subsequence.py` by copying the file to your `submit` directory. The entire test suite in `test_subseq.py` must complete running in under 10 secs on `owl.cs.qc.edu`.

2. **(Approximate Time-Stamp Matching)** Textbook Chapter 4, Exercise 16

   Download `ps4_timestamp.zip`. Complete the implementation of the approximate time-stamp matching algorithm in `timestamp.py` and submit `timestamp.py` by copying the file to your `submit` directory. Run `test_timestamp.py` to help determine if your solutions work. Your algorithm must run in $O(n^2)$ time and the entire test suite in `test_timestamp.py` must complete running in under 6 secs on `owl.cs.qc.edu`.

   (NOTE. You should consider adding your own test cases as there are algorithms that will pass the test suite but are in fact incorrect. In particular, you should convince yourself that the greedy algorithm "earliest t_i first" that associates each event x_j with a compatible transaction with the earliest t_i value may not output the optimal solution.)