Universidad de Las Américas

Facultad de Ingenierías y Ciencias Aplicadas

Ingeniería de Software

1. DATOS DE LOS ALUMNOS:

- Sebastian Luna
- Iñaki Manosalva
- Francis Ríos

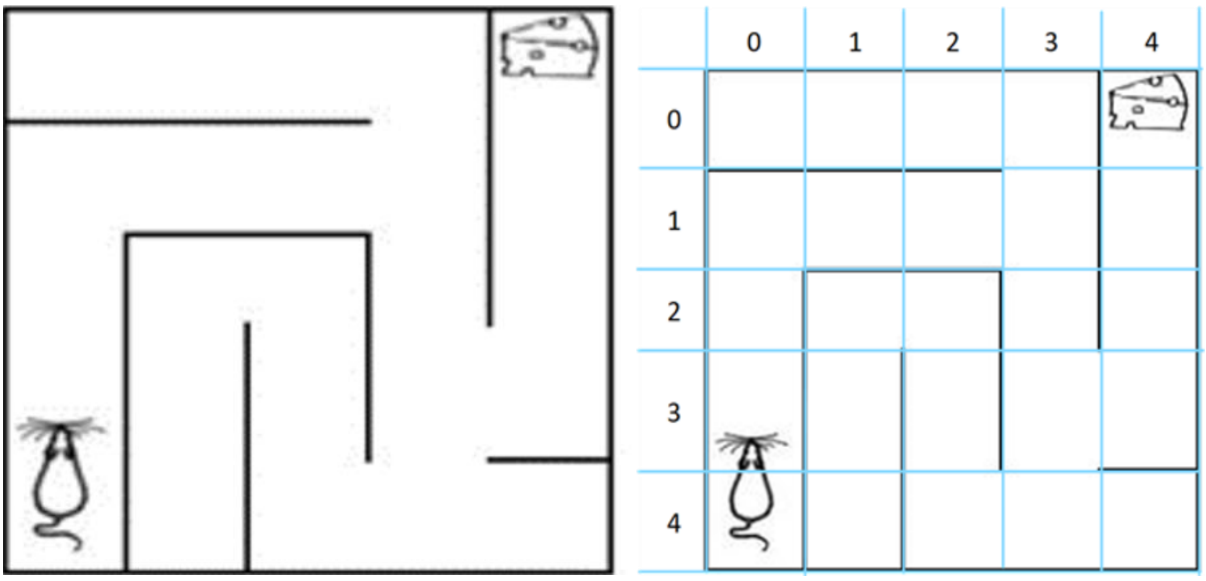
2. TEMA:

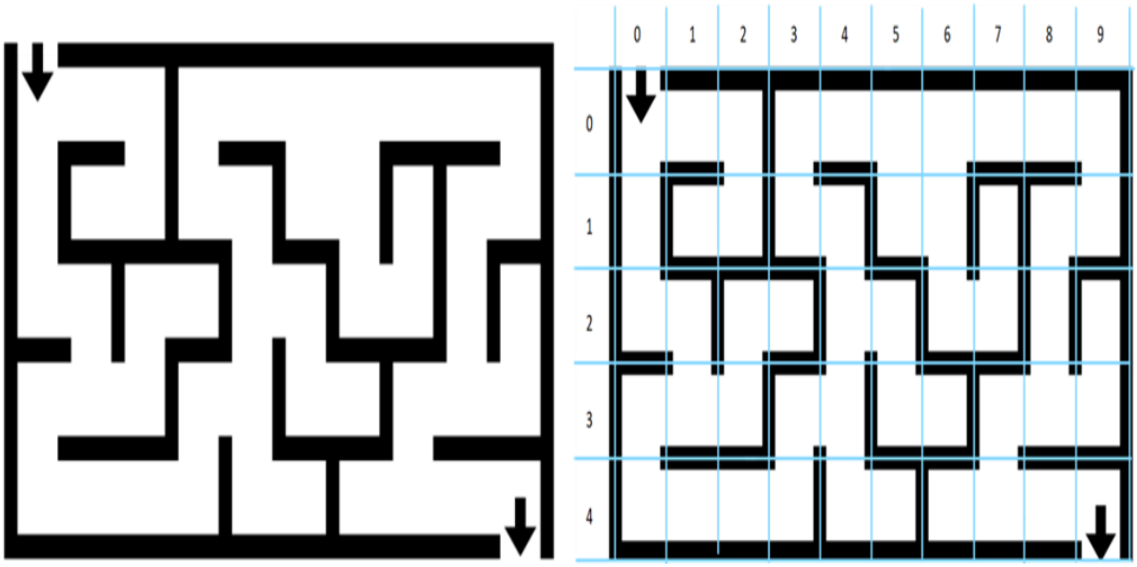
Búsqueda Ciegas y Heurísticas

3. DESARROLLO:

Laberintos a grafos

Para comenzar a resolver el ejercicio primero se transformó a los laberintos (imagenes) a grafos. Esto se realizó dividiendo la imagen en eje X e Y para poder mapear sus coordenadas como nodos.





