In [1]:
```
import numpy as np
import pandas as pd
```

In [26]:
```
# CONSTANTS
STATES = 81
ACTIONS = 4
GAMMA = 0.9925
ACTIONS_LIST = ['←','↑','→','↓']
MAZE = [3,11,12,15,16,17,
        20,22,23,24,26,29,
        30,31,34,35,39,43,
        48,52,53,56,57,58,
        59,60,61,62,66,70,71]
```

In [4]:
```
# Loading input files
def parse_sparse(file_name):
    sparse = np.loadtxt(file_name)
    out_matrix = np.zeros([STATES, STATES])

    for row in sparse:
        out_matrix[int(row[1])-1, int(row[0]-1)] = row[2]

    return out_matrix

prob_a1 = parse_sparse('prob_a1.txt')
prob_a2 = parse_sparse('prob_a2.txt')
prob_a3 = parse_sparse('prob_a3.txt')
prob_a4 = parse_sparse('prob_a4.txt')
transition_mtxs = [prob_a1, prob_a2, prob_a3, prob_a4]
rewards = np.loadtxt('rewards.txt')
```

In [17]:
```
rewards.shape
```

Out[17]:
```
(81,)
```

9.4 (a)

In [18]:
```
# Helper functions

# Policy evaluation
def value_function(pi):

    # construct nxn matrix of GAMMA*P(s'|s,pi(s))
    square = np.zeros([STATES,STATES])
    id_mtx = np.identity(STATES)

    # Update the matrix row by row
    for s in range(STATES):
        p_mtx = transition_mtxs[int(pi[s])]
        square[s,:] = id_mtx[s,:]-GAMMA*p_mtx[:,s]

    #invert square mtx
    inv = np.linalg.inv(square)

    #evaluate V(s) for all s
    V_s = np.dot(inv, rewards)
    return V_s

# Policy improvement
def policy_improvement(s, pi):
    vals = np.repeat(-np.inf, ACTIONS)

    for a in range(ACTIONS):
        vals[a] = np.sum(transition_mtxs[a][:,s] * value_function(pi))

    return np.argmax(vals)

# Policy iteration
def policy_iteration():

    # Initialize ploicy pi at random
    pi = np.random.randint(ACTIONS, size=STATES)

    # Compute the value function based on the policy pi
    Vpi = value_function(pi)

    while True:
        pi_new = np.repeat(1, STATES)

        for s in range(STATES):
            pi_new[s] = policy_improvement(s, pi)
        Vpi_new = value_function(pi_new)

        # Convergence condition
        if all(Vpi == Vpi_new):
            break

        pi = pi_new
        Vpi = Vpi_new
    return pi, Vpi
```

In [42]:
```
# Run the algorithm
pi_opt, V_opt = policy_iteration()
```

In [43]:
```
Vopt_formatted = []
for val in V_opt:
    if val>0:
        Vopt_formatted.append(str(val))
    if val==0:
        Vopt_formatted.append('wall')
    if val<0:
        Vopt_formatted.append(str(val))
pd.DataFrame(np.array(Vopt_formatted).reshape(9,9).T)
```

Out[43]:

	0	1	2	3	4	5	
0	wall	wall	wall	wall	wall	wall	
1	wall	102.37526440102093	103.23462341601052	104.10121204279734	wall	-133.33333333333414	8
2	100.70098072748912	101.5236451489813	wall	104.97507555494724	103.78140737394392	90.98537960093465	9
3	wall	wall	106.77826755022936	105.88853590955102	wall	-133.33333333333385	9
4	wall	wall	107.67462642880358	wall	wall	wall	10
5	wall	109.48993453646308	108.57848711681844	wall	wall	-133.33333333333417	10
6	wall	110.40903296181364	wall	114.16322950263661	115.12155726913032	116.087929588253	12
7	wall	111.33584663396842	112.27044031794429	113.21287932200798	wall	122.02491241481368	12
8	wall	wall	wall	wall	wall	wall	

