Никола Крежески 196011, домашна задача, група 1

- Вештачка интелигенција

1. Задача – Бегство од духови

А) Минимална репрезентација за проблемот - 3 тројки и тоа:

- **(x_p, y_p, orientation$_p$)** – каде првите две променливи x_p и y_p ги енкодираат x и y координатите на pacman-от соодветно, а променливата orientation$_p$ води евиденција за ориентацијата(може да добие една од четирите вредности: горе, долу, лево или десно – за кои можеме да избереме репрезентација по своја волја, на пример броеви од 1 до 4 кои ќе имаат соодветно значење последователно) на pacman-от.
- **(x_{gA}, y_{gA}, position$_{gA}$)** – каде првите две променливи x_{gA}, и, y_{gA} ги енкодираат x и y координатите на ghost A соодветно, position$_{gA}$(може да добие една од четирите вредности: горе, долу, лево или десно – за кои можеме да избереме репрезентација по своја волја, на пример броеви од 1 до 4 кои ќе имаат соодветно значење последователно) води евиденција за следната позиција на ghost A при придвижување на pacman-от.
- **(x_{gB}, y_{gB}, position$_{gB}$)** – каде првите две променливи x_{gB}, и, y_{gB} ги енкодираат x и y координатите на ghost B соодветно, position$_{gB}$(може да добие една од четирите вредности: горе, долу, лево или десно – за кои можеме да избереме репрезентација по своја волја, на пример броеви од 1 до 4 кои ќе имаат соодветно значење последователно) води евиденција за следната позиција на ghost B при придвижување на pacman-от.

***Напомена: Во овој пример промените на состојбата на духовите не зависи директно од акциите кои се превземаат, но таа се менува синхронизирано секогаш кога pacman-от се придвижува, па заради тоа чуваме и состојба за духовите.**

Б) Максималниот број на состојби во просторот на состојби за мојата дефиниција е следен:

- **(MxNx4) * (MxNx4) * (MxNx4)** кое поедноставно запишано би било **(MxNx4)^3** – каде секој множител го претсавува бројот на состојби за pacman-от, првиот и вториот дух – соодветно.

В) Максималната вредост на факторот на разгранување (branching factor) изнесува 4. Нека pacman-от се наоѓа на произволна позиција, тогаш максималниот број на акции кои би ги имале на располагање е 4 и тоа: “Заврти се лево”, “Заврти се десно”, “Придвижи се напред” и “Стоп”. Според тоа, кога би креирале пребарувачко дрво(search tree), тогаш максималниот број на деца на јазолот кој ја претставува моменталната состојба ќе биде 4: првото дете ќе ја претставува состојбата во која би преминале со превземањето на акцијата заврти се лево, второто дете ќе ја претставува состојбата во која би преминале со превземањето на акцијата продолжи право итн.

Г) Почетна состојба(според претходно дефинираната репрезентација):

- **(0, 2, 4) –** Почетната состојба на pacman-от
    - Потсетување: ориентацијата на пакманот е претставена со третата променлива од горенаведената тројка, која променлива може да прими еден од броевите од 1 до 4, каде: 1 е со значење горе, 2 е со значење долу, 3 е со значење лево и 4 е со значење десно. Бидејќи идентична репрезентација користев и за позицијата на духовите, со цел да не се повторувам, истото потсетување важи и за нив.
- **(4, 0, 1) –** Почетната состојба на првиот дух, односно дух А
- **(8, 2, 1) -** Почетната состојба на вториот дух, односно дух В

Крајна состојба:

- **Проблемот се смета за решен кога pacman-от ќе го пронајде патот до својата куќа, односно кога pacman-от ќе застане на полето каде се наоѓа неговата куќа, а тоа поле според мојата репрезентација е полето со координати (5, 4).**

Д) Ова прашање ќе го одговорам со помош на програмски код напишан во програмскиот јазик Python:

