ДОМАШНА ЗАДАЧА 2
ЛОГИЧКО ПРОГРАМИРАЊЕ
БАЗИ НА ПОДАТОЦИ ВО PROLOG
Марија Вецовска 185008

Задача 1.

a)rodeni_razlicen_grad(Kolku)

1) **dete(LiceID, familija(T,M,L))** – предикат проверува дали едно лице со id LiceID е дете во фамилијата **familija(T,M,L).**
lice(LiceID,_,_,_,_,_,_) - прво проверува дали постои ова лице,
familija(T,M,L), member(LiceID, L) - потоа ја бара фамилијата која го содржи ова LiceID во нејзината листа од деца.

```
dete(LiceID, familija(T,M,L)) :-
   lice(LiceID,_,_,_,_,_,_), familija(T,M,L), member(LiceID, L).
```

2) **roditeli(LiceID, Tatko, Majka)** - предикат кој за едно лице со id LiceID ги враќа id на неговиот татко, Tatko, и мајка, Majka.

```
roditeli(LiceID, Tatko, Majka) :- dete(LiceID, familija(Tatko,Majka,_)).
```

3) **grad_ragjanje(LiceID, GradR)** - предикат кој за лицето со id LiceID, го враќа неговиот град на раѓање.

```
grad_ragjanje(LiceID, GradR) :- lice(LiceID,_,_,_,_,GradR,_).
```

4) **razlicen_grad_ragjanje_od_roditeli(LiceID)** – проверува дали лицето со id LiceID е родено во различен град од градовите на раѓање на неговите родители.
Прво го бара градот на раѓање на тоа лице G, па родителите T – татко и M – мајка на тоа лице. За T го бара градот на раѓање TG и за M, MG.
Потоа проверува дали G \= TG, G \= MG – дали лецето се родило во различен град од градовите на раѓање на неговите родители.

```
razlicen_grad_ragjanje_od_roditeli(LiceID):-
   grad_ragjanje(LiceID, G), roditeli(LiceID, T,M),
   grad_ragjanje(T, TG), grad_ragjanje(M, MG), G \= TG, G \= MG.
```

5) **rodeni_razlicen_grad(Num)** - враќа колку лица се родени во град различен од градовите на раѓање на двата родителa.

findall(LiceID, razlicen_grad_ragjanje_od_roditeli(LiceID), Lica) – за секое лице во базата проверува дали е точно razlicen_grad_ragjanje_od_roditeli(LiceID), и ако е точно, LiceID се додава во Lica.
length(Lica, Num) – го враќа броојот на таквите лица.

```
rodeni_razlicen_grad(Num) :-
   findall(LiceID, razlicen_grad_ragjanje_od_roditeli(LiceID), Lica),
   length(Lica, Num).
```

б) predci(Sifra,L)

1) **ist_pol(LiceID, PredokID)** – предикат кој проверува дали лицата со ids LiceID и PredokID се од ист пол.
lice(LiceID,_,_,Pol,_,_,_), lice(PredokID,_,_,PolP,_,_,_) - ги наоѓа Pol за LiceID и PolP за PredokID, па проверува дали се работи за ис пол: Pol = PolP, но за различна личност LiceID \= PredokID.

```
ist_pol(LiceID, PredokID) :-
   lice(LiceID,_,_,Pol,_,_,_), lice(PredokID,_,_,PolP,_,_,_),
   Pol = PolP, LiceID \= PredokID.
```

2) **razlika_sedum_dena(datum(Den1,Mesec1,_), datum(Den2,Mesec2,_))** - Предикат кој проверува дали датумите datum(Den1,Mesec1,_) и datum(Den2,Mesec2,_) имаат разллика 7 дена или помалце.

