Домашна 2

Данило Најков 206033

Задача 1

а)

Идејата е да се соберат сите деца во една фамилија кои што се родени во различен град, да се добие една листа со сите и таа да се собере за да се добие целосен резултат.

```
rodeni_razlicen_grad(S) :-
    findall(S, roden_razlicen_grad(S), L),
    soberi_lista(L,S).
```

Овде се повикува **roden_razlicen_grad**, кој само зема familija и повикува **semejstvo_rodeni**. Вака на секое next би се земала различна фамилија и така се земаат сите со findall во претходниот предикат

```
roden_razlicen_grad(S) :-
    familija(T,M,B),
    semejstvo_rodeni(T,M,B,S).
```

semejstvo_rodeni проверува за секое дете во листата дали е исполнет условот од задачата дефиниран во предикатот **uslov**. Има краен случај кога целата листа е помината и тогаш се поставува првичната вредност на деца родени во различен град на 0. Потоа при враќањето од рекурзијата се проверува условот, и ако е исполнет се зголемува вредноста за 1.

```
semejstvo_rodeni(_,_,[],0).
semejstvo_rodeni(T,M,[K|B],SO) :-
    semejstvo_rodeni(T,M,B,SO),
    not(uslov(T,M,K)).

semejstvo_rodeni(T,M,[K|B],S) :-
    semejstvo_rodeni(T,M,B,SO),
    uslov(T,M,K),
    S is SO + 1.
```

Во **uslov** е дефиниран условот на задачата, односно се преверува дали и за двајцата родители важи дека се родени во различен град од детето

```
uslov(T,M,K) :-
   lice(T,_,_,_,_,TR,_),
   lice(M,_,_,_,_,MR,_),
   lice(K,_,_,_,_,DR,_),
   DR \= MR,
   DR \= TR.
```

Исто така го напишав и предикатот кој едноставно ги собира сите елементи од листата рекурзивно. Овој предикат се користи повторно и во други задачи подоле

```
soberi_lista([],0).
soberi_lista([X|L],K) :-
   soberi_lista(L,KR),
   K is KR + X.
```

б)

Идејата е да се најдат сите предци рекурзивно па потоа да се исфилтрираат според условот. Тоа всушност го прави предикатот **predci**. Тука се користи предикатот и **dodadi** од аудиториските вежби.

```
predci(X,LR) :-
   predci_site(X,L),
   predci_filter(X,L,LR),
   !.
```

Во **predci_site**, најпрвин се наоѓа мајката и таткото на детето, и и за мајката и за таткото, се повикува рекурзивно предикатот за да се најдат и нивните предци. Одкога ќе се најдат, се собираат сите во резултантната листа L, со сите предци на лицето.

```
predci_site(X,[]) :- not(najdi_majka_tatko(X,_,_)).
predci_site(X,L) :-
   najdi_majka_tatko(X,M,T),
   predci_site(M,LM),
   predci_site(T,LT),
   dodadi([M|LM],[T|LT],L).
```

За да се најдат мајката и таткото се зема произволна фамилија, и се проверува се додека не се најде лицето во листата на деца. Ако не се најде во првата унификација, prolog ќе направи redo і ќе се поминат сите фамилии се додека не се најде лицето. Тоа го прават предикатите **najdi_majka_tatko** и **najdi_familija**.

```
najdi_majka_tatko(X,M,T) :-
```

```
    familija(M,T,L),
    najdi_familija(X,L).

najdi_familija(X,[X|_]).
najdi_familija(X,[_|B]) :- najdi_familija(X,B).
```

