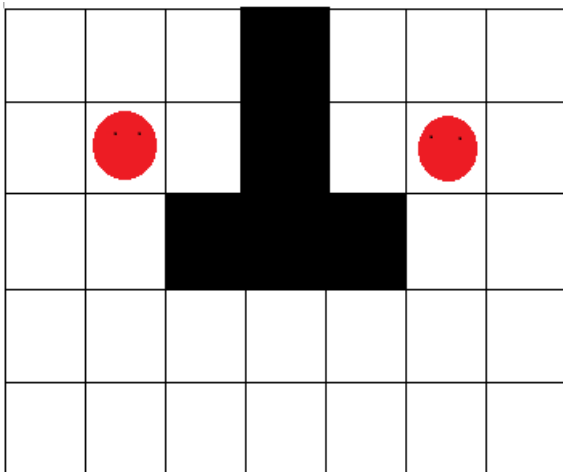


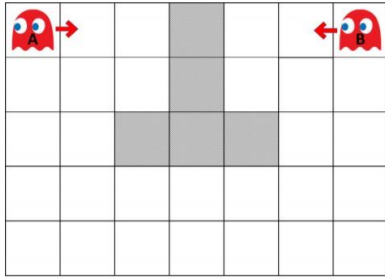
Прва домашна задача по Вештачка интелигенција

Задача 1.

1. Минимална репрезентација на состојба за овој проблем би било претставување на двата духа со координати и нивната насока $((x_a, y_a, \text{насока}), (x_b, y_b, \text{насока}))$ кои ја означуваат нивната моментална позиција во просторот $M \times N$ кој може да го гледаме како координатен систем. Ова е доволно за состојбата бидејќи позициите на пречките во просторот и Пакман се неменливи и не влегуваат во состојбата.
2. За овој конкретен проблем бројот на состојби е $4^2(M \times N)^2$ бидејќи во една состојба се земаат во предвид само позициите на **двата** духа и **нивната насока**. Доколку би имало k духови кои се движат низ просторот бројот на состојби би бил $4^k(M \times N)^k$.
3. Максимална вредност на факторот за разгранување за овој проблем е 16. Од еден јазол (моментална состојба) може да преминеме во најмногу 16 нови состојби, односно според правилата на комбинаторика, 4 акции за едниот дух * 4 акции за вториот дух. Во некои состојби дел од акциите ќе бидат нелегални, но овде станува збор за максимална вредност на факторот за разгранување и затоа разгледуваме состојба од која сите акции се легални во следниот чекор.

Пример за таква состојба е следната илустрација, каде без разлика на која страна се моментално насочени духовите, можат да ги извршат акциите “Заврти се лево”, “Заврти се десно”, “Придвижи се напред” и “Стоп,,.





4. Слика 1. Пример на можна почетна состојба на проблемот

Почетна состојба за сл.1 според мојот начин на репрезентација на состојбите е $((0,4), 'desno'), ((6,4), 'levo')$. Крајна состојба е состојбата во која позицијата на еден од духовите е иста со позицијата на Пакман, т.е $(x_a, y_a) == (x_p, y_p)$ **или** $(x_b, y_b) == (x_p, y_p)$, каде (x_p, y_p) е позицијата на Пакман. При тоа $(x_a, y_a) != (x_b, y_b)$ како што е опишано во условите. Насоката на духот не се зема во предвид при goal test.

5. Легалните акции за произволна состојба ги наоѓаме со проверување на неколку услови:
 - Дали со придвижување напред во насоката во која се насочени духовите ќе излезат од просторот?
 - Дали со придвижување напред во насоката во која се насочени духовите ќе се најдат во исто поле?
 - Дали со придвижување напред во насоката во која е насочен духот, неговите координати ќе бидат исти со координатите на некоја од пречките?
 Ако одговорот на овие три прашања е негативен, тогаш акцијата која сакаме да ја преземеме е легална.

