

Actividad : Resolución de problema mediante búsqueda heurística

Importante

El código siguiente es el que debe usarse para la ejecución de la actividad. En caso de requerir modificaciones se subirán ficheros de sustitución al aula de la asignatura

```
!pip install simpleai flask pydot graphviz
!pip install tabulate

Collecting simpleai
  Downloading simpleai-0.8.3.tar.gz (94 kB)
    94.4/94.4 kB 1.9 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: flask in /usr/local/lib/python3.11/dist-packages (3.1.0)
Requirement already satisfied: pydot in /usr/local/lib/python3.11/dist-packages (3.0.4)
Requirement already satisfied: graphviz in /usr/local/lib/python3.11/dist-packages (0.20.3)
Requirement already satisfied: Werkzeug>=3.1 in /usr/local/lib/python3.11/dist-packages (from flask) (3.1.3)
Requirement already satisfied: Jinja2>=3.1.2 in /usr/local/lib/python3.11/dist-packages (from flask) (3.1.6)
Requirement already satisfied: itsdangerous>=2.2 in /usr/local/lib/python3.11/dist-packages (from flask) (2.2.0)
Requirement already satisfied: click>=8.1.3 in /usr/local/lib/python3.11/dist-packages (from flask) (8.1.8)
Requirement already satisfied: blinker>=1.9 in /usr/local/lib/python3.11/dist-packages (from flask) (1.9.0)
Requirement already satisfied: pyparsing>=3.0.9 in /usr/local/lib/python3.11/dist-packages (from pydot) (3.2.3)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.11/dist-packages (from Jinja2>=3.1.2->flask) (3.0.2)
Building wheels for collected packages: simpleai
  Building wheel for simpleai (setup.py) ... done
  Created wheel for simpleai: filename=simpleai-0.8.3-py3-none-any.whl size=100982 sha256=05189d9048a2f6c7352f7d0930d034d6ba434db5409df8afc99d27d638d4cbd
  Stored in directory: /root/.cache/pip/wheels/ec/02/a7/f0077617a5f73eb1c52e45f12a9da3f0bafff3902bcd91766f
Successfully built simpleai
Installing collected packages: simpleai
Successfully installed simpleai-0.8.3
Requirement already satisfied: tabulate in /usr/local/lib/python3.11/dist-packages (0.9.0)

#!/usr/bin/env python
# coding: utf-8

# 2024 Modified by: Alejandro Cervantes
# Remember installing pyplot and flask if you want to use WebViewer

# NOTA: WebViewer sólo funcionará si ejecutáis en modo local

from __future__ import print_function

import math
from simpleai.search.viewers import BaseViewer, ConsoleViewer, WebViewer
from simpleai.search import SearchProblem, astar, breadth_first, depth_first, uniform_cost

class GameWalkPuzzle(SearchProblem):

    def __init__(self, board, costs, heuristic_number):
        self.board = board
        self.goal = (0, 0)
        self.costs = costs
        self.heuristic_number = heuristic_number
        for y in range(len(self.board)):
            for x in range(len(self.board[y])):
                if self.board[y][x].lower() == "t":
                    self.initial = (x, y)
                elif self.board[y][x].lower() == "p":
                    self.goal = (x, y)

        super(GameWalkPuzzle, self).__init__(initial_state=self.initial)

    def actions(self, state):
        actions = []
        for action in list(self.costs.keys()):
            newx, newy = self.result(state, action)
            if self.board[newy][newx] != "#":
                actions.append(action)
        return actions

    def result(self, state, action):
        x, y = state

        if action.count("up"):
```

```

        y -= 1
    if action.count("down"):
        y += 1
    if action.count("left"):
        x -= 1
    if action.count("right"):
        x += 1

    new_state = (x, y)
    return new_state

def is_goal(self, state):
    return state == self.goal

def cost(self, state, action, state2):
    return self.costs[action]

# Esta función heurística es la distancia entre el estado actual
# el objetivo (único) identificado como self.goal
def heuristic1(self, state):
    x, y = state
    gx, gy = self.goal
    return abs(x - gx) + abs(y - gy)

def heuristic2(self, state):
    x, y = state
    gx, gy = self.goal
    return max(abs(x - gx),abs(y - gy))

def heuristic3(self, state):
    x, y = state
    gx, gy = self.goal
    return 2*(abs(x - gx) + abs(y - gy))

def heuristic(self,state):
    if self.heuristic_number == 1:
        return self.heuristic1(state)
    elif self.heuristic_number == 2:
        return self.heuristic2(state)
    elif self.heuristic_number == 3:
        return self.heuristic3(state)
    else:
        raise Exception("El número de la función heurística debe estar entre 1 y 3. Revise la inicialización del problema.")

def searchInfo (problem,result,use_viewer):
def getTotalCost (problem,result):
    originState = problem.initial_state
    totalCost = 0
    for action,endingState in result.path():
        if action is not None:
            totalCost += problem.cost(originState,action,endingState)
            originState = endingState
    return totalCost

res = "Total length of solution: {0}\n".format(len(result.path()))
res += "Total cost of solution: {0}\n".format(getTotalCost(problem,result))

if use_viewer:
    stats = [{ 'name': stat.replace('_', ' '), 'value': value}
              for stat, value in list(use_viewer.stats.items())]

    for s in stats:
        res+= ' {0}: {1}\n'.format(s['name'],s['value'])
    return res

def resultado_experimento(problem,MAP,result,used_viewer):

# Erika agregó para tratar el error: 'NoneType' object has no attribute 'path'
if result is None:
    print("No se ha encontrado solución") # Imprimir mensaje si no se encontró solución
    return # Salir de la función prematuramente
path = [x[1] for x in result.path()]

for y in range(len(MAP)):
    for x in range(len(MAP[y])):
        if (x, y) == problem.initial:
            print("I", end='')
        elif (x, y) == problem.goal:
            print("P", end='')
        elif (x, y) in path:
            print(".", end='')