Одкога ке се најдат сите предци, ги филтрирам со **predci_filter** во кој за секој елемент се проверува најпрвин дали дали предокот е ист пол со лицето. Ако не е не се додава во листата при враќањето од рекурзијата. Датумот на раѓање се проверува во **proveri_raganje**, во кој се пресметува вкупниот број на денови на кои што е родени лицата (месец * 12 + ден), и се гледа дали абсолутната разликата е помала од 7. Исто така додаден е услов кој проверува дали разликата е околу нова година, за да може да се фати и тој случај.

```
predci_filter(_,[],[]).
predci_filter(X, [P|L], [P|LR]) :-
    predci_filter(X,L,LR),
    lice(X,_,_,XPol,datum(XDen,XMesec,_),_,_),
    lice(P,_,_,XPol,datum(PDen,PMesec,_),_,_),
    proveri_raganje(XDen, XMesec, PDen, PMesec).

predci_filter(X, [P|L], LR) :-
    predci_filter(X,L,LR),
    lice(X,_,_,_,_,_,_),
    lice(P,_,_,_,_,_,_).

proveri_raganje(XDen,12,PDen,1) :-
    XVal is XDen,
    PVal is 30 + PDen,
    Diff is PVal - XVal,
    Diff =< 7,
    Diff >= -7.
proveri_raganje(XDen,1,PDen,12) :-
    XVal is 30 + XDen,
    PVal is PDen,
    Diff is PVal - XVal,
    Diff =< 7,
    Diff >= -7.
proveri_raganje(XDen, XMesec, PDen, PMesec) :-
    XVal is XMesec*30 + XDen,
    PVal is PMesec*30 + PDen,
    Diff is PVal - XVal,
    Diff =< 7,
    Diff >= -7.
```

Задача 2

а)

Во оваа задача ги користам **dolzina**, **clen** и **otstrani_duplikati** од аудиториските вежби. Идејата е да се најдат сите бреови заедно со вкупната комуникација остварена со сите други броеви, да се најде максимум и да се врати името и презимето на лицето со тој телефон. Тоа го прави предикатот **najbroj. modificiran_maksimum** е сличен со предикатот од аудиториски но сега прима листа од листи, каде секоја листа е торка (Х, број_комуникација). Така ја враќа торката со најголем број_комуникација.

```
najbroj(X,Y):-
   findall([B,BP], broj_povikuvanja(B,BP), L),
   modificiran_maksimum(L,[B,_]),
   telefon(B,X,Y,_).

modificiran_maksimum([X|O],M) :- modificiran_maks(O,X,M),!.
modificiran_maks([[_,X]|O],[YBr,Y],M) :- X<Y, modificiran_maks(O,[YBr,Y],M).
modificiran_maks([[XBr,X]|O],[_,Y],M) :- X>=Y, modificiran_maks(O,[XBr,X],M).
modificiran_maks([ ],M,M).
```

broj_povikuvanja ги собира сите појдовни и дојдовни повици, ги отстранува дупликатите (бидејќи не сакаме ист број да се број 2 пати), и пресметува должина на резултантната листа. Така добиваме со колку броеви бил во контакт влезниот број.

```
broj_povikuvanja(T,B):-
   broj_povikuvanja_pojdovni(T,BP),
   broj_povikuvanja_dojdovni(T,BD),
   dodadi(BP,BD,BTemp),
   otstrani_duplikati(BTemp, BOut),
   dolzina(BOut, B).
```

broj_povikuvanja_pojdovni ја зема листата на појдовни повици, и со **mapiraj_vo_broj**, се мапира од предиктатот **povik** во самиот број (не ни е потребно времетраењето). Така враќа листа со броеви од појдовни повици.

```
broj_povikuvanja_pojdovni(T, LR) :-
   telefon(T,_,_,L),
   mapiraj_vo_broj(L,LR).

mapiraj_vo_broj([],[]).
mapiraj_vo_broj([povik(BR,_)|L],[BR|LR]) :- mapiraj_vo_broj(L,LR).
```

broj_povikuvanja_dojdovni ги наоѓа сите броеви во чија листа на повици е влезниот број. Тоа се прави преку **findall** и **broj_povikuvanja_dojdovni_posebno** кој проверува дали еден број е во листата на појдовни повици на друг број и ако да го враќа тој број. Проверката за членство се прави преку **modificiran_clen** бидејќи елементите на листа со повици се **povik** а не број.

```
broj_povikuvanja_dojdovni(T, L) :-
   findall(B, broj_povikuvanja_dojdovni_posebno(T,B), L).

broj_povikuvanja_dojdovni_posebno(T,BT) :-
   telefon(BT,_,_,L),
   modificiran_clen(T,L).

modificiran_clen(X,[povik(X,_)|_]) :- ! .
modificiran_clen(X,[_|L]) :- modificiran_clen(X,L).
```

б)