- Најпрво го претставувам кодот на помошните глобални функции кои ќе ги користам во successor функцијата (која функција ни ги враќа сите легални, односно допуштени акции кога проблемот се наоѓа во некоја произволна состојба):

```
# M and N represent the size of the table
def up(pac_x, pac_y, gA_x, gA_y, gB_x, gB_y, obstacles):
    if pac_y < N - 1 and (pac_x, pac_y + 1) != (gA_x, gA_y) and (pac_x,
pac_y + 1) != (gB_x, gB_y) and (pac_x, pac_y + 1) not in obstacles:
        pac_y += 1
    return pac_y


def down(pac_x, pac_y, gA_x, gA_y, gB_x, gB_y, obstacles):
    if pac_y > 0 and (pac_x, pac_y - 1) != (gA_x, gA_y) and (pac_x,
pac_y - 1) != (gB_x, gB_y) and (pac_x, pac_y - 1) not in obstacles:
        pac_y -= 1
    return pac_y


def right(pac_x, pac_y, gA_x, gA_y, gB_x, gB_y, obstacles):
    if pac_x < M - 1 and (pac_x + 1, pac_y) != (gA_x, gA_y) and (pac_x +
1, pac_y) != (gB_x, gB_y) and (pac_x + 1, pac_y) not in obstacles:
        pac_x += 1
    return pac_x


def left(pac_x, pac_y, gA_x, gA_y, gB_x, gB_y, obstacles):
    if pac_x > 0 and (pac_x - 1, pac_y) != (gA_x, gA_y) and (pac_x - 1,
pac_y) != (gB_x, gB_y) and (pac_x - 1, pac_y) not in obstacles:
        pac_x -= 1
    return pac_x
```

- Потоа го претставувам кодот на самата successor функција:

