Trabajo Práctico 3: Búsqueda no informada

FrozenLake[1] es un entorno de cuadrícula de la biblioteca `gymnasium` de `python`, donde un agente debe trasladarse desde un punto de inicio hasta una meta por un lago congelado, evitando caer en los agujeros del hielo, a lo largo de una «vida» de, por defecto, 100 acciones como máximo.

La dimensión de la cuadrícula se conoce de antemano, pero la distribución de los agujeros no se conocen (son aleatorios). Las casillas permanecen sólidas una vez cruzadas, y el objetivo del agente es alcanzar la meta sin caer en un agujero.

Las acciones permitidas son:

- 0: mueve a la izquierda.
- 1: mueve hacia abajo.
- 2: mueve a la derecha.
- 3: mueve hacia arriba.

El agente percibe su ubicación y si esa casilla contiene un agujero o es la meta. A continuación se observa un ejemplo del entorno para una cuadrícula de 4×4.

Exploración del entorno

1. Instalar la biblioteca y crear el entorno por defecto:

```
import gymnasium as gym

env = gym.make('FrozenLake-v1', render_mode='human')
```

[1] `https://gymnasium.farama.org/environments/toy_text/frozen_lake/`

2. Obtener información del entorno:

```
print("Número de estados:", env.observation_space.n)
print("Número de acciones:", env.action_space.n)
```

3. Ejecutar un episodio básico:

```
state = env.reset()
print("Posición inicial del agente:", state[0])

done = truncated = False
while not (done or truncated):
    action = env.action_space.sample()  # Acción aleatoria
    next_state, reward, done, truncated, _ = env.step(action)
    print(f"Acción: {action}, Nuevo estado: {next_state}, Recompensa: {reward}")
    print(f"¿Ganó? (encontró el objetivo): {done}")
    printf(f"¿Frenó? (alcanzó el máximo de pasos posible): {truncated}\n")
    state = next_state
```

4. Modificar el entorno:

   a) La función `make` tiene como argumento `is_slippery`. ¿Qué controla dicho argumento? ¿Cuál es su valor por defecto?

   b) FrozenLake ya tiene algunos mapas precargados de distintos tamaños. Los cuales pueden ser especificados mediante la opción `map_name` a la hora de crear el entorno:

   ```
   gym.make('FrozenLake-v1', desc=None, map_name="4x4", render_mode='human')
   ```

   Para crear mapas aleatorios con distinto porcentajes de agujeros, puede especificarse un mapa personalizado a través del parámetro `desc`.

   - Definiendo una lista:

     ```
     desc=["SFFF", "FHFH", "FFFH", "HFFG"]
     gym.make('FrozenLake-v1', desc=desc, render_mode='human')
     ```

   - A partir de la función `generate_random_map`:

     ```
     from gymnasium.envs.toy_text.frozen_lake import generate_random_map

     gym.make('FrozenLake-v1', desc=generate_random_map(size=8), render_mode='human')
     ```

   ¿Cómo son los entornos anteriores? Describa el tamaño, cantidad de agujeros, posición inicial del agente y del objetivo.

   c) Arme una nueva función `generate_random_map_custom` que permita definir el tamaño de la grilla, la probabilidad que una casilla sea de hielo, y ubique de forma aleatoria la posición inicial del agente y del objetivo (el entorno creado a partir de dicha función podría no tener solución).

5. Modificar la «vida» del agente:

   a) Para modificar la cantidad de pasos máxima que puede dar un agente se utiliza un *wrapper*:

   ```
   import gymnasium as gym
   from gym import wrappers
   ```

```
nuevo_limite = 10
env = gym.make('FrozenLake-v1', render_mode='human').env
env = wrappers.TimeLimit(env, nuevo_limite)
```

Búsqueda no informada

1. Crear un entorno determinista, aleatorio, de 100×100, cuyas celdas representen un obstáculo para el agente (agujeros en el hielo) con probabilidad 0,08. Modificar la vida del agente a 1000 acciones.

2. Implementar un agente basado en objetivos que dado un punto de inicio y un punto destino, encuentre el camino óptimo (si es posible). Considere los siguientes escenarios posibles:

   - Escenario 1: Cada acción tiene costo 1.
   - Escenario 2: Las acciones tienen como costo asociado un entero más que el que representa a la acción, es decir, moverse a la izquierda tiene costo 1, moverse hacia abajo tiene costo 2, etc.

   a) El agente deberá ser capaz de resolver el problema planteado mediante los siguientes algoritmos de búsqueda no informada:
      i) Búsqueda por Anchura.
      ii) Búsqueda por Profundidad.
      iii) Búsqueda por Profundidad Limitada (límite = 10).
      iv) Búsqueda de Costo Uniforme.

   b) Al finalizar el proceso de formulación se deberán imprimir por pantalla:
      i) El entorno generado.
      ii) La secuencia de estados completa para llegar desde el estado inicial al estado objetivo (si es posible).