Главната идеја е за секој број да се најдат сите броеви со нивниот пресметан score и да се најде максимум. Тоа е направено во **omilen**, и повторно се користи **modificiran_maksimum** од минатата задача.

```
omilen(X,Y) :-
   findall([YTemp,Sum], omilen_posebno(X,YTemp,Sum), LOut),
   modificiran_maksimum(LOut, [Y,_]).
```

omilen_posebno го пресметува пресметува score-от меѓу два броја како сума од појдовни повици, дојдовни повици, пратени смс и примени смс. За sms се користи **findall** за да се најдат сите пратени и примени пораки (може еден број да пратил повеќе пати на друг, или да примил повеќе пати од еден) и се сумира score-от. За повиците не е потребно да се прави **findall** бидејќи не можат да се појават повеќе од еднаш според условите на задачата. Исто така нашишав и предикат **soberi** кој само ги сумира сите вредности од една листа.

```
omilen_posebno(X,Y,Sum) :-
   telefon(Y, _, _, _),
   omilen_pojdoven(X,Y,PSum),
   omilen_dojdoven(X,Y,DSum),
   findall(S,omilen_sms_sender(X,Y,S),LOut),
   soberi(LOut, SSSum),
   findall(Sk,omilen_sms_reciever(X,Y,Sk),LkOut),
   soberi(LkOut, SRSum),
   Sum is PSum + DSum + SSSum + SRSum.
soberi([],0).
```

```
soberi([X|L],Sum) :-
   soberi(L,SumO),
   Sum is SumO + X.
```

И **omilen_pojdoven** и **omilen_dojdoven** внатре го користат предикатот **omilen_povik_vnatre_lista** кој проверува дали одреден број е присутен во повиците на другиот број. Ако бројот е најден се става вредноста на траењето на бројот, а ако не тогаш се става 0. Единствената разлика кај **omilen_pojdoven** и **omilen_dojdoven** е тоа што во едниот случај се земаат повиците што ги направил X а во другиот повиците што ги направил Y. Така добиваме информации за сите повици меѓу X и Y.

```
omilen_pojdoven(X,Y,PSum) :-
   telefon(X,_,_,L),
   omilen_povik_vnatre_lista(L,Y,PSum),
   !.

omilen_dojdoven(X,Y,DSum) :-
   telefon(Y,_,_,L),
   omilen_povik_vnatre_lista(L,X,DSum),
   !.

omilen_povik_vnatre_lista([],_,0).
omilen_povik_vnatre_lista([povik(Y,Val)|_], Y, Val).
omilen_povik_vnatre_lista([povik(_,_)|L], Y, Val) :-
   omilen_povik_vnatre_lista(L,Y,Val).
```

Идејата за SMS е идентична со тие за повици. Има **omilen_sms_sender** и **omilen_sms_reciever** каде единствената разлика е кој пратил и кој примил порака. **omilen_sms_vnatre_lista** е исто така слично со предикатот **omilen_povik_vnatre_lista**, само што тука не го враќаме времетраењето на повикот, туку 100 како што е дефинирано во задачата.

```
omilen_sms_sender(X, Y, SSSum) :-
   sms(X,L),
   omilen_sms_vnatre_lista(L,Y, SSSum).

omilen_sms_reciever(X, Y, SSSum) :-
   sms(Y,L),
   omilen_sms_vnatre_lista(L,X, SSSum).

omilen_sms_vnatre_lista([],_,0).
omilen_sms_vnatre_lista([Y|_], Y, 100).
omilen_sms_vnatre_lista([_|L], Y, Val) :-
   omilen_sms_vnatre_lista(L,Y,Val).
```

Задача 3

а)

За оваа задача го дефинирав предикатот **izbroj_lokacija** кој со помош на findall, наоѓа колку пати била таа локација почетна или крајна во рамки на еден клиент, па потоа ги прави сума на резултантната листа од сите клиенти, за да ја најде вкупната бројка.

```
izbroj_lokacija(X,Br) :-
   findall(N, izbroj_lokacija_posebno(X,N), LOut),
   soberi_lista(LOut, Br).
```

izbroj_lokacija_posebno во секое извршување ке земе еден клиент, и со **izbroj_pojavuvanja**, ќе изброи колку пати се појавил X во листата на услуги

```
izbroj_lokacija_posebno(X,Br) :-
   klient(_,_,_,Uslugi),
   izbroj_pojavuvanja(X,Uslugi, Br).
```

izbroj_pojavuvanja ја поминува листата со услуги при што се дефинирани 3 услови

- Еднаш кога X се појавува како почетна локација - зголеми го бројачот
- Еднаш кога X се појавува како крајна локација - зголеми го бројачот
- Еднаш кога X не е најден во погорните услови - остави го бројачот ист.

