Wild

Esame di Ingegneria della Conoscenza

Greco Arianna, mat.591455

Greco Alberto, mat.640450

INDICE **pagina**

1-INTRODUZIONE

Il nostro progetto, Wild, è un Sistema Esperto basato su conoscenza che consiglia il percorso ottimale in un parco naturale in base alle condizioni meteo e al tipo di attività che si vuole fare.

Per implementarlo abbiamo scelto di utilizzare l'algoritmo di ricerca A* che permette di avere il percorso minimo in un grafo, in base a una euristica appropriata, partendo dal nodo Start per arrivare al nodo Goal.

Acquisiamo tramite il Datalog la conoscenza del nostro sistema e creiamo random il nostro grafo e i nostri nodi così da avere sempre un percorso diverso come output.

2-PYTHON

Il nostro sistema è progettato in Python 3.9 e il Datalog è stato creato su Jupyter Lab.

3-DATALOG

Datalog è un linguaggio di interrogazione per basi di dati derivato da Prolog basato su regole di deduzione per l'estrazione dell'informazione e l'integrazione dei dati.

E' stato utilizzato in questo lavoro per acquisire informazioni sui vari tipi di terreni nel parco naturale in caso di sole o di pioggia, per consigliare al meglio il tipo di percorso da intraprendere in base al tipo di passeggiata che si vuole fare.

Abbiamo utilizzato la libreria pyDatalog in Python, che si aggiunge alle librerie di Python per il paradigma di programmazione logica.

Abbiamo creato i nostri termini:

X, *(variabile utilizzata per le query)*

Sole, Pioggia *(per indicare il meteo)*

tipoTerreno, Sabbioso, Breccioso, Terroso, Fangoso, Erboso, Alberato, Roccioso, *(per indicare il tipo di terreno)*

modo, APiedi, InBici, InMotoCross*, (per indicare in che modo si intende fare il percorso)*

attraversabileApiedi,attraversabileInbici,attraversabileInmoto,

tipoPercorso, Passeggiata, Avventura, Panoramico, Percorso*(per indicare che tipo di passeggiata si vuole fare)*

Abbiamo creato le nostre clausole: tipoTerreno, modo*(indica il modoin cui si intende fare il percorso)*, tipoPercorso

E le nostre query: attraversabileApiedi, attraversabileInbici, attraversabileInmoto, Percorso

Abbiamo integrato tutto il Datalog nel nostro codice.

4-PROGETTAZIONE

Prima di definire l'algoritmo di ricerca abbiamo creato il grafo dove l'algoritmo andrà a lavorare.

La classe graph ci permette di sapere i vicini in ogni posizione nel grafo

```
class Graph(Protocol):
    def neighbors(self, id: Location) → List[Location]: pass


class SimpleGraph:
    def __init__(self):
        self.edges: Dict[Location, List[Location]] = {}

    def neighbors(self, id: Location) → List[Location]:
        return self.edges[id]
```

Abbiamo creato i nodi random attribuendo ad ognuno un peso specifico in base alla scelta dell'utente.

Creazione del Grafo random

```
def randgraph(start, goal):
    for i in range(diagram4.height):
        for y in range(diagram4.width):
            if (not i == 0) & (not i == diagram4.height-1) & (not y == 0) & (not y == diagram4.width-1):
                if randint(1, 15) < 3:
                    if (not start.__eq__((i, y))) & (not goal.__eq__((i, y))):
                        diagram4.walls.append((i, y))
                        diagramnode[(i, y)] = Node(-1, False)
                else:
                    typenode((i, y))
            else:
                typenode((i, y))
```

Attribuzione del costo specifico in base alla scelta dell'utente

```
def setweights(results,me,type):
    for i in range(diagram4.height):
        for y in range(diagram4.width):
            diagram4.weights[(i, y)] = 9
    for i in range(diagram4.height):
        for y in range(diagram4.width):
            x = groundType[diagramnode[(i, y)].ground]
            for st in results.data:
                    if x == st[0]:
                        if me == 'Sole':
                            u=weightsforsun[type][x]
                            diagram4.weights[(i, y)] = weightsforsun[type][x]
                        else:
                            diagram4.weights[(i, y)] = weightsforrain[type][x]
```

Abbiamo creato una coda con priorità per la frontiera di A* così da restituire sempre l'elemento con costo minore utilizzando la tupla[priority,item] e i metodi standard di una coda.

```
class PriorityQueue:
    def __init__(self):
        self.elements: List[Tuple[float, T]] = []

    def empty(self) -> bool:
        return not self.elements

    def put(self, item: T, priority: float):
        heapq.heappush(self.elements, (priority, item))

    def get(self) -> T:
        return heapq.heappop(self.elements)[1]
```

5-ALGORITMO DI RICERCA A*

Abbiamo scelto l'algoritmo di ricerca A* perché combina sia l'algoritmo di ricerca a costo minimo, sia la ricerca Best-first euristica.