Prosiguiendo con el ejercicio, la tarea tenía ficheros adjuntos con un código base, a partir de ese código integramos más funciones que eran necesarias. Para los métodos de BFS y DFS se reutilizó código visto en clase y se les realizaron pequeñas modificaciones a los métodos.

```
def BFS(self, wgraph, s, target):
    parents = {}
    parents[s] = None
    visited = { gi: False for gi in self.edges.keys() }
    queue = []
    queue.append(s)
    visited[s] = True

    while queue:
        s = queue.pop(0)
        draw_square(wgraph, s)

        if s == target:
            break

        for i in self.edges[s]:
            if not visited[i]:
                queue.append(i)
                visited[i] = True
                parents[i] = s

    return parents
```

```
def DFS(self, wgraph, s, target):
    parents = {}
    parents[s] = None
    visited = { gi: False for gi in self.edges.keys() }
    queue = []
    queue.append(s)
    visited[s] = True

    while queue:
        s = queue.pop()
        draw_square(wgraph, s)

        if s == target:
            break

        for i in self.edges[s]:
            if not visited[i]:
                queue.append(i)
                visited[i] = True
                parents[i] = s

    return parents
```

Se modificó también el método “heuristic”, a esté se le agregó dos diccionarios con información. Los valores **key** corresponden a las coordenadas de cada nodo, los valores **value** corresponden a información aleatoria puesto que no teníamos una base sólida sobre que basar nuestros datos de heurística. El diccionario **H1** corresponde al laberinto más grande, mientras que el diccionario **H2** corresponde al laberinto más pequeño.

```python
def heuristic(H ,id):
    """
    Builds 'H' heuristic
    """
    H1 = {
        '0,0': 3.215, '0,1': 4.363, '0,2': 1.426, '0,3': 1.14,  '0,4': 4.541,
        '1,0': 5.261, '1,1': 1.941, '1,2': 1.666, '1,3': 1.657, '1,4': 4.103,
        '2,0': 4.845, '2,1': 1.046, '2,2': 2.308, '2,3': 1.311, '2,4': 5.221,
        '3,0': 5.172, '3,1': 1.482, '3,2': 5.727, '3,3': 2.208, '3,4': 2.17,
        '4,0': 2.364, '4,1': 5.858, '4,2': 3.88,  '4,3': 1.659, '4,4': 1.204,
        '5,0': 4.217, '5,1': 1.766, '5,2': 3.968, '5,3': 4.957, '5,4': 3.913,
        '6,0': 1.448, '6,1': 5.651, '6,2': 2.655, '6,3': 4.211, '6,4': 3.575,
        '7,0': 1.15,  '7,1': 1.999, '7,2': 5.679, '7,3': 4.27,  '7,4': 3.122,
        '8,0': 1.495, '8,1': 1.963, '8,2': 3.933, '8,3': 2.439, '8,4': 5.741,
        '9,0': 4.556, '9,1': 3.521, '9,2': 5.934, '9,3': 4.35,  '9,4': 2.705
    }

    H2 = {      You, 5 minutes ago • Uncommitted changes
        '0,0': 3.48, '0,1': 1.76, '0,2': 5.07, '0,3': 5.45, '0,4': 4.44,
        '1,0': 5.47, '1,1': 2.28, '1,2': 3.79, '1,3': 1.75, '1,4': 2.37,
        '2,0': 5.95, '2,1': 3.55, '2,2': 4.42, '2,3': 2.38, '2,4': 3.56,
        '3,0': 2.26, '3,1': 2.85, '3,2': 1.57, '3,3': 1.94, '3,4': 4.95,
        '4,0': 5.15, '4,1': 1.67, '4,2': 5.87, '4,3': 5.89, '4,4': 5.62
    }

    return H1[id] if H else H2[id]
```

Se modificó el método “geo_pos”, de igual manera se le agregaron dos diccionarios con información. El diccionario **G1** corresponde al laberinto más grande, mientras que el diccionario **G2** corresponde al laberinto más pequeño. Los valores **key** corresponden a las coordenadas de cada nodo, los valores **value** corresponden a la información del pixel aproximado de cada nodo en su correspondiente **imagen.**

```python
def geo_pos(G, id):
    """
    Builds 'G's' positional information
    The map is a png image used as backgroud,
    the position corresponds to an approximated pixel
    for each coord
    """
    G1 = {
        '0,0': (65, 100), '0,1': (65, 190), '0,2': (65, 270), '0,3': (65, 360), '0,4': (65, 450),
        '1,0': (115, 100), '1,1': (115, 190), '1,2': (115, 270), '1,3': (115, 360), '1,4': (115, 450),
        '2,0': (160, 100), '2,1': (160, 190), '2,2': (160, 270), '2,3': (160, 360), '2,4': (160, 450),
        '3,0': (200, 100), '3,1': (200, 190), '3,2': (200, 270), '3,3': (200, 360), '3,4': (200, 450),
        '4,0': (250, 100), '4,1': (250, 190), '4,2': (250, 270), '4,3': (250, 360), '4,4': (250, 450),
        '5,0': (295, 100), '5,1': (295, 190), '5,2': (295, 270), '5,3': (295, 360), '5,4': (295 , 450),
        '6,0': (340, 100), '6,1': (340, 190), '6,2': (340, 270), '6,3': (340, 360), '6,4': (340, 450),
        '7,0': (395, 100), '7,1': (395, 190), '7,2': (395, 270), '7,3': (395, 360), '7,4': (395, 450),
        '8,0': (440, 100), '8,1': (440, 190), '8,2': (440, 270), '8,3': (440, 360), '8,4': (440, 450),
        '9,0': (490, 100), '9,1': (490, 190), '9,2': (490, 270), '9,3': (490, 360), '9,4': (490, 450)
    }

    G2 = {
        '0,0': (110, 85), '0,1': (110, 180), '0,2': (110, 260), '0,3': (110, 350), '0,4': (110, 450),
        '1,0': (200, 85), '1,1': (200, 180), '1,2': (200, 260), '1,3': (200, 350), '1,4': (200, 450),
        '2,0': (290, 85), '2,1': (290, 180), '2,2': (290, 260), '2,3': (290, 350), '2,4': (290, 450),
        '3,0': (380, 85), '3,1': (380, 180), '3,2': (380, 260), '3,3': (380, 350), '3,4': (380, 450),
        '4,0': (470, 85), '4,1': (470, 180), '4,2': (470, 260), '4,3': (470, 350), '4,4': (470, 450)
    }

    return G1[id] if G else G2[id]
```