Крајниот чекор на рекурзијата го иницијализира бројачот на 0, и со враќањето од рекурзијата според горонаведените услови се добива бројка за листата дадена како влезен аргумент.

```
izbroj_pojavuvanja(_,[],0).
izbroj_pojavuvanja(X, [usluga(X,_,_,_,_) | L], Br) :-
   izbroj_pojavuvanja(X, L, BrO),
   Br is BrO + 1, !.
izbroj_pojavuvanja(X, [usluga(_,X,_,_,_) | L], Br) :-
   izbroj_pojavuvanja(X, L, BrO),
   Br is BrO + 1, !.
izbroj_pojavuvanja(X, [usluga(_,_,_,_,_) | L], BrO) :-
   izbroj_pojavuvanja(X, L, BrO), !.
```

б)

Идејата е да се најде за секој клиент колку км поминал (**findall** со **pominati_km_po_user**), и да се најде максимумот **maksimum_km**.

```
najmnogu_kilometri(X,Y) :-
   findall([Id, Km],pominati_km_po_user(Id, Km), LOut),
   maksimum_km(LOut,[IdOut,_]),
   klient(IdOut, X, Y, _).
```

maksimum_km е скоро идентичен со максимумот од аудиториските вежби само што дополнително се одредува максимум на торката [KlientId, PominatiKm], за потоа да се земе името и презимето на клиентот со најмногу km.

```
maksimum_km([[Id,Broj]|O],M) :- maks_km(O,[Id,Broj],M),!.
maks_km([[_,Broj1]|O],[Id2,Broj2],M) :- Broj1=<Broj2, maks_km(O,[Id2,Broj2],M).
maks_km([[Id1,Broj1]|O],[_,Broj2],M) :- Broj1>Broj2, maks_km(O,[Id1,Broj1],M).
maks_km([ ],M,M).
```

pominati_km_po_user зема еден клиент и го повикува **suma_od_km**, за го најде вкупниот број поминати километри за тој клиент.

```
pominati_km_po_user(Id, Km) :-
   klient(Id, _, _, Uslugi),
   suma_od_km(Uslugi, Km).
```

Идејата за наоѓање на поминатиот пат во км, е да се најдат сите патишта, и најкраткиот да се земе. Тоа го прави **suma_od_km**, која со **findall** na **presmetaj_dolzina**, ги наоѓа сите патишта помеѓу две точки и **najdi_najkratko** е само едноставен минимум за да се земе најмалото растојание.

```
suma_od_km([],0).
suma_od_km([usluga(A,B,_,_,_) | L], Km) :-
   suma_od_km(L, KmOut),
   findall(Dolzina,presmetaj_dolzina(A,B,Dolzina,[]),LOut),
   najdi_najkratko(LOut,100000,Min),
   Km is Min + KmOut.

najdi_najkratko([],Min,Min).
najdi_najkratko([X|L], Min, MinOut) :-
   X < Min,
   najdi_najkratko(L, X, MinOut).
najdi_najkratko([X|L], Min, MinOut) :-
   X >= Min,
```

```
najdi_najkratko(L, Min, MinOut).
```

Првите 2 услови во **presmetaj_dolzina**, е ако постои директен пат помеѓу A и B, притоа бидејќи се смета дека патиштата се двонасочни, се зема и растојанието ако постои и од B до A.
Во другите случаеви, се проверува дали патот што се испитува е веќе проверен. (оваа информација се чува во листата Videno, и со секое рекурзивно повикување провереният пат се додава во неа за да се избегне бесконечен рекурзивен циклус). Потоа се зема растојанието до некое проезволно C кое што пролог го унифицира во некоја точка во графот и рекурзивно се проверува растојанието од таа точка до B. Првичното растојание се сумира со рекурзивниот најден пат за да се дојде до вкупното растојание. Последното дефинирање на предикатот е за да се провери и во другата насока (не само A->C, туку и C->A)

```
presmetaj_dolzina(A, B, Km, _) :- rastojanie(A, B, Km).
presmetaj_dolzina(A, B, Km, _) :- rastojanie(B, A, Km).
presmetaj_dolzina(A,B, Km, Videno) :-
    not(clen([A,B], Videno)),
    rastojanie(A,C,KmD),
    A \=B,
    presmetaj_dolzina(C,B,KmO, [[A,B]| Videno]),
    Km is KmD + KmO.
presmetaj_dolzina(A,B, Km, Videno) :-
    not(clen([A,B], Videno)),
    rastojanie(C,A,KmD),
    A \=B,
    presmetaj_dolzina(C,B,KmO, [[A,B]| Videno]),
    Km is KmD + KmO.
```

