**Raptor Algorithm:**

**Problem definition:**

We need to find the fasted path (the path with less time for traveling) from one station to one station go through some bus routes with some additional conditions: departure time, number of transfers.

**Introduction:**

This problem can be solved by using Raptor algorithm. Raptor algorithm is an algorithm based on paper named “Round-Based Public Transit Routing”. This paper is published by researchers Daniel Delling, Thomas Pajor and Renato F. Werneck on 2012.

Above problem can be known as computing all Pareto-optimal journeys in a dynamic public transit network for multiple criteria such as departure time and number of transfers.

Existing algorithms consider this as a graph problem, and solve it using variants of Dijkstra algorithm such as Layered Dijkstra (2004), multi-label-correcting algorithm (2008).

Starting from arrival time and number of transfers as criteria, this algorithm can be extended to handle more arbitrary additional criteria such as arrival time, ticket price, change time at each station or reliability of transfers.

**Attribute Definition:**

In below algorithm explanation, we use some notations:

* maximum bus transfers for this search. We can choose this dynamically when searching.
* is set of city’s stations.
* is set of city’s routes.
* is set of trips of all routes. And is set of trips of route . In our algorithm, we will sort those trips of each route based on departure time of each trip for searching optimization.
* earliest arrival time that we can come to this station using at most k transfers. Our algorithm will optimize those set of values. Note that two points:

1. This time is real time base on departure time (ie: 5:30 PM, 8:15 AM)
2. The total transfers can be any from to . If value equals 0, it means we cannot reach this station on time by using any routes.

* earliest arrival time to this station. After loops finish,
* be the earliest trip in route r that one can catch at station
* be the departure time of trip t at . Note that, currently, our bus system, arrival time and departure time of trip t at are equals. That means no wait time at each station of each trip.
* be the arrival time of trip t at . Note that, currently, our bus system, arrival time and departure time of trip t at are equals. That means no wait time at each station of each trip.

We also have some variables which means:

* **Q**: a map. This map has key is a route and value is a station in this route which previous round has been optimized.
* **TraceUsedRoute[i][j] = R** means: Best way come to j-station using at most i transfers using route R.
* **TraceFromStation[i][j] = S** means: best way come to j-station using at most i transfers must go on at station S.
* **Result[i][j] = T** means: best way come to j-station using at most i transfers it T times.

**Solution:**

Method signature: **Raptor**(source s, target t, departure time r, maximum transfer K)

*Step 1: initialize all variables Q, TraceUsedRoute, TraceFromStation, Result*

*Step 2: optimize each stations for total K rounds. After K rounds, we can have best time go to each stations at maximum K transfers*.

**For all** k 🡨 1,2,3,…K **do**

*Step 3: find all previous round optimize stations. Find all routes go through this station.*

**For all** marked station p **do**

**For all** routes r go through station p **do**

**If** (r, p’) in Q for some p’ **then**

**If** r.order(p) < r.order(p’) **then**

Change (r, p’) -> (r,p) in Q

**Endif**

**Else**

Add (r, p’) 🡪 Q

**Endif**

**endfor**

**endfor**

*Step 4: travel again each route from marked station and optimize time at each station.*

**For all** pair (r,p) in Q **do**

Trip t = null

**Forall** stop beginning with p in route r **do**

**If** t != null **and** < **then**

=

Add to marked stations for next round

*Step 5: save to trace array for print result later, if necessary*

traceFromStation[k][ = p

traceUsedRoute[k][ = r

result[k][ = t

**endif**

**if**  **then**

**endif**

**endfor**

*step 6. Check if no optimize stations anymore*

**if** no marked station in this turn **then**

**break;**

**endif**

**endfor**

**endfor**

#### Algorithm complexity

The worst-case running time of our algorithm can be bounded as follows:

In every round, algorithm scans each route at most once. If is the number of stations of route , then we look at stations in total to process the route. For each station , we must find the earliest trip. If we keep the list of trips serving r sorted by time, while traversing r we can find all values with a single sweep over this list, since can only decrease. In total, RAPTOR takes time for each round.

If we call sum of all stations and all trips (of all routes) in city equals to . Total algorithm complexity is: )