Al método “draw_square” se le agregó un parámetro adicional “graph” para poder saber con que grafo estamos trabajando los métodos necesarios.

```python
def draw_square(graph, node_id, correction = 500, color="medium sea green", scale=1, ts=None, text=None):
    """
    Draw a square in turtle over the map background to
    indicate the exapnsion of the search and the shortest path
    node_id: the corresponding node in the graph (city)
    correction: corrects the origin of the geo_pos pixels
    ts: a turtle object can be passed to the function,
    if not a turtle object is created
    To animante the shortest path a single turtle object is
    define outside the function and passed as parameter
    """

    # correctionx = 600 if graph else 182
    # correctiony = 327 if graph else 196

    if ts == None:
        ts = t.Turtle(shape="square")
    ts.shapesize(0.5, 0.5)
    ts.color(color)
    ts.penup()
    x, y = geo_pos(graph, node_id)
    ts.goto(x*scale, correction - y*scale)
    if text != None:
        ts.write(str(text), font=("Arial", 20, "normal"))

def costOfPath(path, graph):
    # Returns the cumulated cost of path
    cum_costs = [0]
    for i in range(len(path)-1):
        cum_costs += [graph.cost(path[i], path[i+1])]

    return cumsum(cum_costs)
```

Se implementó el método “greedy” correspondiente a la búsqueda avara, se reutilizó el código del método “aStar”, sin embargo, se le quitó la invocación al cálculo heurístico puesto que la búsqueda avara no toma en cuenta este cálculo.

```python
def greedy(wgraph, graph, start, target):
    openL = []
    openL.append((start, 0))
    parents = {}
    costSoFar = {}
    parents[start] = None
    costSoFar[start] = 0

    while bool(len(openL)):
        # print(openL)
        current = findleastF(openL)
        draw_square(wgraph, current)  # Draw search expansion

        if current == target:
            break

        for successor in graph.neighbors(current):
            newCost = costSoFar[current] + graph.cost(current, successor)
            if successor not in costSoFar or newCost < costSoFar[successor]:
                costSoFar[successor] = newCost
                priority = newCost
                openL.append((successor, priority))
                parents[successor] = current

    return parents
```

Se implementaron cuatro métodos adicionales uno por cada búsqueda solicitada. Estos métodos dibujarán gráficamente en las imágenes la expansión de cada búsqueda y su camino más corto correspondientemente.

```python
def drawBFS(graph, G1, G2, startNode, endGraph1, endGraph2):
    # Define screen and world coordinates
    screen = t.Screen()
    screen.title("BUSQUEDA EN ANCHURA")
    screen.setup(500, 500)
    t.setworldcoordinates(0, 0, 500, 500)
    # Use image as backgroud (image is 500x500 pixels)
    bg = './img/grafo_laberinto_1.png' if graph else './img/grafo_laberinto_2.png'
    t.bgpic(bg)

    # Get image anchored to left-bottom corner (sw: southwest)
    canvas = screen.getcanvas()
    canvas.itemconfig(screen._bgpic, anchor="sw")

    parentsBFS = G1.BFS(graph, startNode, endGraph1) if graph else G2.BFS(graph, startNode, endGraph2)

    shortest_path = G1.pathfromOrigin(startNode, endGraph1, parentsBFS) if graph else G2.pathfromOrigin(startNode, endGraph2, parentsBFS)

    print("\nBUSQUEDA EN ANCHURA")
    print(f"Parents: {parentsBFS}")
    print(f"Shortest Path: {shortest_path}")

    # Calculating the cost of the shortest path
    cost_tsp = costOfPath(shortest_path, G1) if graph else costOfPath(shortest_path, G2)

    # Draw shortest path
    for ni in shortest_path:
        draw_square(graph ,ni, color="salmon")

    # Animate shortest path agent and include cost
    tsp = t.Turtle(shape="square")

    for i, ni in enumerate(shortest_path):
        draw_square(graph ,ni, color="dodger blue", ts=tsp, text=cost_tsp[i])

```

```python
def drawDFS(graph, G1, G2, startNode, endGraph1, endGraph2):
    # Define screen and world coordinates
    screen = t.Screen()
    screen.title("BUSQUEDA EN PROFUNDIDAD")

    screen.setup(500, 500)
    t.setworldcoordinates(0, 0, 500, 500)
    # Use image as backgroud (image is 500x500 pixels)
    bg = './img/grafo_laberinto_1.png' if graph else './img/grafo_laberinto_2.png'
    t.bgpic(bg)

    # Get image anchored to left-bottom corner (sw: southwest)
    canvas = screen.getcanvas()
    canvas.itemconfig(screen._bgpic, anchor="sw")

    parentsDFS = G1.DFS(graph, startNode, endGraph1) if graph else G2.DFS(graph, startNode, endGraph2)

    shortest_path = G1.pathfromOrigin(startNode, endGraph1, parentsDFS) if graph else G2.pathfromOrigin(startNode, endGraph2, parentsDFS)

    print("\nBUSQUEDA EN PROFUNDIDAD")
    print(f"Parents: {parentsDFS}")
    print(f"Shortest Path: {shortest_path}")

    # Calculating the cost of the shortest path
    cost_tsp = costOfPath(shortest_path, G1) if graph else costOfPath(shortest_path, G2)

    # Draw shortest path
    for ni in shortest_path:
        draw_square(graph ,ni, color="salmon")

    # Animate shortest path agent and include cost
    tsp = t.Turtle(shape="square")

    for i, ni in enumerate(shortest_path):
        draw_square(graph ,ni, color="dodger blue", ts=tsp, text=cost_tsp[i])
```

```python
def drawGreedy(graph, G1, G2, startNode, endGraph1, endGraph2):
    # Define screen and world coordinates
    screen = t.Screen()
    screen.title("BUSQUEDA AVARA")

    screen.setup(500, 500)
    t.setworldcoordinates(0, 0, 500, 500)
    # Use image as backgroud (image is 500x500 pixels)
    bg = './img/grafo_laberinto_1.png' if graph else './img/grafo_laberinto_2.png'
    t.bgpic(bg)

    # Get image anchored to left-bottom corner (sw: southwest)
    canvas = screen.getcanvas()
    canvas.itemconfig(screen._bgpic, anchor="sw")

    parentsGreedy = greedy(graph, G1, startNode, endGraph1) if graph else greedy(graph, G2, startNode, endGraph2)

    # Calculating and printing the shortest path
    shortest_path = G1.pathfromOrigin(startNode, endGraph1, parentsGreedy) if graph else G2.pathfromOrigin(startNode, endGraph2, parentsGreedy)

    print("\nBUSQUEDA AVARA")
    print(f"Parents: {parentsGreedy}")
    print(f"Shortest Path: {shortest_path}")

    # Calculating the cost of the shortest path
    cost_tsp = costOfPath(shortest_path, G1) if graph else costOfPath(shortest_path, G2)

    # Draw shortest path
    for ni in shortest_path:
        draw_square(graph ,ni, color="salmon")

    # Animate shortest path agent and include cost
    tsp = t.Turtle(shape="square")

    for i, ni in enumerate(shortest_path):
        draw_square(graph ,ni, color="dodger blue", ts=tsp, text=cost_tsp[i])
```

```python
def drawAstar(graph, G1, G2, startNode, endGraph1, endGraph2):
    # Define screen and world coordinates
    screen = t.Screen()
    screen.title("BUSQUEDA A*")

    screen.setup(500, 500)
    t.setworldcoordinates(0, 0, 500, 500)
    # Use image as backgroud (image is 500x500 pixels)
    bg = './img/grafo_laberinto_1.png' if graph else './img/grafo_laberinto_2.png'
    t.bgpic(bg)

    # Get image anchored to left-bottom corner (sw: southwest)
    canvas = screen.getcanvas()
    canvas.itemconfig(screen._bgpic, anchor="sw")

    parentsAstar = aStar(graph, G1, startNode, endGraph1) if graph else aStar(graph, G2, startNode, endGraph2)

    # Calculating and printing the shortest path
    shortest_path = G1.pathfromOrigin(startNode, endGraph1, parentsAstar) if graph else G2.pathfromOrigin(startNode, endGraph2, parentsAstar)

    print("\nBUSQUEDA A*")
    print(f"Parents: {parentsAstar}")
    print(f"Shortest Path: {shortest_path}")

    # Calculating the cost of the shortest path
    cost_tsp = costOfPath(shortest_path, G1) if graph else costOfPath(shortest_path, G2)

    # Draw shortest path
    for ni in shortest_path:
        draw_square(graph ,ni, color="salmon")

    # Animate shortest path agent and include cost
    tsp = t.Turtle(shape="square")

    for i, ni in enumerate(shortest_path):
        draw_square(graph ,ni, color="dodger blue", ts=tsp, text=cost_tsp[i])
```

Ya finalizando con los cambios, para el método “main” se modificó el input de la información, se crearon dos variables que representan la “meta” para cada grafo en

particular, por último, se crearon dos grafos. **G1** corresponde al laberinto más grande y **G2** al laberinto más pequeño.

```
394  def main(argv):
395
396      """
397      Usage:
398        python main_bc_h.py graph startNode
399        Target for graph 0 will always be (4,0)
400        Target for graph 1 will always be (9,4)
401        <0> <startNode>:  Any coord from (0,0) to (4,0)
402        <1> <startNode>:  Any coord from (0,0) to (9,4)
403
404      Example:
405        python main_bc_h.py 0 0,4
406        python main_bc_h.py 1 0,0
407      """
408
409      if len(argv) != 2:
410          print(main.__doc__)
411      else:
412          graph = int(argv[0])
413          startNode = argv[1]
414
415          # Always Bucarest due to heuristic is given for this end city
416          endGraph1 = '9,4'
417          endGraph2 = '4,0'
418
419          G1 = Graph()  # Builds a larger maze Graph
420          G2 = Graph()  # Builds a shorter maze Graph
421
```