X is Mesec1- Mesec2 – во X се одзимаат месеците.
(1) – ако разликата е 0, се работи за ист месец.
Y is Den1-Den2 – во Y се добива разликата меѓу деновите, ако е помала од |7|, меѓу датумите нема разлика поголема од 7 дена.
(X == 0, Y is Den1-Den2, Y >= -7, Y =< 7).
(2) – ако разликата е 1, Mesec1 доаѓа по Mesec2.
Y is 30-Den2+Den1 – во Y ги собира деновите до завршување на Mesec2(30-Den2) и ги собира со оние поминати од Mesec1. Ако Y е 7 или помало, меѓу датумите нема разлика поголема од 7 дена.
(3) – истото како (2), но ако Mesec1 доаѓа пред Mesec2.
(4) – исто како (2), во случај кога Mesec1 е декември а Mesec2 јануари.
(5) – исто како (3), во случај кога Mesec1 е јануари а Mesec2 декември.

```
razlika_sedum_dena(datum(Den1,Mesec1,_), datum(Den2,Mesec2,_)) :-          (1)
   X is Mesec1- Mesec2,
  (X == 0, Y is Den1-Den2, Y >= -7, Y =< 7).
razlika_sedum_dena(datum(Den1,Mesec1,_), datum(Den2,Mesec2,_)) :-          (2)
   X is Mesec1- Mesec2,
  (X == 1, Y is 30-Den2+Den1, Y =< 7).
razlika_sedum_dena(datum(Den1,Mesec1,_), datum(Den2,Mesec2,_)) :-          (3)
   X is Mesec1- Mesec2,
  (X == -1, Y is 30-Den1+Den2, Y =< 7).
razlika_sedum_dena(datum(Den1,Mesec1,_), datum(Den2,Mesec2,_)) :-          (4)
   X is Mesec1- Mesec2,
  (X == 11, Y is 30-Den2+Den1, Y =< 7).
razlika_sedum_dena(datum(Den1,Mesec1,_), datum(Den2,Mesec2,_)) :-          (5)
   X is Mesec1- Mesec2,
  (X == -11, Y is 30-Den1+Den2, Y =< 7).
```

3) **rodeni_ista_nedela(LiceID, PredokID)** - проверува дали нема растојание поголемо од 7 дена меѓу деновите на раѓање на лицата со ids LiceID и PredokID.

```
rodeni_ista_nedela(LiceID, PredokID) :-
   lice(LiceID,_,_,_,DatumL,_,_),
   lice(PredokID,_,_,_,DatumP,_,_),
   razlika_sedum_dena(DatumL, DatumP),
```

LiceID \= PredokID.

4) **samo_predok(LiceID, PredokID)** – предикат кој проверува дали PredokID е предок на LiceID.
(1) ако PredokID е татко на LiceID, точно.
(2) ако PredokID е мајка на LiceID, точно.
(3) ако (1) и (2) не важат, проверка дали PredokID е предок на таткото на LiceID.
(3) ако (1) ,(2) и (3) не важат, проверка дали PredokID е предок на мајката на LiceID.

```
samo_predok(LiceID, PredokID):- roditeli(LiceID, PredokID, _).                (1)
samo_predok(LiceID, PredokID):- roditeli(LiceID, _, PredokID).                (2)

samo_predok(LiceID, PredokID) :- roditeli(LiceID, Tatko, _),                  (3)
   samo_predok(Tatko, PredokID).
samo_predok(LiceID, PredokID) :- roditeli(LiceID, _, Majka),                  (4)
   samo_predok(Majka, PredokID).
```

5) predok(LiceID, PredokID) - : проверка дали PredokID е предок на LiceID, дали се од истиот пол и дали денот на раѓање на предокот е најмногу една недела пред или после денот на раѓање на лицето.

```
predok(LiceID, PredokID):-samo_predok(LiceID, PredokID),
      ist_pol(LiceID, PredokID), rodeni_ista_nedela(LiceID, PredokID).
```

6) **predci(LiceID, Predci)** – предикат кој на влез прима шифра на некое лице и за тоа лице на излез треба да ги врати сите негови предци (нивните шифри) за кои се исполнети следните услови: предокот да биде од истиот пол како и лицето и денот на раѓање на предокот да биде најмногу една недела пред или после денот на раѓање на лицето (годината е небитна и претпоставете дека сите месеци имаат по 30 денови).

```
predci(LiceID, Predci) :- findall(PredokID, predok(LiceID, PredokID), Predci).
```

Задача 2.

a) najbroj(X, Y)

1)**povikani(X, Pov)**- предикат кој враќа листа од торки Pov во форма: (Br, T), за секој povik(Br, T) на бројот X во листата на повици L.
Со **telefon(X,_,_,L) –** се зима листата L за бројот X.
findall((Br, T), member(povik(Br, T), L), Pov) – за сите повици во L, во Pov се додава торка (Br, T).

```
povikani(X, Pov) :- telefon(X,_,_,L),
   findall((Br, T), member(povik(Br, T), L), Pov).
```

