(a) To do this, we build a graph H as follows. H has the same nodes as all the G_i, and it consists precisely of those edges that occur in every one of $G_0, \ldots, G_b$. In this graph H, we simply perform breadth-first search to find the shortest path from s to t (if any s-t path exists).

Note that this idea, generalized to any sequence of graphs $G_i, \ldots, G_j$, will be useful in part (b).

(b) We are given graphs $G_0, \ldots, G_b$. While trying to find the last path P_b, we have several choices. If G_b contains $P_{b\ 1}$, then we may use $P_{b\ 1}$, adding $l(P_{b\ 1})$ to the cost function (but not adding the cost of change K.) Another option is to use the shortest s-t path, call it S_b, in G_b. This adds $l(S_b)$ and the cost of change K to the cost function. However, we may want to make sure that in $G_{b\ 1}$ we use a path that is also available in G_b so we can avoid the change penalty K. This effect of G_b on the earlier part of the solution is hard to anticipate in a greedy-type algorithm, so we'll use dynamic programming.

We will use subproblems $Opt(i)$ to denote minimum cost of the solution for graphs $G_0, \ldots, G_i$.

To compute $Opt(n)$ it seems most useful to think about where the last changeover occurs. Say the last changeover is between graphs G_i and G_{i+1}. This means that we use the path P in graphs $G_{i+1}, \ldots, G_b$, hence the edges of P must be in every one of these graphs.

Let $G(i,j)$ for any $0 \leq i \leq j \leq b$ denote the graph consisting of the edges that are common in $G_i, \ldots, G_j$; and let $\ell(i,j)$ be the length of the shortest path from s to t in this graph (where $\ell(i,j) - \infty$ if no such path exists).

If the last change occurs between graphs G_i and G_{i+1} then we get that $Opt(b) = Opt(i) + (b-i)\ell(i+1,b) + K$. We have to deal separately with the special case when there are no changes at all. In that case $Opt(b) - (b+1))\ell(0,b)$.

So we get argued that $Opt(b)$ can be expressed via the following recurrence:

$$Opt(b) = \min[(b+1))\ell(0,b), \min_{1 \leq i < b} Opt(i) + (b-i)\ell(i+1,b) + K].$$

Our algorithm will first compute all $G(i,j)$ graphs and $\ell(i,j)$ values for all $1 \leq i \leq j \leq b$. There are $O(b^2)$ such pairs and to compute one such subgraph can take $O(n^2 b)$ time, as there are up to $O(n^2)$ edges to consider in each of at most b graphs. We can compute the shortest path in each graph in linear time via BFS. This is a total of $O(n^2 b^3)$ time, polynomial but really slow. We can speed things up a bit to $O(b^2 n^2)$ by computing the graphs $G(i,j)$ and $\ell(i,j)$ for a fixed value of i in order of $j = i \ldots b$.

Once we have precomputed these values the algorithm to compute the optimal values is simple and takes only $O(b^2)$ time. We will use $M[0 \ldots b]$ to store the optimal values.

```
For i=0,...,b
    M[i] = min((i  1)ℓ(0,i); min_{1≤j<i} M[j]  (i    j)ℓ(j  1,i))
EndFor
```

[1] ex377.520.504