Implementacion de busqueda usando SIMPLE-AI

Introducción a la práctica y elementos de programación

Práctica Búsqueda. 2017-18 LEG
Departamento de Informática

Universidad Carlos III de Madrid

Esquema

Esquema

Resumen y objetivo

- Veremos una aplicación práctica de búsquedas
- El proyecto se evaluará mediante una **entrega** y un **examen** que se realizará coincidiendo con la convocatoria ordinaria de la asignatura.
- Usaremos un paquete de aplicaciones para IA escrito en Python, SIMPLE-AI
- **Plataformas soportadas**:
  - La plataforma de referencia es Linux
  - La versión de Python de referencia es la 2.7

Esquema

Estructura

- `simple-ai-0.8.1`: Soporte de algoritmos de AI (del libro)
- `game-vX.X`: Librerías del juego
- `student`: Ficheros modificables por el alumno
- `tutorial`: Ficheros del tutorial

Dependencias:

- `simple-ai-0.8.1`: Requiere *Flask* para el interfaz web (instalado en laboratorio)
- `game-vX.X`: Requiere el módulo *pygame* (instalado en laboratorio)

Instalación requerida:

- `pydot`: Instalar desde línea de comando: `pip install pydot`

Python vs. Java

- Importante: El código debe ser **obligatoriamente indentado** (no se usan llaves). Usar tabulador o 4 espacios, pero **no ambos**
- Los comentarios empiezan por `#` o bien en bloques identificados por tres comillas simples.
- Muchos errores aparecerán en tiempo de ejecución
- Los operadores lógicos son `and`,`not`,`or`
- Los strings se definen con comillas simples o dobles. El operador de concatenación de strings es '+'. Admiten acceso con la sintaxis de slices.
- Las funciones se invocan sobre variables. Pueden devolver una lista de resultados: `res1, ..., resn = funcion(vars)`
- Los métodos se invocan sobre objetos (y tienen variables): `objeto.metodo(vars)`
- Para imprimir usaremos la función `print()` con un solo argumento: `print("String")`
- Para crear strings que muestren resultados, se usa el método `<string>.format(<lista>).`

```
print ("El resultado numero {0} es {1}".format(1,"un resultado"))
```

Hay abundante documentación on-line en castellano
`https://wiki.python.org/moin/SpanishLanguage`

Cargando el intérprete

- Entrar y salir del intérprete interactivo (observar que es la versión 2.7)

```
 ~$ python
Python 2.7.3 (default, Jun 22 2015, 19:43:34)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> quit()
~$
```

  Nota: el prompt >>> indica que estamos *dentro* del intérprete

- También se puede ejecutar un módulo indicando el nombre de fichero (extensión ".py") y las opciones `~$ python gameAI.py`
- Se puede navegar la ayuda usando la función `help()` o buscar ayuda sobre un elemento concreto `help(list)`

Cargando módulos

- Se carga otro fichero en el intérprete con `import <module> as <alias>`, se accede a sus funciones como `<alias>.funcion` y se recarga si se modifica con `reload(<alias>)`
- Se puede usar la función `dir(<alias>)` para mostrar los elementos de un módulo o clase.

```
>>> import os,sys
>>> sys.path.append(os.path.abspath("../simpleai-0.8.1"))
>>> from simpleai.search import SearchProblem
>>> import simpleai.search as sai
>>> dir(sai)
>>> ['CspProblem', 'HIGHEST_DEGREE_VARIABLE', 'LEAST_CONSTRAINING_VALUE',
     'MOST_CONSTRAINED_VARIABLE', 'SearchProblem',
     '__builtins__', '__doc__', '__file__', '__name__', '__package__', '__path__',
     'astar', 'backtrack', 'beam', 'breadth_first', 'convert_to_binary', 'csp',
     'depth_first', 'genetic', 'greedy',
     'hill_climbing', 'hill_climbing_random_restarts', 'hill_climbing_stochastic',
     'iterative_limited_depth_first', 'limited_depth_first', 'local',
     'min_conflicts', 'models', 'simulated_annealing', 'traditional', 'uniform_cost', 'utils']
```

Búsqueda en un Mapa

- Vamos a crear una clase MapProblem donde se realizará búsqueda sobre un grafo (mapa) como se hace en los ejercicios de clase.
- La nueva clase tiene que implementar los siguientes métodos:
  - *actions(s)*: Devuelve una LISTA de acciones posibles en cierto estado *s*
  - *result(s,a)*: Devuelve un ESTADO, resultado de aplicar una acción *a* en un estado *s*
  - *is_goal (s)*: Devuelve un BOOLEANO, True si *s* es un estado final
  - *cost (s,a,s')*: Devuelve un valor numérico positivo, coste de ir de un estado (*s*) a otro (*s'*) mediante cierta acción *a*
  - *heuristic (s)*: Devuelve un valor numérico, valor de la heurística *h()* para el estado *s*

