**Universidad Icesi**

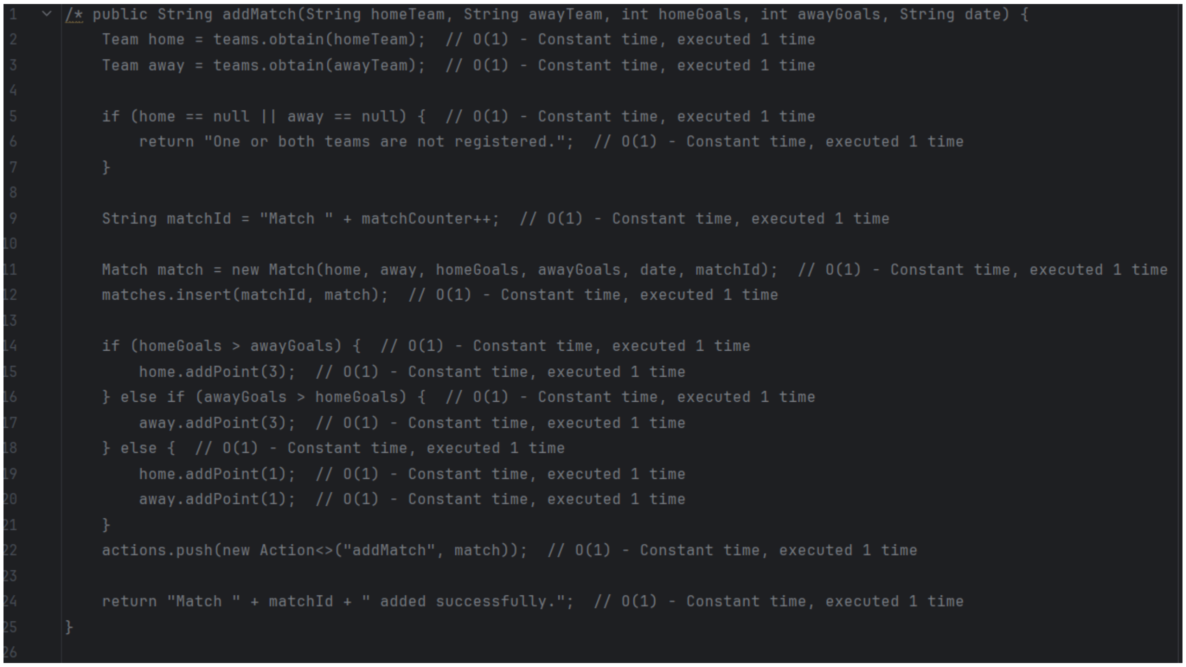
**Departamento de Computación y Sistemas Inteligentes**

