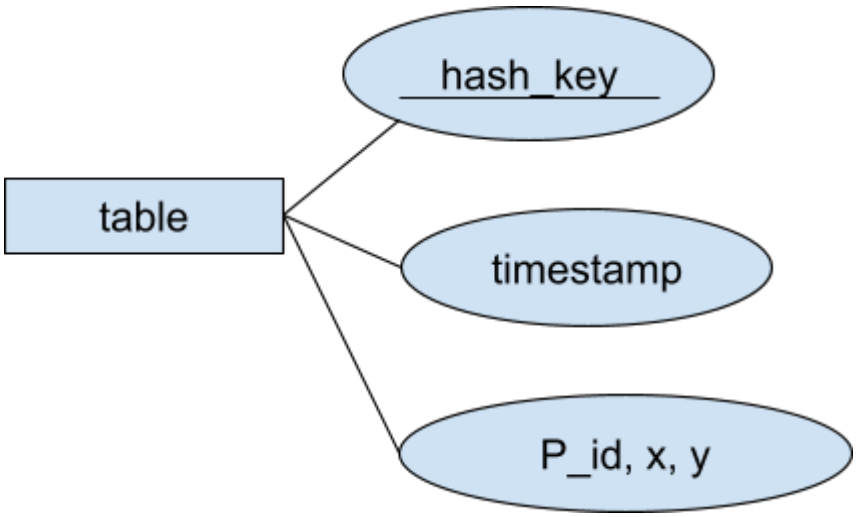
Assignment 2: Indexing

By Paulius Zabinskas
ID 20120267

Question 1:

We can use hash tables to index data timestamps and tuples. Divide the field into 20x10 squares assuming the field is standard rectangle shape. Each square will be given a spatial location identifying squares' location on the field i.e. ([x_1,y_1], [x_2,y_1] … [x_20, y_10]). The hash function could concatenate P_id, timestamp, x_i with y_i coordinate values to be used as an index. Assuming that we are working with given tuples without taking action to normalise the data set.

Potential table schema:



Potential algorithm:
Given player_id return hashset containing grid coordinates and integer with value of how many times player was in grid coordinates.

Pseudo code:

```
Function getPlayerLocationSortedByTime(int p_id):
    Initialise a HashMap (playerCoordinateMap) to store coordinates as keys and their counts as values
    Initialise a List (playerCoordinateList) to store player coordinates

    // First, collect all coordinates for the given player ID
    For each entry in table:
        If entry.p_id equals p_id:
```

```
            Add entry.coordinates to playerCoordinateList

    // Next, count occurrences of each coordinate
    For each coordinates in playerCoordinateList:
        If playerCoordinateMap contains coordinates:
            Increment the count for coordinates in playerCoordinateMap
        Else:
            Add coordinates to playerCoordinateMap with a count of 1

    Return playerCoordinateMap
```

Question 2:

We can increase the number of CPUs used to execute a function. Then divide the dataset / table into smaller, equal subset tables for each CPU. Now each subset table can be processed independently. After all processes have been completed, combine the result into a single hashmap. If the same coordinates have been detected, sum up their counts.

```
Function parallelGetPlayerLocationSortedByTime(int p_id, table):
    Divide table into subsets (tableSubsets)

    // Parallel processing
    Initialise an empty HashMap (combinedPlayerCoordinateMap)
    Parallel forEach subset in tableSubsets:
        Call getPlayerLocationSortedByTime(p_id, subset)
        Synchronise and merge results into combinedPlayerCoordinateMap

    Return combinedPlayerCoordinateMap

Function getPlayerLocationSortedByTime(int p_id, subset):
    // Same as before
```

Question 3:

After subdividing the pitch into 9 rectangles, should include all arias in table given some unique name (rec_1, rec_2,...., rec_9) these values can be used as hash parameters to form an index with player id, similarly as it was mentioned in question 1 solution.
The algorithm must return times of a given player in a given rec_i.

Pseudo-code:

```
Function getPlayerTimesInRectangle(player_id, rectangle, table):
    // Initialize a list to store timestamps when the player is in the specified rectangle
    Initialise a List (timestampsInRectangle)

// Iterate over each record in the table
    For each tuple t in table:
        // Check if the tuple corresponds to the target player
        If t.player_id equals player_id:
            // Check if the player's position falls within the specified rectangle
            If isWithinRectangle(t.position, rectangle):
                // Add the timestamp to the list
                Add t.timestamp to timestampsInRectangle
            End
        End
    End

    Return timestampsInRectangle
```

Question 4:

This time we need to check for all player positions and if any of the players are in the same rectangle at a given time. We can use the hash of rac_i and time attributes as an indexing strategy.

Pseudo-code:

```
Function findPlayersInSameGridLocation(table):
    // Initialize a HashMap with keys as grid locations and values as lists of player-time tuples
    Initialise a HashMap (gridTimeMap)

    // Iterate over each record in the table
    For each tuple t in table:
        // Add tuple (player_id, timestamp) to the list in gridTimeMap for this index(t.hash)
        Add (t.player_id, t.timestamp) to gridTimeMap[t.hash]
    End

    // Initialize a list to store instances of two players in the same grid
    Initialise a List (playersInSameGrid)

    // Iterate over each grid location to find overlapping player times
    For each gridLocation in gridTimeMap:
        // Nested loops to compare every pair of player-time tuples in the same grid
        For each playerTimeTuple1 in gridTimeMap[gridLocation]:
            For each playerTimeTuple2 in gridTimeMap[gridLocation]:
                // Check for different players at the same time in the same grid
```

```
            If playerTimeTuple1.player_id != playerTimeTuple2.player_id AND
playerTimeTuple1.timestamp == playerTimeTuple2.timestamp:
                Add (playerTimeTuple1, playerTimeTuple2) to playersInSameGrid
            End
        End
    End
End

Return playersInSameGrid
```