Michael Regan : 22112111

CA208 Assignement 2 - Fixture Scheduler

I hereby declare that all work is my own as per DCU academic policy.

Here is my source code

-------------------------------------------------------------------------------------------------------------------------------------------------------------------

%Name: Michael regan

%Studend ID: 22112111

%I declare this work is solely my own according to the DCU plagiarism policy

:- use\_module(library(dicts)).

% Defineing the groups of NFL teams

groups([

group(a, [chiefs, buccaneers, packers, bills, ravens]),

group(b, [saints, rams, seahawks, fortyniners, steelers]),

group(c, [titans, dolphins, cardinals, cowboys, chargers]),

group(d, [vikings, falcons, eagles, raiders, panthers]),

group(e, [giants, broncos, bengals, texans, lions]),

group(f, [browns, coltsn, patriots, jets, jaguars])

]).

% Check if two teams are in the same group

sameGroup(Home, Away) :-

member(group(\_, Teams), \_),%getting a list of teams where home and away are both part of

member(Home, Teams), %checking if home is in the same group as away

member(Away, Teams), %checking if away is in the same group as home

Home \= Away. %checking if home and away not the same team

% selects a random group

randomGroup(Group) :-

groups(Groups),

random\_select(group(\_,Group),Groups, \_).

% selects two random teams from a random group

randomTeams(Home, Away) :-

groups(G),

member(group(a, Teams), G),

random\_select(Home,Teams, Rest),

random\_select(Away, Rest, \_).

% creating a fixture between two teams in the same group where day is between 1-20

generateFixture(Fixture):-

randomTeams(Home,Away), %generating random home/away team

random\_between(1,20,Day), %picking a random number between 1 and 20

Fixture = fixture(Home, Away, Day). %creating Fixture

schedule(S) :-

schedule(0, [], S). %starting with an empty list

%when we reach 60 fixtures meaning all teams have been slected

schedule(10, S, S) :-

(scheduleValidator(S)%validating schedule

-> write(S),

!; schedule(0, [], \_)).%starting agian with an empty list

%adding fixtures to schedule

schedule(N, S, \_) :-

generateFixture(Fixture), %generating a fixture

append(S, [Fixture], NewSchedule), %appending Fixture to the list

N1 is N + 1,

schedule(N1, NewSchedule, NewSchedule).

scheduleValidator(S) :-

dict\_create(MyDict, my\_dict, \_{test:[]}),%creating a dict to store the the days each team played

getHome(S, Home),%creating a list of all teams playing Home

getAway(S,Away),%creating a list of all teams playing Away

homeAway(Home,Home,Away),%making sure every team plays an equal amount of home/away fixtures

myAppend(Home,Away,HA), %creating a list of all teams in each fixture

sort(HA,Teams), %removing duplicates to get a list of all teams

addTeamsToDict(Teams,MyDict,NewDict), %Creating a key-value for each team where the key is the team and the value is []

allocateDaysToTeam(S,NewDict,NewDict1),%changing the value associated with each team to the days they played aswell as checking that each team is in there allocated group

validatingRestDays(NewDict1, Teams),%validating each team gets at least 4 rest days between fixtures

getDays(S,Days), %Creating a list of all days where theres a fixture

countAll(Days). %Ensuring there is at most, 3 fixtures a day

% Counting each day in a list of days to ensure there are no more then 3 fixtures a day

countAll([H | T]):-

count(H,[H | T],Count), %counting how many times the head of the list appears

countAll(T),%iterating through days, if were recount a day it will return an icorrect Count however this is ok as we would have already validated if it.

acceptable(Count). %checking there is no more the 3 fixtures a day

% Once we counted every day

countAll([]).

% Checking if the number of rest days is acceptable

acceptable(X) :- X =< 3.%Checking there is only 3 fixtures in a day

%If X is the head of the list

count(X, [X | T], Count) :-

count(X, T, Count1), %continue iterating

Count is Count1 + 1, !. %incrment Count by 1

% If X is not the head of the list

count(X, [\_ | T], Count) :- count(X, T, Count). %continue iterating

% if list is empty return count as 0

count(\_, [], 0).

% Check if each team layed an equal amount of home and away matches

homeAway([H | T], Home, Away) :-

compareHomeAway(H, Home, Away), %Comparing that each element in Home appeared the same amount of times in home and away

homeAway(T, Home, Away). %iterates through the first list until its empty

% Once we incremented over every team in home

homeAway([],\_,\_).

%Comparing if X appears equally in Home and Away

compareHomeAway(X, Home, Away):-

count(X, Home, Count1), %counting how many times X appears in Home

count(X , Away, Count2), %counting how many times X appears in Away

Count1 = Count2, %checking if the counts are equal

!.

% Getting a list of each team who played home

getHome(S, Home) :-

maplist(getFirstArgument, S, Home).%passing each fixture in S through getFirstArgument to get a list of the home teams

% Returning the first argument in fixture()

getFirstArgument(fixture(Home,\_,\_),Home).

% Getting a list of each team who played away

getAway(S, Away) :-

maplist(getSecondArgument, S, Away).%passing each fixture in S through getSecondArgument to get a list of away teams

% Returning the second argument in fixture()

getSecondArgument(fixture(\_,Away,\_),Away).

% Getting a list of the days each fixture is on

getDays(S, Days) :-

maplist(getThirdArgument, S, Days).%passing each fixture in S through getSecondArgument to get a lis of away teams

% Returning the third argument in fixture

getThirdArgument(fixture(\_, \_, Day), Day).