**Computación y estructuras discretas I**

# Complexity analysis

- ## Time Complexity Analysis

### 1. Method addMatch()



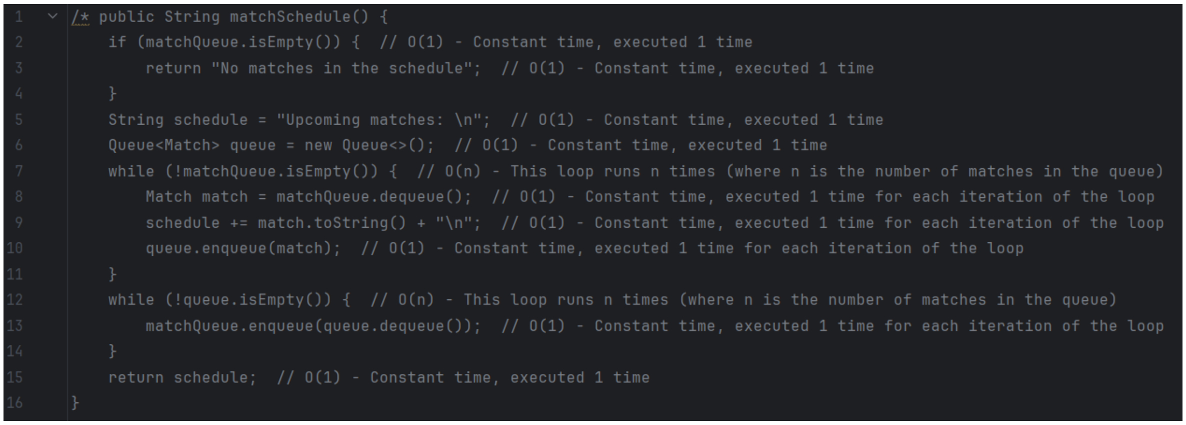
```
1   /* public String addMatch(String homeTeam, String awayTeam, int homeGoals, int awayGoals, String date) {
2       Team home = teams.obtain(homeTeam);  // O(1) - Constant time, executed 1 time
3       Team away = teams.obtain(awayTeam);  // O(1) - Constant time, executed 1 time
4
5       if (home == null || away == null) {  // O(1) - Constant time, executed 1 time
6           return "One or both teams are not registered.";  // O(1) - Constant time, executed 1 time
7       }
8
9       String matchId = "Match " + matchCounter++;  // O(1) - Constant time, executed 1 time
10
11      Match match = new Match(home, away, homeGoals, awayGoals, date, matchId);  // O(1) - Constant time, executed 1 time
12      matches.insert(matchId, match);  // O(1) - Constant time, executed 1 time
13
14      if (homeGoals > awayGoals) {  // O(1) - Constant time, executed 1 time
15          home.addPoint(3);  // O(1) - Constant time, executed 1 time
16      } else if (awayGoals > homeGoals) {  // O(1) - Constant time, executed 1 time
17          away.addPoint(3);  // O(1) - Constant time, executed 1 time
18      } else {  // O(1) - Constant time, executed 1 time
19          home.addPoint(1);  // O(1) - Constant time, executed 1 time
20          away.addPoint(1);  // O(1) - Constant time, executed 1 time
21      }
22      actions.push(new Action<>("addMatch", match));  // O(1) - Constant time, executed 1 time
23
24      return "Match " + matchId + " added successfully.";  // O(1) - Constant time, executed 1 time
25  }
26
```

**Line-by-Line Analysis**:

- **Lines 2-3**: teams.obtain(homeTeam) and teams.obtain(awayTeam) are search operations in a hash table. In the **best case**, the complexity is **O(1)**. In the **worst case**, it can be **O(n)**, but we assume average hash table performance.
- **Lines 5-9**: The goal comparison and points update operations are constant, i.e., **O(1)**.
- **Lines 11-12**: The insertion into the matches hash table is **O(1)** in an efficient hash table.
- **Lines 14-20**: The goal comparison and point update operations for the teams are **O(1)**.
- **Line 22**: The actions.push() operation is **O(1)**.

**Overall Time Complexity for addMatch()**: **O(1)**, since all operations in the method are constant time and executed only once.

## 2. Method matchSchedule()



```
1   /* public String matchSchedule() {
2       if (matchQueue.isEmpty()) {  // O(1) - Constant time, executed 1 time
3           return "No matches in the schedule";  // O(1) - Constant time, executed 1 time
4       }
5       String schedule = "Upcoming matches: \n";  // O(1) - Constant time, executed 1 time
6       Queue<Match> queue = new Queue<>();  // O(1) - Constant time, executed 1 time
7       while (!matchQueue.isEmpty()) {  // O(n) - This loop runs n times (where n is the number of matches in the queue)
8           Match match = matchQueue.dequeue();  // O(1) - Constant time, executed 1 time for each iteration of the loop
9           schedule += match.toString() + "\n";  // O(1) - Constant time, executed 1 time for each iteration of the loop
10          queue.enqueue(match);  // O(1) - Constant time, executed 1 time for each iteration of the loop
11      }
12      while (!queue.isEmpty()) {  // O(n) - This loop runs n times (where n is the number of matches in the queue)
13          matchQueue.enqueue(queue.dequeue());  // O(1) - Constant time, executed 1 time for each iteration of the loop
14      }
15      return schedule;  // O(1) - Constant time, executed 1 time
16  }
```