2)**povikal(X, Y, P)** - Предикат кој го враќа (Y, Traenje) ако постои повик од X до Y.
Ова се постигнува така што:
povikani(X, L) – ги враќа сите торки за повици на бројот X кон други броеви во L.
member(povik(Y,Traenje), L) – проверува дали во L има повик до Y со некое Traenje.

```
povikal(X, Y, (Y,Traenje)) :-
      povikani(X, L), member((Y,Traenje), L).
```

3)povikuvaci(X, Pov) - Предикат кој враќа листа на торки: (Br, T) за сите броеви кои го повикале X, каде Br е број кој го повикал X, а T е траењето на повикот.

Анализа на:
findall((Br, T), povikal(Br, X,(X,T)), Pov).
I) **povikal(Br, X,(X,T))** – проверува дали Br го повикал X, ако да, (X, T) го репрезентира повикот.
Кога I) е точно, во листата Pov се зачувува (Br, T).
Findall го прави ова за сите броеви кои го имаат повикано X.

```
povikuvaci(X, Pov) :-
   findall((Br, T), povikal(Br, X,(X,T)), Pov).
```

4)**soberi_vreme(L, P2, P, L1)** – L е листа од торки, P2 е торка: (Br, T2).
Предикатот бара торка (Br, T1) во листата L за во P да врати торка (Br, T1+T2).
Значи, предикатот бара две торки кои претставуваат разговор со ист број и враќа една торка со нивните собрани времиња. Во L1 ја враќа L со отстрането (Br, T1).

(1) – Кога во L ќе се појави торка (Br, T1), (Br, T) се иницијализира во P а остатокот од L се иницијализира во L1, па при рекурзивно враќање се додаваат и останатите членови од L во L1 – без (Br, T1).
(2) – Членот X на L не е торка (Br, T1) , па продолжува да бара рекурзивно. Во L1 се додава членот X од L.

```
soberi_vreme([(Br, T1)|L], (Br, T2), (Br, T), L) :- T is T1+T2.          (1)
soberi_vreme([X|L], (Br, T2), P, [X|L1]) :-
   soberi_vreme(L,(Br, T2), P, L1).                                      (2)
```

5) **spoj(L1, L2, L)** – L1 и L2 се листи од торки, за сите торки во двете листи кои имаат ист број, во L се става торка со тој број и вкупното траење во двете торки, за оние торки кои немаат соодветен пар, се враќаат истите.
(1) – кога ќе се испразни листата L2, значи дека сите торки кои останале во L1 немаат пар во L2, па L ја иницијализира со L1.
(2) - soberi_vreme(L1, X, P, LN) – Ако за торката X има соодветна торка во L1, времињата им се совираат и новата торка се враќа во P. Во LN се враќа L1 со отстранетат торка која соодветствува на X.
P се додава во L и се повикува spoj(LN, L2, L) – L1 се заменува со LN.
(3) – Ако (2) не се извршило, значи за X немало соодветна торка во L1, па X може да се додаде во L.

```
spoj(L1, [], L1).                                                        (1)
spoj(L1, [X|L2], [P|L]) :- soberi_vreme(L1, X, P, LN), spoj(LN, L2, L),  !.   (2)
spoj(L1, [X|L2], [X|L]) :- spoj(L1, L2, L), !.                           (3)
```

6)**razgovaral_so(X, P)** - предикат кој во P ги враќа торките за сите броеви со кои разговарал бројот X, со сумирано време во случај да имало повеќе повици.
povikani(X, Pojdovni) – во Pojdovni ги враќа сите торки за броеви кои ги повикал X.
povikuvaci(X, Dojdovni) – во Dojdovni ги враќа сите торки за броеви кои го повикале X.
spoj(Pojdovni, Dojdovni, P) – ги спојува.

```
razgovaral_so(X, P) :- povikani(X, Pojdovni),
   povikuvaci(X, Dojdovni), spoj(Pojdovni, Dojdovni, P).
```

7) **br_razgovori(L)** - L е листа од торки (X, Len), X е број а Len е бројот на различни броеви со кои има остварено повик X.
(razgovaral_so(X, Br),length(Br, Len)) – за број X во Br се враќа листата од торки за броеви со кои разговарал тој, па во Len се враќа нејзината должина – бројот на броеви со кои разговарал X.
Findall овозможува ова да се изврши за сите броеви во базата и за нив да се зачува торка (X, Len) во L.

```
br_razgovori(L) :-
   findall((X, Len),(razgovaral_so(X, Br),length(Br, Len)), L).
```

