Round Robin Tournament Scheduler

Introduction

The Round Robin Tournament Scheduler is a software tool designed to generate a fair, conflict-free schedule for sports leagues using a round-robin format. It ensures that each participant competes against every other participant a set number of times in a round-robin tournament. The tool efficiently manages logistical constraints such as venue availability, predefined time slots, and rules like mandatory rest periods to create a well-structured and balanced schedule.

By automating the scheduling process, this tool simplifies league planning, ensuring efficiency, fairness, and optimal venue utilization while dynamically handling complex constraints. It minimizes scheduling conflicts, ensures participants have adequate rest between matches, and organizes matches in a logical, structured manner. Designed as an essential resource for league organizers, the scheduler enhances tournament management by eliminating the challenges of manual scheduling and ensuring a seamless competition flow.

Problem Instance

A soccer league wants to host a single round-robin tournament that features five teams in total. The tournament will span over two days, and the league has access to two venues throughout the span of the tournament. Matches can start as early as 9 AM, and matches can finish as late as 5 PM. To prevent the teams from playing consecutive matches and to allow for travel to different venues if necessary, there is a minimum 90-minute rest period between matches for every team.

Inputs

To generate the tournament schedule, all the input will be read from a file that contains the following details:

- **Round-robin type**
- **Participants**
- **Duration** (number of days)
- **Start Time** (24-hour format)
- **End Time** (24-hour format)
- **Match Length** (in minutes)
- **Number of Venues Available**
- **Rest Period** (in minutes)

Example Input File for Soccer Round-Robin Tournament (.txt file):

```
1                          -- round-robin type (1 indicates single round robin)
TEAM A                  -- list of participants
TEAM B
```

```
TEAM C
TEAM D
TEAM E
END                     -- finish reading list of teams
3                        -- number of days
9:00                    -- start time
17:00                   -- end time
90                      -- match length (in minutes)
2                       -- number of venues available
90                      -- rest period (in minutes)
```

"–" is used to represent comments in the input file.

Valid Solution

Based on the participants, the program will get the number of participants represented by `numParticipants`, and use that value to compute the total number of matches that need to be scheduled during the tournament. To determine the number of matches that will occur, the following formula will be used:

Total number of matches = `(numParticipants`^{2} `- numParticipants) / 2`

Each participant in the tournament will play `numParticipants - 1` matches in a single round-robin tournament. In the case of the soccer tournament example, since there are five teams, there will be a total of ten matches, with each team playing four matches. The following matches will need to be scheduled:

`TEAM A vs TEAM B`

`TEAM A vs TEAM C`

`TEAM A vs TEAM D`

`TEAM A vs TEAM E`

`TEAM B vs TEAM C`

`TEAM B vs TEAM D`

`TEAM B vs TEAM E`

`TEAM C vs TEAM D`

`TEAM C vs TEAM E`

`TEAM D vs TEAM E`

The program must get all the matches that need to be scheduled, then it will take the first match (`TEAM A vs TEAM B`) and place it in the first available spot (`DAY 1, Venue 1, 09:00-10:30`). It will then move on to the next match, where it searches for an available spot in `DAY 1, Venue 1`. Since it is another match played by `TEAM A`, there has to be at least a 90-minute gap after the first match. The program continues scheduling matches, filling up spots in different

venues, but still making sure that the constraints, such as rest periods, are being taken into account.

Example Output for Soccer Round-Robin Tournament (printed onto console):

```
DAY 1:
Venue 1:
 09:00-10:30: TEAM A vs TEAM B
 12:00-13:30: TEAM A vs TEAM C
 15:00-16:30: TEAM A vs TEAM D
Venue 2:
 09:00-10:30: TEAM C vs TEAM D
 12:00-13:30: TEAM B vs TEAM D
 15:00-16:30: TEAM B vs TEAM C

DAY 2:
Venue 1:
 09:00-10:30: TEAM A vs TEAM E
 12:00-13:30: TEAM B vs TEAM E
 15:00-16:30: TEAM C vs TEAM E

DAY 3:
Venue 1:
 09:00-10:30: TEAM D vs TEAM E
```

In the output, notice that:

- No team plays another game until after a minimum of 90 minutes have passed since their previous match has ended, regardless of venue.
- No team has multiple matches on the same day at the same time at different venues.
- Teams can have multiple matches at the same time, but on different days.
- No matches are scheduled on `Day 2 at Venue 2`, hence not printed onto the console.