Nella selezione del percorso, A* segue molti percorsi distinti, fino a trovare un percorso a costo minimo, per questo è ottimo per risolvere il nostro problema di ricerca del percorso in una foresta.

```
def a_star_search(graph: WeightedGraph, start: Location, goal: Location, type):
    frontier = PriorityQueue()
    frontier.put(start, 0)
    came_from: Dict[Location, Optional[Location]] = {}
    cost_so_far: Dict[Location, float] = {}
    came_from[start] = None
    cost_so_far[start] = 0

    while not frontier.empty():
        current: Location = frontier.get()

        if current == goal:
            break

        for next in graph.neighbors(current):
            new_cost = cost_so_far[current] + graph.cost(current, next)
            if next not in cost_so_far or new_cost < cost_so_far[next]:
                cost_so_far[next] = new_cost
                priority = new_cost + heuristic(next, goal, type)
                frontier.put(next, priority)
                came_from[next] = current

    return came_from, cost_so_far
```

L'euristica utilizzata è il calcolo del teorema di Pitagora

```
def heuristic(a: GridLocation, b: GridLocation, type) -> float:
    (x1, y1) = a
    (x2, y2) = b
    return sqrt(abs(x1 - x2) ** 2 + abs(y1 - y2) ** 2)
```

Abbiamo scelto di avere un nodo di partenza e un solo nodo Goal che corrispondono alle posizioni (0,0) nodo di partenza e (9,9) nodo di arrivo.

6-CASI DI TEST

All’avvio apparirà questa schermata che permette di scegliere il tipo di percorso che si vuole fare.

```
  ♠ W I L D ♠

Ti aiuteremo a scegliere il percorso adeguato per la tua giornata in un parco naturale in base alle tue preferenze e al meteo

Scegli se vuoi fare una:
 - Passeggiata
 - Panoramico
 - Avventura
 Ricorda di inserire la tua scelta con la prima lettera maiuscola
```

Una volta scelto il percorso e inserito correttamente, apparirà questo

```
  ♠ W I L D ♠

Ti aiuteremo a scegliere il percorso adeguato per la tua giornata in un parco naturale in base alle tue preferenze e al meteo

Scegli se vuoi fare una:
 - Passeggiata
 - Panoramico
 - Avventura
 Ricorda di inserire la tua scelta con la prima lettera maiuscola
Passeggiata


Il meteo di oggi è
(Scegli inserendo 1 oppure 2):
1.Soleggiato o Abbastanza soleggiato
2.Piovoso o ha appena smesso di piovere
```

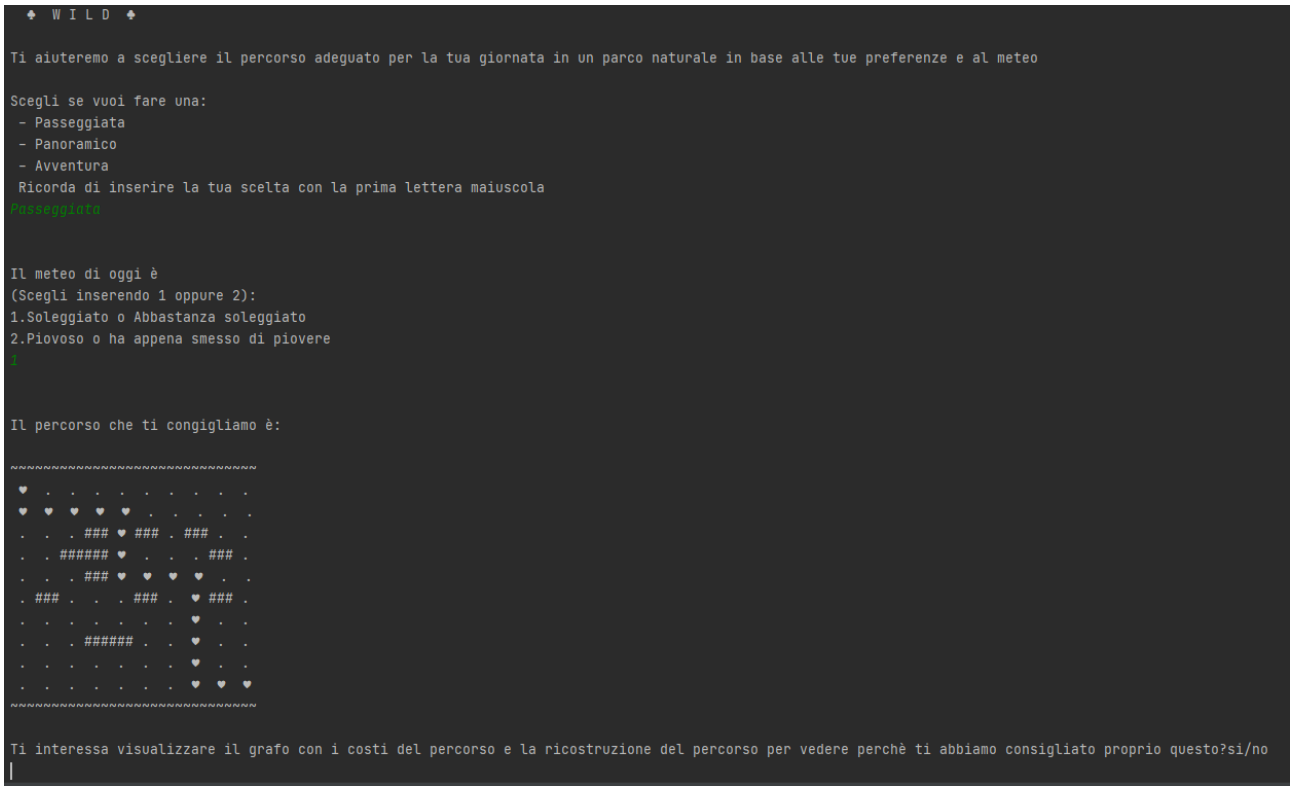
Dove l’utente dovrà scegliere il meteo del giorno per avere il percorso migliore in base alle condizioni del terreno.

```
Il meteo di oggi è
(Scegli inserendo 1 oppure 2):
1.Soleggiato o Abbastanza soleggiato
2.Piovoso o ha appena smesso di piovere
1
Il percorso che ti congigliamo è:

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 ♥ ♥ . . . . . . . . .
 . ♥ ♥ ♥ . . ### . . .
 . . . ♥ . ### . . . .
 . . . ♥ ###### . . . .
 . . . ♥ . . . . . .
 . . ### ♥ ### . . . . .
 . ### . ♥ . . . . . .
 . . ### ♥ . . . . . .
 . . . ♥ ♥ ♥ . . . . .
 . . . . . ♥ ♥ ♥ ♥ ♥
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Ti interessa visualizzare il grafo con i costi del percorso e la ricostruzione del percorso per vedere perchè ti abbiamo consigliato proprio questo?si/no
```

Una volta ottenuto il percorso, c’è la possibilità di visualizzare il grafo in maniera diversa. Infatti si potrà scegliere se visualizzare il grafo con i costi in ogni posizione e il grafo con la ricostruzione del percorso partendo dal nodo goal, fino al nodo di partenza.



Se la scelta sarà si,verranno visualizzati i due grafi,

```
Grafo con i costi:

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 S  9  18 21 30 33 42 45 51 54
 3  9  18 24 30 39 45 54 60 63
 9  12 15### 33### 54### 69 66
 15 21###### 42 51 60 63### 69
 24 27 33### 45 54 57 60 69 72
 33### 42 51 54### 66 63### 75
 42 51 45 54 63 66 75 66 69 78
 45 54 54###### 75 75 69 78 87
 51 54 63 69 78 84 81 78 84 93
 54 63 72 72 81 90 90 81 87 X
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~



Grafo ricostruito partendo dal nodo Goal:


~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 S  <  <  <  <  <  <  <  <
 ^  <  <  <  <  <  <  ^  ^
 ^  ^  < ### ^ ### ^ ### ^  ^
 ^  ^ ###### ^  <  <  v ### ^
 ^  ^  < ### ^  <  <  <  <  <
 ^ ### ^  <  ^ ### ^  ^ ### ^
 ^  <  ^  <  ^  <  >  ^  <  <
 ^  <  ^ ###### ^  >  ^  <  <
 ^  <  ^  <  <  ^  ^  ^  <  <
 ^  ^  ^  ^  <  <  >  ^  <  X
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

ripetere esecuzione?si/no
```

Se la scelta sarà no verrà chiesto all'utente se vuole richiedere un altro percorso

```
ripetere esecuzione?si/no
si
Scegli se vuoi fare una:
 - Passeggiata
 - Panoramico
 - Avventura
 Ricorda di inserire la tua scelta con la prima lettera maiuscola
```

Se la scelta sarà no, l’esecuzione finirà.

```
Ti interessa visualizzare il grafo con i costi del percorso e la ricostruzione del percorso per vedere perchè ti abbiamo consigliato proprio questo?si/no
no

ripetere esecuzione?si/no
no
Arrivederci!

Process finished with exit code 0
```