También se crearon la lista de adyacencia (**edges**) y la lista de pesos (**weights**) para cada grafo (**G1, G2**).

```
# Adding edges (adjacency list)
G1.edges = {···

G2.edges = {···

# Adding weights to edges
G1.weights = {···

G2.weights = {···
```

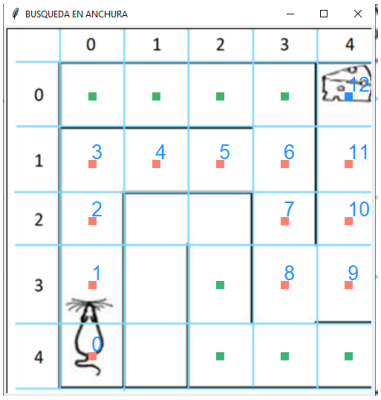
Se realizan validaciones a los inputs que llegan por consola y finalmente ejecutamos los métodos correspondientes para dibujar cada búsqueda en pantalla.

```
        if graph:
            if startNode not in G1.edges.keys():
                return print("Coordenada no existe en grafo 1")
        else:
            if startNode not in G2.edges.keys():
                return print("Coordenada no existe en grafo 2")

        # DRAW ALL SEARCHS

        drawBFS(graph, G1, G2, startNode, endGraph1, endGraph2)
        time.sleep(2)
        t.clearscreen()

        drawDFS(graph, G1, G2, startNode, endGraph1, endGraph2)
        time.sleep(2)
        t.clearscreen()

        drawGreedy(graph, G1, G2, startNode, endGraph1, endGraph2)
        time.sleep(2)
        t.clearscreen()

        drawAstar(graph, G1, G2, startNode, endGraph1, endGraph2)
        time.sleep(2)
        # t.clearscreen()

        """
        CODIGO
        """

        t.exitonclick()  # Al hacer clic sobre la ventana grafica se cerrara


if __name__ == "__main__":
    import sys
    main(sys.argv[1:])
```

