Децата на Пакман во потрага по богатство

а) Минимална репрезентација на состојбата на проблемот е к двојки од вредности ((x, y)) кои ги енкодираат x и y координатите на к-те деца на Pacman. Во играта исто така има еден дух со x и y координати, па вкупно има к + 1 двојки од вредности ((x, y)). Покрај духот и децата мора да има и богатство, поради ова има една логичка променлива за некое поле во лавиринтот што укажува на кое поле е богатството.

б) ((M x N)^(k + 1)) x 2^(1)

в) Фактор на разгранување е бројот на деца на секој јазол (надворешниот степен на јазолот)(Максималниот број следни состојби од секој јазол). Во секој момент еден од агентите може да преземе акција “Заврти се лево”, “Заврти се десно”, “Придвижи се напред” и “Стоп”. Значи во секој момент може да превземе една од 4 акции кои ке произведат следна состојба, па според ова максималната вредност на факторот на разгранување е 4.

г) Во состојбата од слика 1, к = 3, односно има 3 деца на пакман. Големината на табелата е 9 x 6. Децата се на координатите (2, 5), (0, 2), (3, 0). Богатсвото е скриено па не ни се познати неговите координати. Духот е на позиција со координати (8, 1). Целна состојба би била секоја состојба на табелата кога еден од децата на пакман ќе го најде богатсвото односно неговите координати (x, y) ќе бидат еднакви со координатите на богатството.

д) Пред да ја извршиме акцијата “Придвижи се напред”, мора да провериме дали полето на кое ќе се придвижи агентот е валидно поле. Ова ќе го направиме со некоја функција која ќе изгледа вака:

```
def check_if_position_valid(x, y, ghost):
    if (x, y) in obstacles or (x, y) == ghost:
        return False

    if x < 0 or x > M or y < 0 or y > N:
        return False

    return True
```

Оваа функција проверува дали полето кое се наоѓа на координати (x, y) е поле на кое има препрека, проверува дали полето е надвор од границите на табелата и проверува дали полето (x, y) е всушност полето на кое се наоѓа духот. Ако барем една од трите проверки е вистина враќа False, во спротивно враќа True. Функцијата за “Придвижи се напред” ќе изгледа вака:

```
def continue_straight(x, y, direction, ghost, actions):
    if direction == "istok":
        new_x = x + 1
        new_y = y
    elif direction == "zapad":
        new_x = x - 1
        new_y = y
    elif direction == "sever":
        new_x = x
        new_y = y + 1
    elif direction == "jug":
        new_x = x
        new_y = y - 1

    if check_if_position_valid(new_x, new_y, ghost):
        actions["ProdolzhiPravo"] = (new_x, new_y, direction, ghost)
```

Други легални акции се “Заврти се лево” и “Заврти се десно” :

```
def turn_left(x, y, direction, ghost, actions):
    if direction == "istok":
        new_x = x
        new_y = y + 1
        new_direction = "sever"
    elif direction == "zapad":
        new_x = x
        new_y = y - 1
        new_direction = "jug"
    elif direction == "sever":
        new_x = x - 1
        new_y = y
        new_direction = "zapad"
```

```
    elif direction == "jug":
        new_x = x + 1
        new_y = y
        new_direction = "istok"

    if check_if_position_valid(new_x, new_y, ghost):
        actions["SvrtiLevo"] = (new_x, new_y, new_direction, ghost)
```

```
def turn_right(x, y, direction, ghost, actions):
    if direction == "istok":
        new_x = x
        new_y = y - 1
        new_direction = "jug"
    elif direction == "zapad":
        new_x = x
        new_y = y + 1
        new_direction = "sever"
    elif direction == "sever":
        new_x = x + 1
        new_y = y
        new_direction = "istok"
    elif direction == "jug":
        new_x = x - 1
        new_y = y
        new_direction = "zapad"

    if check_if_position_valid(new_x, new_y, ghost):
        actions["SvrtiDesno"] = (new_x, new_y, new_direction, ghost)
```

Последна легална акција е “Стоп”.

ѓ) Во нашиот проблем, секоја од акциите има иста цена на чинење, т.е. има цена 1. Тривијална евристика е термин за евристичка функција која назначува константна вредност на секој јазол, односно секогаш враќа ист број. Не можеме да дефинираме нетривијална допустлива евристика. Можеме да дефинираме тривијална допустлива евристика.