```

```
        else:
            print(MAP[y][x], end='')
        print()

    info=searchInfo(problem,result,used_viewer)
    print(info)

def main(MAP_ASCII,COSTS,algorithms,heuristic_number=1):
    MAP = [list(x) for x in MAP_ASCII.split("\n") if x]

    for algorithm in algorithms:
        problem = GameWalkPuzzle(MAP,COSTS,heuristic_number)
        used_viewer=BaseViewer()
        # Probad también ConsoleViewer para depurar
        # No podréis usar WebViewer en Collab para ver los árboles

        # Mostramos tres experimentos
        print ("Experimento con algoritmo {}".format(algorithm))

        result = algorithm(problem, graph_search=True,viewer=used_viewer)
        resultado_experimento(problem,MAP,result,used_viewer)
```

```
#!/usr/bin/env python
# coding: utf-8

# 2024 Modified by: Alejandro Cervantes
# Configuración y llamada para el caso 1
# Se ejecutan los algoritmos de búsqueda en amplitud y búsqueda en profundidad

MAP_ASCII = """
#####
#   P #
# #### #
# T # #
# ##  #
#     #
#####
"""

COSTS = {
    "left": 1.0,
    "right": 1.0,
    "up": 1.0,
    "down": 1.0,
}

algorithms=(breadth_first,depth_first)
main (MAP_ASCII,COSTS,algorithms)
```

```
Experimento con algoritmo <function breadth_first at 0x7b35fc272480>:
#####
#...P #
#.### #
#..T # #
# ##  #
#     #
#####
Total length of solution: 9
Total cost of solution: 8.0
max fringe size: 6
visited nodes: 23
iterations: 23

Experimento con algoritmo <function depth_first at 0x7b35fc2734c0>:
#####
#   P.#
# ###.#
# T.#.#
# ##. .#
#   ..#
#####
Total length of solution: 11
Total cost of solution: 10.0
max fringe size: 4
visited nodes: 11
iterations: 11
```

```
#!/usr/bin/env python
# coding: utf-8
```

```
# 2024 Modified by: Alejandro Cervantes
# Configuración y llamada para el caso 2
# Se utiliza el mismo mapa pero se varían los costes

MAP_ASCII = """
#####
#   P #
#  ### #
#   T # #
#  ##  #
#      #
#####
"""

COSTS = {
    "left": 2.0,
    "right": 2.0,
    "up": 5.0,
    "down": 5.0,
}

algorithms=(breadth_first,uniform_cost,astar)
main (MAP_ASCII,COSTS,algorithms)
```



Experimento con algoritmo <function breadth_first at 0x7b35fc272480>:

```
#####
#...P #
#.### #
#..T # #
#  ##  #
#      #
#####
Total length of solution: 9
Total cost of solution: 22.0
max fringe size: 6
visited nodes: 23
iterations: 23
```

Experimento con algoritmo <function uniform_cost at 0x7b35fc2fc220>:

```
#####
#...P #
#.### #
#..T # #
#  ##  #
#      #
#####
Total length of solution: 9
Total cost of solution: 22.0
max fringe size: 6
visited nodes: 22
iterations: 22
```

Experimento con algoritmo <function astar at 0x7b35fc2fc360>:

```
#####
#...P #
#.### #
#..T # #
#  ##  #
#      #
#####
Total length of solution: 9
Total cost of solution: 22.0
max fringe size: 6
visited nodes: 20
iterations: 20
```

```
#!/usr/bin/env python
# coding: utf-8

# 2024 Modified by: Alejandro Cervantes
# Configuración y llamada para el caso 3
# Se utiliza el mismo mapa y se usan diferentes heurísticas

MAP_ASCII = """
#####
#   P #
#  ### #
#   T # #
#  ##  #
#      #
#####
"""

COSTS = {
```

```
"left": 2.0,
"right": 2.0,
"up": 1.0,
"down": 1.0,
}

algorithms=(astar,)
main (MAP_ASCII,COSTS,algorithms,1)
main (MAP_ASCII,COSTS,algorithms,2)
main (MAP_ASCII,COSTS,algorithms,3)
```