Најпрвин се дефинираат помошни функции за пресметување на позицијата и насоката на духовите во новата состојба. Истите функции се користат и за двата духови бидејќи акциите се исти.

Функцијата продолжи право на влез ги добива координатите на духот и ги враќа новите координати, а насоката останува иста кога се движат право. Во оваа функција се проверува дали акцијата Продолжи право е легална (дали е во границите на просторот, означен како table и дали новите позиции се во пречките). Ако акцијата не е легална враќа None.

```

def ghost_prodolziPravo(x,y,nasoka,table, obstacles):
    if nasoka == 'desno':
        if x+1 < table[0] and (x+1,y) not in obstacles:
            return tuple([tuple([x+1,y]),nasoka])
    elif nasoka == 'levo':
        if x-1 >= 0 and (x-1,y) not in obstacles:
            return tuple([tuple([x-1,y]),nasoka])
    elif nasoka == 'gore':
        if y+1 < table[1] and (x,y+1) not in obstacles:
            return tuple([tuple([x,y+1]),nasoka])
    elif nasoka == 'dolU':
        if y-1 >= 0 and (x,y-1) not in obstacles:
            return tuple([tuple([x,y-1]),nasoka])
    return None

```

Функциите за Сврти Лево и Сврти Десно се секогаш легални бидејќи не се придвижуваме од моменталната позиција, само ја менуваме насоката.

```

def ghost_Levo(x,y,nasoka):
    xy = tuple([x,y])
    if nasoka == 'desno':
        return tuple([xy, 'gore'])
    elif nasoka == 'levo':
        return tuple([xy, 'dolU'])
    elif nasoka == 'gore':
        return tuple([xy, 'levo'])
    elif nasoka == 'dolU':
        return tuple([xy, 'desno'])

def ghost_Desno(x,y,nasoka):
    xy = tuple([x,y])
    if nasoka == 'desno':
        return tuple([xy, 'dolU'])
    elif nasoka == 'levo':
        return tuple([xy, 'gore'])
    elif nasoka == 'gore':
        return tuple([xy, 'desno'])
    elif nasoka == 'dolU':
        return tuple([xy, 'levo'])

```

Пречките и големината на просторот ги задаваме во конструкторот бидејќи се неменливи.

Во successor се дефинираат новите состојби. Првин ги пресметуваме новите позиции и насоки со користење на горе-споменатите помошни функции.

Проверуваме дали состојбите кои ја вклучуваат акцијата Продолжи право вратиле None. Доколку да, тие се нелегални и понатаму не ги разгледуваме. Ако вратиле нова позиција и насока, тогаш се проверува уште условот дали двата духа се на различно поле. Ако и тој услов е исполнет, акцијата е легална.

