# Vorarbeit

Timetable:   
quadrupel (S,C,T,F) Stops, Connections, Trips, Footpaths

Stops:

Connections: quintuple (c\_dep\_stop, c\_arr\_stop, c\_dep\_time, c\_arr\_time, ctrip) verbindung ohne zwischenhalt eines zuges

Trips: Ein Zug der einen Weg abfährt

Footpaths: triple(f\_dep\_stop, f\_arr\_stop, f\_dur) – bilden eine footpath graph

Transfers: Traveler standing at Stop is a pair (s,t)

Journeys:

Zusamengesetzt aus legs. Legs L = pair of connections (lenter, lexit) mit dem selben trip

Pareto-optimal:

X ist pareto-optimal wenn kein y existiert welches x dominiert

Dominieren:

X dominiert y wenn y keine komponente hat welche strikt kleiner als die von x ist und x mindestens eine komponente hat welche strikt kleiner ist als die von y

# Gedanken

## Earliest Arrival Problem:

Zeit für die kürzeste reise bei einer fixen Abfahrtszeit. 🡪 Keine Fixe Zeit sondern Zeitfenster

## Earliest Arrival Profile Problem:

* Eingabe timetable, source stop, target stop, minimum dep time, maximum arr time
* Alle tupel (j\_dep\_time, j\_arr\_time) welche erfüllen:
  + J abfahrt nicht vor min dep time
  + J ankunft nicht nach max arr time
  + Tupel ist pareto-optimal
  + J hat mindestens 1 Leg (algorithmus vereinfachung)

## Pareto Profile Problem

# Algorithmus

## Earliest Arrival Connection Scan

Voraussetzungen: Connections in array aus quintupeln sortiert nach departure time, Footpaths in einem adjacency array

Funktion: Speichert für eine Abfahrtsort und Abfahrtszeit die frühstmögliche Ankunftszeit für alle Stops. Dann kann für einen Stop der Journey extrahiert werden. Ankunftszeit als einziges primäres kriterium

Journey Extraction: Journey Pointer setzen. Journe pointer können von zielStop den Journey aufbauten

Journey Pointer: (final enter connection, final exit connection, final footpath)

## Earliest Profile Connection Scan

Profile: pair von (p\_dep\_time, p\_arr\_time)

Profile für Journey Extraction: quadrupel (p\_dep\_time, p\_arr \_time, l\_enter, l\_exit)

Trip für Journey Extraxtion: pair (t\_arr\_time, t\_exit\_con\_id)

Basiert auf den Tatsachen:

* Es gibt 3 Möglichkeiten was ein Passagier tun kann wenn er in einem Zug sitzt
  + Aussteigen und zum Ziel laufen (Variante1)
  + Sitzen bleiben (Variante2)
  + Aussteigen und in einen anderen Zug einsteigen (Variante3)
* Wenn mann c absteigend nach c\_dep\_time durchläuft kann zuvor noch keine connection mit niedrigerer departure time gefunden worden sein.

# Gesamtablauf

1. GTFS-Daten in Timetable umwandeln
2. Timetable.Connections nach Timetable.Connections.DepartureTime sortieren.
3. Server Läuft
4. Benutzeranfrage Annehmen
5. Benutzerinstanz des Timetables erzeugen
6. Algorithmusvariante auf dem erzeugten Timetable ausführen
7. Journey in die vom Server verlangte Form bringen
8. Journey an den Server zurückgeben

# Earliest Arrival connection Scan auf dem erzeugten Timetable ausführen

1. Array mit EAT für alle Stops anlegen und auf Unendlich setzen
2. Array mit Conecctions für alle Trips anlegen und auf leer setzen
3. Array mit Journey Pointer für jeden Stop anlegen und auf Leer setzen
4. EAT für alle vom startStop erreichbare Stops auf startzeit + Laufzeit setzen
5. Erste mögliche Connection finden
6. Alle möglichen Connections durchlaufen (aufsteigend c\_dep\_time)
   1. Schleife verlassen wenn c\_dep\_time >= EAT vom Zielpunkt
   2. Testen ob connection reachable ist (triparraypunkt nicht leer? Oder stop von startstop erreichber?)
      1. Prüfen ob der Triparraypunkt leer ist
         1. Connection in den Triparraypunkt eintragen(eintritspunkt in den Trip)
      2. Für alle footpaths vom stop aus überprüfen: laufzeit + ankunftszeit < alte ankunftszeit des stops
         1. Laufzeit + ankunftszeit als neue Ankunftszeit in EAT Array eintragen
         2. Eintrittspunkt, Connection, Footpath in den Journey Pointer Array eintragen.
7. Leeren journey array anlegen
8. Schleife solange J[t] nicht leer ist. (Anfangs ist t der zielpunkt)
   1. J[t] mit prepend in j einfügen
   2. J[t] Eintrittspunkt in t eintragen
9. Den initial-footpath mit prepend an j anfügen
10. Rückgabe von j

