# CS5800: Algorithms Spring 2018
# Assignment 7.1

Saptaparna Das

April 10, 2018

**(a)** For the given problem, the underlying datastructure should be a directed graph where edges indicate bus routes and nodes indicate bus stops.
Since we are given a 24 hours schedule of all buses at every stop, we can subtract the time at one stop from previous stop to get the time in which the bus can reach from one stop to another. So, we can use this time as weight of every edge in the graph.

**(b)** The technical problem is to get a path for a given pair of nodes which takes the least time.

**(c)** We will use Dijkstra's algorithm to calculate shortes path from a given source node.
Since, we also have take into consideration the transition time of an user if he/she has to change the bus to reach the destination, we can do that by adding transition time as weight to the edges along with existing weights.Now this transition time depends on the bus route and particular source to destination address. So, this should be dynamic in nature. The solution to this problem is, we can preprocess and create the graph once, but for every pair of source to destination nodes, we need to recalculate the weights based on transition time. So, in standard Dijkstra's algorithm when for every adjacent node of a source node, we update the weight, we need to see if the node is in a particular bus route or different bus route, in which case we have to add transition time to it along with predefined time to reach one stop to another.(Assuming we have the information if a node is part of a particular bus route).
If we can assume that the user will always choose a source bus stop from which theres a direct transit till destination, we don't need to recalculate the weights for every input. In that case, we can use standard Dikjstra algorithm with predefined edge weights as time taken by a bus to go from one stop to other (Assuming every bus takes same time to go a particular distance on their respective route i.e bus 1 and bus2 take same time to go from stop A to stop B)

**(d) Analysis:** The time taken by standard Dijkstra's algorithm depends on how it is implemented. Since this is definitely a dense graph, we can use fibonacci heap to implement the priority queue. The amortized cost of each of the $|V|$ EXTRACT-MIN operations is O(log V), and each DECREASEKEY call, of which there are at most $|E|$ takes only O(1) amortized time. So, total running time is $O(VlogV + E)$. Space complexity $O(E + V)$

**(e)** For the given problem, though it basically has to calculate shortest path between any pair of nodes, the weights of the graph change dynamically. So, we can not precompute the values and use. Hence, standard Floyd algorithm might not work. Also, for a dense graph Floyd will consume more memory.Again, the weights indicate time here, which cannot be negative. So, Dijkstra's algorithm can be used. For each run of Dijkstra it calculates the shortest paths from a fixed source to all other nodes. We don't need to save this result for the next run since, the weights might change depending on given source node. For every input of source node and destination point, we run Dijkstra with modified weights. So, it always picks the edges with smaller weights in the process and that leads to shortest path from source to destination.