Problem Statement:

A travel agency has a list of flights. Each flight has a flight number, origin airport, destination airport, departure time and arrival time. Write a program to assist the travel agency in determining the earliest arrival time for the destination given an origin airport and start time. Hint: use a shortest path algorithm.

Problem Overview:

This is the shortest path algorithm problem as we need to calculate the earliest arrival time, with a given start time. So, we can take the problem as a di graph where all the airports are nodes and the flights are the di edges with weight of time interval i.e. the time difference between arrival time of destination airport and departure time of source airport.

Here we need to take care that while going from origin airport to destination airport, and while taking flights from one airport to another, we can take only those flights which can be caught. So, time plays an important role in this problem as we can take only the flights whose departure time from the airport is later than our arrival time at the airport.

So, to solve this we will use Dijkstra's Algorithm with a little modification.

Dijkstra's Algorithm:

Dijkstra's Algorithm is a shortest path algorithm used to calculate the shortest path between nodes in a graph. This algorithm was created and published by computer scientist Dr. Edsger W. Dijkstra. The algorithm exists in many variants. Dijkstra's original algorithm was to find the shortest path between two given nodes, but a more common variant fixes a single node as the "source" node and finds shortest paths from the source to all other nodes in the graph, producing a shortest-path tree.

In Dijkstra's algorithm, we start from a source node and initialize its distance by zero. Next, we push the source node to a priority queue with a cost equal to zero. After that, we perform multiple steps. In each step**,** we extract the node with the lowest cost, update its neighbors' distances, and push them to the priority queue if needed. Each of the neighboring nodes is inserted with its respective new cost, which is equal to the cost of the extracted node plus the edge we just passed through. We continue to visit all nodes until there are no more nodes to extract from the priority queue. Then, we return the calculated distances.

Algorithm:

The algorithm for this problem is the slight modification of Dijkstra's Algorithm. Here we have the database about airports and the flights. There is information about flight number, origin airport and destination, the flights have departure and arrival time. So, given origin and destination airports and start time we need to make our data structure properly to keep the data and find the shortest arrival time.

Here the di graph is used where the airports are the vertices and flights are di-edges with weights: departure time and arrival time. The distance should be the function of earliest arrival times at airports. Then based upon those earliest arrival time, we use a minimum priority queue for airports. For the initial origin airport let the earliest arrival time be start time and for others it will be infinity in the beginning. Now, until the queue is not empty, adjacent loop is to be formed where we can only select the flights which can be caught, and we also want the flight with minimum arrival time. So, for that we create another priority queue of flights where its departure time should be later than arrival time of another flight at the airport. And we also use another variable time, which is used to update the flight priority queue. And finally, we perform relaxation, where if the above time is less than the earliest arrival time at adjacent airport, then we change the earliest arrival time of the airport to be the time and update in the airport queue accordingly.

Pseudocode:

From http://www.csl.mtu.edu/cs2321/www/newLectures/30_More_Dijkstra.htm

```
function flightAgency (Flight F, Graph G, Vertex s, Vertex d, Time startT):
    initialize arrival time, T{}
    T[s] ← startT

    for all vertex, v ≠ s do
        T[v] ← ∞
    make Priority Queue, Q, of vertices keyed by T

    while Q is not empty do
        v ← Q.removeMin()

        for all vertices, w, adjacent to v and in Q do
            make Priority Queue, P, of fights, f, with f.departT ≥ T[v] keyed by f.arriveT
            Time t ← ∞
            if P is not empty then
                t ← P.min().arriveT
            if t < T[w] then
                T[w] ← t
                update w in Q

    return T[d]
```

Main Program:

So, for our main program we made following setup:

- Airport A, B, C, D, E as the vertices of the Graph G.
- The list of 7 flights:
  FN-101: From airport A to B with departure time 02:00 and arrival time 06:00
  FN-102: From airport A to C with departure time 02:00 and arrival time 08:00
  FN-103: From airport B to D with departure time 12:00 and arrival time 13:00
  FN-104: From airport B to E with departure time 11:00 and arrival time 17:00
  FN-105: From airport C to B with departure time 09:00 and arrival time 10:00
  FN-106: From airport C to D with departure time 06:00 and arrival time 10:00
  FN-107: From airport D to E with departure time 13:00 and arrival time 14:00
- The flights are the edges.
- Airport A as the origin airport.
- Airport E as the destination airport.
- 02:00 as the start time.