е) Сега можеме да дефинираме нетривијална допустлива евристика. Бидејќи ја знаеме колоната, можеме да искористиме евристика како Manhattan минимум или максимум за да одлучиме кое поле од колоната ќе го пребараме за богатството прво.

ж) Најдобар за овој проблем би бил astar search (A*) алгоритмот. A* search алгоритмот е и комплетен и оптимален, се додека му доделиме соодветна евристика. Тој е комбинација од доброто од сите други стратегии за

пребарување, ја вклучува големата брзина на greedy search со оптималноста на Uniform Cost Search. Може да се искористи и Uniform Cost Search поради неговата тривијалност. Може да се искористи и BFS(Breadth First Search) кој е алгоритам за неинформирано пребарување, со него секогаш ќе го најдеме богатсвото, но ќе биде по спор од A*.

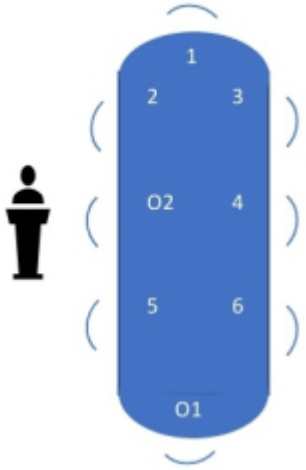
Дипломатска вечера

а)

Променливи се дипломатите VIP1, VIP2, VIP3, VIP4.

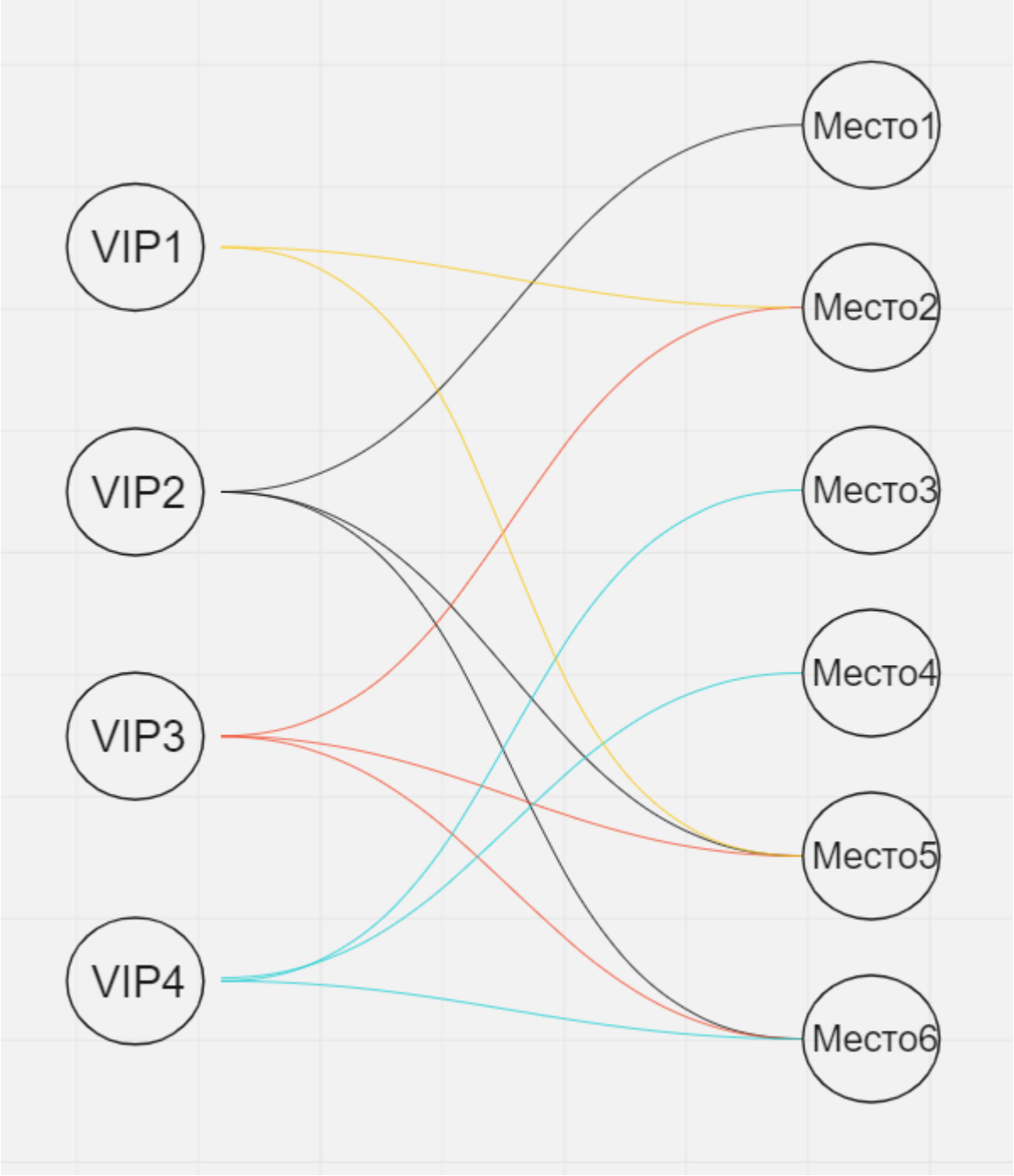
На масата за 8 лица веќе се зафатени 2 места за домаќините O1 и O2. Па домени се местата 1, 2, 3, 4, 5 ,6.