Estados

- Los estados tienen que ser un elemento **inmutable**
- El **tipo** se puede ver con la función `type(<variable>)`
- Los elementos inmutables básicos en Python son: números, strings y tuplas de números o strings. Estos elementos se **copian** en las asignaciones.
- Los strings pueden ir entre comillas simples o dobles
- Las tuplas (vectores) van entre paréntesis `(0,'otronumero',2.1)` y sus elementos se **leen** mediante *slices*: `v[inicio:fin]`, donde **fin es el elemento siguiente al último**.
  - El **primer** elemento tiene índice 0.
  - Se pueden usar valores negativos: el último elemento tiene índice -1 (ej: `v[-1]`)
  - El elemento de índice `fin` **no se incluye en el slice**.
  - Inicio y fin son **opcionales**: si no se ponen el slice se extiende desde o hasta el límite correspondiente.

```
estado_inicial='A'
estado_final='H'
```

```
una_tupla=('A',1,'Un string')
una_tupla[1:2]
(1,)
una_tupla[1:3]
(1, 'Un string')
una_tupla[2]='Otro string'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

Problema

- El problema de búsqueda viene dado por un grafo explícito.
- Podemos codificarlo usando un **diccionario**, que es un tipo **mutable**
- Importante: Las variables a las que se asigna un objeto **mutable** son **referencias** (no copias)
- Las constantes de tipo diccionario van entre corchetes `{ clave1: valor1, clave2:valor2 }`
- Las claves deben ser **inmutables**, pero los valores pueden ser **mutables**
- Se accede a un elemento con la sintaxis: `objeto[<clave>]`.
- Se puede borrar una clave con la función `del`

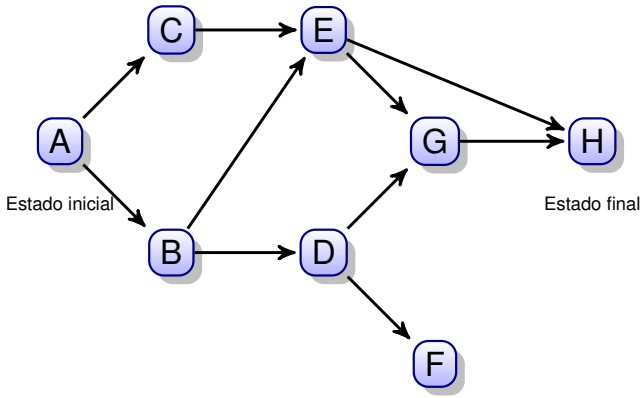
```
>>> un_diccionario={'atributo1':'valor1','atributo2':'val
>>> d1=un_diccionario
>>> d2=un_diccionario
>>> d1['atributo1']
'valor1'
>>> d1['atributo1']='valor1_modificado'
>>> d2['atributo1']
'valor1_modificado'
>>> d1['atributo3']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'atributo3'
>>> d1.keys()
['atributo1', 'atributo2']
>>> d1.values()
['valor1_modificado', 'valor2']
>>> d1.items()
[('atributo1', 'valor1_modificado'),
 ('atributo2', 'valor2')]
```

```
sucesoresA= { 'accion-1':'B', 'accion-2':'C' }
sucesoresB= { 'accion-1':'D', 'accion-2':'E' }
mapa = { 'A': sucesoresA, 'B': sucesoresB}
print(mapa)
>>> {'A': {'accion-1': 'B', 'accion-2': 'C'},
'B': {'accion-1': 'D', 'accion-2': 'E'}}
print (mapa['A'])
>>> {'accion-1': 'B', 'accion-2': 'C'}
print (mapa['A']['accion-1'])
>>> B
```

Definición del mapa

```
sucesoresA= { 'accion-1':'B', 'accion-2':'C' }
sucesoresB= { 'accion-1':'D', 'accion-2':'E' }
sucesoresC= { 'accion-1':'E' }
sucesoresD= { 'accion-1':'F', 'accion-2':'G' }
sucesoresE= { 'accion-1':'G', 'accion-2':'H' }
sucesoresF= {  }
sucesoresG= { 'accion-1':'H' }
sucesoresH= {  }
mapa = { 'A': sucesoresA, 'B': sucesoresB , 'C': sucesoresC,
         'D': sucesoresD, 'E': sucesoresE , 'F': sucesoresF,
         'G': sucesoresG, 'H': sucesoresH ,
    }