в)

Главната идеја во **najmnogu_zarabotil** е:

- Да се најдат сите можни unique таксиња во базата **najdi_site_taksinja**
- За секое да се најде колку заработил **najdi_najvekje_zarabotil**
- Да се најде максимумот на овие со помош на **modificiran_maksimum** дефиниран во втората задача

```
najmnogu_zarabotil(X) :-
    najdi_site_taksinja(L),
    najdi_najvekje_zarabotil(L,LOut),
    modificiran_maksimum(LOut, [X,_]).
```

Сите таксиња се наоѓаат со тоа што од секој клиент се враќа таксињата во услугите на тој клиент (**taksi_po_klient**), се израмнува листата да биде во 1D, и се отстрануваат дупликатите за да се најдат unique таксињата.

```
najdi_site_taksinja(L) :-
   findall(LV, taksi_po_klient(LV), LOut),
   izramni(LOut, LIzramneto),
   otstrani_duplikati(LIzramneto, L).
```

taksi_po_klient зема еден клиент, и со **mapiraj_taksi** се поминува секој елемент од листата од услуги и се враќа бројот на такси во листа.

```
taksi_po_klient(L) :-
   klient(_,_,_,LUslugi),
   mapiraj_taksi(LUslugi, L).

mapiraj_taksi([],[]).
mapiraj_taksi([usluga(_,_,_,_,T)|LUslugi], [T|L]) :-
   mapiraj_taksi(LUslugi, L).
```

najdi_najvekje_zarabotil ја зема листата од такси броеви, и за секој пресметува колку заработил и при вракањето од рекурзијата ја става торката [taksi, zarabotuvacka] во резултантната листа.

```
najdi_najvekje_zarabotil([],[]).
najdi_najvekje_zarabotil([X|L], [[X,Z]|LOut]) :-
   najdi_zarabotuvacka(X, Z),
   najdi_najvekje_zarabotil(L,LOut).
```

najdi_zarabotuvacka го користи **najdi_zarabotuvacka_po_klient** за да најде за секој клиент, колку пари потрошил со тоа такси. Тоа се наоѓа за секој клиент со **findall**, и се сумира за целосна бројка.

```
najdi_zarabotuvacka(X, Z) :-
   findall(ZK,najdi_zarabotuvacka_po_klient(X,ZK),LOut),
   soberi(LOut, Z).

najdi_zarabotuvacka_po_klient(X, Z) :-
   klient(_,_,_,LUslugi),
   soberi_pari_od_uslugi_cut(X,LUslugi, Z).
```

soberi_pari_od_uslugi_cut е само wrapper со cut оператор на **soberi_pari_od_uslugi** за да се врати само првиот резултат (вкупната цена). Без овој услов **soberi_pari_od_uslugi**

враќа повеќе резултати со секое Next, при што така се скокаат некои услуги и цената не е точна. Овде cut операторот е задолжителен.

```
soberi_pari_od_uslugi_cut(X,LUslugi,Z) :-
    soberi_pari_od_uslugi(X,LUslugi,Z),
    !.
```

soberi_pari_od_uslugi го користи **presmetaj_dolzina** од истата задача под б, за да се најде цената на услугата патот се множи со цената по километар. Ова рекурзивно се повторува за секоја услуга од листата за која што е точен условот дека таксито што ја извршила услугата е таксито барано од горниот предикат и датумот е декември 2015, а останатите се скокаат.

```
soberi_pari_od_uslugi(_,[],0).
soberi_pari_od_uslugi(X, [usluga(_,_,_,_,Y) | L], Pari) :-
    X \= Y,
    soberi_pari_od_uslugi(X, L, Pari).
soberi_pari_od_uslugi(X, [usluga(Od,Do,Cena,datum(_,12,2015),X) | L], Pari) :-
    soberi_pari_od_uslugi(X,L,PariRec),
    findall(Dolzina,presmetaj_dolzina(Od,Do,Dolzina,[]),LOut),
    najdi_najkratko(LOut,100000,Min),
    Pari is PariRec + (Min*Cena).
soberi_pari_od_uslugi(X, [usluga(_,_,_,_,X) | L], Pari) :-
    soberi_pari_od_uslugi(X, L, Pari),
    !.
```