Code Jam 2008 - Qualification Round

Analysis: Train Timetable

This problem can be solved with a greedy strategy. The simplest way to do this is by scanning through a list of all the trips, sorted by departure time, and keeping track of the set of trains that will be available at each station, and when they will be ready to take a trip.

When we examine a trip, we see if there will be a train ready at the departure station by the departure time. If there is, then we remove that train from the list of available trains. If there is not, then our solution will need one new train added for the departure station. Then we compute when the train taking this trip will be ready at the other station for another trip, and add this train to the set of available trains at the other station. If a train leaves station A at 12:00 and arrives at station B at 13:00, with a 5-minute turnaround time, it will be available for a return journey from B to A at 13:05.

We need to be able to efficiently identify the earliest a train can leave from a station; and update this set of available trains by adding new trains or removing the earliest train. This can be done using a heap data structure for each station.

Sample Python code provided below that solves a test case for this problem:

```
def SolveCase(case_index, case):
  T, (tripsa, tripsb) = case
  trips = []
  for trip in tripsa:
    trips.append([trip[0], trip[1], 0])
  for trip in tripsb:
    trips.append([trip[0], trip[1], 1])

  trips.sort()

  start = [0, 0]
  trains = [[], []]

  for trip in trips:
    d = trip[2]
    if trains[d] and trains[d][0] <= trip[0]:
      # We're using the earliest train available, and
      # we have to delete it from this station's trains.
      heappop(trains[d])
    else:
      # No train was available for the current trip,
      # so we're adding one.
      start[d] += 1
    # We add an available train in the arriving station at the
    # time of arrival plus the turnaround time.
    heappush(trains[1 - d], trip[1] + T)

  print "Case #%d: %d %d" % (case_index, start[0], start[1])
```

Luckily Python has methods that implement the heap data structure operations. This solution takes O(n log n) time, where n is the total number of trips, because at each trip we do at most one insert or one delete operation from the heaps, and heap operations take O(log n) time.