% Adding an element to the top of a list

addList(X, L, [X | L]).

% myAppend function from lab3 combines two lists together

myAppend([H | T], L, [H | L3]) :- myAppend(T,L,L3).

myAppend([],L, L).

% Creating a key-value pair in a dictionary for each team

addTeamsToDict([H | T],Dict, NewDict):-

put\_dict(H, Dict, [], NewDict1), %creating a key-value pair for each team where the team is the key

addTeamsToDict(T,NewDict1,NewDict). %iterating for every team

% When there is only one team left in the list

addTeamsToDict([H | []],Dict, NewDict):- put\_dict(H,Dict, [], NewDict).%return NewDict

% Getting every argument from fixture()

getAllArguments(fixture(Home,Away,Day),Dict,NewDict) :-

sameGroup(Home,Away), %checking each Team is competing against a team in its allocated group

addDaysToDict(Home,Day,Dict,NewDict1), %adding the day the the value associated with the team playing home

addDaysToDict(Away,Day,NewDict1,NewDict), %adding the day to the value associated with the team playing away

!.

% Updating a dictionary of the teams so that there values are a list of days where they had a fixture

allocateDaysToTeam([H | T], Dict, NewDict) :-

getAllArguments(H,Dict,NewDict1), %passsing the fixture to getAllArguments

allocateDaysToTeam(T,NewDict1,NewDict). %iterating through all of the fixtures

allocateDaysToTeam([H | []], Dict, NewDict) :-

getAllArguments(H,Dict,NewDict). %passing the last fixture in S to getAllArguments

% updating the value of a key-value pair by appending a day to the value

addDaysToDict(Team,Day,Dict,NewDict):-

get\_dict(Team, Dict, Value), %getting the list of asscoiated with the team

addList(Day,Value,Result),%adding the next day they played

put\_dict(Team, Dict, Result, NewDict).%updating the dict with the updated list of days played

% validating each team has 4 rest days between fixtures

validatingRestDays(Dict,[H | T ]):-

get\_dict(H, Dict, Value), %getting the final list of the days the team played

sort(Value,Days), %sorting the days in order

validRestDay(Days), %validating each team gets at least 4 rest days between their fixtures

validatingRestDays(Dict,T). %iterating for each team

% Team list is empty

validatingRestDays(\_,[]).

% Taking a list and making sure each element in the list is =< by 4 then the previous element

validRestDay([Day1, Day2 | T]) :-

Test is Day1+4, %declarng test

Test =< Day2, %ensuring there is at least 4 rest days

validRestDay([Day2 | T]), %iterating through each day

!.

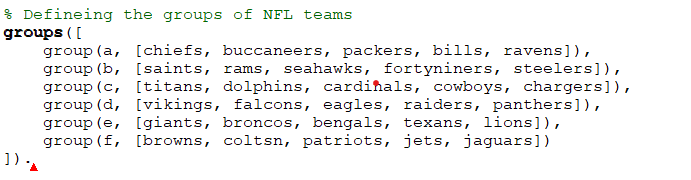
%when we reach the last day we return true

validRestDay([ \_ | []]).

-------------------------------------------------------------------------------------------------------------------------------------------------------------------

Schedule/1 Produces a valid schedule S

It does this by generating a random list of fixtures using generateFixture/1 where every fixture has a Home team, Away Team and day. E.g. Fixture(chiefs,buccaneers,1).

group contains a list of 5 NFL teams and groups contains 6 groups . 

generateFixture/1 uses randomTeams/2 which uses random Group/1



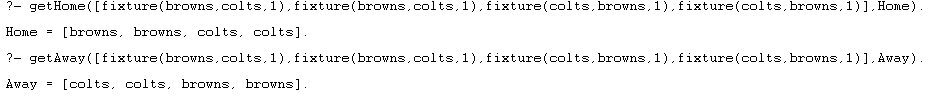




Random fixtures are then added to S until it reaches a length of 60 it then is ran through scheduleValidator/1 and if S is valid, S is printed and true is returned, else S is made [] and the process of generating fixtures starts over again.

scheduleValidator/1 accepts a schedule S and returns true if S is valid. A valid schedule has each team play an equal amount of home and away fixtures. Every fixture must be between two teams in the same group. There must be no more than 3 fixtures per day and every team needs at least 4 days of rest before fixtures.

scheduleValidator/1 first creates a dictionary to store each team as a key and the days it plays as the value.

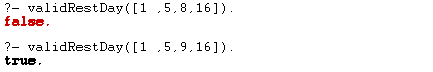
getHome/2 and getAway/2 both use maplist/3 to get a list of the first arguments and second arguments of the fixtures in S.

myAppend/3 from lab3 and sort/2 are used to create a full list of every team competing for use as keys in the dictionary in addTeamstoDict/3.

The values of the dictionary are updated using allocateDaystoTeam/3 which is called recursively for every fixture in S and updates the dictionary using getAllArguments/3.

getAllArguments/3 ensures that the Home and Away team are in the same group using sameGroup/2 and then updates both of the values in the dictionary to the Day using addDaysTDict/4.

validatingRestDays/2 accepts the dictionary and a list of teams and iterates through each team in the dictionary, sorting the days in order and then using validRestDay /1 to ensure there is at least 4 days between fixtures.



getDays/2 is used similarly to getHome/2 and getAway/2 to produce a list of all the third arguments of the fixtures in S.

Then countAll/1 is used to produce a count of every day to ensure that there is no more then 3 fixtures a day

Graphical user interface

Description automatically generated with low confidence