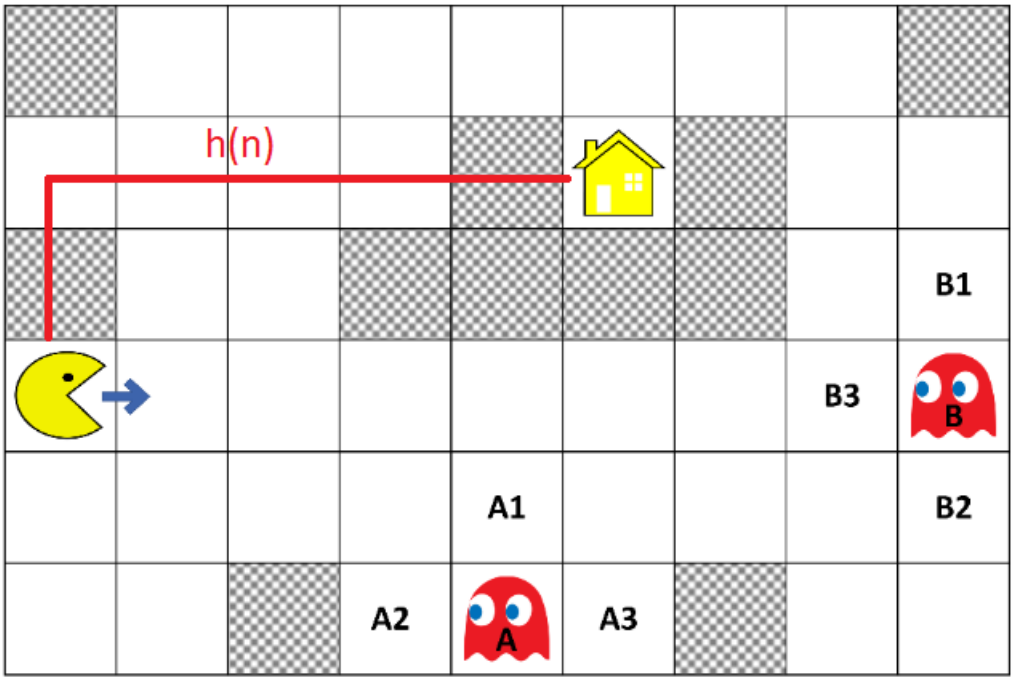
```
def successor(self, state):
    successors = dict()
    pac_info = state[0]
    pac_x = pac_info[0]
    pac_y = pac_info[1]
    orientation = pac_info[2]
    gA_info = state[1]  # se odnesuva na trojkata so koja e pretstaven
duhot
    gB_info = state[2]  # se odnesuva na trojkata so koja e pretstaven
duhot
    gA_x = gA_info[0]
    gA_y = gA_info[1]
    gA_pos = gA_info[2]
    gB_x = gB_info[0]
    gB_y = gB_info[1]
    gB_pos = gB_info[2]

    # Postavuvanje na vrednostite za slednata pozicija na duhovite
    if gA_pos == 'gore':
        gA_y += 1
    elif gA_pos == 'dolu':
        gA_y -= 1
    elif gA_pos == 'levo':
        gA_x -= 1
    elif gA_pos == 'desno':
        gA_x += 1

    if gB_pos == 'gore':
        gB_y += 1
    elif gB_pos == 'dolu':
        gB_y -= 1
    elif gB_pos == 'levo':
        gB_x -= 1
    elif gB_pos == 'desno':
        gB_x += 1

    if orientation == 'desno':
        new_x = right(pac_x, pac_y, gA_x, gA_y, gB_x, gB_y,
self.obstacles)
        if pac_x != new_x:
            successors['PridviziSeNapred'] = ((new_x, pac_y, 'desno'),
(gA_x, gA_y, gA_pos), (gB_x, gB_y, gB_pos))

        successors['SvrtiLevo'] = ((pac_x, pac_y, 'gore'), (gA_x, gA_y,
gA_pos), (gB_x, gB_y, gB_pos))
        successors['SvrtiDesno'] = ((pac_x, pac_y, 'dolu'), (gA_x, gA_y,
gA_pos), (gB_x, gB_y, gB_pos))
        successors['Stop'] = ((pac_x, pac_y, 'desno'), (gA_x, gA_y,
gA_pos), (gB_x, gB_y, gB_pos))

    if orientation == 'dolu':
        new_y = down(pac_x, pac_y, gA_x, gA_y, gB_x, gB_y,
self.obstacles)
        if pac_y != new_y:
            successors['PridviziSeNapred'] = ((pac_x, new_y, 'dolu'),
(gA_x, gA_y, gA_pos), (gB_x, gB_y, gB_pos))

        successors['SvrtiLevo'] = ((pac_x, pac_y, 'desno'), (gA_x, gA_y,
gA_pos), (gB_x, gB_y, gB_pos))
```

```
        successors['SvrtiDesno'] = ((pac_x, pac_y, 'levo'), (gA_x, gA_y,
gA_pos), (gB_x, gB_y, gB_pos))
        successors['Stop'] = ((pac_x, pac_y, 'dolu'), (gA_x, gA_y,
gA_pos), (gB_x, gB_y, gB_pos))

    if orientation == 'gore':
        new_y = up(pac_x, pac_y, gA_x, gA_y, gB_x, gB_y, self.obstacles)
        if pac_y != new_y:
            successors['PridviziSeNapred'] = ((pac_x, new_y, 'gore'),
(gA_x, gA_y, gA_pos), (gB_x, gB_y, gB_pos))

        successors['SvrtiLevo'] = ((pac_x, pac_y, 'levo'), (gA_x, gA_y,
gA_pos), (gB_x, gB_y, gB_pos))
        successors['SvrtiDesno'] = ((pac_x, pac_y, 'desno'), (gA_x,
gA_y, gA_pos), (gB_x, gB_y, gB_pos))
        successors['Stop'] = ((pac_x, pac_y, 'gore'), (gA_x, gA_y,
gA_pos), (gB_x, gB_y, gB_pos))

    if orientation == 'levo':
        new_y = left(pac_x, pac_y, gA_x, gA_y, gB_x, gB_y,
self.obstacles)
        if pac_y != new_y:
            successors['PridviziSeNapred'] = ((pac_x, new_y, 'levo'),
(gA_x, gA_y, gA_pos), (gB_x, gB_y, gB_pos))

        successors['SvrtiLevo'] = ((pac_x, pac_y, 'dolu'), (gA_x, gA_y,
gA_pos), (gB_x, gB_y, gB_pos))
        successors['SvrtiDesno'] = ((pac_x, pac_y, 'gore'), (gA_x, gA_y,
gA_pos), (gB_x, gB_y, gB_pos))
        successors['Stop'] = ((pac_x, pac_y, 'levo'), (gA_x, gA_y,
gA_pos), (gB_x, gB_y, gB_pos))

    return successors
```