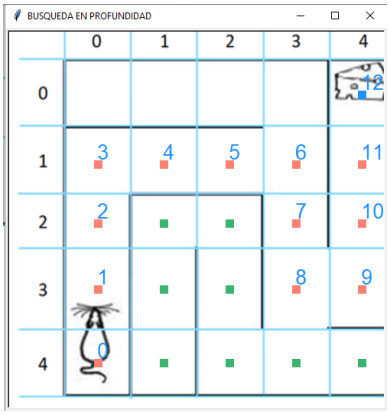
Ejecuciones

Laberinto pequeño

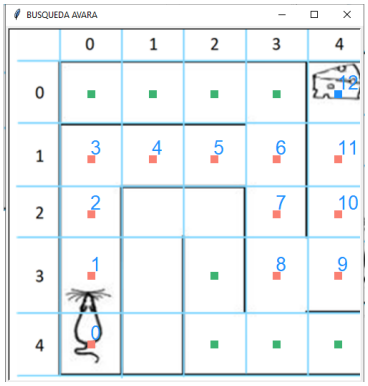
Búsqueda en amplitud



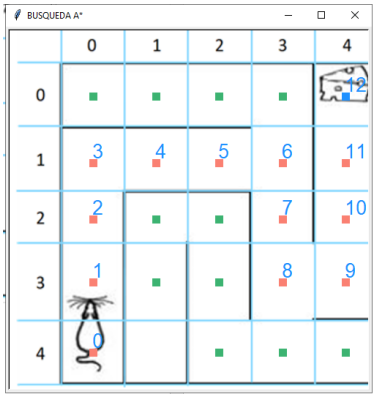
Búsqueda en profundidad



Búsqueda avara



Búsqueda A*

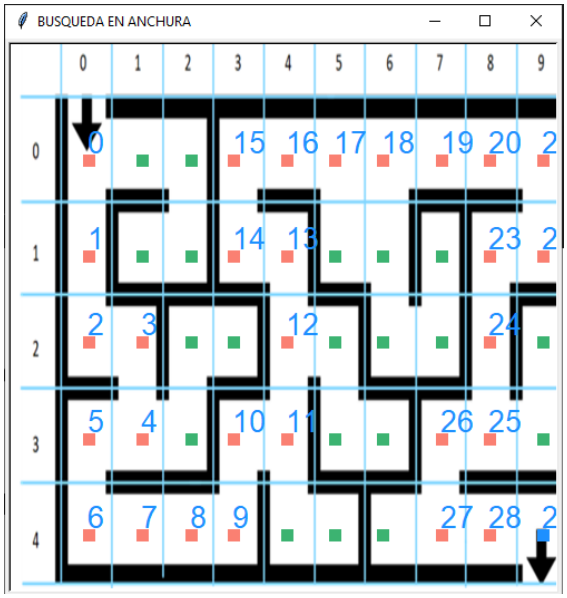


Consola

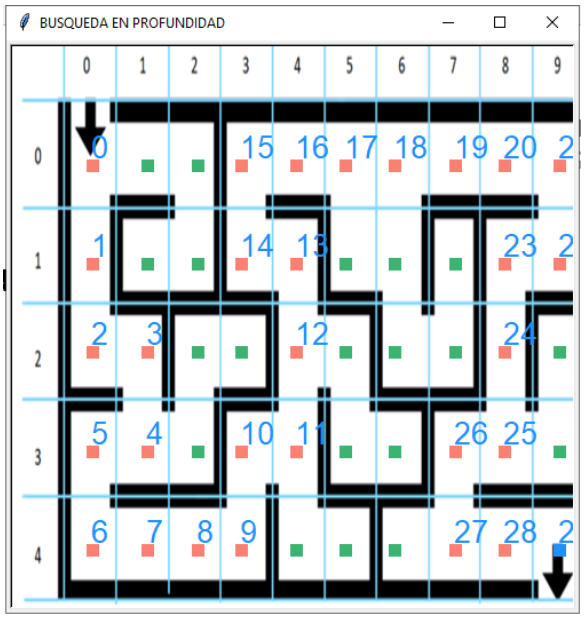
```
IAcode990DCPZX2K2 MINGW64 ~/Desktop/DOCUMENTOS UDLA/OCTAVO SEMESTRE/Inteligencia Artificial/P2/AP2S7_Búsquedas_ciegas_heuristicas (master)
$ python main_bc_h.py 0 0,4

BUSQUEDA EN ANCHURA
Parents: {'0,4': None, '0,3': '0,4', '0,2': '0,3', '0,1': '0,2', '1,1': '0,1', '2,1': '1,1', '3,1': '2,1', '3,0': '3,1', '3,2': '3,1', '2,0': '3,0', '3,3': '3,2', '1,0': '2,0', '4,3': '3,3', '3,4': '3,3', '0,0': '1,0', '4,2': '4,3', '4,4': '3,4', '2,4': '3,4', '4,1': '4,2', '2,3': '2,4', '4,0': '4,1', '2,2': '2,3'}
Shortest Path: ['0,4', '0,3', '0,2', '0,1', '1,1', '2,1', '3,1', '3,2', '3,3', '4,3', '4,2', '4,1', '4,0']

BUSQUEDA EN PROFUNDIDAD
Parents: {'0,4': None, '0,3': '0,4', '0,2': '0,3', '0,1': '0,2', '1,1': '0,1', '2,1': '1,1', '3,1': '2,1', '3,0': '3,1', '3,2': '3,1', '3,3': '3,2', '4,3': '3,3', '3,4': '3,3', '4,4': '3,4', '2,4': '3,4', '2,3': '2,4', '2,2': '2,3', '1,2': '2,2', '1,3': '1,2', '1,4': '1,3', '4,2': '4,3', '4,1': '4,2', '4,0': '4,1'}
Shortest Path: ['0,4', '0,3', '0,2', '0,1', '1,1', '2,1', '3,1', '3,2', '3,3', '4,3', '4,2', '4,1', '4,0']

BUSQUEDA AVARA
Parents: {'0,4': None, '0,3': '0,4', '0,2': '0,3', '0,1': '0,2', '1,1': '0,1', '2,1': '1,1', '3,1': '2,1', '3,0': '3,1', '3,2': '3,1', '2,0': '3,0', '3,3': '3,2', '1,0': '2,0', '4,3': '3,3', '3,4': '3,3', '0,0': '1,0', '4,2': '4,3', '4,4': '3,4', '2,4': '3,4', '4,1': '4,2', '2,3': '2,4', '4,0': '4,1', '2,2': '2,3'}
Shortest Path: ['0,4', '0,3', '0,2', '0,1', '1,1', '2,1', '3,1', '3,2', '3,3', '4,3', '4,2', '4,1', '4,0']

BUSQUEDA A*
Parents: {'0,4': None, '0,3': '0,4', '0,2': '0,3', '0,1': '0,2', '1,1': '0,1', '2,1': '1,1', '3,1': '2,1', '3,0': '3,1', '3,2': '3,1', '3,3': '3,2', '2,0': '3,0', '4,3': '3,3', '3,4': '3,3', '1,0': '2,0', '4,4': '3,4', '2,4': '3,4', '2,3': '2,4', '2,2': '2,3', '0,0': '1,0', '4,2': '4,3', '4,1': '4,2', '4,0': '4,1', '1,2': '2,2', '1,3': '1,2', '1,4': '1,3'}
Shortest Path: ['0,4', '0,3', '0,2', '0,1', '1,1', '2,1', '3,1', '3,2', '3,3', '4,3', '4,2', '4,1', '4,0']
Program finished.
```

Laberinto grande

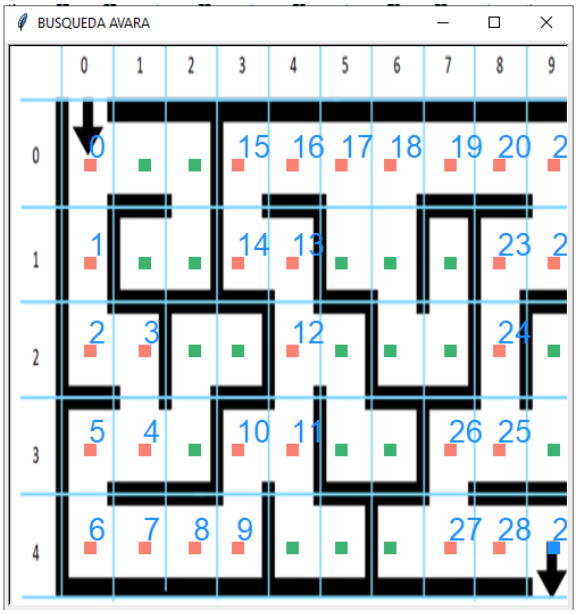
Búsqueda en amplitud



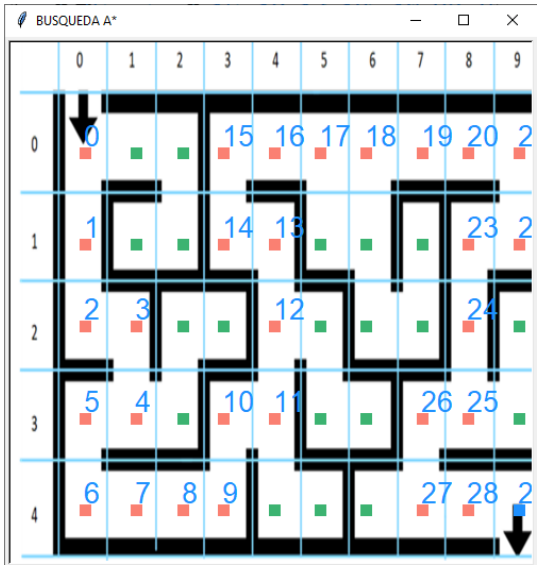
Búsqueda en profundidad



Búsqueda avara



Búsqueda A*

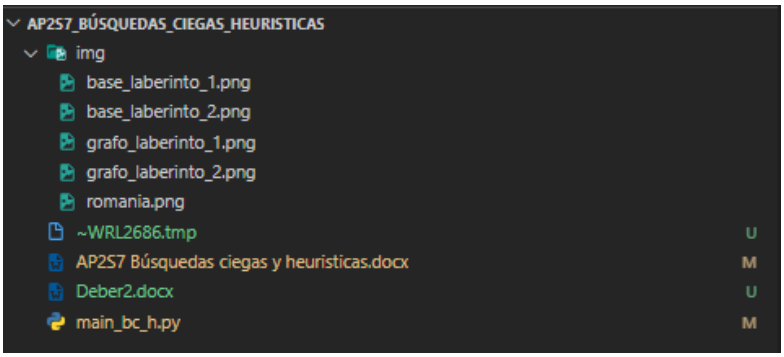


Consola

```
IAcode99@DCPZX2K2 MINGW64 ~/Desktop/DOCUMENTOS UDLA/OCTAVO SEMESTRE/Inteligencia Artificial/P2/AP2S7_Búsquedas_ciegas_heuristicas (master)
$ python main_bc_h.py 1 0,0

BUSQUEDA EN ANCHURA
Parents: {'0,0': None, '0,1': '0,0', '1,0': '0,0', '0,2': '0,1', '2,0': '1,0', '1,2': '0,2', '2,1': '2,0', '1,3': '1,2', '1,1': '2,1', '0,3': '1,3', '2,3': '1,3', '0,4': '0,3', '2,2': '2,3', '1,4': '0,4', '3,2': '2,2', '2,4': '1,4', '3,4': '2,4', '3,3': '3,4', '4,3': '3,3', '4,2': '4,3', '4,4': '4,3', '4,1': '4,2', '5,2': '4,2', '5,4': '4,4', '3,1': '4,1', '5,3': '5,2', '3,0': '3,1', '6,3': '5,3', '4,0': '3,0', '5,0': '4,0', '6,0': '5,0', '5,1': '5,0', '7,0': '6,0', '6,1': '6,0', '8,0': '7,0', '6,2': '6,1', '9,0': '8,0', '7,2': '6,2', '9,1': '9,0', '7,1': '7,2', '8,1': '9,1', '8,2': '8,1', '8,3': '8,2', '7,3': '8,3', '9,3': '8,3', '7,4': '7,3', '9,2': '9,3', '8,4': '7,4', '6,4': '7,4', '9,4': '8,4'}
Shortest Path: ['0,0', '0,1', '0,2', '1,2', '1,3', '0,3', '0,4', '1,4', '2,4', '3,4', '3,3', '4,3', '4,2', '4,1', '3,1', '3,0', '4,0', '5,0', '6,0', '7,0', '8,0', '9,0', '9,1', '8,1', '8,2', '8,3', '7,3', '7,4', '8,4', '9,4']

BUSQUEDA EN PROFUNDIDAD
Parents: {'0,0': None, '0,1': '0,0', '1,0': '0,0', '2,0': '1,0', '2,1': '2,0', '1,1': '2,1', '0,2': '0,1', '1,2': '0,2', '1,3': '1,2', '0,3': '1,3', '2,3': '1,3', '2,2': '2,3', '3,2': '2,2', '0,4': '0,3', '1,4': '0,4', '2,4': '1,4', '3,4': '2,4', '3,3': '3,4', '4,3': '3,3', '4,2': '4,3', '4,4': '4,3', '5,4': '4,4', '4,1': '4,2', '5,2': '4,2', '5,3': '5,2', '6,3': '5,3', '3,1': '4,1', '3,0': '3,1', '4,0': '3,0', '5,0': '4,0', '6,0': '5,0', '5,1': '5,0', '6,1': '5,1', '6,2': '6,1', '7,2': '6,2', '7,1': '7,2', '7,0': '6,0', '8,0': '7,0', '9,0': '8,0', '9,1': '9,0', '8,1': '9,1', '8,2': '8,1', '8,3': '8,2', '7,3': '8,3', '9,3': '8,3', '9,2': '9,3', '7,4': '7,3', '8,4': '7,4', '6,4': '7,4', '9,4': '8,4'}
Shortest Path: ['0,0', '0,1', '0,2', '1,2', '1,3', '0,3', '0,4', '1,4', '2,4', '3,4', '3,3', '4,3', '4,2', '4,1', '3,1', '3,0', '4,0', '5,0', '6,0', '7,0', '8,0', '9,0', '9,1', '8,1', '8,2', '8,3', '7,3', '7,4', '8,4', '9,4']

BUSQUEDA AVARA
Parents: {'0,0': None, '0,1': '0,0', '1,0': '0,0', '0,2': '0,1', '2,0': '1,0', '1,2': '0,2', '2,1': '2,0', '1,3': '1,2', '1,1': '2,1', '0,3': '1,3', '2,3': '1,3', '0,4': '0,3', '2,2': '2,3', '1,4': '0,4', '3,2': '2,2', '2,4': '1,4', '3,4': '2,4', '3,3': '3,4', '4,3': '3,3', '4,2': '4,3', '4,4': '4,3', '4,1': '4,2', '5,2': '4,2', '5,4': '4,4', '3,1': '4,1', '5,3': '5,2', '3,0': '3,1', '6,3': '5,3', '4,0': '3,0', '5,0': '4,0', '6,0': '5,0', '5,1': '5,0', '7,0': '6,0', '6,1': '6,0', '8,0': '7,0', '6,2': '6,1', '9,0': '8,0', '7,2': '6,2', '9,1': '9,0', '7,1': '7,2', '8,1': '9,1', '8,2': '8,1', '8,3': '8,2', '7,3': '8,3', '9,3': '8,3', '7,4': '7,3', '9,2': '9,3', '8,4': '7,4', '6,4': '7,4', '9,4': '8,4'}
Shortest Path: ['0,0', '0,1', '0,2', '1,2', '1,3', '0,3', '0,4', '1,4', '2,4', '3,4', '3,3', '4,3', '4,2', '4,1', '3,1', '3,0', '4,0', '5,0', '6,0', '7,0', '8,0', '9,0', '9,1', '8,1', '8,2', '8,3', '7,3', '7,4', '8,4', '9,4']

BUSQUEDA A*
Parents: {'0,0': None, '0,1': '0,0', '1,0': '0,0', '0,2': '0,1', '1,2': '0,2', '1,3': '1,2', '0,3': '1,3', '2,3': '1,3', '0,4': '0,3', '2,0': '1,0', '2,2': '2,3', '2,1': '2,0', '1,1': '2,1', '3,2': '2,2', '1,4': '0,4', '2,4': '1,4', '3,4': '2,4', '3,3': '3,4', '4,3': '3,3', '4,2': '4,3', '4,4': '4,3', '5,4': '4,4', '4,1': '4,2', '5,2': '4,2', '5,3': '5,2', '3,1': '4,1', '3,0': '3,1', '6,3': '5,3', '4,0': '3,0', '5,0': '4,0', '6,0': '5,0', '5,1': '5,0', '7,0': '6,0', '6,1': '6,0', '8,0': '7,0', '9,0': '8,0', '6,2': '6,1', '7,2': '6,2', '9,1': '9,0', '8,1': '9,1', '8,2': '8,1', '7,1': '7,2', '8,3': '8,2', '7,3': '8,3', '9,3': '8,3', '7,4': '7,3', '8,4': '7,4', '6,4': '7,4', '9,2': '9,3', '9,4': '8,4'}
Shortest Path: ['0,0', '0,1', '0,2', '1,2', '1,3', '0,3', '0,4', '1,4', '2,4', '3,4', '3,3', '4,3', '4,2', '4,1', '3,1', '3,0', '4,0', '5,0', '6,0', '7,0', '8,0', '9,0', '9,1', '8,1', '8,2', '8,3', '7,3', '7,4', '8,4', '9,4']
Program finished.
```

OBSERVACIONES

En la medida de lo posible mantener esta estructura de carpetas y archivos al ejecutar, si decide no apegarse a esta estructura, asegúrese de que dentro del archivo "main_bc_h.py" usted invoque las rutas y/o nombres de archivos correctamente.



Las instrucciones para ejecutar las encontrará en el "doc string" del archivo .py, o aparecerán en caso de que ingrese mal un comando.

```
$ python main_bc_h.py

    Usage:
      python main_bc_h.py graph startNode
      Target for graph 0 will always be (4,0)
      Target for graph 1 will always be (9,4)
      <0> <startNode>:  Any coord from (0,0) to (4,0)
      <1> <startNode>:  Any coord from (0,0) to (9,4)

    Example:
      python main_bc_h.py 0 0,4
      python main_bc_h.py 1 0,0
```