In [44]:
```
directions = []
for i in range(0,81):
    if i+1 not in MAZE:
        directions.append("W")
    else:
        directions.append(ACTIONS_LIST[pi_opt[i]])

print(np.array(directions).reshape(9,9).T)
```

```
[['W' 'W' 'W' 'W' 'W' 'W' 'W' 'W' 'W']
 ['W' '→' '→' '↓' 'W' 'W' '↓' 'W' 'W']
 ['→' '↑' 'W' '↓' '←' '←' '↓' '←' 'W']
 ['W' 'W' '↓' '←' 'W' 'W' '↓' 'W' 'W']
 ['W' 'W' '↓' 'W' 'W' 'W' '↓' 'W' 'W']
 ['W' '↓' '←' 'W' 'W' 'W' '↓' 'W' 'W']
 ['W' '↓' 'W' '→' '→' '→' '→' '→' 'W']
 ['W' '→' '→' '↑' 'W' '→' '→' '↑' 'W']
 ['W' 'W' 'W' 'W' 'W' 'W' 'W' 'W' 'W']]
```

9.4 (b)

In [54]:
```
# Helper functions

def value_iteration():
    # Initialize V0 to all zeros
    Vk = np.repeat(0,STATES)

    #initialize Vk_new
    Vk_new = np.repeat(-np.inf,STATES)

    #initialize optimal policy pi*
    pi_opt = np.repeat(-np.inf,STATES)

    iter_count = 1

    while True:
        vals = np.full((ACTIONS,STATES), -np.inf)

        for a in range(ACTIONS):
            vals[a,:] = [np.sum(transition_mtxs[a][:,s]*Vk) for s in range(STATES)]

        Vk_new = np.array([rewards[s] + GAMMA * np.amax(vals[:,s]) for s in range(STATES)])

        # Convergence condition
        if all(Vk_new == Vk):
            # Deal with optimal policy
            vals = np.full([ACTIONS,STATES], -np.inf)
            for a in range(ACTIONS):
                vals[a,:] = [np.sum(transition_mtxs[a][:,s]*Vk_new) for s in range(STATES)]
            pi_opt = np.array([np.argmax(vals[:,s]) for s in range(STATES)])
            break

        Vk = Vk_new
        iter_count += 1

    return Vk, pi_opt
```

In [49]:
```
# Run the algorithm
Vopt2, pi_opt2 = value_iteration()
```

In [50]:
```
Vopt2_formatted = []
for val in Vopt2:
    if val>0:
        Vopt2_formatted.append(str(val))
    if val==0:
        Vopt2_formatted.append('wall')
    if val<0:
        Vopt2_formatted.append(str(val))
pd.DataFrame(np.array(Vopt2_formatted).reshape(9,9).T)
```

Out[50]:

	0	1	2	3	4	5	
0	wall	wall	wall	wall	wall	wall	
1	wall	102.37526440101948	103.23462341600904	104.10121204279588	wall	-133.33333333333232	8
2	100.70098072748769	101.52364514897985	wall	104.97507555494576	103.78140737394247	90.98537960093337	9
3	wall	wall	106.77826755022787	105.88853590954956	wall	-133.33333333333232	9
4	wall	wall	107.67462642880207	wall	wall	wall	10
5	wall	109.48993453646152	108.5784871168169	wall	wall	-133.33333333333232	1
6	wall	110.40903296181204	wall	114.16322950263499	115.12155726912869	116.08792958825137	12
7	wall	111.33584663396682	112.27044031794267	113.21287932200636	wall	122.02491241481191	12
8	wall	wall	wall	wall	wall	wall	

In [53]:
```
directions2 = []
for i in range(0,81):
    if i+1 not in MAZE:
        directions2.append("W")
    else:
        directions2.append(ACTIONS_LIST[pi_opt2[i]])

print(np.array(directions2).reshape(9,9).T)
```

```
[['W' 'W' 'W' 'W' 'W' 'W' 'W' 'W' 'W']
 ['W' '→' '→' '↓' 'W' 'W' '↓' 'W' 'W']
 ['→' '↑' 'W' '↓' '←' '←' '↓' '←' 'W']
 ['W' 'W' '↓' '←' 'W' 'W' '↓' 'W' 'W']
 ['W' 'W' '↓' 'W' 'W' 'W' '↓' 'W' 'W']
 ['W' '↓' '←' 'W' 'W' 'W' '↓' 'W' 'W']
 ['W' '↓' 'W' '→' '→' '→' '→' '→' 'W']
 ['W' '→' '→' '↑' 'W' '→' '→' '↑' 'W']
 ['W' 'W' 'W' 'W' 'W' 'W' 'W' 'W' 'W']]
```

In [ ]: