Intel·ligència Artificial

Exercise 1: Heuristic Search

NEUS OLLER MATAS

Enginyeria Informàtica

Universitat Rovira i Virgili

1. Formalitzeu el problema definint els estats i els operadors.

- Els estats seran les caselles del mapa proporcionat de dimensions $10 * 10$. Els estats seran qualsevol configuració de les 100 caselles de la matriu, per tant, el número d'estats possibles serà: $(10 * 10)! = 100!$
- Els operadors seran el moviment que pot fer l'algorisme per avançar de casella, en el nostre cas, el desplaçament vertical o horitzontal.

2. Doneu 3 heurístiques ben diferenciades (no tenen per què ser les 3 millors, però han de ser ben diferents) per intentar trobar el camí/camins més ràpids des de l'estat inicial al final.
3. Per cada heurística, indiqueu si són o no admissibles respecte al temps. No cal que les 3 ho siguin, però almenys n'hi hauria d'haver una d'admissible.

- Heurística de Distància de Manhattan: aquesta heurística calcula la distància de Mamhattan entre la casella actual i la casella final. És admissible ja que mai sobreestima el cost, ja que només permet moviments horitzontals i verticals. Però, no té en compte el temps de viatge.
- Heurística de Menor Temps en Línia Recta: aquesta heurística estima el temps mínim per arribar a la casella final en línia recta. Assumeix que es pot viatjar en línia recta en el menor temps per arribar a la casella final. Pot ser inadmissible ja que no considera restriccions de mitjans de transport.
- Heurística de Temps Mínim Estimat: aquesta heurística considera el temps mínim estimat des de la casella actual fins la casella final, tenint en compte el mitjà de transport disponible. És admissible ja que sempre subestima el temps requerit.

4. Feu un programa que resolgui el problema fent una cerca heurística amb el mètode best first.

```
public ArrayList<Position> BFSearch(Heuristic H) {
        Position finalPos = new Position(9,9);

        ArrayList<State> remaining = new ArrayList<>();   // conté els estat que s'han d'explorar
        remaining.add(new State(new Position(0,0), new State(), 0));

        ArrayList<State> processed = new ArrayList<>();   // conté els estat que ja s'han explorat

        boolean found = false;

        while (!found && !remaining.isEmpty()){

            State s = remaining.get(0);
            remaining.remove(0);

            if(s.samePosition(finalPos)) return s.getPath();

            for(State n : lookNeighbours(s, H)){

                if(!contains(processed, n) && !contains(remaining, n)){

                    remaining.add(n);

                    // amb Best-First s'ordenen els estats amb les heurístiques més baixes
                    remaining.sort((o1, o2) -> {
                        if(o1.getHeuristic() > o2.getHeuristic())
                            return 1;
                        if(o1.getHeuristic() < o2.getHeuristic())
                            return -1;
                        return 0; // o1 == o2
                    });
                }
                processed.add(n);
            }
        }
        return null;
    }
}
```

5. Feu un programa que resolgui el problema fent una cerca heurística amb el mètode A*.

```
public ArrayList<Position> ASSearch(Heuristic H) {

        Position finalPos = new Position(9,9);

        ArrayList<State> remaining = new ArrayList<>(); // conté els estat que s'han d'explorar
        remaining.add(new State(new Position(0,0), new State(), 0));

        ArrayList<State> processed = new ArrayList<>(); // conté els estat que ja s'han explorat

        boolean found = false;

        while (!found && !remaining.isEmpty()){

            State s = remaining.get(0);
            remaining.remove(0);

            if(s.samePosition(finalPos)) return s.getPath();

            for(State n : lookNeighbours(s, H)){

                if(!contains(processed, n)){
                    remaining.add(n);

                    // amb A* es busca explorar primer els camins que tenen un menor cost acumulat
                    remaining.sort((o1, o2) -> {
                        // si o1 > o2
                        if(o1.getAccumulate() + o1.getHeuristic() > o2.getAccumulate() + o2.getHeuristic())
                            return 1;
                        // si o1 > o2
                        if(o1.getAccumulate() + o1.getHeuristic() < o2.getAccumulate() + o2.getHeuristic())
                            return -1;
                        return 0;
                    });
                }
                processed.add(n);
            }
        }
        return null;
    }
}
```

6. Proveu ambdós algorismes amb les 3 heurístiques per a diferents problemes (el de l'enunciat i, almenys, un altre mapa que dissenyeu vosaltres) indicant, per a cada prova:
   - la solució (camí) que s'ha trobat, indicant el temps associat,

Heurística basada en mirar el menor TEMPS.

   - el nombre de nodes que ha "tractat" l'algorisme de cerca per trobar el camí (és a dir, el nombre d'iteracions de cerca que ha fet),

Heurística basada en mirar el menor COST.

   - si la solució trobada és òptima o no respecte al temps.

Heurística anomenada OTHERS on es considera que $1 * temps = 2 * cost$

7. Comenteu els resultats anteriors en relació al disseny de cada heurística i les seves propietats en vers l'algorisme de cerca on s'ha emprat.

mapa.txt

Best-First Search Algorithm

```
*** BFS per COST ***
WBBBBSSBWW
WBSSSSSBBB
BBSSSSSBWW
WWWWWWWBWW
TTTTTTTTTT
STBWWTWWWW
STBWSSSSSS
STBWSSSSSS
STBWWBBBBB
STBBBBWWWW
Nodes: 19 Cost: 11
Time: 153 Other: 158
```

```
*** BFS per TIME ***
WBBBBSSBWW
WBSSSSSBBB
BBSSSSSBWW
WWWWWWWBWW
TTTTTTTTTT
STBWWTWWWW
STBWSSSSSS
STBWSSSSSS
STBWWBBBBB
STBBBBWWWW
Nodes: 23 Cost: 61
Time: 85 Other: 111
```

```
*** BFS per OTHER ***
WBBBBSSBWW
WBSSSSSBBB
BBSSSSSBWW
WWWWWWWBWW
TTTTTTTTTT
STBWWTWWWW
STBWSSSSSS
STBWSSSSSS
STBWWBBBBB
STBBBBWWWW
Nodes: 23 Cost: 61
Time: 85 Other: 111
```

A Algorithm*

```
*** A* per COST ***
WBBBBSSBWW
WBSSSSSBBB
BBSSSSSBWW
WWWWWWWBWW
TTTTTTTTTT
STBWWTWWWW
STBWSSSSSS
STBWSSSSSS
STBWWBBBBB
STBBBBWWWW
Nodes: 19 Cost: 9
Time: 160 Other: 164
```

```
*** A* per TIME ***
WBBBBSSBWW
WBSSSSSBBB
BBSSSSSBWW
WWWWWWWBWW
TTTTTTTTTT
STBWWTWWWW
STBWSSSSSS
STBWSSSSSS
STBWWBBBBB
STBBBBWWWW
Nodes: 19 Cost: 44
Time: 70 Other: 88
```

```
*** A* per OTHER ***
WBBBBSSBWW
WBSSSSSBBB
BBSSSSSBWW
WWWWWWWBWW
TTTTTTTTTT
STBWWTWWWW
STBWSSSSSS
STBWSSSSSS
STBWWBBBBB
STBBBBWWWW
Nodes: 21 Cost: 42
Time: 72 Other: 90
```

Comentari dels resultats

- "per COST": si fem una comparativa entre els algorismes BFS i A*, podem observar que el cost té dues unitats de diferència el temps en té sis ($2 * 3$). Per tant, es podria dir que l'algorisme A* ens dona més bons resultats en comparació amb l'algorisme BFS.
- "per TEMPS": ens torbem, també, que A* torna a ser més òptim ja que té 15 unitats de temps menys respecte BFS

mapa1.txt

Best-First Search Algorithm

```
*** BFS per COST ***
WTBBTTBTTT
WTSBSTSBSW
WTTSSSWWWW
BBBBBBTWSW
SSSSSTSSSS
SBTTBTWWWW
WSSSBTBSSW
WBSWWBTBWW
WBBSTBTWWW
WBWWWBTTTW
Nodes: 19 Cost: 18
Time: 130 Other: 138
```

```
*** BFS per TIME ***
WTBBTTBTTT
WTSBSTSBSW
WTTSSSWWWW
BBBBBBTWSW
SSSSSTSSSS
SBTTBTWWWW
WSSSBTBSSW
WBSWWBTBWW
WBBSTBTWWW
WBWWWBTTTW
Nodes: 19 Cost: 46
Time: 47 Other: 64
```

```
*** BFS per OTHER ***
WTBBTTBTTT
WTSBSTSBSW
WTTSSSWWWW
BBBBBBTWSW
SSSSSTSSSS
SBTTBTWWWW
WSSSBTBSSW
WBSWWBTBWW
WBBSTBTWWW
WBWWWBTTTW
Nodes: 19 Cost: 46
Time: 47 Other: 64
```

A Algorithm*

```
*** A* per COST ***
WTBBTTBTTT
WTSBSTSBSW
WTTSSSWWWW
BBBBBBTWSW
SSSSSTSSSS
SBTTBTWWWW
WSSSBTBSSW
WBSWWBTBWW
WBBSTBTWWW
WBWWWBTTTW
Nodes: 19 Cost: 18
Time: 130 Other: 138
```

```
*** A* per TIME ***
WTBBTTBTTT
WTSBSTSBSW
WTTSSSWWWW
BBBBBBTWSW
SSSSSTSSSS
SBTTBTWWWW
WSSSBTBSSW
WBSWWBTBWW
WBBSTBTWWW
WBWWWBTTTW
Nodes: 19 Cost: 46
Time: 47 Other: 64
```

```
*** A* per OTHER ***
WTBBTTBTTT
WTSBSTSBSW
WTTSSSWWWW
BBBBBBTWSW
SSSSSTSSSS
SBTTBTWWWW
WSSSBTBSSW
WBSWWBTBWW
WBBSTBTWWW
WBWWWBTTTW
Nodes: 19 Cost: 46
Time: 47 Other: 64
```

Comentari dels resultats

No hi ha cap millora ni cap algorisme sembla donar millors resultats que l'altre.

8. Per a cada heurística que heu dissenyat, hauríeu trobat la solució si haguéssiu aplicat hill climing? No cal implementar l'algorisme hill climbing només justificar-ho

Segurament donaria una solució diferent, ja que al no tenir cua de pendents i agafar directament el que tingui millor heurística, no aniria pel camí més òptim globalment parlant.