print(mapa)
{'A': {'accion-1': 'B', 'accion-2': 'C'},
 'C': {'accion-1': 'E'},
 'B': {'accion-1': 'D', 'accion-2': 'E'},
 'E': {'accion-1': 'G', 'accion-2': 'H'},
 'D': {'accion-1': 'F', 'accion-2': 'G'},
 'G': {'accion-1': 'H'}, 'F': {}, 'H': {}
}
```



Acciones

actions(s): Devuelve una LISTA de acciones posibles en cierto estado *s*

- Las funciones se declaran con el nombre y parámetros, seguidos de dos puntos (:)
- Se puede acceder a las variables y funciones de la clase con el prefijo `self`.
- Importante: El cuerpo de la función tiene que **indentarse**
- El tipo *lista* es mutable, va entre corchetes (`[ ]`) y se **leen y modifican** los elementos mediante la notacion de slices
- Las listas admiten métodos `append()`, `pop()`, etc.
- Se puden concaternar listas con el operador '+'.
- Los diccionarios admiten los métodos `keys()`, `values()`, `items()`, todos ellos devuelven listas.

```
def actions(self, state):
    # ESTE METODO DEBE DEVOLVER UNA LISTA DE ACCIONES POSIBLES
    return self.mapaProblema[state].keys()
```

Transición entre estados

result(s,a): Devuelve un ESTADO, resultado de aplicar una acción *a* en un estado *s*

- El resultado de una acción determinada se obtiene facilmente accediendo a la definición del mapa

```
def result(self, state, action):
    return self.mapaProblema[state][action]
```

is_goal (s): Devuelve un BOOLEANO, True si *s* es un estado final

- El resultado de una acción determinada se obtiene facilmente accediendo a la definición del mapa

```
def is_goal(self, state):
    # ESTE METODO DEBE DEVOLVER UN BOOLEANO (True: estado final)
    return state == self.estado_final
```

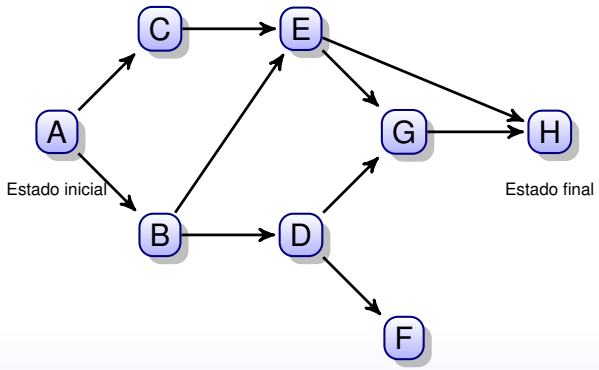
Transición entre estados

- Para ejecutar la búsqueda hay primero que crear un objeto de esta clase
- Se le pasa como parámetro el estado inicial
- Se cargan luego las variables del objeto con el nombre del objeto como cualificador
- Se invoca la función proporcionada `ejercicio_mapa`, que ejecuta la búsqueda e imprime las estadísticas
- `ejercicio_mapa` recibe como parámetro el problema, la función de búsqueda y un “visor”.
- Si se usa el visor *WebViewer()*, se puede seguir la ejecución mediante un navegador (ver línea comentada en el código solución).

```
problema=MapProblem(estado_inicial)
problema.mapaProblema=mapa
problema.estado_final= estado_final
ejercicioMapa(problem,algorithm=breadth_first,use_viewer=BaseViewer())
```

Resultado

```
Estado final:H
Camino: [(None, 'A'), ('accion-1', 'B'), ('accion-2', 'E'), ('accion-2', 'H')]
Coste: 3
visited nodes: 8
iterations: 8
max fringe size: 3
```



Continuación

- Utilizar y comparar otros algoritmos de búsqueda
- Implementar otros laberintos
- Implementar otros ejercicios de búsqueda no informada
- Implementar coste y heurística y reproducir ejercicios de búsqueda heurística.

Errores frecuentes en Python

- **Problema**: ImportError: No module named py

  Al usar import, no se incluye la extensión ".py"

- **Problema**: NameError: name 'MYVARIABLE' is not defined

  Para acceder a una variable hay que poner el nombre o alias del módulo: `modulo.MYVARIABLE` o `modulo.MYFUNCTION`

- **Problema**: TypeError: 'dict' object is not callable

  Para obtener el valor en una variable tipo `dictionary` se usan corchetes `dict["elemento"]`, no paréntesis

- **Problema**: ValueError: too many values to unpack

  En una asignación múltiple no coincide el número de elementos a derecha e izquierda `a,b = (1,2,3)`

- **Problema**: AttributeError: 'list' object has no attribute 'length' (o algo similar)

  La longitud de una lista se obtiene con la función `len(LISTA)`, **no** con un método `LISTA.len()`

- **Problema**: No funcionan los cambios hechos en un fichero

  Hay que salvar los cambios y recargar el módulo `reload(MODULO)` o `reload(ALIAS-MODULO)`