Challenge problem 1

In [ ]:

```
### Challenge problem 1 - solution

### nodes, edges and size of the network
nodes = np.array(ns['node'])
edges = np.array(hwn['edge'])
nn = len(nodes)
ne = len(edges)

print ('Nodes, edges and size of the network:')
print ('Nb of nodes: %d'%nn)
print ('First three and last three elements of the list of nodes: %r ... %r'%(nodes[:3], nodes[-3:]))
print ('Nb of edges: %d'%ne)


### computation of the weighted degrees
weighted_degrees = np.zeros(nn)

# loop over the edges
for c in range(ne):

    # get weight of edge c
    i = hwn['i'].iloc[c]
    j = hwn['j'].iloc[c]
    w = hwn['weight'].iloc[c]

    # add weight to both nodes i and j
    weighted_degrees[i] += w
    weighted_degrees[j] += w

# check that we made the right computation
if not np.sum(weighted_degrees)/2==np.sum(hwn['weight']):
    print ('ERROR: sum of weights is different from the sum of weighted degrees divided by 2')

# mean and standard deviation of the weighted degrees
print ('\nWeighted degrees:')
print ('mean: %.0f'%np.mean(weighted_degrees))
print ('standard deviation: %.0f'%np.std(weighted_degrees))

# plot distribution of the weighted degrees
plt.hist(weighted_degrees, bins=30)
plt.title('Distrib. of weighted degrees')
plt.show()
```

Challenge problem 2

In [ ]:

```
### Challenge problem 2 - solution

### list of nodes by status
administratives = np.array(ns[ns['status']=='ADM']['node'])
medicals = np.array(ns[ns['status']=='MED']['node'])
nurses = np.array(ns[ns['status']=='NUR']['node'])
patients = np.array(ns[ns['status']=='PAT']['node'])

nadm = len(administratives)
nmed = len(medicals)
nnur = len(nurses)
npat = len(patients)

print ('Nodes by status:')
print ('ADM: ', nadm)
print ('MED: ', nmed)
print ('NUR: ', nnur)
print ('PAT: ', npat)
print ('total: ', nadm+nmed+nnur+npat)

### arrays of weighted degrees by status
adm_wd = weighted_degrees[administratives] # get the degrees stored in weighted_degrees corresponding to
                                           # the administrative nodes
med_wd = weighted_degrees[medicals]
nur_wd = weighted_degrees[nurses]
pat_wd = weighted_degrees[patients]

print ('\nWeighted degrees by status (mean and standard deviation):')
print ('ADM: \t%.0f \t%.0f'%(np.mean(adm_wd), np.std(adm_wd)))
print ('MED: \t%.0f \t%.0f'%(np.mean(med_wd), np.std(med_wd)))
print ('NUR: \t%.0f \t%.0f'%(np.mean(nur_wd), np.std(nur_wd)))
print ('PAT: \t%.0f \t%.0f'%(np.mean(pat_wd), np.std(pat_wd)))

### plot distributions of weighted degrees by status
fig, axs = plt.subplots(1,4,figsize=(15,4))
fig.suptitle('Weighted degree distributions by status')
bins = np.arange(0,4350,150)

# plot weighted degree distibutions for all the nodes
axs[0].hist(weighted_degrees, histtype='step', color='w', edgecolor='k', bins=bins, label= 'all statuses')
axs[1].hist(weighted_degrees, histtype='step', color='w', edgecolor='k', bins=bins, label= 'all statuses')
axs[2].hist(weighted_degrees, histtype='step', color='w', edgecolor='k', bins=bins, label= 'all statuses')
axs[3].hist(weighted_degrees, histtype='step', color='w', edgecolor='k', bins=bins, label= 'all statuses')

# plot weighted degree distributions by status
colors = {'ADM':'green', 'MED':'blue', 'NUR':'red', 'PAT':'gold'}
axs[0].hist(adm_wd, bins=bins, color=colors['ADM'], label='ADM')
axs[1].hist(med_wd, bins=bins, color=colors['MED'], label='MED')
axs[2].hist(nur_wd, bins=bins, color=colors['NUR'], label='NUR')
axs[3].hist(pat_wd, bins=bins, color=colors['PAT'], label='PAT')

axs[0].set_title('Administratives')
axs[1].set_title('Medical doctors')
axs[2].set_title('Nurses')
axs[3].set_title('Patients')

axs[0].legend()
axs[1].legend()
axs[2].legend()
axs[3].legend()

plt.show()
```

BONUS Challenge problem 3

In [ ]:

```
### BONUS Challenge problem 3 - solution

### compute the contact matrix
status_index = {'ADM':0, 'MED':1, 'NUR':2, 'PAT':3}

mat_contacts = np.zeros((4,4))

for e in range(ne):

    i = hwn['i'].iloc[e]
    j = hwn['j'].iloc[e]
    w = hwn['weight'].iloc[e]
    si = ns['status'][i]
    sj = ns['status'][j]

    if si==sj:   # fill the diagonal
        mat_contacts[status_index[si],status_index[sj]] += w
    else:        # fill cells outside the diagonal
        mat_contacts[status_index[si],status_index[sj]] += w
        mat_contacts[status_index[sj],status_index[si]] += w

### plot the heatmap
plt.figure(figsize=(5,4))

# plot the heatmap
plt.pcolormesh(mat_contacts, cmap=plt.cm.Blues)
# add values in the cells
for i in range(4):
    for j in range(4):
        plt.text(i+0.3, j+0.4, '%d'%mat_contacts[j][i], fontsize=14)

plt.title('matrix of weighted contacts', fontsize=16)
plt.xticks(np.array(range(4)) + 0.5, status_index.keys(), fontsize=14)
plt.yticks(np.array(range(4)) + 0.5, status_index.keys(), fontsize=14)

plt.show()
```

Challenge problem 4

In [ ]:

```
### Challenge problem 4 - solution


    ### initialisation
    N = len(nodes)

    # infectious status of a node: 's'=susceptible; 'i'=infectious; 'r'=recovered
    inf_status = np.array(['s']*N)
    inf_status[initial_infected] = 'i'
    inf_status[initial_recovereds] = 'r'
    new_inf_status = inf_status.copy()

    # lists to store results
    time = []
    S = []
    I = []
    R = []

    # at time 0
    time.append(0)
    S.append(N-1-len(initial_recovereds))
    I.append(1)
    R.append(len(initial_recovereds))
    t = 0

    ### evolution of the epidemics
    while True:
        t += 1
        time.append(t)

        # transmission and recovery
        for n in nodes:

            # if n is infected, maybe it recovers
            if inf_status[n]=='i':
                if random.uniform(0,1)<mu:
                    new_inf_status[n] = 'r'

            # if n is suceptible, maybe a neighbour can infect him
            elif inf_status[n]=='s':
                for m,w in zip(neighbours[n],weights[n]):
                    if inf_status[m]=='i':
                        prob_of_infection = 1 - np.power(1-beta, w)
                        if random.uniform(0,1)<prob_of_infection:
                            new_inf_status[n]='i'
                            break

        # update infectious status
        inf_status = new_inf_status.copy()

        # compute total number of susceptible, infectious, recovered
        nsuscep = len(inf_status[inf_status=='s'])
        ninfect = len(inf_status[inf_status=='i'])
        nrecov = len(inf_status[inf_status=='r'])
        S.append(nsuscep)
        I.append(ninfect)
        R.append(nrecov)

        # end simulation if no more infectious
        if ninfect==0:
            break
```

