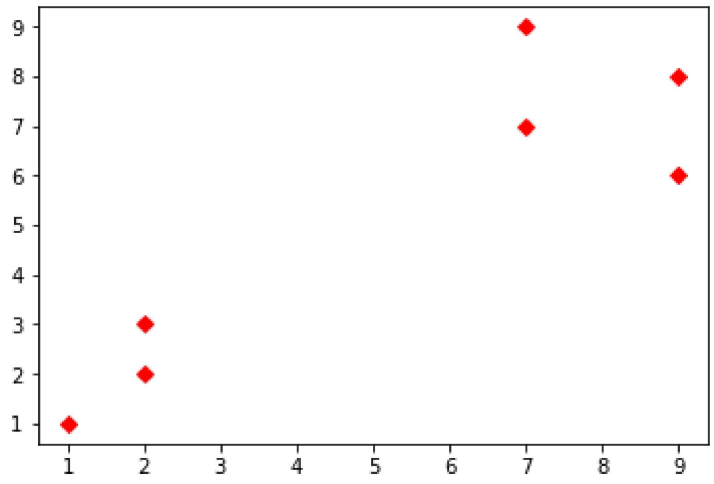
Import the libraries

In [7]:
```
import numpy as np
from scipy import linalg
import math
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
```

(1) plot the data

In [8]:
```
A = np.array([[1.0,1.0],[2.0,2.0],[2.0,3.0],[7.0,9.0],[7.0,7.0],[9.0,8.0],[9.0
,6.0]])
for point in A:
    plt.plot(point[0],point[1],'rD-')
```



Euclidean distance function

In [9]:
```
def dist(vec1,vec2):
    # computing the distance, vec1 and vec2 should have the same size; c is th
e centroid; Ai is the element in A
    d=np.linalg.norm(vec1-vec2)
    return d
```

In [10]:
```
a = dist(A[0],A[1])
a
```

Out[10]: 1.4142135623730951

e-neighborhood function

In [11]:
```
def epsilon(vi,vj,e):
    s=dist(vi,vj)
    if s<0.5:
        return 0
    else:
        return s
```

Gaussian kernel distance function

In [20]:
```
d = matrixM(A)
#for i in range(7):
    #print(d[i][i])
dd = matrixMM(A)
d==dd
```

Out[20]:
```
array([[ True,  True,  True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True,  True,  True]])
```

In [19]:
```
def matrixMM(A):
    (m,n)=np.shape(A)
    b=np.zeros((m,m))
    for i in range(m):
        for j in range(m):
            b[i][j]=np.linalg.norm(A[i]-A[j])
    return b
```

In [12]:
```
def matrixM(A):
    (m,n)=np.shape(A)
    b=np.zeros((m,m))
    for i in range(m):
        for j in range(m):
            b[i][j]=dist(A[i],A[j])
    return b
```

In [21]:
```
def matrixS(M):
    (m,n)=np.shape(M)
    b=np.zeros((m,m))
    for i in range(m):
        for j in range(m):
            b[i][j]=math.exp((-np.square(M[i][j]))/(2*np.square(2.1)))
    return b
```

In [22]:
```
def matrixW(S):
    (m,n)=np.shape(S)
    b=np.zeros((m,m))
    for i in range(m):
        for j in range(m):
            if S[i][j]<0.5 or S[i][j]==1:
                b[i][j]=0
            else:
                b[i][j]=S[i][j]
    return b
```

In [23]:
```
#diag matr
def matrixD(A):
    (m,n)=np.shape(A)
    D=np.zeros((m,m))
    for i in range(m):
        for j in range(m):
            if A[i][j] != 0:
                D[i][i]+=A[i][j]
    return D
```

(2) generate distance matrix M

In [24]:
```
M=matrixM(A)
```

(3) Print the matrix with at most 2 decimal places

In [25]:
```
np.set_printoptions(precision=2) #print("%.2f" % M)
print(M)
```

```
[[ 0.    1.41  2.24 10.    8.49 10.63  9.43]
 [ 1.41  0.    1.    8.6   7.07  9.22  8.06]
 [ 2.24  1.    0.    7.81  6.4   8.6   7.62]
 [10.    8.6   7.81  0.    2.    2.24  3.61]
 [ 8.49  7.07  6.4   2.    0.    2.24  2.24]
 [10.63  9.22  8.6   2.24  2.24  0.    2.  ]
 [ 9.43  8.06  7.62  3.61  2.24  2.    0.  ]]
```

(4) generate similarity matrix S based on M

In [41]:
```
S=matrixS(M)
```

(5) Print the similarity matrix S with at most 2 decimal places

In [42]:
```
np.set_printoptions(suppress=True)
np.set_printoptions(precision=2)
print(S)#(5)
```

```
[[1.   0.8  0.57 0.   0.   0.   0.  ]
 [0.8  1.   0.89 0.   0.   0.   0.  ]
 [0.57 0.89 1.   0.   0.01 0.   0.  ]
 [0.   0.   0.   1.   0.64 0.57 0.23]
 [0.   0.   0.01 0.64 1.   0.57 0.57]
 [0.   0.   0.   0.57 0.57 1.   0.64]
 [0.   0.   0.   0.23 0.57 0.64 1.  ]]
```

(6) generate weight matrix W based on S

In [43]:
```
W= matrixW(S)
print(W)
```

```
[[0.   0.8  0.57 0.   0.   0.   0.  ]
 [0.8  0.   0.89 0.   0.   0.   0.  ]
 [0.57 0.89 0.   0.   0.   0.   0.  ]
 [0.   0.   0.   0.   0.64 0.57 0.  ]
 [0.   0.   0.   0.64 0.   0.57 0.57]
 [0.   0.   0.   0.57 0.57 0.   0.64]
 [0.   0.   0.   0.   0.57 0.64 0.  ]]
```

(7) Plot the graph associated to W

(8) Compute the Laplacian matrix L based on W

In [45]:
```
D=matrixD(W)
print(D)
print()
L=D-W
print(L)
```

```
[[1.36 0.   0.   0.   0.   0.   0.  ]
 [0.   1.69 0.   0.   0.   0.   0.  ]
 [0.   0.   1.46 0.   0.   0.   0.  ]
 [0.   0.   0.   1.2  0.   0.   0.  ]
 [0.   0.   0.   0.   1.77 0.   0.  ]
 [0.   0.   0.   0.   0.   1.77 0.  ]
 [0.   0.   0.   0.   0.   0.   1.2 ]]

[[ 1.36 -0.8  -0.57  0.    0.    0.    0.  ]
 [-0.8   1.69 -0.89  0.    0.    0.    0.  ]
 [-0.57 -0.89  1.46  0.    0.    0.    0.  ]
 [ 0.    0.    0.    1.2  -0.64 -0.57  0.  ]
 [ 0.    0.    0.   -0.64  1.77 -0.57 -0.57]
 [ 0.    0.    0.   -0.57 -0.57  1.77 -0.64]
 [ 0.    0.    0.    0.   -0.57 -0.64  1.2 ]]
```

(9) Compute the eigenvalues and eigenvectors of L

In [66]:
```
eivalue,eivector=np.linalg.eig(L)
print(eivalue)
print(eivector)
```

```
[-0.    1.97  2.55  0.    1.2   2.41  2.34]
[[ 0.58  0.76  0.3   0.    0.    0.    0.  ]
 [ 0.58 -0.12 -0.81  0.    0.    0.    0.  ]
 [ 0.58 -0.64  0.51  0.    0.    0.    0.  ]
 [ 0.    0.    0.    0.5   0.71 -0.5  -0.04]
 [ 0.    0.    0.    0.5   0.04  0.5   0.71]
 [ 0.    0.    0.    0.5  -0.04  0.5  -0.71]
 [ 0.    0.    0.    0.5  -0.71 -0.5   0.04]]
```

(10) Sort the eigenvalues and eigenvectors in increasing order

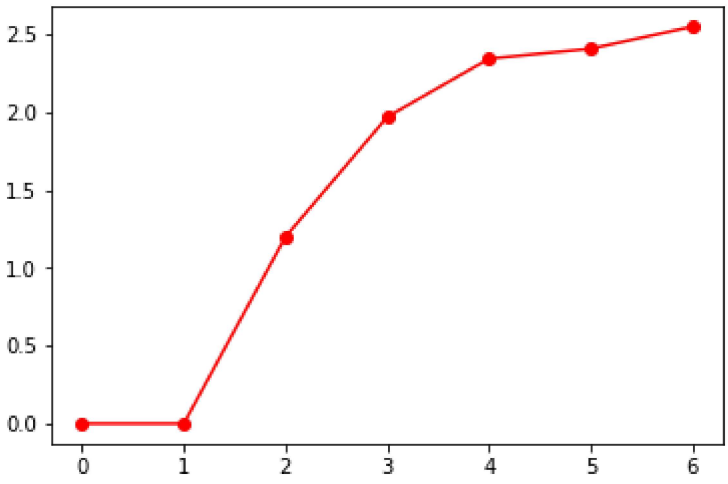
In [76]:
```
eivalue_sorted=np.sort(eivalue)
res=sorted(zip(eivalue, eivector))
print(res[1][1])
```

```
[ 0.    0.    0.    0.5   0.71 -0.5  -0.04]
```

(11) Plot the eigenvalues in increasing order

In [78]:
```
plt.plot(eivalue_sorted,'ro-')
```

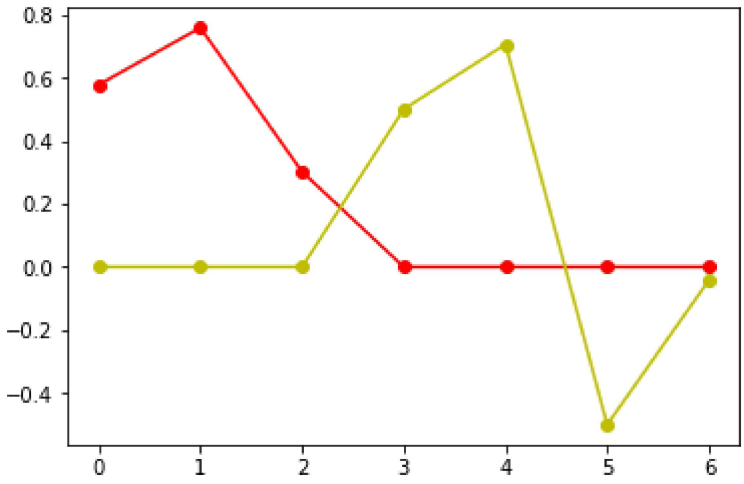
Out[78]: [<matplotlib.lines.Line2D at 0x236e409bf98>]



(12) Plot the first eigenvector and the second eigenvector

In [81]:
```
plt.plot(res[0][1],'ro-')
plt.plot(res[1][1],'yo-')
```

Out[81]: [<matplotlib.lines.Line2D at 0x236e5476630>]



(14) Propose a thredholding mechanism based on the mean value of the second eigenvertor

In [96]:
```
# mean value of the second eigenvector named "vec"
vec=res[1][1]
mean=np.mean(vec)
print(mean)
threshold=mean
print(vec)
group1 = []
group2 = []
for i in range(len(vec)):
    if vec[i]<threshold:
        group1.append(i)
    else:
        group2.append(i)
```

```
0.094803769799918
[ 0.    0.    0.    0.5   0.71 -0.5  -0.04]
```

(15) Draw the clustering

In [99]:
```
for i in group1:
    plt.plot(i,vec[i],'bo')
for i in group2:
    plt.plot(i,vec[i],'ro')
```