8) **sporedi(T1, T2, T)** - Предикат кој споредува торки во форма (X, Y) и го враќа она со поголемо Y. *Ако има повеќе торки со исто Y, ја враќа последната.

```
sporedi((X1, Y1), (_, Y2), (X1, Y1)) :- Y1>=Y2.
sporedi((_, _), (X2, Y2), (X2, Y2)).
```

9) **najmnogu_razgovori(T)** - Предикат кој ја враќа торката T за бројот кој имал најмногу разговори со различни броеви.
(1) – br_razgovori(L) враќа листа од торки за сите броеви во базата,
najmnogu_razgovori(T, L) – ја враќа торката T за бројот кој имал најмногу разговори со различни броеви во листата L.
(2) – кога во L ќе остане еден елемент, T се иницијализира со тој елемент.
(3) – при рекурзивно враќање, спореди ја торката T1 која била најголема во минатиот повик, спореди ја со актуелната торка во овој повик-T2: sporedi(T1, T2, T), и во T врати ја поголемата торка, таа торка проследи ја понатаму.

```
najmnogu_razgovori(T) :- br_razgovori(L), najmnogu_razgovori(T, L).          (1)
najmnogu_razgovori(T, [T]).                                                  (2)
najmnogu_razgovori(T, [T2|L]) :- najmnogu_razgovori(T1, L),                  (3)
   sporedi(T1, T2, T), !.
```

10) **najbroj(X, Y)** - ќе се врати името X и презимето Y за бројот кој остварил комуникација со најмногу други броеви.
najmnogu_razgovori((Br, _)) - (Br, _) е торката за бројот кој остварил најмногу разговори со различни броеви.
telefon(Br,X,Y,_) - во базата го бара записот за Br за да ги зема името X и презимето Y.

```
najbroj(X, Y) :- najmnogu_razgovori((Br, _)), telefon(Br,X,Y,_).
```

б) omilen(X,Y)

11)**pratil_poraka_pom(X, Poraki)** – за еден запис за порака во базата, листата на примачи на пораката L ја трансформира во листа (Br, 100), за секој Br во L и резултатот го враќа во Poraki.

```
pratil_poraka_pom(X, Poraki) :- sms(X,L),
   findall((Br, 100), member(Br, L), Poraki).
```

```
pratil_poraka_pom(X, []).
```

12) **spoj_poraki(L1, L)** – L1 е листа од листи од торки. Овој предикат ја трансформира L1 така што ја претвара во листа од торки. За сите торки (Br, T) кои имаат ист Br во L1, во L се враќа една торка (Br, Tzbir), каде Tzbir е збир на T во сите такви торки.

spoj_poraki(L1, LR) – во LR се враќа трансформираната L1.
spoj(X, LR, L) – листата-елемент X ја спојува со LR, на начин да нема торки со ист Br. Резултатот L го проследува за понатамошна трансформација.

```
spoj_poraki([], []).                                                    (1)
spoj_poraki([X|L1], L) :- spoj_poraki(L1, LR), spoj(X, LR, L),  !.      (2)
```

13) **pratil_poraka(X, L)** – L е листата на торки (Br, T) за сите броеви кон кои пратил порака X. T е A*100, каде A е бројот на пораки кои X му ги пратил на Br.
bagof(Poraki, pratil_poraka_pom(X, Poraki), L1) – во Poraki враќа листа од листи од торки за секој запис за смс кој го пратил X во Л1.
spoj_poraki(L1, L) – соодветно ја трансформира L1 во L.

```
pratil_poraka(X, L):-
   bagof(Poraki, pratil_poraka_pom(X, Poraki), L1),
   spoj_poraki(L1, L).
```

14) **dobil_poraka(X, L)** - L е листата на торки (Br, T) за сите броеви кои му пратиле порака на X. T е A*100, каде A е бројот на пораки кои Br му ги пратил на X.

```
dobil_poraka(X, L) :-
   findall((Br, T), (pratil_poraka(Br, Poraki), member((X, T), Poraki)), L).
```

(pratil_poraka(Br, Poraki), member((X, T), Poraki)) – за бројот Br во базата враќа листа на торки за броеви кон кои пратил порака - Poraki. Пота проверува дали има запис (X, T) во Poraki, т.е. дали Br му пратил порака на X. Ако да, додава (Br, T) во L.

