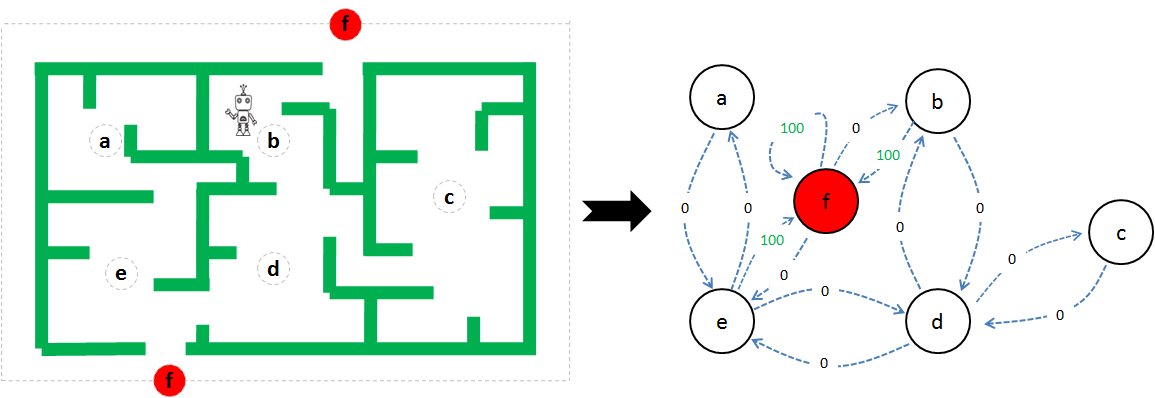
**Example**

Let’s consider an example where an agent is trying to come out of a maze. It can move one random square or area in any direction, and get a reward if exits. The most common way to formalize a reinforcement problem is to represent it as Markov decision process. Assume the agent is in state b (maze area) and the target is to reach state f. So within one step agent can reach from b to f, let’s put a reward of 100 (otherwise 0) for links between nodes that allows agents to reach target state.



import numpy as np

from random import randint

import matplotlib.pyplot as plt

from matplotlib.collections import LineCollection

# defines the reward/link connection graph

R = np.matrix([[-1, -1, -1, -1, 0, -1],

[-1, -1, -1, 0, -1, 100],

[-1, -1, -1, 0, -1, -1],

[-1, 0, 0, -1, 0, -1],

[ 0, -1, -1, 0, -1, 100],

[-1, 0, -1, -1, 0, 100]])

Q = np.zeros\_like(R)

#The -1's in the table means there isn't a link between nodes. For example, State 'a' #cannot go to State 'b'.

# learning parameter

gamma = 0.8

# Initialize random\_state

initial\_state = randint(0,4)

# This function returns all available actions in the state given as an argument

def available\_actions(state):

current\_state\_row = R[state,]

av\_act = np.where(current\_state\_row >= 0)[1]

return av\_act

# This function chooses at random which action to be performed within the range

# of all the available actions.

def sample\_next\_action(available\_actions\_range):

next\_action = int(np.random.choice(available\_act,1))

return next\_action

# This function updates the Q matrix according to the path selected and the Q

# learning algorithm

def update(current\_state, action, gamma):

max\_index = np.where(Q[action,] == np.max(Q[action,]))[1]

if max\_index.shape[0] > 1:

max\_index = int(np.random.choice(max\_index, size = 1))

else:

max\_index = int(max\_index)

max\_value = Q[action, max\_index]

# Q learning formula

Q[current\_state, action] = R[current\_state, action] + gamma \* max\_value

# Get available actions in the current state

available\_act = available\_actions(initial\_state)

# Sample next action to be performed

action = sample\_next\_action(available\_act)

### Training

# Train over 100 iterations, re-iterate the process above).

for i in range(100):

current\_state = np.random.randint(0, int(Q.shape[0]))

available\_act = available\_actions(current\_state)

action = sample\_next\_action(available\_act)

update(current\_state,action,gamma)

# Normalize the "trained" Q matrix

print ("Trained Q matrix: \n", Q/np.max(Q)\*100)

### Output

Trained Q matrix:

[[ 0. 0. 0. 0. 45.9833795 0. ]

[ 0. 0. 0. 63.71191136 0. 100. ]

[ 0. 0. 0. 63.71191136 0. 0. ]

[ 0. 79.77839335 50.96952909 0. 36.28808864 0. ]

[ 28.80886427 0. 0. 57.61772853 0. 45.42936288 ]

[ 0. 79.77839335 0. 0. 45.9833795 100. ]]

### Testing

current\_state = 2

steps = [current\_state]

while current\_state != 5:

next\_step\_index = np.where(Q[current\_state,] == np.max(Q[current\_state,]))[1]

if next\_step\_index.shape[0] > 1:

next\_step\_index = int(np.random.choice(next\_step\_index, size = 1))

else:

next\_step\_index = int(next\_step\_index)

steps.append(next\_step\_index)

current\_state = next\_step\_index

# Print selected sequence of steps

print ("Best sequence path: ", steps)

### Output

Best sequence path: [2, 3, 1, 5]