Challenge problem 6

In [ ]:

```
### Challenge problem 6 - solution

vacc1 = random.sample(list(nodes), ndoses)        # randomly sample 'ndoses' elements from list 'nodes'
vacc2 = random.sample(list(patients), ndoses)
vacc3 = random.sample(list(nurses)+list(medicals), ndoses)
vacc4 = np.argsort(weighted_degrees)[-ndoses:]    # get indexes of the 'ndoses' nodes with highest degree
notvacc1 = [n for n in nodes if not n in vacc1]
notvacc2 = [n for n in nodes if not n in vacc2]
notvacc3 = [n for n in nodes if not n in vacc3]
notvacc4 = [n for n in nodes if not n in vacc4]
```

Challenge problem 7

In [ ]:

```
### Challenge problem 7 - solution

# check time of execution
start_time = time.time()

# matrices to store results of simulations, one for each strategy
nsimulations = ncampaigns*niter_per_seed*(nn-ndoses)
mat1 = np.zeros((nsimulations, tmax))
mat2 = np.zeros((nsimulations, tmax))
mat3 = np.zeros((nsimulations, tmax))
mat4 = np.zeros((nsimulations, tmax))

c = 0   # simulation counter
for campaign in range(ncampaigns):

    # define people to be vaccinated according to the 4 strategies
    vacc1 = random.sample(list(nodes), ndoses)
    vacc2 = random.sample(list(patients), ndoses)
    vacc3 = random.sample(list(nurses)+list(medicals), ndoses)
    vacc4 = np.argsort(weighted_degrees)[-ndoses:]
    notvacc1 = [n for n in nodes if not n in vacc1]
    notvacc2 = [n for n in nodes if not n in vacc2]
    notvacc3 = [n for n in nodes if not n in vacc3]
    notvacc4 = [n for n in nodes if not n in vacc4]

    # for each not vaccinated node as seed
    for n in range(nn-ndoses):

        # iterating several times for each seed
        for i in range(niter_per_seed):

            # simulations
            t1, S1, I1, R1 = SIR(nodes, neighbours, weights, beta, mu, initial_infected=notvacc1[n],
                                 initial_recovereds=vacc1)
            t2, S2, I2, R2 = SIR(nodes, neighbours, weights, beta, mu, initial_infected=notvacc2[n],
                                 initial_recovereds=vacc2)
            t3, S3, I3, R3 = SIR(nodes, neighbours, weights, beta, mu, initial_infected=notvacc3[n],
                                 initial_recovereds=vacc3)
            t4, S4, I4, R4 = SIR(nodes, neighbours, weights, beta, mu, initial_infected=notvacc4[n],
                                 initial_recovereds=vacc4)

            # fix R
            R1 = fix_R(R1, tmax)
            R2 = fix_R(R2, tmax)
            R3 = fix_R(R3, tmax)
            R4 = fix_R(R4, tmax)

            # save results
            mat1[c,:] = R1
            mat2[c,:] = R2
            mat3[c,:] = R3
            mat4[c,:] = R4
            c += 1

print("--- %s seconds ---" % (time.time() - start_time))

### Plots
fig = plt.figure(figsize=(9,6))

x = np.arange(0,tmax)

# epidemic without vaccination
plt.plot(x, np.mean(mat_novax, axis=0), color='k', ls='--', lw=3, label='no vaccination')

# different vaccination strategies
plt.plot(x, np.mean(mat1, axis=0)-ndoses, color='orange', lw=3, label='%d random vaccinated'%ndoses)
plt.plot(x, np.mean(mat2, axis=0)-ndoses, color=colors['PAT'], lw=3, label='%d PAT vaccinated'%ndoses)
plt.plot(x, np.mean(mat3, axis=0)-ndoses, color=colors['MED'], lw=3, label='%d MED/NUR vaccinated'%ndoses)
plt.plot(x, np.mean(mat4, axis=0)-ndoses, color='violet', lw=3, label='%d highest degrees vaccinated'%ndoses)

plt.xlabel('time')
plt.ylabel('Nb of recovered after be infected')
plt.legend(title=r'$\beta=$%4.3f, $\mu=$%2.1f'%(beta, mu), loc='lower right')
plt.title('Comparison of different vaccination strategies, %d campaigns'%ncampaigns)

fig.tight_layout()
```