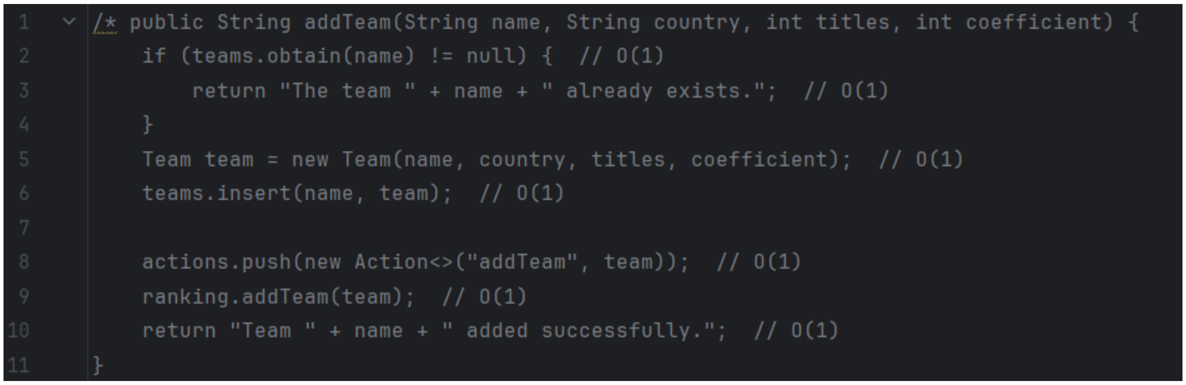
**Line-by-Line Analysis**:

- **Lines 2-3**: The matchQueue.isEmpty() operation and the return have **O(1)** complexity.
- **Line 6**: Creating a new queue is **O(1)**.
- **Lines 7-11**: The while (!matchQueue.isEmpty()) loop iterates through all elements in matchQueue. If there are **n** matches, this loop runs **n** times. Each operation inside the loop (dequeue(), schedule +=, enqueue()) is **O(1)**, so the loop has **O(n)** complexity.
- **Lines 12-13**: The second while (!queue.isEmpty()) loop also iterates through all matches in the temporary queue. Similar to the previous loop, it has **O(n)** complexity.

**Overall Time Complexity for matchSchedule()**: **O(n)**, where **n** is the number of matches in the matchQueue.

- ## Space Complexity Analysis

### 1. Method addTeam()



```
1   /* public String addTeam(String name, String country, int titles, int coefficient) {
2       if (teams.obtain(name) != null) {  // O(1)
3           return "The team " + name + " already exists.";  // O(1)
4       }
5       Team team = new Team(name, country, titles, coefficient);  // O(1)
6       teams.insert(name, team);  // O(1)
7
8       actions.push(new Action<>("addTeam", team));  // O(1)
9       ranking.addTeam(team);  // O(1)
10      return "Team " + name + " added successfully.";  // O(1)
11  }
```

| Tipo | Variable | Tamaño de un valor atómico | Cantidad de valores atómicos |
|---|---|---|---|
| **Entrada** | **homeTeam** | **32 bits (String)** | **1** |
| **Entrada** | **awayTeam** | **32 bits (String)** | **1** |
| **Entrada** | **homeGoals** | **32 bits (int)** | **1** |
| **Entrada** | **awayGoals** | **32 bits (int)** | **1** |
| **Entrada** | **date** | **32 bits (String)** | **1** |
| **Auxiliar** | **home** | **32 bits (Object reference)** | **1** |
| **Auxiliar** | **away** | **32 bits (Object reference)** | **1** |
| **Auxiliar** | **matchId** | **32 bits (String)** | **1** |
| **Auxiliar** | **match** | **32 bits (Object reference)** | **1** |
| **Auxiliar** | **actions** | **32 bits (Object reference)** | **n (depends on number of actions)** |
| **Salida** | **result** | **32 bits (String)** | **1** |