Услови (Множество на ограничувања):



1. Секој гостин добива едно место на масата :).
2. VIP3 сака да биде сместен до еден од организаторите на настанот.
3. Организаторот O2 и VIP1 се пријатели и VIP1 бара да биде сместен до него.
4. VIP4 бара да не биде сместен спротивно од VIP1.
5. VIP2 сака да биде на чело на масата или до организаторот O1.
6. VIP4 не сака да биде на чело на масата.
7. VIP4 бара да не биде свртен со грб кон говорницата.

б)


VIP1
VIP2
VIP3
VIP4
Место1
Место2
Место3
Место4
Место5
Место6

в)

	1	2	3	4	5	6
VIP1		T			T	
VIP2	T				T	T
VIP3		T			T	T
VIP4			T	T		T

Ќе ја примениме евристиката MRV (Minimum remaining values) за избор на следна променлива.

VIP1

Сега ќе ја искористиме евристиката LCV (Least Constraining Value) за избор на вредност која ќе и се додели на променливата избрана според евристиката MRV.

2

	1	2	3	4	5	6
VIP1		**T**				
VIP2	T				T	T
VIP3		T			T	T
VIP4			T	T		T

Во следните чекори ќе го примениме алгоритамот за проверка нанапред (Forward Checking).

	1	2	3	4	5	6
VIP1		**T**				
VIP2	T				T	T
VIP3		T			T	T
VIP4			T	T		T

	1	2	3	4	5	6
VIP1		**T**				
VIP2	T				T	T
VIP3					T	T
VIP4			T	T		T

	1	2	3	4	5	6
VIP1		**T**				
VIP2	T				T	T
VIP3					T	T
VIP4				T		T

По примена на сите унарни услови од проблемот, за домените на секоја променлива, преостануваат само вредносите прикажани во табелата.

г)

Со применување на алгоритамот за проверка нанапред во претходните чекори ја проверивме конзистентноста на проблемот, односно ја проверивме конзистентноста на ребрата.

	1	2	3	4	5	6
VIP1		**T**				
VIP2	T				T	T
VIP3					T	T
VIP4				T		T

д)

	1	2	3	4	5	6
VIP1		**T**				
VIP2	T				T	T
VIP3					T	T
VIP4				T		T

Ќе ја примениме евристиката MRV (Minimum remaining values) за избор на следна променлива.

VIP3

Сега ќе ја искористиме евристиката LCV (Least Constraining Value) за избор на вредност која ќе и се додели на променливата избрана според евристиката MRV.

5

	1	2	3	4	5	6
VIP1		**T**				
VIP2	T				T	T
VIP3					**T**	
VIP4				T		T

Во следните чекори ќе го примениме алгоритамот за проверка нанапред (Forward Checking).

	1	2	3	4	5	6
VIP1		**T**				
VIP2	T				T	T
VIP3					**T**	
VIP4				T		T

	1	2	3	4	5	6
VIP1		**T**				
VIP2	T					T
VIP3					**T**	
VIP4				T		T

Чекорите се повторуваат

	1	2	3	4	5	6
VIP1		**T**				
VIP2	**T**					
VIP3					**T**	
VIP4				T		T

VIP4 може да биде седнат на која било од двете позиции 4 и 6. Па едно решение е:

	1	2	3	4	5	6
VIP1		**T**				
VIP2	**T**					
VIP3					**T**	
VIP4				**T**		