15) **razmenil_poraki(X, L)** - L е листата на торки (Br, T) за сите броеви со кои X разменил порака. T е A*100, каде A е бројот на пораки кои X ги разменил со Br.

```
razmenil_poraki(X, L) :- pratil_poraka(X, Prateni),
   dobil_poraka(X, Primeni), spoj(Prateni, Primeni, L).
```

16) **ostvaril_komunikacija(X, L)** - L е листата на торки (Br, T) за сите броеви со кои X остварил комуникација.
T е мерка за големината на комуникацијата.

```
ostvaril_komunikacija(X, L) :- razgovaral_so(X, Povici),
   razmenil_poraki(X, Poraki), spoj(Povici, Poraki, L).
```

17) **najmnogu_komunikacija(L, T)** – Предикат кој ја враќа торката T од листата L, (Br, T1), која има најголемо L1 во споредба со останатите торки.
*Овој предикат функционира на ист начин како 9)najmnogu_razgovori(T1, L).

```
najmnogu_komunikacija([T], T).
najmnogu_komunikacija([T2|L], T) :- najmnogu_komunikacija(L, T1),
   sporedi(T1, T2, T), !.
```

18) **omilen(X, Y) -** за даден број X проследен на влез ќе го врати неговиот омилен број Y. Омилен број е оној со кој има остварено најголемо вкупно траење на повици (се собираат и појдовни и дојдовни). При пресметувањето на омилениот број секоја СМС порака да се разгледува како остварен повик со траење 100 (оној кој ја испраќа пораката е појдовниот број, оние до кој се испраќа пораката се дојдовни).

ostvaril_komunikacija(X, L) – ги враќа торките за сите броеви со кои комуницирал X во L.
najmnogu_komunikacija(L, (Y, _)) – ја наоѓа најголемата торка во L и ја враќа во Y.

omilen(X, Y) :- ostvaril_komunikacija(X, L), najmnogu_komunikacija(L, (Y, _)).

Задача 3

a) izbroj_lokacija(Lok,Br)

1)site_uslugi(L) - враќа листа L од сите услуги остварени од сите клиенти.
findall(U, klient(_,_,_,U), L1) – за секој клиент од базата на податоци ја зима листата на услуги U и ја става во листа L1.
spoj_listi(L1, L) - затоа што L1 е листа од листи, таа треба да се спои во една листа L.

site_uslugi(L) :- findall(U, klient(_,_,_,U), L1), spoj_listi(L1, L) .

2) spoj_listi(L1, L) – листата од листи L1 ја спојува во една листа – L.
(1) кога листата која треба да се спојува е празна, листата која се враќа исто така се иницијализира на празна листа.
(2) При враќање од рекурзија, L2 е листа од сите споени елементи пред E, па L2 се спојува со E и резултатот се враќа во L. Потоа L се проследува понатаму за да се спои со елементот пред E.

```
spoj_listi([], []).                                          (1)
spoj_listi([E|L1], L) :- spoj_listi(L1, L2), append(L2, E, L).   (2)
```

3) izbroj_lokacija(Lok,Br) - предикат кој за локација Lok која се задава на влез ќе пресмета колку пати таа локација била почетна или крајна за некоја услуга.

site_uslugi(L) - во L ги враќа сите услуги за сите клиенти.
findall(Lok, member(usluga(Lok,_,_,_,_), L), L1), length(L1, Br1) – со memeber() се пристапува до една услуга-член на L, usluga(Lok,_,_,_,_) е точно кога Lok се совпаѓа со појдовната локацијата во таа усуга, и Lok се зачувува како член на L1.
Должината на L1 - Br1 е број на појавувања на Lok како појдовна локација во услуги.
findall(Lok, member(usluga(_,Lok,_,_,_), L), L2), length(L2, Br2) – исто ккао погоре, со тоа што Lok претставува завршна локација и Br2 е број на појавувања на Lok како завршна локација во услуги.
Br is Br1+Br2 – Br е бројот на појавувања на Lok како појдовна или завршна локација.

```
izbroj_lokacija(Lok,Br) :-
   site_uslugi(L),
   findall(Lok, member(usluga(Lok,_,_,_,_), L), L1), length(L1, Br1),
   findall(Lok, member(usluga(_,Lok,_,_,_), L), L2), length(L2, Br2),
   Br is Br1+Br2.
```

+ Најкраток пат од а до б во граф
najkratok_pat(P,K,Y)

4) pokratok((X1, Y1), (X2, Y2), (X, Y)) – во (X, Y) ја враќа торака со поголемо Y.

```
pokratok((X1, Y1), (_, Y2), (X1, Y1)) :- Y1<Y2.
pokratok((_, _), (X2, Y2), (X2, Y2)).
```

5)najkratok(L, X) – во X ја враќа торката (A, B) која има најмало B во листата торки L.

