Burak Furkan Akşahin

151044094

CSE341

Programming Languages HW 2

Q1 & Q2) Source codes of Q1 and Q2 in the 151044094_part1.

For route predicate, i wrote extra 3 big predicates named connected, path and dublicate_control.

Connected covers flight(istanbul,izmir,C) – flight(izmir,İstanbul,C), conflict.

Route6 works like bfs, looks for connected citites if these cities are not visited call route6 recursively and look for other cities until we found our destination. While searching our destination, cost is calculated too. So all paths can be found, and their costs.

Dublicate_control is a tool i wrote that sorts a list. After sorting dublicate elements will dissapear so we can check if there is a dublicate element in this list.

To print all routes from X to Y -> route(X,Y,C). (press enter)

X = bla Y = bla C = bla, (press n)

For croute predicate, i used my route predicate. Took

After getting all paths, my minimum predicate look for routes and get the minimum cost path for croute predicate.

Q3) Source code of Q3 in the 151044094_part3.

When, where and enrollment rules are given by me. Extra sessions and students were added.

Schedule predict, combines enrollment,when and where predicates as student given to enrollment predicate and this returns all courses of this student. When predicate takes all courses and return all course times and where predicate gives places.

Usage predict, combines where and when for given place. Course can be found by where, and times can be found by where.

Conflict predict looks for places of these given courses. If their places are the same look for times too. If their time difference is less than 2 hours there is a conflict return True or False.

Meet predict looks for students Schedule. If they can be in the same place, look for their course times, If the difference is less than 2 hours they can meet return True or False.

Q4) Source code of Q4 in the 151044094_part4.

Intersect and union work pattern are the same except starting input of subresult and appending strategy.

Intersect: Starting subresult is []

Look head of the one list if the head element is in the other list it is a intersect. Append this to subresult. Call intersect again with tail now until tail’s size is 0.

Union: Starting subresult is L1

Look head of the L2 if the head element is not in the L1 we must append it to subresult. Call union again for tail until tail’s size is 0.

Flatten:

Look head of the list, if head is not a list append it to our subresult and call flatten again for tail until tail’s size is 0. But if head is a list, we must flatten it first to append it our result. To do that i called flatten with our head get its flatten version, and call flatten with tail and this flatten version.