Output:

```
OK
PS C:\A1\X\KU\Algo\mini> py .\main.py
The earliest arrival time for the airport E from airport A is 14:00.
PS C:\A1\X\KU\Algo\mini>
```

We got the output that, the earliest arrival time for the airport E from airport A is 14:00. So, starting at 02:00 from airport A the shortest path to reach airport E, the earliest arrival time is 14:00.

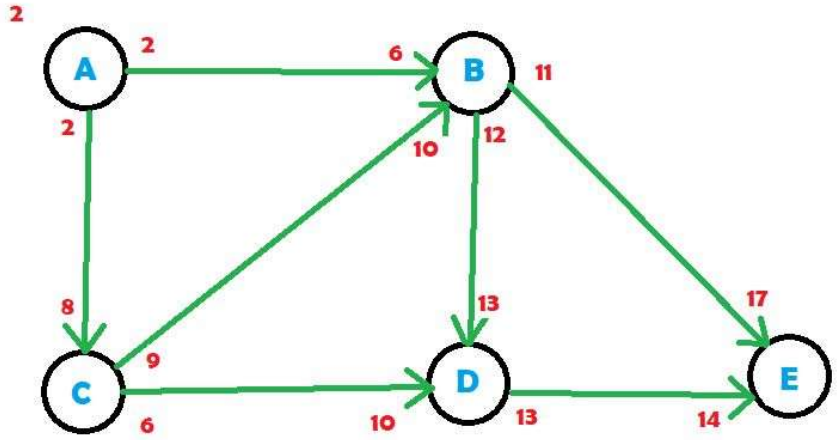
Similarly, if we had tried the same setup with start time 10:00 then we get the result as:

```
OK
PS C:\A1\X\KU\Algo\mini> py .\main.py
The earliest arrival time for the airport E from airport A is 14:00.
PS C:\A1\X\KU\Algo\mini> py .\main.py
No flight from airport A to airport E after 10:00.
PS C:\A1\X\KU\Algo\mini>
```

We got the output as, No flight from airport A to airport E after 10:00. Also, we would get similar result, if we try to go from origin to destinatin airport where there is no flight like from B to A.

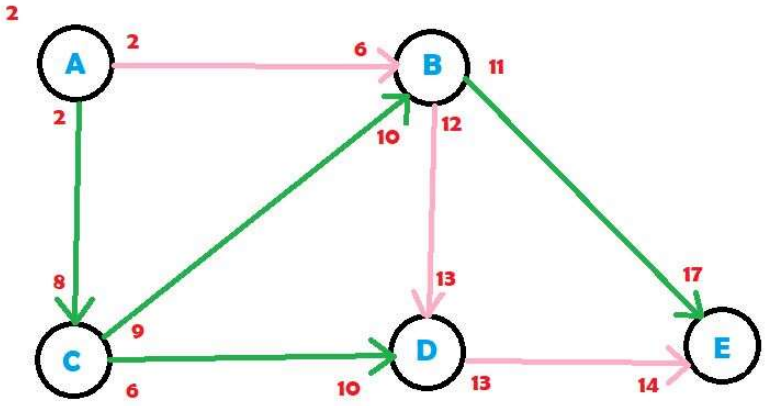
Analysis:

So, based upon the above setup, we would have got a graph like below:



In above graph, there is a flight that leaves from airport A at 2 and reaches C at 8 and another flight leaves airport A at 2 and reach airport B at 6 and so on.

From airport A, we can go to airport B and C. The best possible arrival time for airport B and C would be 6 and 8 respectively. The flight from C to D departs at 6. So, starting at time 2 and reaching airport C at 8, there is no way we can catch that flight. So, from C, now we can only go to B. We can reach airport B either from A or from C, but the earliest arrival time would be from airport A at 6. Similarly, from B we can go to D or E and so on. So, finally using the algorithm the best possible shortest pathway we can reach airport E from airport A would be via route A – B – D – E.



And, using those flights, we would reach airport E at 14, which is the answer given by our program.