(1) кога во L има една торка, таа има најмало B.
(2) во Y1 се враќа најголемата торка од минатиот повик, се споредува со торката X и во Y се враќа пократката торка, која се проследува понатака.

```
najkratok([X], X).                                          (1)
najkratok([X|L], Y) :- najkratok(L, Y1), pokratok(X, Y1, Y).   (2)
```

6)dfs(P,K,T,Pat,R) – Наоѓа пат во граф од P до K, T е променлива која чува листа на веќе поминати темиње, Pat е листа од сите изминати темиња од P до K, а R е должината на тој пат.

(1) – кога ќе се повика предикатот со ист почеток и крај, Pat иницијализирај го со T – листа на поминати темиња, а должината се иницијализира на 0 за при враќање да се пресмета.
(2) – најди во базата ребро со почеток во P и крај во некое X, провери дали X се појавува во листата на посетени темиња, ако не(Значи нема да се створи циклус) додај го X на листата на поминати темиња T и повикај го dfs() со почетно теме X. При враќање од рекурзија се пресметува должината на патот: R is R1+RX, R1 е должината од минатиот повик а RX е должината на реброто кое се разгедува во овој повик.
(3) – исто како **(2)**, тука се применува својството дека ребрата се ненасочени:
rastojanie(P,X,RX) е истото ребро со rastojanie(X,P,RX).

```
dfs(K,K,T,T,0).                                                    (1)
dfs(P,K,T,Pat,R) :- rastojanie(P,X,RX), not(member(X,T)), append(T, [X], T1),
   dfs(X,K,T1,Pat,R1), R is R1+RX.                                 (2)
dfs(P,K,T,Pat,R) :- rastojanie(X,P,RX), not(member(X,T)), append(T, [X], T1),
   dfs(X,K,T1,Pat,R1), R is R1+RX.                                 (3)
```

7) najkratok_pat(P,K,Y) – го наоѓа најкраткиот пат со почеток во P и крај во K во графот и неговата должина ја враќа во Y.
findall((Pat, R), dfs(P,K,[P],Pat,R), O) – со помош на дфс преберување ги наоѓа сите можни патишта од P до K, и за секој ваков пат во листата O зачувува торка (Pat, R), каде Pat е листа од темињата кои се изминуваат, а R е должина на патот.
najkratok(O, Y) – ја наоѓа торката со најмала должина.

```
najkratok_pat(P,K,Y) :-
   findall((Pat, R), dfs(P,K,[P],Pat,R), O), najkratok(O, Y), !.
```

б)najmnogu_kilometri(X,Y)

8)sum(L, S) – во S се враќа сума од сите елементи на листата L.

```
sum([], 0).
sum([X|L], S) :- sum(L, S1), S is S1+X.
```

9) klient_kilometri(ID, Y) - за клиентот со ид ID го враќа бројот на километри кој го има поминато возејќи се со такси – Y.
klient(ID,_,_,U) – ја зима листата на услуги за тој клиент во U.
findall(D, (member(usluga(P,K,_,_,_), U), najkratok_pat(P,K,(_, D))), All) – за секоја услуга на тој клиент – една услуга се добива со member(), со предикатот najkratok_pat() ја зима должината на патот кој е поминат при услугата. Сите вакви должини ги става во листата All.
sum(All, Y) – ги собира во Y должините за сите услуги кои ги користел клиентот.

```
klient_kilometri(ID, Y) :- klient(ID,_,_,U),
   findall(D, (member(usluga(P,K,_,_,_), U), najkratok_pat(P,K,(_, D))), All),
   sum(All, Y).
```