***Објаснување на кодот:** Значи ако проблемот се наоѓа во некоја произволна состојба, тогаш: најпрво треба да видиме на каде следно ќе се придвижат духовите, со цел да имаме предвид кои полиња ни се дозволени кога го придвижуваме пакманот. Пакманот има можни 4 акции и во зависност на моменталната ориентација во која тој се наоѓа, овие акции ни се дефинирани при што само за акцијата 'PridviziSeNapred' проверуваме дали истата е легална, затоа што останатите 3 акции по default се легални, бидејќи немаме никакво придвижување на пакманот со тие акции кои би можеле да го доведат пакманот во нелегална состојба. Проверката за акцијата 'PridviziSeNapred' се врши на тој начин што проверувам дали следната позиција на пакманот нема да го одведе пакманот надвор од таблата т.е координатниот систем, па исто така проверувам дали потенцијално следната позиција не е во листата на пречки и последно имам уште 2 услови кои проверуваат дали следната позиција на пакманот е различна со таа на духовите. Ако сите овие услови се исполнети, тогаш акцијата е легална.

***Напомена:** Во кодот погоре како вредности на променливите се користат стрингови, (за разлика од мојата првична дефиниција) за означување на ориентацијата на пакманот и позициите на духовите со цел за полесна разбирливост на кодот.

Ѓ) Евристика која би ја користел за проблемот, а истата не е тривијална е менхетеновото растојание од моменталната позиција на пакманот до неговата куќа. Оваа евристика е допустлива, затоа што ја решава „релаксираната“ верзија на проблемот. Што тоа значи? Тоа значи дека евристиката не е „свесна“ (aware) за никакви пречки или духови кои можат да го попречат движењето на пакманот. Оттука можеме да заклучиме дека хевристиката **h(n) е по default ≥ 0**, затоа што растојанието е физичка мерка и никогаш не може да биде негативна и **h(n) е секогаш ≤ h*(n)** поради причините кои ги наведов погоре. Исто така ќе го наведам и следниот цртеж со цел за полесно толкување на тоа што го напишав:



Слика 1: Вредноста која ја пресметува хевристиката (Manhattan distance)

Е) Иако пакманот би можел да се придвижува за 1, 2 или 3 полиња нанапред во зависност од неговата ориентација, дефинираната евристика сè уште би била допустлива. Во овој проблем кога пакманот ќе се придвижува за 1, 2 или 3 полиња, менхетеновото растојание помеѓу моменталната позиција на пакманот и позицијата на куќичката ќе се зголемува или намалува соодветно, но бидејќи и овој проблем за хевристиката би бил „релаксиран“, односно хевристиката не е „свесна“ за никакви пречки, духови или било какви други препреки кои би го загушиле придвижувањето на пакманот, менхетеновото растојание секогаш ќе биде помало или еднакво на вистинското растојание од пакманот до неговата куќа. Со тоа докажуваме дека хевристиката останува допустлива и за овој изменет проблем.

Ж) Кој од следните алгоритми за пребарување ќе гарантира оптималност на решението?