Experimento con algoritmo <function astar at 0x7b35fc2fc360>:

```
#####
#   P.#
#   ####.
#   T.#.#
#   ##...#
#   #
#####
Total length of solution: 9
Total cost of solution: 12.0
max fringe size: 5
visited nodes: 19
iterations: 19
```

Experimento con algoritmo <function astar at 0x7b35fc2fc360>:

```
#####
#   P.#
#   ####.
#   T.#.#
#   ##...#
#   #
#####
Total length of solution: 9
Total cost of solution: 12.0
max fringe size: 5
visited nodes: 22
iterations: 22
```

Experimento con algoritmo <function astar at 0x7b35fc2fc360>:

```
#####
#   P.#
#   ####.
#   T.#.#
#   ##...#
#   #
#####
Total length of solution: 9
Total cost of solution: 12.0
max fringe size: 4
visited nodes: 12
iterations: 12
```

✓ PROBANDO CASO 2: teoría de búsqueda con coste (ERIKA Y EZEQUIEL)

Aunque el software ya está dado (con cuadernos Jupyter y scripts que simulan el entorno), el objetivo no es programar desde cero, sino:

- Ejecutar los algoritmos en distintos mapas/casos.
- Analizar comparativamente sus comportamientos.
- Justificar con teoría conceptos como eficiencia, optimalidad, admisibilidad y completitud.
- Mostrar resultados en tablas y discutir por qué cada algoritmo funciona como lo hace.
- Prepara la discusión sobre **admisibilidad y eficiencia**.

Legenda:

- BFS (búsqueda amplitud): breadth_first
- DFS (búsqueda en profundidad): depth_first,
- UCS (Dijkstra): uniform_cost
- A*: astar

FUNCIÓN DE EVALUACIÓN IMPLEMENTADA

Hace la evaluación automáticamente en base a estos criterios:

| Concepto | ¿Cómo lo justificas en tu actividad? |
|-------------|--|
| Eficiencia | Comparamos nodos expandidos, tiempo de ejecución y tamaño máximo de la frontera para ver cuál algoritmo resuelve con menos recursos. |
| Optimalidad | Analizamos si el coste de la solución coincide con el mínimo encontrado por UCS (Dijkstra), garantizando el camino más barato. |

| Concepto | ¿Cómo lo justificas en tu actividad? | | | | | |
|---------------|--|--|--|--|--|--|
| Admisibilidad | Comparamos el coste de A* con el de UCS; al usar heurística Manhattan, verificamos que nunca sobreestima y produce solución óptima. | | | | | |
| Compleitud | Verificamos que cada algoritmo devuelve solución (o informa "No") en todos los mapas diseñados, asegurando que explora todo el espacio buscable. | | | | | |

| Algoritmo | Tipo | Estructura de datos | Criterio de expansión | Considera costes | Usa heurística | Optimalidad |
|--|---------------|---------------------|---|------------------|------------------|---|
| Búsqueda en amplitud (BFS) | No informada | Cola (FIFO) | Explora todos los nodos a la misma profundidad antes de bajar | No | No | Sólo óptima en grafos no ponderados (coste = 1 en cada arista) |
| Búsqueda en profundidad (DFS) | No informada | Pila (LIFO) | Profundiza en una rama hasta el final antes de retroceder | No | No | No garantiza óptimo, puede quedarse en ramas infinitas |
| Uniform-Cost Search (UCS) (también Dijkstra) | No informada* | Cola de prioridad | Extrae siempre el nodo de menor coste acumulado $g(n)$ | Sí | No | Óptimo para costes ≥ 0 (es el algoritmo de Dijkstra clásico) |
| A* | Informada | Cola de prioridad | Extrae el nodo minimizando $f(n) = g(n) + h(n)$ | Sí | Sí (h admisible) | Óptimo si la heurística $h(n)$ no sobreestima el coste real |

- **BFS** = *Breadth-First Search*, explora nivel a nivel.
- **UCS** corresponde al algoritmo de **Dijkstra** cuando todas las aristas tienen coste no negativo.
- En muchas bibliotecas (incluida SimpleAI), `uniform_cost` es precisamente la implementación de Dijkstra.
- El asterisco en “No informada*” para UCS indica que, aunque no use heurística, sí tiene en cuenta el coste real de cada acción.

Detalles:

En este caso se deben probar y comparar los **algoritmos de búsqueda amplitud (BFS), Dijkstra y A*** cuando hay costes distintos de 1.