10) najmnogu(L , X) – предикат кој ја враќа торката X (има форма (A, B)) за онаа торка во L која има најголемо B.

```
pogolemo((X1, Y1), (_, Y2), (X1, Y1)) :- Y1>=Y2.
pogolemo((_, _), (X2, Y2), (X2, Y2)).

najmnogu([X], X).
najmnogu([X|L], Y) :- najmnogu(L, X1), pogolemo(X, X1, Y), !.
```

11)najmnogu_kilometri(K) - го враќа Idto – K на клиентот кој има поминато најмногу километри со такси компанијата .
findall((ID, Y), klient_kilometri(ID, Y), L) – за секој клиент во базата во листата L се зачувува торка (ID, Y), каде Y е бројот на километри кој тој го има поминато возејќи се со такси.

najmnogu(L, (K, _)) - ја враќа торката од L која има најголемо Y – таа го претставува клиентот кој има поминато најмногу километри со такси компанијата

```
najmnogu_kilometri(K) :- findall((ID, Y), klient_kilometri(ID, Y), L),
   najmnogu(L, (K, _)), !.
```

12) najmnogu_kilometri(X,Y) - името и презимето на клиентот кој има поминато најмногу километри со такси компанијата.

```
najmnogu_kilometri(X,Y) :- klient(Z,X,Y,_), najmnogu_kilometri(Z), !.
```

в) najmnogu_zarabotil(X)

13) uslugi_dekemvri2015(L) - ги враќа сите услуги на сите клиети од базата кои се случиле во декември 2015.

findall(U, klient(_,_,_,U), LU) – во LU се сместуваат листи од услуги за секој клиент посебно.
spoj_listi(LU, L1) – Листата од листи LU се спојува во листа L1.
findall(usluga(P,K,C,datum(Den,12,2015),D),
member(usluga(P,K,C,datum(Den,12,2015), D), L1), L) – од L1 се зимаат сите улуги чиј датум е во формат datum(Den,12,2015) и се сместуваат во L.

```
uslugi_dekemvri2015(L) :- findall(U, klient(_,_,_,U), LU),
   spoj_listi(LU, L1),
   findall(usluga(P,K,C,datum(Den,12,2015),D),
         member(usluga(P,K,C,datum(Den,12,2015), D), L1), L).
```

14) uslugi_cena(L1) - враќа листа од торки (Br, Len) – каде Br е број на возило а Len е должина на рутата. За секоја услуга извршена во декември 2015 се генерира ваква торка.

member(usluga(P,K,C,_,Br), L), najkratok_pat(P,K,(_, Y)), Len is Y*C – за една услуга usluga(P,K,C,_,Br) од листата L, najkratok_pat(P,K,(_, Y)) во Y ја враќа должината на патот за таа услуга, а со Len is Y*C во Len се пресметува цената на услугата (Y е километри, C е цена за еден км).

findall((Br, Len),
(member(usluga(P,K,C,_,Br), L), najkratok_pat(P,K,(_, Y)), Len is Y*C),
L1). - за секоја услуга во L, генерира торка **(Br, Len) –** (Бр. Возило, цена на услуга) и ваквите торки ги залчувува во L1.

```
uslugi_cena(L1) :- uslugi_dekemvri2015(L),
   findall((Br, Len),
         (member(usluga(P,K,C,_,Br), L), najkratok_pat(P,K,(_, Y)), Len is Y*C),
         L1).
```

15) zarabotil(Br,P) – за возилото со број Br, враќа колку заработило во месец декември 2015 во P.

uslugi_cena(L1) - враќа листа од торки (Br, Len) за секоја услуга извршена во декември 2015.
bagof(C, member((Br,C), L1), L) – за секоја торка во L1 која се однесуваат на возило Br се зачувува C – цена на услугата, во L.
sum(L, P) – враќа сума на сите цени во P.

```
zarabotil(Br,P) :- uslugi_cena(L1),
   bagof(C, member((Br,C), L1), L), sum(L, P).
```

16) najmnogu_zarabotil(X) – го наоѓа бројот на такси возилото X кое заработило најмногу во текот на месец декември 2015 година.

findall((Br,P), zarabotil(Br,P), L) – за секое возило со број Br кое извршило услуга во L додава торка (Br, P), каде P е заработката на тоа возило.
najmnogu(L, (X,_)) - го враќа бројот на возилото кое има најголемо P = најмногу заработено.

```
najmnogu_zarabotil(X) :-
   findall((Br,P), zarabotil(Br,P), L),
   najmnogu(L, (X,_)).
```