Останатите состојби кои **не** ја вклучуваат акцијата Продолжи право се секогаш легални.

```
class Pacman(Problem):
    def __init__(self, initial, goal):
        super().__init__(initial, goal)
        self.table = (7,5)
        self.obstacles = ((2,2), (3,2), (3,3), (3,4), (4,2))
        self.Pacman = goal

    def successor(self, state):
        successors = dict()
        ghostA_xy = list(state[0][0])
        ghostB_xy = list(state[1][0])
        ghost_A_orient = state[0][1]
        ghost_B_orient = state[1][1]

        ghost_A_new_Pravo = ghost_prodolziPravo(ghostA_xy[0], ghostA_xy[1], ghost_A_orient, self.table, self.obstacles)
        ghost_B_new_Pravo = ghost_prodolziPravo(ghostB_xy[0], ghostB_xy[1], ghost_B_orient, self.table, self.obstacles)
        ghost_A_new_Desno = ghost_Desno(ghostA_xy[0], ghostA_xy[1], ghost_A_orient)
        ghost_B_new_Desno = ghost_Desno(ghostB_xy[0], ghostB_xy[1], ghost_B_orient)
        ghost_A_new_Levo = ghost_Levo(ghostA_xy[0], ghostA_xy[1], ghost_A_orient)
        ghost_B_new_Levo = ghost_Levo(ghostB_xy[0], ghostB_xy[1], ghost_B_orient)

        if ghost_A_new_Pravo != None:
            if ghost_B_new_Pravo != None and ghost_A_new_Pravo[0] != ghost_B_new_Pravo[0]:
                successors[('ProdolziPravo', 'ProdolziPravo')] = (ghost_A_new_Pravo, ghost_B_new_Pravo)
            if ghost_A_new_Pravo[0] != ghost_B_new_Levo[0]:
                successors[('ProdolziPravo', 'SvrtiLevo')] = (ghost_A_new_Pravo, ghost_B_new_Levo)
            if ghost_A_new_Pravo[0] != ghost_B_new_Desno[0]:
                successors[('ProdolziPravo', 'SvrtiDesno')] = (ghost_A_new_Pravo, ghost_B_new_Desno)
            if ghost_A_new_Pravo[0] != tuple(ghostB_xy):
                successors[('ProdolziPravo', 'Stop')] = (ghost_A_new_Pravo, tuple([tuple(ghostB_xy), ghost_B_orient]))

        if ghost_B_new_Pravo != None:
            if ghost_B_new_Pravo[0] != ghost_A_new_Levo[0]:
                successors[('SvrtiLevo', 'ProdolziPravo')] = (ghost_A_new_Levo, ghost_B_new_Pravo)
            if ghost_B_new_Pravo[0] != ghost_A_new_Desno[0]:
                successors[('SvrtiDesno', 'ProdolziPravo')] = (ghost_A_new_Desno, ghost_B_new_Pravo)
            if ghost_B_new_Pravo[0] != tuple(ghostA_xy):
                successors[('Stop', 'ProdolziPravo')] = (tuple([tuple(ghostA_xy), ghost_A_orient]), ghost_B_new_Pravo)

        successors[('SvrtiDesno', 'SvrtiDesno')] = (ghost_A_new_Desno, ghost_B_new_Desno)
        successors[('SvrtiDesno', 'SvrtiLevo')] = (ghost_A_new_Desno, ghost_B_new_Levo)
        successors[('SvrtiDesno', 'Stop')] = (ghost_A_new_Desno, tuple([tuple(ghostB_xy), ghost_B_orient]))

        successors[('SvrtiLevo', 'SvrtiDesno')] = (ghost_A_new_Levo, ghost_B_new_Desno)
        successors[('SvrtiLevo', 'SvrtiLevo')] = (ghost_A_new_Levo, ghost_B_new_Levo)
        successors[('SvrtiLevo', 'Stop')] = (ghost_A_new_Levo, tuple([tuple(ghostB_xy), ghost_B_orient]))

        successors[('Stop', 'SvrtiDesno')] = (tuple([tuple(ghostA_xy), ghost_A_orient]), ghost_B_new_Desno)
        successors[('Stop', 'SvrtiLevo')] = (tuple([tuple(ghostA_xy), ghost_A_orient]), ghost_B_new_Levo)
        successors[('Stop', 'Stop')] = (tuple([tuple(ghostA_xy), ghost_A_orient]), tuple([tuple(ghostB_xy), ghost_B_orient]))

        return successors
```

- Тривијална допустлива хевристика би било $h(s) = 0$. Тривијална бидејќи е константна, допустлива бидејќи е исполнет условот $0 \leq h(s) \leq h^*(s)$ каде $h^*(s)$ е вистинската „цена„. Нетривијална допустлива хевристика не постои бидејќи не знаеме точно каде е Пакман, односно ова е неинформирано пребарување.

Математички приказано: $h(s) = \min(\text{abs}(k-a) , \text{abs}(k-b))$

8. Образложување на примената на различни алгоритми за овој проблем (го разгледувам основниот проблем, без проширување со информацијата за колоната) :

- i. DFS (Depth First Search): Овој алгоритам наоѓа решение за проблемот, но користи многу чекори.