Vea que los ficheros contienen costes modificados para este caso.

Para A*, observe que el código proporciona una función heurística basada en la distancia de Manhattan.

1. Muestre una tabla con los datos de todas las ejecuciones.
2. ¿Obtiene **UCS (Dijkstra)** el camino de coste óptimo?
3. ¿Obtiene A* el camino de coste óptimo?
4. ¿Cuál de los dos algoritmos (**UCS o A***) es más eficiente en el caso planteado?
5. ¿Se puede afirmar que las respuestas 2 y 3 siempre serán de este modo aunque se varíe el mapa?
6. ¿Se puede afirmar que las respuestas 2 y 3 siempre serán de este modo aunque se varíen los costes? Justifique su respuesta

```
from tabulate import tabulate

def evaluar_algoritmos(MAP_ASCII, COSTS, heuristic_number=1, escenario="No especificado"):
    # Prepara el tablero
    MAP = [list(line) for line in MAP_ASCII.strip().split("\n")]

    # Metadatos de los algoritmos (incluimos 'optimalidad' para la tabla)
    algoritmos = {
        "Búsqueda en amplitud (BFS)": {
            "func": breadth_first,
            "tipo": "No informada",
            "estructura": "Cola (FIFO)",
            "criterio": "Número de pasos",
            "costes": "No",
            "heurística": "No",
            "optimalidad": "Sólo si costes=1",
        },
        "Búsqueda en profundidad (DFS)": {
            "func": depth_first,
            "tipo": "No informada",
            "estructura": "Pila (LIFO)",
            "criterio": "Profundidad",
            "costes": "No",
            "heurística": "No",
            "optimalidad": "No",
        },
        "Uniform-Cost Search (UCS) (Dijkstra)": {
            "func": uniform_cost,
            "tipo": "Informada*",
            "estructura": "Cola de prioridad",
            "criterio": "Coste acumulado",
            "costes": "Sí",
            "heurística": "No",
            "optimalidad": "Sí",
        },
        "A*": {
            "func": astar,
            "tipo": "Informada",
            "estructura": "Cola de prioridad",
            "criterio": "g + h",
            "costes": "Sí",
            "heurística": "Sí",
            "optimalidad": "Sí (h admisible)",
        },
    }
```

```
resultados = {}

# Primero, obtener coste óptimo de referencia usando UCS
ucs_meta = algoritmos["Uniform-Cost Search (UCS) (Dijkstra)"]
prob_ucs = GameWalkPuzzle(MAP, COSTS, heuristic_number)
viewer_ucs = BaseViewer()
res_ucs = ucs_meta["func"](prob_ucs, graph_search=True, viewer=viewer_ucs)
if res_ucs:
    origin = prob_ucs.initial_state
    coste_opt = 0
    for action, state2 in res_ucs.path():
        if action:
            coste_opt += prob_ucs.cost(origin, action, state2)
            origin = state2
else:
    coste_opt = None

# Ejecutar cada algoritmo y recopilar datos
for nombre, meta in algoritmos.items():
    prob = GameWalkPuzzle(MAP, COSTS, heuristic_number)
    viewer = BaseViewer()
    outcome = meta["func"](prob, graph_search=True, viewer=viewer)

    completitud = "Sí" if outcome else "No"
    cost = "-"
    length = "-"
    exp = viewer.stats.get("visited_nodes", "-")
    maxf = viewer.stats.get("max_fringe_size", "-")
    admisible = "No"

    if outcome:
        # calcular coste y longitud
        origin = prob.initial_state
        c = 0
        for action, state2 in outcome.path():
            if action:
                c += prob.cost(origin, action, state2)
                origin = state2
        cost = c
        length = len(outcome.path())
        # admisibilidad
        if coste_opt is not None and abs(c - coste_opt) < 1e-6:
            admisible = "Sí"

    resultados[nombre] = {
        **meta,
        "escenario": escenario,
        "completitud": completitud,
        "longitud": length,
        "coste": cost,
        "nodos_exp": exp,
        "tamaño_frontera": maxf,
        "admisible": admisible
    }

# Montar la tabla con las columnas en el orden solicitado
tabla = []
for nom, d in resultados.items():
    tabla.append([
        d["escenario"],
        nom,
        d["criterio"],
        d["tipo"],
        d["estructura"],
        d["costes"],
        d["heurística"],
        d["longitud"],
        d["nodos_exp"],
        d["tamaño_frontera"],
        d["optimalidad"],
        d["admisible"],
        d["completitud"],
        d["coste"]
    ])

headers = [
    "Escenario", "Algoritmo", "Criterio de expansión", "Tipo", "Estructura de datos",
    "Considera costes", "Usa heurística", "Longitud", "Nodos expandidos", "Tamaño máx. (Eficiencia)",
    "Optimalidad", "Admisible", "Completitud", "Coste solución"
]
print(tabulate(tabla, headers, tablefmt="pipe"))
```

Executa el código y realiza la evaluación automaticamente de todos los escenarios para contestar las preguntas:

- ¿Se puede afirmar que las respuestas 2 y 3 siempre serán de este modo aunque se **varíe el mapa**?
- ¿Se puede afirmar que las respuestas 2 y 3 siempre serán de este modo aunque se **varíen los costes**?