# Earliest Arrival Profile Algorithm (Ohne Interstop Footpaths, Unoptimized, ohne Journey Extraction)

1. Array für jeden Stop anlegen und mit Profilarrays füllen. Profilarrays mit {(∞,∞)} füllen.
2. Array für jeden Trip anlegen und mit ∞ füllen.
3. Alle connections durchlaufen (absteigend c\_dep\_time)
   1. Variante1
      1. Testen ob c\_arr\_stop = target
         1. T1 = c\_arr\_time + c\_arr\_stop\_change\_time(footpath loops)
      2. Else
         1. T1 = ∞
   2. Variante2
      1. T2 = Triparraywert
   3. Variante3
      1. Frühestes Profil für die Station auslesen.
      2. Solange ein späteres Profil für die Station wählen bis p\_dep\_time < c\_arr\_time
      3. T3 = p\_arr\_time
   4. Tc = min(T1,T2,T3)
   5. Tc in Stoparray eintragen
      1. p = (c\_dep\_time – c\_dep\_stop\_chang\_time(footpath loops), Tc)
      2. q = frühestes Profil des c\_dep\_stops
      3. Prüfen ob q p dominiert
         1. Prüfen ob q\_dep\_time = p\_dep\_time
            1. p in Stoparray einfügen
         2. Else
            1. q mit p ersetzen
   6. Tc in Triparray eintragen

# Earliest Arrival Profile Algorithm (Ohne Interstop Footpaths, Unoptimized)

1. Array für jeden Stop anlegen und mit Profilarrays füllen. Profilarrays mit {(∞,∞,null,null)} füllen.
2. Array für jeden Trip anlegen und mit {(∞,null)} füllen.
3. Alle connections durchlaufen (absteigend c\_dep\_time)
   1. Variante1
      1. Testen ob c\_arr\_stop = target
         1. T1 = c\_arr\_time + c\_arr\_stop\_change\_time(footpath loops)
      2. Else
         1. T1 = ∞
   2. Variante2
      1. T2 = Triparraywert
   3. Variante3
      1. Frühestes Profil für die Station auslesen.
      2. Solange ein späteres Profil für die Station wählen bis p\_dep\_time < c\_arr\_time
      3. T3 = p\_arr\_time
   4. Tc = min(T1,T2,T3)
   5. Prüfen ob Tc < T2
      1. t\_exit\_con\_id = current connection
   6. Tc in Stoparray eintragen
      1. p = (c\_dep\_time – c\_dep\_stop\_chang\_time(footpath loops), current connection, t\_exit\_con\_id)
      2. q = frühestes Profil des c\_dep\_stops
      3. Prüfen ob q p dominiert
         1. Prüfen ob q\_dep\_time = p\_dep\_time
            1. p in Stoparray einfügen
         2. Else
            1. q mit p ersetzen
   7. Tc in Triparray eintragen
4. Leeren ResultArray für die Journey Arrays anlegen
5. Alle Profile im StartStop durchlaufen(p)
   1. Im ResultArray einen JounreyArray für das jetztige Profil anlegen
   2. px = p
   3. s = StartStop
   4. [px.l\_enter, px.l\_exit] mit append in j einfügen
   5. while(px.l\_exit.arr\_stop ungleich target)
      1. s = px.l\_exit.arr\_stop
      2. Alle Profile in s durchlaufen(p2)
         1. If p2\_arr\_time = px\_arr\_time
            1. Px = p2
      3. [px.l\_enter, px.l\_exit] mit append in j einfügen
   6. Den initial-footpath mit prepend an j anfügen
6. Rückgabe des ResultArrays