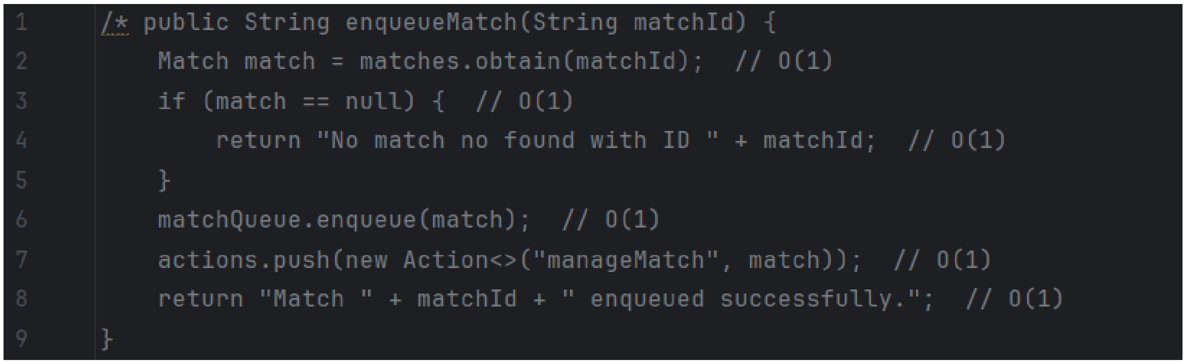
**Line-by-Line Space Analysis**:

- **Lines 2-3**: Checking if the team already exists doesn't require additional space.
- **Line 5**: Creating a new Team object takes up space based on its attributes (name, country, titles, coefficient). This takes **O(1)** space.
- **Line 6**: The insertion into the teams hash table takes up space proportional to the number of teams, i.e., **O(n)**.

- **Line 8**: Pushing an action onto the actions stack requires space **O(n)**.
- **Line 9**: Adding the team to the ranking takes **O(n)** space.

**Overall Space Complexity for addTeam()**: **O(n)**, where **n** is the number of teams and the number of actions in the stack.

### Method enqueueMatch()



```
/* public String enqueueMatch(String matchId) {
    Match match = matches.obtain(matchId);  // O(1)
    if (match == null) {  // O(1)
        return "No match no found with ID " + matchId;  // O(1)
    }
    matchQueue.enqueue(match);  // O(1)
    actions.push(new Action<>("manageMatch", match));  // O(1)
    return "Match " + matchId + " enqueued successfully.";  // O(1)
}
```

| Tipo | Variable | Tamaño de un valor atómico | Cantidad de valores atómicos |
|---|---|---|---|
| **Entrada** | **matchQueue** | **32 bits (Queue)** | **n (depends on number of matches)** |
| **Auxiliar** | **schedule** | **32 bits (String)** | **1** |
| **Auxiliar** | **queue** | **32 bits (Queue)** | **n (depends on number of matches)** |
| **Auxiliar** | **match** | **32 bits (Object reference)** | **n (depends on number of matches)** |
| **Auxiliar** | **matchString** | **32 bits (String)** | **n (depends on number of matches)** |
| **Salida** | **schedule** | **32 bits (String)** | **1** |

**Line-by-Line Space Analysis**:

- **Lines 2**: The retrieval of a match from the matches hash table does not require additional space.
- **Lines 3-4**: Checking if the match exists doesn't require extra space.
- **Line 6**: The matchQueue.enqueue() operation adds the match to the queue. If there are **n** matches in the queue, this takes **O(n)** space.
- **Line 7**: Pushing an action onto the actions stack takes **O(n)** space.

**Overall Space Complexity for enqueueMatch()**: **O(n)**, where **n** is the number of matches in the queue and the number of actions in the stack.

## Summary of Complexities:

**Time:**

1. **addMatch()**: **O(1)**, since all operations inside the method are constant time and executed only once.
2. **matchSchedule()**: **O(n)**, where **n** is the number of matches in the matchQueue.

**Space:**

1. **addTeam()**: **O(n)**, where **n** is the number of teams and the number of actions in the stack.
2. **enqueueMatch()**: **O(n)**, where **n** is the number of matches in the queue and the number of actions in the stack.