Cada bloque:

- Define MAP_ASCII y COSTS para el escenario.
- Llama a main(...) para imprimir caminos y estadísticas.
- Llama a evaluar_algoritmos(...) para generar la tabla completa con todos los metadatos y resultados.

Así tendrás 7 celdas (1 original + 6 variaciones) perfectamente reutilizables.

```
# Escenario 1: Caso 2 - original
MAP_ASCII = """"\
#####
#   P   #
#   ### #
#   T   #
#  ##   #
#       #
#####
""""

COSTS = {"left":2.0,"right":2.0,"up":5.0,"down":5.0}
algorithms = (breadth_first, uniform_cost, astar)

print("### Resultados Caso 2 - original ###")
main(MAP_ASCII, COSTS, algorithms, heuristic_number=1)

print("### Tabla resumen Caso 2 - original ###")
evaluar_algoritmos(
    MAP_ASCII,
    COSTS,
    heuristic_number=1, # Si quieres probar otras heurísticas cambiar el numero para 1, 2 o 3
    escenario="Original: Costes H=2, V=5"
)
```

```
##> ### Resultados Caso 2 - original ###
Experimento con algoritmo <function breadth_first at 0x7b35fc272480>:
#####
#...P #
#.### #
#..T # #
#  ##  #
#      #
#####
Total length of solution: 9
Total cost of solution: 22.0
max fringe size: 6
visited nodes: 23
iterations: 23

Experimento con algoritmo <function uniform_cost at 0x7b35fc2fc220>:
#####
#...P #
#.### #
#..T # #
#  ##  #
#      #
#####
Total length of solution: 9
Total cost of solution: 22.0
max fringe size: 6
visited nodes: 22
iterations: 22

Experimento con algoritmo <function astar at 0x7b35fc2fc360>:
#####
#...P #
#.### #
#..T # #
#  ##  #
#      #
#####
Total length of solution: 9
Total cost of solution: 22.0
max fringe size: 6
visited nodes: 20
iterations: 20

### Tabla resumen Caso 2 - original ###
```