3. Ejecutar un total de 30 veces cada algoritmo en escenarios aleatorios con las características descriptas en el ejercicio 1. Evaluar cada uno de los algoritmos sobre los mismos entornos generados.

   a) Calcular la media y la desviación estándar de:
      - Cantidad de estados explorados para llegar al objetivo (si es que fue posible).
      - Costo total de las acciones tomadas para las soluciones obtenidas.
      - Tiempo empleado (segundos).

   b) Presentar los resultados en gráficos de cajas y bigotes (boxplots).

4. Repetir el procedimiento descrito en el ejercicio anterior, para el caso de un agente con comportamiento aleatorio. Esto es, el agente elige cada una de sus acciones al azar.

5. ¿Cuál de los algoritmos considera más adecuado para resolver el problema planteado? Justificar la respuesta.

6. Forma de entrega:

   a) Dentro del repositorio `ia-uncuyo-2024`, crear una nueva carpeta de nombre `tp3-busquedas-no-informadas`.

   b) Dentro de la carpeta `tp3-busquedas-no-informadas` crear una nueva carpeta `code` para el proyecto desarrollado en `python`.

   c) Dentro de la carpeta `tp3-busquedas-no-informadas` crear un archivo de nombre `no-informada-results.csv` en formato csv (*comma separated values*) con los resultados de las 30 ejecuciones de cada uno de los algoritmos evaluados. El formato deberá ser el siguiente:
      - `algorithm_name`: nombre del algoritmo (BFS, DFS, DLS, UCS_e1, UCS_e2) (cadena).

- `env_n`: número del entorno aleatorio (entero).
- `states_n`: cantidad de estados explorados (entero).
- `cost_e1`: costo total de las acciones en el escenario 1 (entero).
- `cost_e2`: costo total de las acciones en el escenario 2 (entero).
- `time`: tiempo empleado (float).
- `solution_found`: indicar si se encontró la solución (booleano).

d) Dentro de la carpeta `tp3-busquedas-no-informadas`, crear una nueva carpeta `images`, que incluya todos los gráficos e imágenes a utilizar en el reporte final.

Trabajo Práctico 4: Búsqueda informada

1. Dado el entorno FrozenLake y los ambientes creados en el Trabajo Práctico N° 3, implementar un agente que resuelva el problema planteado mediante un algoritmo de búsqueda A*.

   a) Proponer una heurística admisible y consistente para el problema.

   b) Al finalizar el proceso de formulación se deberán imprimir por pantalla:

      i) El entorno generado.

      ii) La secuencia de estados completa para llegar desde el estado inicial al estado destino (si es posible).

2. Ejecutar un total de 30 veces el algoritmo A* en los mismos escenarios aleatorios construidos en el Trabajo Práctico N°3.

   a) Calcular la media y la desviación estándar de:

      - Cantidad de estados explorados para llegar al objetivo (si es que fue posible).
      - Costo total de las acciones tomadas para las soluciones obtenidas.
      - Tiempo empleado (segundos).

   b) Presentar los resultados en un gráfico de cajas y bigotes o boxplots. Incluya también los resultados obtenidos en el Trabajo Práctico N°3.

3. Forma de entrega:

   a) Dentro del repositorio `ia-uncuyo-2024`, crear una carpeta con el nombre `tp4-busquedas-informadas`.

   b) Dentro de la carpeta `tp4-busquedas-informadas` crear una nueva carpeta `code` para el proyecto desarrollado en `python`.

   c) Dentro de la carpeta `tp4-busquedas-informadas` crear un archivo de nombre `informada-results.csv` en formato csv (*comma separated values*) con los resultados de las 30 ejecuciones. El formato deberá ser el siguiente:

      - `algorithm_name`: nombre del algoritmo (cadena).
      - `env_n`: número del entorno aleatorio (entero).
      - `states_n`: cantidad de estados explorados (entero).
      - `cost_e1`: costo total de las acciones en el escenario 1 (entero).
      - `cost_e2`: costo total de las acciones en el escenario 2 (entero).
      - `time`: tiempo empleado (float).
      - `solution_found`: indicar si se encontró la solución (booleano).

   d) Dentro de la carpeta `tp4-busquedas-informadas`, colocar un archivo con el nombre `tp3y4-reporte.md` con la evaluación de desempeño de los ejercicios (3), (4) y (5) del TP3 y del ejercicio (2) del TP4. Incluya en este reporte, gráficos que den cuenta de la comparación. Organice el reporte siguiendo la estructura:

      - Introducción: descripción general del problema.
      - Marco teórico: descripción teórica y general del funcionamiento del entorno y de los agentes, y sus principales elementos.
      - Diseño experimental: descripción de los experimentos.

- Análisis y discusión de resultados: presentar los resultados obtenidos en los experimentos, realizando un breve análisis de dichos resultados, e incluyendo gráficas o tablas que los resuman.
- Conclusiones: conclusiones finales de los resultados obtenidos.

e) Dentro de la carpeta `tp4-busquedas-informadas`, crear una nueva carpeta `images`, que incluya todos los gráficos e imágenes utilizados en el reporte final.