- **DFS(Depth First Search) –** DFS е систематски алгоритам за пребарување, односно изминување на граф или дрво, со тоа што секогаш започнува од јазелот – корен истражувајќи најдлабоко во секоја гранка од дрвото пред да се наврати назад и да пребарува во следната гранка, односно следното поддрво на почетниот јазел. DFS не е оптимален алгоритам за пребарување во општ случај, од што следува дека DFS нема да биде оптимален ни за конкретниот проблем кој што го обработуваме, затоа што DFS секогаш го изминува дрвото или графот на ист, систематски однапред дефиниран начин, без оглед на длабочината или цената на чинење на акциите. Односно, DFS секогаш го пронаоѓа „најлевото" решение, без да зема во предвид останати фактори.
- **BFS(Breadth First Search) –** BFS е систематски алгоритам за пребарување, односно изминување на граф или дрво, со тоа што секогаш започнува од јазелот – корен истражувајќи ги соседните јазли на тековното ниво(се мисли на ниво на дрво) пред да премине на јазел од следното ниво. BFS е оптимален алгоритам за пребарување секогаш кога цената на чинење на акциите е иста, т.е има цена 1 во општ случај. Па така и за конкретниот случај, BFS е оптимален.
- **UCS(Uniform Cost Search) –** UCS е алгоритам за пребарување на „weighted" дрва или графови. Примарната цел на овој алгоритам е да го пронајде патот до целта кој има најмала кумулативна цена. UCS започнува од јазелот – корен. Во општ случај UCS е оптимален, па ќе биде оптимален и за конкретниот случај.
- **A* (A Star Search) –** A* е алгоритам за информирано пребарување кој претставува микс од UCS и Best-Fit Search алгоритмот, односно неговата функција на пребарување f(n) претставува збир на кумулативната цена за дадениот јазол и вредност на хевристичката функција која враќа апроксимација за тоа каде целната позиција се наоѓа, односно математички напишано f(n) = g(n) + h(n), каде g(n) се однесува на кумулативната цена за јазелот, а h(n) е вредноста на хевристичката функција. A* алгоритмот во општ случај е оптимален секогаш кога хевристичката функција која ја користи е допустлива. За конкретниот случај, A* е оптимален, ако претпоставиме дека како хевристичка функција ја користи погоренаведената хевристичка функција, односно менхетеновото растојание од моменталната позиција на пакменот до позицијата на неговата куќа.

Дали во вашите сценарија некои од алгоритмите се меѓусебно еквивалентни?

- Во даденото сценарио, алгоритмот UCS(Uniformed Cost Search) е еквивалентен со алгоритмот BFS(Breadth first search) затоа што цената на чинење на акциите е секогаш 1, па UCS се однесува идентично како и BFS, затоа што цената на акциите е незначителна.

Доколку имате избор, кој од понудените алгоритми би го избрале како најдобар за проблемот?

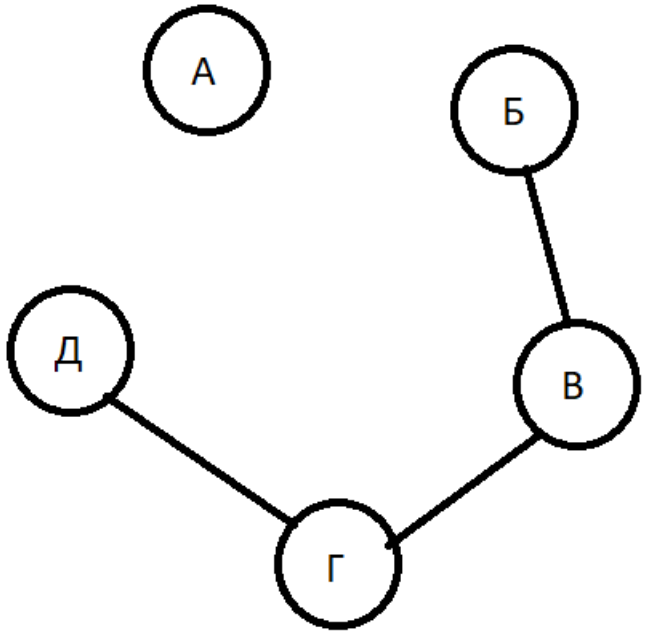
- Како заклучок од досега напишаното, за конкретниот проблем како најдобар алгоритам би го избрал A*, од причина дека тоа е единствен алгоритам за информирано пребарување од сите овие понудени, па истиот би ја пронашол оптималната цел за многу пократко време за разлика од преостанатите алгоритми. За разлика од A*, DFS не ни го земаме во предвид, затоа што е неоптимален, а UCS кој во случајов е еквивалентен со BFS не би го одбрал како најдобар, затоа што е алгоритам за неинформирано пребарување, и во зависност од тоа каде се наоѓа целната позиција, можно е на алгоритмот да му треба многу време за да ја пронајде целта, која секако ќе биде оптимална.