| Escenario | Algoritmo | Criterio de expansión | Tipo | Estructura de datos | Considera costes | Usa heurística | Longitud | Nodos expandidos | Tamaño máx. (Eficiencia) | Optimalidad | Admisible | Completitud | Coste solución |
|---------------------------|--------------------------------------|-----------------------|--------------|---------------------|------------------|----------------|----------|------------------|--------------------------|------------------|-----------|-------------|----------------|
| Original: Costes H=2, V=5 | Búsqueda en amplitud (BFS) | Número de pasos | No informada | Cola (FIFO) | No | No | 9 | 23 | 6 | Sólo si costes=1 | Sí | Sí | 22 |
| Original: Costes H=2, V=5 | Búsqueda en profundidad (DFS) | Profundidad | No informada | Pila (LIFO) | No | No | 11 | 11 | 4 | No | No | Sí | 38 |
| Original: Costes H=2, V=5 | Uniform-Cost Search (UCS) (Dijkstra) | Coste acumulado | Informada* | Cola de prioridad | Sí | No | 9 | 22 | 6 | Sí | Sí | Sí | 22 |
| Original: Costes H=2, V=5 | A* | g + h | Informada | Cola de prioridad | Sí | Sí | 9 | 20 | 6 | Sí (h admisible) | Sí | Sí | 22 |

```
# Escenario 2: Mapa estrecho (estructura lineal)
MAP_ASCII = """"\
#####
#T  #
#   #
#   #
#   P#
#####
""""

print("### Resultados - Mapa estrecho ###")
main(MAP_ASCII, COSTS, algorithms, heuristic_number=1)
print("### Tabla resumen - Mapa estrecho ###")
evaluar_algoritmos(MAP_ASCII, COSTS, heuristic_number=1, escenario="Mapa con estructura lineal")
```



```
### Resultados - Mapa estrecho ###
Experimento con algoritmo <function breadth_first at 0x7b35fc272480>:
#####
#T...#
#   .#
#   .#
#   P#
#####
Total length of solution: 7
Total cost of solution: 18.0
max fringe size: 2
visited nodes: 12
iterations: 12

Experimento con algoritmo <function uniform_cost at 0x7b35fc2fc220>:
#####
#T...#
#   .#
#   P#
#####
Total length of solution: 7
Total cost of solution: 18.0
max fringe size: 2
visited nodes: 12
iterations: 12

Experimento con algoritmo <function astar at 0x7b35fc2fc360>:
#####
#T...#
#   .#
#   P#
#####
Total length of solution: 7
Total cost of solution: 18.0
max fringe size: 2
visited nodes: 12
iterations: 12

### Tabla resumen - Mapa estrecho ###
| Escenario | Algoritmo | Criterio de expansión | Tipo | Estructura de datos | Considera costes | Usa heurística | Longitud | Nodos expandidos | Tamaño máx. (Eficiencia) | Optimalidad | Admisible | Completitud | Coste solución |
|:-----|:-----|:-----|:-----|:-----|:-----|:-----|:-----|:-----|:-----|:-----|:-----|:-----|:-----|
| Mapa con estructura lineal | Búsqueda en amplitud (BFS) | Número de pasos | No informada | Cola (FIFO) | No | No | 7 | 12 | 2 | Sólo si costes=1 | Sí | Sí | 18 |
| Mapa con estructura lineal | Búsqueda en profundidad (DFS) | Profundidad | No informada | Pila (LIFO) | No | No | 7 | 7 | 2 | No | No | Sí | Sí | 18 |
| Mapa con estructura lineal | Uniform-Cost Search (UCS) (Dijkstra) | Coste acumulado | Informada* | Cola de prioridad | Sí | No | 7 | 12 | 2 | Sí | Sí | Sí | 18 |
| Mapa con estructura lineal | A* | g + h | Informada | Cola de prioridad | Sí | Sí | 7 | 12 | 2 | Sí (h admisible) | Sí | Sí | 18 |
```

```
# Escenario 3: Mapa con muchas bifurcaciones
MAP_ASCII = """"\
#####
#T  # #
#   # # #
#   # # #
#   # # P#
#####
""""

print("### Resultados - Mapa con bifurcaciones ###")
main(MAP_ASCII, COSTS, algorithms, heuristic_number=1)
print("### Tabla resumen - Mapa con bifurcaciones ###")
evaluar_algoritmos(MAP_ASCII, COSTS, heuristic_number=1, escenario="Mapa con bifurcaciones")
```




```
### Resultados - Mapa con bifurcaciones ###
Experimento con algoritmo <function breadth_first at 0x7b35fc272480>:
No se ha encontrado solución
Experimento con algoritmo <function uniform_cost at 0x7b35fc2fc220>:
No se ha encontrado solución
```

```
Experimento con algoritmo <function astar at 0x7b35fc2fc360>:
No se ha encontrado solución
### Tabla resumen – Mapa con bifurcaciones ###
| Escenario | Algoritmo | Criterio de expansión | Tipo | Estructura de datos | Considera costes | Usa heurística | Longitud | Nodos expandidos | Tamaño máx. (Eficiencia) | Optimalidad | Admisible | Completitud | Coste solución |
|:-----|:-----|:-----|:-----|:-----|:-----|:-----|:-----|:-----|:-----|:-----|:-----|:-----|:-----|
| Mapa con bifurcaciones | Búsqueda en amplitud (BFS) | Número de pasos | No informada | Cola (FIFO) | No | No | - | 12 | 4 | Sólo si costes=1 | No | No | - |
| Mapa con bifurcaciones | Búsqueda en profundidad (DFS) | Profundidad | No informada | Pila (LIFO) | No | No | - | 12 | 3 | No | No | No | - |
| Mapa con bifurcaciones | Uniform-Cost Search (UCS) (Dijkstra) | Coste acumulado | Informada* | Cola de prioridad | Sí | No | - | 12 | 3 | Sí | No | No | - |
| Mapa con bifurcaciones | A* | g + h | Informada | Cola de prioridad | Sí | Sí | - | 12 | 3 | Sí (h admisible) | No | No | - |

# Escenario 4: Mapa completamente abierto
MAP_ASCII = """\
#####
#T      P#
#####
""

print("### Resultados – Mapa abierto ###")
main(MAP_ASCII, COSTS, algorithms, heuristic_number=1)
print("### Tabla resumen – Mapa abierto ###")
evaluar_algoritmos(MAP_ASCII, COSTS, heuristic_number=1, escenario="Mapa completamente abierto")
```