- ii. Алгоритмите UCS (Uniform Cost Search) и BFS (Breadth First Search) се однесуваат исто за овој проблем, бидејќи цената на секое ребро е 1. Овие алгоритми се оптимални и наоѓаат решение во помалку чекори во споредба со DFS, но и тие имаат големи разгранување бидејќи макс. фактор на разгранување е 16.

Споредба на решенија со **DFS** и **BFS** за иста иницијална состојба:

[illegible]

```
[('ProdolziPravo', 'ProdolziPravo'), ('ProdolziPravo', 'SvrtiDesno'), ('SvrtiLevo', 'ProdolziPravo')]
```

- iii. Алгоритмот A^* не може да се искористи за дадениот проблем бидејќи тој е погоден за информирано пребарување, додека во овој случај имаме неинформирано пребарување (немаме информација за позицијата на Пакман).

Ако проблемот е проширен, со информација за колоната на Пакман, тогаш A^* би можел да се искористи за наоѓање оптимално решение под услов да имаме оптимална хевристика.

Целиот код за подобар преглед:

https://github.com/Tijana37/AI_homework1

ЗАДАЧА 2

1. Формално дефинирање на проблемот како проблем на исполнување услови означува определување на променливите/актерите во проблемот, вредностите кои тие можат да ги примаат и кои се условите кои мора да бидат задоволени при доделување на вредностите на променливите за одредена секвенца да ја означиме како решение. Goal state во овој тип на проблеми е состојба во која секоја од променливите има доделена вредност и сите услови се исполнети.

Променливи: N1, N2, N3, N4, N5 (членовете)

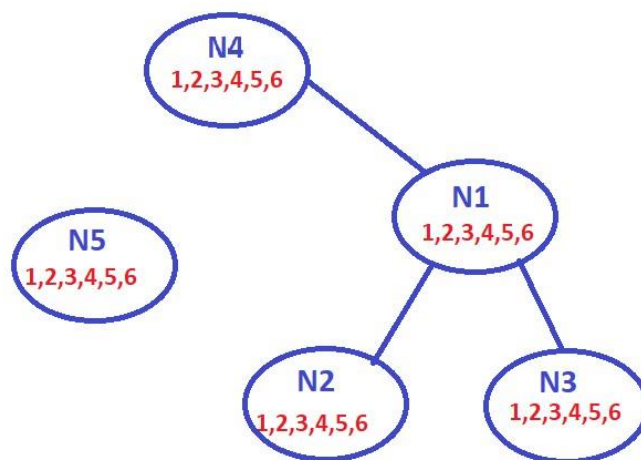
Домен: 1,2,3,4,5,6 (задачите означени со реден број)

Услови:

- Членот N4 мора да работи само на прашањата 1, 2 или 3.
- Членот N1 мора да работи на прашања кои се наоѓаат на листата пред прашањата на кои работи N3.
- Членот N5 не сака да прегледува комплексни прашања, затоа мора да работи на прашањето со T/F одговори.
- Членот N2 мора да работи на прашања кои се наоѓаат на листата по прашањата на кои работи N1.
- Членот N2 не може да работи на прашањата 4, 5 и 6.
- Членот N3, поради суеверие, не сака да работи на прашањата со непарни броеви.
- Членот N1 не може да работи на прашањето 6.
- Членовите N1 и N4 не треба да работат на исто прашање.

*Во овој проблем доменот е ист за сите променливи, доколку беше поинаку дефиниран би требало да се опишат домените за секоја од променливи посебно.

2. Графот на ограничувања е претставен со променливите и вредностите кои може да ги примаат во јазлите. Врските меѓу јазлите го означуваат односот меѓу јазлите. Доколку два јазли се ограничени со бинарен услов, ги поврзуваме меѓусебно. Графот не носи детална информација за условите, но служи за согледување на ограничувањата.