2. Задача – Хуманитарна акција

А) Формално дефинирање на проблемот

- **Променливи:** пријателите **А(Александар), Б(Бојан), В(Влатко), Г(Горан) и Д(Дејан).**
- **Домен: 1**(Храна), **2**(Хигиенски производи), **3**(Облека), **4**(Мебел), **5**(Апарати за домаќинство) и **6**(Хемиски производи)
- **Нотација:** на пример – Item(А) = 1
- **Услови:**
  - Item(В) =/= Item(Б)
  - Item(А) =/= 1
  - Item(Б) = {2, 3, 6}
  - Item(В) =/= {2, 4, 6}
  - Item(Г) = {1, 2, 3, 4}
  - Item(Д) < Item(Г)
  - Item(Д) =/= 6
  - If Item(Г) == 1 Item(В) = 5 elif Item(В) == 1 Item(Г) = 5

Б) Граф на ограничувања



В) Во следната табела се прикажани вредностите кои остануваат за променливите по пропагирање, односно исполнување на унарните услови

*********	Александар	Бојан	Влатко	Горан	Дејан
Храна(1)			✓	✓	✓
Хигиенски производи(2)	✓	✓		✓	✓
Облека(3)	✓	✓	✓	✓	✓
Мебел(4)	✓			✓	✓
Апарати за домаќинство(5)	✓		✓		✓
Хемиски производи(6)	✓	✓			

Г) Тука итеративно ќе го претставам методот за проверка на конзистентноста на целиот проблем:

#########	Александар	Бојан	Влатко	Горан	Дејан
Храна(1)			✓		✓
Хигиенски производи(2)	✓	✓		✓	✓
Облека(3)	✓	✓	✓	✓	✓
Мебел(4)	✓			✓	✓
Апарати за домаќинство(5)	✓		✓		✓
Хемиски производи(6)	✓	✓			

Горан → Дејан – Заради условот Item(Д) < Item(Г), кај Горан се поткаструва храната, затоа што ако Горан би донирал храна, тогаш Дејан нема што да донира

#########	Александар	Бојан	Влатко	Горан	Дејан
Храна(1)			✓		✓
Хигиенски производи(2)	✓	✓		✓	✓
Облека(3)	✓	✓	✓	✓	✓
Мебел(4)	✓			✓	
Апарати за домаќинство(5)	✓		✓		
Хемиски производи(6)	✓	✓			

Дејан → Горан – Заради условот Item(Д) < Item(Г), кај Дејан ќе се скастрат нештата под реден број 4 и 5, заради тоа што Горан никогаш нема да може да донира нешто под реден број 5 и 6.

#########	Александар	Бојан	Влатко	Горан	Дејан
Храна(1)					✓
Хигиенски производи(2)	✓	✓		✓	✓
Облека(3)	✓	✓	✓	✓	✓
Мебел(4)	✓			✓	
Апарати за домаќинство(5)	✓		✓		
Хемиски производи(6)	✓	✓			

Влатко → Горан – Поради условот **If Item(Г) == 1 Item(В) = 5 elif Item(В) == 1 Item(Г) = 5,** се скаструва храната кај Влатко, затоа што ако Влатко се одлучи да донира храна, тогаш условот не може да биде исполнет, затоа што Горан не знае дека тој може да донира апарати за домаќинство.