 ### Resultados – Mapa abierto ###

```
Experimento con algoritmo <function breadth_first at 0x7b35fc272480>:
#####
#T.....P#
#####
Total length of solution: 11
Total cost of solution: 20.0
max fringe size: 1
visited nodes: 11
iterations: 11


Experimento con algoritmo <function uniform_cost at 0x7b35fc2fc220>:
#####
#T.....P#
#####
Total length of solution: 11
Total cost of solution: 20.0
max fringe size: 1
visited nodes: 11
iterations: 11

Experimento con algoritmo <function astar at 0x7b35fc2fc360>:
#####
#T.....P#
#####
Total length of solution: 11
Total cost of solution: 20.0
max fringe size: 1
visited nodes: 11
iterations: 11

### Tabla resumen – Mapa abierto ###
| Escenario | Algoritmo | Criterio de expansión | Tipo | Estructura de datos | Considera costes | Usa heurística | Longitud | Nodos expandidos | Tamaño máx. (Eficiencia) | Optimalidad | Admisible | Completitud | Coste solución |
|:-----|:-----|:-----|:-----|:-----|:-----|:-----|:-----|:-----|:-----|:-----|:-----|:-----|:-----|
| Mapa completamente abierto | Búsqueda en amplitud (BFS) | Número de pasos | No informada | Cola (FIFO) | No | No | 11 | 11 | 1 | Sólo si costes=1 | Sí | Sí | 20 |
| Mapa completamente abierto | Búsqueda en profundidad (DFS) | Profundidad | No informada | Pila (LIFO) | No | No | 11 | 11 | 1 | No | Sí | Sí | 20 |
| Mapa completamente abierto | Uniform-Cost Search (UCS) (Dijkstra) | Coste acumulado | Informada* | Cola de prioridad | Sí | No | 11 | 11 | 1 | Sí | Sí | Sí | 20 |
| Mapa completamente abierto | A* | g + h | Informada | Cola de prioridad | Sí | Sí | 11 | 11 | 1 | Sí (h admisible) | Sí | Sí | 20 |
```

```
# Escenario 5: Costes muy dispares (horizontal barato, vertical caro)
MAP_ASCII = """\
#####
#   P   #
# #### #
# T # #
# ##  #
#     #
#####
""

COSTS_DISP = {"left":1.0,"right":1.0,"up":100.0,"down":100.0}
print("### Resultados – Costes muy dispares ###")
main(MAP_ASCII, COSTS_DISP, algorithms, heuristic_number=1)
print("### Tabla resumen – Costes muy dispares ###")
evaluar_algoritmos(MAP_ASCII, COSTS_DISP, heuristic_number=1, escenario="Costes muy dispares")
```

 ### Resultados – Costes muy dispares ###

```
Experimento con algoritmo <function breadth_first at 0x7b35fc272480>:
#####
#...P #
#.### #
#..T # #
# ##  #
#     #
```

```
#####
Total length of solution: 9
Total cost of solution: 206.0
max fringe size: 6
visited nodes: 23
iterations: 23
```

```
Experimento con algoritmo <function uniform_cost at 0x7b35fc2fc220>:
#####
#...P #
#.### #
#..T # #
#  # #
#    #
#####
Total length of solution: 9
Total cost of solution: 206.0
max fringe size: 6
visited nodes: 21
iterations: 21
```

```
Experimento con algoritmo <function astar at 0x7b35fc2fc360>:
#####
#...P #
#.### #
#..T # #
#  # #
#    #
#####
Total length of solution: 9
Total cost of solution: 206.0
max fringe size: 6
visited nodes: 17
iterations: 17
```

| ### Tabla resumen - Costes muy dispares ### | | | | | | | | | | | | | |
|---|--------------------------------------|-----------------------|--------------|---------------------|------------------|----------------|----------|------------------|--------------------------|------------------|-----------|------------|----------------|
| Escenario | Algoritmo | Criterio de expansión | Tipo | Estructura de datos | Considera costes | Usa heurística | Longitud | Nodos expandidos | Tamaño máx. (Eficiencia) | Optimalidad | Admisible | Compleitud | Coste solución |
| Costes muy dispares | Búsqueda en amplitud (BFS) | Número de pasos | No informada | Cola (FIFO) | No | No | 9 | 23 | 6 | Sólo si costes=1 | Sí | Sí | 206 |
| Costes muy dispares | Búsqueda en profundidad (DFS) | Profundidad | No informada | Pila (LIFO) | No | No | 11 | 11 | 4 | No | No | Sí | 604 |
| Costes muy dispares | Uniform-Cost Search (UCS) (Dijkstra) | Coste acumulado | Informada* | Cola de prioridad | Sí | No | 9 | 21 | 6 | Sí | Sí | Sí | 206 |
| Costes muy dispares | A* | g + h | Informada | Cola de prioridad | Sí | Sí | 9 | 17 | 6 | Sí (h admisible) | Sí | Sí | 206 |

```
# Escenario 6: Costes no simétricos (aleatorios)
COSTS RAND = {"left":3.0,"right":1.0,"up":7.0,"down":2.0}
print("### Resultados - Costes aleatorios ###")
main(MAP_ASCII, COSTS RAND, algorithms, heuristic_number=1)
print("### Tabla resumen - Costes aleatorios ###")
evaluar_algoritmos(MAP_ASCII, COSTS RAND, heuristic_number=1, escenario="Costes no simétricos")
```



```
### Resultados - Costes aleatorios ###
Experimento con algoritmo <function breadth_first at 0x7b35fc272480>:
#####
#...P #
#.### #
#..T # #
#  # #
#    #
#####
Total length of solution: 9
Total cost of solution: 24.0
max fringe size: 6
visited nodes: 23
iterations: 23

Experimento con algoritmo <function uniform_cost at 0x7b35fc2fc220>:
#####
#...P #
#.### #
#..T # #
#  # #
#    #
#####
Total length of solution: 9
Total cost of solution: 24.0
max fringe size: 5
visited nodes: 22
iterations: 22

Experimento con algoritmo <function astar at 0x7b35fc2fc360>:
#####
#...P #
#.### #
#..T # #
#  # #
#    #
#####
Total length of solution: 9
```