3. Иницијалната состојба во Constraint Satisfaction Problems е празна листа (табела) од доделени вредности.

Иницијална состојба:

	N1	N2	N3	N4	N5
1					
2					
3					
4					
5					

При решавање на овој тип на проблеми секогаш започнуваме од унарните услови кои ограничуваат само една променлива.

Во задавава унарни услови се:

- Членот N4 мора да работи само на прашањата 1, 2 или 3.
- Членот N5 не сака да прегледува комплексни прашања, затоа мора да работи на прашањето со T/F одговори.
- Членот N2 не може да работи на прашањата 4, 5 и 6.
- Членот N3, поради суеверие, не сака да работи на прашањата со непарни броеви.
- Членот N1 не може да работи на прашањето 6.

Вредности кои преостануваат по примена на сите унарни услови од проблемот:

*со X се означени вредностите кои не смеат да бидат доделени на соодветната променлива

	N1	N2	N3	N4	N5
1			X		X
2					X
3			X		X
4		X		X	X
5		X	X	X	X
6	X	X		X	

4. Преостанатите услови се нарекуваат бинарни услови бидејќи две променливи се ограничуваат меѓусебно.

Преостанати неисполнети услови:

- Членот N1 мора да работи на прашања кои се наоѓаат на листата пред прашањата на кои работи N3.
- Членот N2 мора да работи на прашања кои се наоѓаат на листата по прашањата на кои работи N1.
- Членовите N1 и N4 не треба да работат на исто прашање

Оптимизација на програмот може да се обезбеди со алгоритмот кој ја проверува конзистентноста на ребрата пред да додели вредност. Правило: Реброто $X \rightarrow Y$ е конзистентно ако за секоја вредност x во задниот дел од стрелката, постои вредност y во главата на стрелката која може да биде доделена без да биде прекршен услов. Ребрата меѓу променливите кои не се ограничени со услов исто така мора да бидат проверени за конзистентност.

Ја разгледуваме конзистентноста на реброто N1 во однос на N3.

Според условот:

- Членот N1 мора да работи на прашања кои се наоѓаат на листата пред прашањата на кои работи N3.


Согледуваме дека за секоја од вредностите кои се останати во доменот на N1, постои вредност која може да се додели на N3 и при тоа условот да е исполнет.

Пр: Ако на N1 се додели 5, тогаш на N3 може да се додели вредноста 6.

Ако на N1 се додели 4, тогаш на N3 може да се додели вредноста 6.

Ако на N1 се додели 3, тогаш на N3 може да се доделат вредностите 4 и 6.

Ако на N1 се додели 2, тогаш на N3 може да се доделат вредностите 4 и 6.
 Ако на N1 се додели 1, тогаш на N3 може да се доделат вредностите 2, 4 и 6.
 Заклучок: N1 е конзистентен во однос на N3.



	N1	N2	N3	N4	N5
1			X		X
2					X
3			X		X
4		X		X	X
5		X	X	X	X
6	X	X		X	


Ако N3 прими вредност 2, N1 ќе ја прими вредноста 1. Истото следува и за другите вредности на N3. Следува, реброто N3-> N1 е конзистентно.

	N1	N2	N3	N4	N5
1			X		X
2					X
3			X		X
4		X		X	X
5		X	X	X	X
6	X	X		X	

Следно, ја разгледуваме конзистентноста на реброто N2 во однос на N1, која е ограничена од условот: • Членот N2 мора да работи на прашања кои се наоѓаат на листата по прашањата на кои работи N1.

***Ако условот се разгледува како членот N2 мора да работи на прашања кои се по најголемото прашање (по реден број во листата) на кое работи N1, тогаш проблемот нема решение. При бришење на вредностите од доменот поради конзистентност на ребрата, прашањето 5 не е во доменот на ниту еден член!**

Ги отстрануваме вредностите 3,4,5 од доменот на N1 за реброто N1->N2 да е конзистентно. Бидејќи ако ја доделиме вредноста 3 на N1, тогаш не постои вредност во доменот на N2 која би можела да се додели без да биде прекршен условот.




	N1	N2	N3	N4	N5
1			X		X
2					X
3	X		X		X
4	X	X		X	X
5	X	X	X	X	X
6	X	X		X	

***Ако условот се разгледува како членот N2 мора да работи на прашања кои се по најмалото прашање (по реден број во листата) на кое работи N1.**

Се отстранува само вредноста 1 од доменот на N2, бидејќи нема прашање кое може да се додели на N1 кое е пред прашањето 1. За останатите вредности, реброто е конзистентно.

	N1	N2	N3	N4	N5
1		X	X		X
2					X
3			X		X
4		X		X	X
5		X	X	X	X
6	X	X		X	



Бидејќи направивме промени во домените, повторно ја проверува конзистентноста на ребрата кои веќе ги прогласивме за конзистентни со истата постапка.

Неисполнет останува само условот: Членовите N1 и N4 не треба да работат на исто прашање.


Затоа ја проверуваме конзистентноста на ребрата N1->N4 и N4->N1.

Ако N1 работи на прашањето 1, N4 би можел да работи на 2 и 3.

Ако N1 работи на прашањето 2, N4 би можел да работи на 1 и 3.

Ако N1 работи на прашањата 1 и 2, N4 би можел да работи на 3.

Реброто е конзистентно.



	N1	N2	N3	N4	N5
1		X	X		X
2					X
3			X		X
4		X		X	X
5		X	X	X	X
6	X	X		X	

Во обратна насока, реброто е конзистентно.

По arc consistency, почнуваат да се доделуваат вредности со хевристиките. При секое доделување со forward checking проверуваме дали треба да се отстрани некоја вредност од доменот на променливите.

5. Почнуваме со евристиката Minimum remaining values. Членот N5 има само една задача која може да му се додели. Заклучуваме дека N5 мора да работи на прашањето 6.

	N1	N2	N3	N4	N5
1		X	X		X
2					X
3			X		X
4		X		X	X
5		X	X	X	X
6	X	X		X	✓

Со примена на алгоритмот forward checking проверуваме дали доменот на некоја од променливите е променет и заклучуваме дека нема никаква промена бидејќи повеќе членови може да работат на една задача.

Според degree heuristic N1 е член кој е присутен во најголем дел од условите, затоа него му доделуваме вредност. Заклучуваме дека N1 мора да работи на прашањето 5.

На N1 се доделува вредноста 1, tie break за доменот на N4.

За да бидат исполнети условите, N2 и N3 мора да работат на задачи кои се наоѓаат после најмалата вредност на N1. Доделуваме задача 1 на член N1 и задача 6 на N3.

Уште еднаш ги проверуваме условите.

✓ Членот N4 мора да работи само на прашањата 1, 2 или 3.

✓ Членот N1 мора да работи на прашања кои се наоѓаат на листата пред прашањата на кои работи N3.

✓ Членот N5 не сака да прегледува комплексни прашања, затоа мора да работи на прашањето со T/F одговори.

✓ Членот N2 мора да работи на прашања кои се наоѓаат на листата по прашањата на кои работи N1.

✓ Членот N2 не може да работи на прашањата 4, 5 и 6.

✓ Членот N3, поради суеверие, не сака да работи на прашањата со непарни броеви.

✓ Членот N1 не може да работи на прашањето 6.

✓ Членовите N1 и N4 не треба да работат на исто прашање

Дополнително, сите задачи се вклучени и сите членови работат на барем една задача.

Едно можно решение на проблемот е:

	N1	N2	N3	N4	N5
1	✓	X	X	X	X
2	X	X	X	✓	X
3	X	✓	X	✓	X
4	✓	X	X	X	X
5	✓	X	X	X	X
6	X	X	✓	X	✓