If there is no possible schedule for the given input, then the console will print:

```
A schedule was not able to be generated based on the input
```

Project Implementations

This project is implemented in both C++ and Haskell, offering two different approaches to solving the tournament scheduling problem.

C++ Implementation

Project Structure and Components

Core Data Structures (`data_structure.h` and `data_structure.cpp`)

- `Time` struct: Represents a specific time of day with hour and minute.
- `Match` struct: Contains details about a single match, including participants, venue, day, start and end times.
- `Tournament` struct: Holds tournament configuration details like participants, number of days, start/end times, match length, venues, and rest period.

Input Handling (`read_input.h` and `read_input.cpp`)

- `read_input()`: Reads tournament configuration from an input file.
- Validates input parameters such as:
  - Tournament type
  - Participant list
  - Number of days
  - Daily start and end times
  - Match length
  - Number of venues
  - Rest period between matches

Scheduling Algorithm (`scheduler.h` and `scheduler.cpp`)

- Implements a backtracking algorithm to schedule matches
- `schedule_matches()`: Initiates the scheduling process
- `solve()`: Recursively attempts to schedule matches respecting constraints
- `is_valid()`: Checks if a match can be scheduled without conflicts

Output Generation (`print_output.h` and `print_output.cpp`)

- `print_schedule()`: Prints the generated match schedule to the console
- Formats output to show matches by day, venue, and time

Main Program (`round_robin.cpp`)

- Handles program execution
- Reads input file
- Generates matchups
- Invokes scheduling and output functions

Haskell Implementation

Project Structure and Components

Core Data Structures

- `Time.hs`: Defines time representation and time calculation functions
- `Match.hs`: Contains match data type and related operations
- `Tournament.hs`: Stores tournament configuration and parameters

Input Handling (`ReadInput.hs`)

- `readInputFile`: Reads and parses tournament configuration from input files
- Validates all input parameters including:
  - Tournament type
  - Participant list
  - Number of days
  - Daily start and end times
  - Match length
  - Number of venues
  - Rest period between matches

Scheduling Algorithm (`Scheduler.hs`)

- Implements a backtracking algorithm using State monad
- `scheduleMatches`: Main scheduling function
- `solve`: Recursive backtracking solver
- `isValid`: Checks for scheduling conflicts
- Manages constraints including:
  - Minimum rest periods
  - Venue time conflicts
  - Participant availability

Output Generation (`PrintOutput.hs`)

- `printSchedule`: Formats and displays the tournament schedule
- Generates console output showing matches by:
  - Day
  - Venue
  - Time slot

Main Program (`RoundRobin.hs`)

- Program entry point
- Coordinates all components:
  - Reads input file
  - Generates match pairings
  - Executes scheduling algorithm
  - Produces final output
- Handles command-line arguments

Scheduling Algorithm

Both implementations use a depth-first search (backtracking) approach to: - Explore possible match assignments - Prioritize venue and time slot filling - Ensure all constraints are met - Backtrack when a valid schedule cannot be found

Error Handling

Both implementations perform comprehensive error checking:

Input Validation

- Invalid tournament type
- Insufficient or excessive participants
- Duplicate team names
- Invalid date/time formats
- End time before start time
- Match length exceeding available time
- Invalid venue count
- Invalid rest period duration

Scheduling Constraints

- Impossible time slot assignments
- Venue double-booking
- Insufficient rest periods between matches
- Participant scheduling conflicts
- Exceeding maximum tournament duration

System Limitations

- Maximum participant limit exceeded
- Maximum match count exceeded
- File I/O errors
- Memory allocation issues (C++)
- Stack overflow (deep recursion cases)

User Feedback

- Clear error messages for invalid inputs
- Notification when scheduling fails
- Success confirmation when schedule generated
- Help text for proper usage

Usage

C++ Implementation

```
./round_robin <input_file>
```

Example:

```
./round_robin input1.txt
```

Haskell Implementation (Same as C++)

```
./round_robin <input_file>
```

Example:

```
./round_robin input1.txt
```

Requirements

For C++ Implementation

- C++ compiler (g++)
- Makefile

For Haskell Implementation

- GHC

Compilation

C++ Implementation

Use the provided Makefile:

```
make
```

Haskell Implementation

To compile the Haskell implementation:

```
ghc -o round_robin RoundRobin.hs
```

Limitations

- The backtracking algorithm may become inefficient for large tournaments
- Suitable for small to medium-sized tournaments
- May require optimization for a large number of participants