Total cost of solution: 24.0
max fringe size: 5
visited nodes: 22
iterations: 22

| ### Tabla resumen - Costes aleatorios ### | | | | | | | | | | | | | | |
|---|--------------------------------------|-----------------------|--------------|---------------------|------------------|----------------|----------|------------------|--------------------------|------------------|-----------|-------------|----------------|--|
| Escenario | Algoritmo | Criterio de expansión | Tipo | Estructura de datos | Considera costes | Usa heurística | Longitud | Nodos expandidos | Tamaño máx. (Eficiencia) | Optimalidad | Admisible | Completitud | Coste solución | |
| Costes no simétricos | Búsqueda en amplitud (BFS) | Número de pasos | No informada | Cola (FIFO) | No | No | 9 | 23 | 6 | Sólo si costes=1 | Sí | Sí | 24 | |
| Costes no simétricos | Búsqueda en profundidad (DFS) | Profundidad | No informada | Pila (LIFO) | No | No | 11 | 11 | 4 | No | No | Sí | 38 | |
| Costes no simétricos | Uniform-Cost Search (UCS) (Dijkstra) | Coste acumulado | Informada* | Cola de prioridad | Sí | No | 9 | 22 | 5 | Sí | Sí | Sí | 24 | |
| Costes no simétricos | A* | g + h | Informada | Cola de prioridad | Sí | Sí | 9 | 22 | 5 | Sí (h admisible) | Sí | Sí | 24 | |

```
# Escenario 7: Costes invertidos (vertical barato, horizontal caro)
COSTS_INV = {"left":10.0,"right":10.0,"up":1.0,"down":1.0}
print("### Resultados - Costes invertidos ###")
main(MAP_ASCII, COSTS_INV, algorithms, heuristic_number=1)
print("### Tabla resumen - Costes invertidos ###")
evaluar_algoritmos(MAP_ASCII, COSTS_INV, heuristic_number=1, escenario="Costes invertidos")
```

```
### Resultados - Costes invertidos ###
Experimento con algoritmo <function breadth_first at 0x7b35fc272480>:
#####
#...P #
#.### #
#..T # #
# # #
# #
#####
Total length of solution: 9
Total cost of solution: 62.0
max fringe size: 6
visited nodes: 23
iterations: 23

Experimento con algoritmo <function uniform_cost at 0x7b35fc2fc220>:
#####
# P.#
# ###.#
# T.#.#
# ##...#
# #
#####
Total length of solution: 9
Total cost of solution: 44.0
max fringe size: 5
visited nodes: 22
iterations: 22

Experimento con algoritmo <function astar at 0x7b35fc2fc360>:
#####
# P.#
# ###.#
# T.#.#
# ##...#
# #
#####
Total length of solution: 9
Total cost of solution: 44.0
max fringe size: 5
visited nodes: 21
iterations: 21
```

| ### Tabla resumen - Costes invertidos ### | | | | | | | | | | | | | | |
|---|--------------------------------------|-----------------------|--------------|---------------------|------------------|----------------|----------|------------------|--------------------------|------------------|-----------|-------------|----------------|--|
| Escenario | Algoritmo | Criterio de expansión | Tipo | Estructura de datos | Considera costes | Usa heurística | Longitud | Nodos expandidos | Tamaño máx. (Eficiencia) | Optimalidad | Admisible | Completitud | Coste solución | |
| Costes invertidos | Búsqueda en amplitud (BFS) | Número de pasos | No informada | Cola (FIFO) | No | No | 9 | 23 | 6 | Sólo si costes=1 | No | Sí | 62 | |
| Costes invertidos | Búsqueda en profundidad (DFS) | Profundidad | No informada | Pila (LIFO) | No | No | 11 | 11 | 4 | No | No | Sí | 46 | |
| Costes invertidos | Uniform-Cost Search (UCS) (Dijkstra) | Coste acumulado | Informada* | Cola de prioridad | Sí | No | 9 | 22 | 5 | Sí | Sí | Sí | 44 | |
| Costes invertidos | A* | g + h | Informada | Cola de prioridad | Sí | Sí | 9 | 21 | 5 | Sí (h admisible) | Sí | Sí | 44 | |