Д)

Прво доделување: Според хевристиката **MRV(Minimum remaining values)** ја избирам променливата **В(Влатко)** за доделување на вредност. Сега, со хевристиката **LCV(Least constraining value)** која ја избира вредноста која ќе придонесе за минимално кастрење на вредности од домените на останатите променливи ја избираме вредноста **5(Апарати за домаќинство).** Со forward checking можеме да видеме дека нема да има кастрење во доменот на ниту една променлива, а при тоа останатите услови остануваат непрекршени.

#########	Александар	Бојан	Влатко	Горан	Дејан
Храна(1)					✓
Хигиенски производи(2)	✓	✓		✓	✓
Облека(3)	✓	✓		✓	✓
Мебел(4)	✓			✓	
Апарати за домаќинство(5)	✓		✓		
Хемиски производи(6)	✓	✓			

Второ доделување: Според хевристиката **MRV(Minimum remaining values)** можам да изберам една од три променливи, па по азбучен ред ја избирам променливата **Б(Бојан).** За доделување на вредност на променливата, ја користам хевристиката **LCV(Least constraining value)** која ја избира вредноста која ќе придонесе за минимално кастрење на вредности од домените на останатите променливи. Со LCV можам да изберам повеќе вредности, затоа што ниту една од вредностите нема да предизвикаат кастрење на ниту една вредност од домените на другите променливи, па ја избирам најмалата вредност, односно **2(Хигиенски производи).** Со forward checking можеме да видеме дека нема да има кастрење во доменот на ниту една променлива, а при тоа останатите услови остануваат исполнети.

#########	Александар	Бојан	Влатко	Горан	Дејан
Храна(1)					✓
Хигиенски производи(2)	✓	✓		✓	✓
Облека(3)	✓			✓	✓
Мебел(4)	✓			✓	
Апарати за домаќинство(5)	✓		✓		
Хемиски производи(6)	✓				

Трето доделување: Според хевристиката **MRV(Minimum remaining values)** можам да изберам една од две променливи, па по азбучен ред ја избирам променливата **Г(Горан).** За доделување на вредност на променливата, ја користам хевристиката **LCV(Least constraining value)** која ја избира вредноста која ќе придонесе за минимално кастрење на вредности од домените на останатите променливи. Со LCV се избира вредноста **4(Мебел),** затоа што не предизвикува кастрење на доменот на вредности на променливата Д(Дејан). Со forward checking, можеме да провериме дека со избирање на вредноста Мебел, нема да има поткастрување од доменот на променливата Дејан.

#########	Александар	Бојан	Влатко	Горан	Дејан
Храна(1)					✓
Хигиенски производи(2)	✓	✓			✓
Облека(3)	✓				✓
Мебел(4)	✓			✓	
Апарати за домаќинство(5)	✓		✓		
Хемиски производи(6)	✓				

Четврто доделување: Според хевристиката **MRV(Minimum remaining values)** ја избирам променливата **Д(Дејан),** затоа што има најмалку преостанати вредности во својот домен. Со хевристиката **LCV(Least constraining value)** можеме да избереме било која од вредностите во доменот на променливата Д(Дејан), па затоа ја избирам најмалата вредност **1(Храна).** Со forward-checking можеме да видеме дека нема да се скастри ниту една вредност од домените на останатите променливи.

#########	Александар	Бојан	Влатко	Горан	Дејан
Храна(1)					✓
Хигиенски производи(2)	✓	✓			
Облека(3)	✓				
Мебел(4)	✓			✓	
Апарати за домаќинство(5)	✓		✓		
Хемиски производи(6)	✓				

Петто доделување: Ни преостанува уште една променлива, па затоа не користиме хевристика за избор на променливи. За избор на вредностите можеме да ја избереме било која, па затоа ќе ја изберам произволно, односно ќе ја изберам вредноста **6(Хемиски производи)** – for the sake of diversity.

**********	Александар	Бојан	Влатко	Горан	Дејан
Храна(1)					✓
Хигиенски производи(2)		✓			
Облека(3)					
Мебел(4)				✓	
Апарати за домаќинство(5)			✓		
Хемиски производи(6)	✓				