Code :-

```
import math
n = int (input('Enter the numbers of the symbols:-'))
a = dict(input("Enter symbols and Probabilities: ").split() for _ in
range(n))
for k, v in a.items():
   a[k] = float(v)
print('our Input sequence is', a)
sequence = a.copy()
class shano_fanno:
   def check(self,eg):
      assert (sum(eg.values())) == 1.0
      if (len(eg) == 2):
         return dict(zip(eg.keys(), ['0', '1']))
      return eg
   def descendingorder(self,eg):
      sorted_p = sorted(eg.items(), key=lambda x: x[1],
reverse=True)
      return sorted_p
   def spilt(self,x):
      diff = []
      diff_values = []
      for i in range(len(x)):
         res1 = dict(list(x.items())[i:])
         res2 = dict(list(x.items())[:i])
         def compare(a, b):
            s = round(sum(a.values()) - sum(b.values()), 2)
            diff.append(abs(s))
            diff_values.append(abs(s))
         compare(res1, res2)
         y = []
         y.append(list(res2.items()))
         y.append(list(res1.items()))
         diff.append(y)
      def high_prob(diff, min_diff, diff_values):
         diff1 = diff.copy()
         index = [i for i, x in enumerate(diff_values) if x ==
min_diff]
         if len(index) >= 2:
            pro = []
            mult_lst = [i * 2 for i in index]
            for ii in range(len(mult_lst)):
               aa = mult_lst[ii]
               pos1 = aa + 1
               yy = diff1[pos1]
               pro.append(yy)
            flatList = [item for elem in pro for item in elem] #
flattening the nested lists
            a1 = dict(flatList[0])
            a2 = dict(flatList[2])
            s1 = round(sum(a1.values()), 2)
            s2 = round(sum(a2.values()), 2)
            if s1 > s2:
               pos = mult_lst[0]
               last = diff.pop(pos + 1)
               print(last, 'last')
               return last
            else:
               pos = mult_lst[1]
               last = diff.pop(pos + 1)
               print(last, 'last')
               return last

         else:
            pos = diff.index(min_diff)
            last = diff.pop(pos + 1)
            return last

      min_diff = min(diff_values)
      last = high_prob(diff, min_diff, diff_values)
      res1, res2 = map(list, zip(last))
      res1 = res1.pop()
      res2 = res2.pop()

      def covert(a):

         if len(a) >= 2:
            return dict(a)
         elif len(a) == 1:
            return dict(a)

         elif len(a) == 0:

            raise Exception('there is no number')

      a = covert(res1)
      b = covert(res2)

      return a, b

   def shano(self,a, b):
      final = []

      def codes(a, b):
         a_copy = a.copy()
         b_copy = b.copy()
         for k, v in a.items():
            a[k] = str('0')
         for k, v in b.items():
            b[k] = str('1')

         newdict = a | b

         for k, v in newdict.items():
            newdict[k] = list(v)

         x1 = newdict
         final.append(x1)
         return a_copy, b_copy

      a_copy, b_copy = codes(a, b)

      def checklength(a_copy, b_copy):

         if len(a_copy) == 1:

            x1 = b_copy
            a, b = obj.spilt(x1)

            a_copy, b_copy = codes(a, b)

            if len(a_copy) + len(b_copy) > 2:
               return checklength(a_copy, b_copy)
            else:
               return final


         elif len(b_copy) == 1:

            x1 = a_copy
            a, b = obj.spilt(x1)
            a_copy, b_copy = codes(a, b)
            if len(a_copy) + len(b_copy) > 2:
               return checklength(a_copy, b_copy)
            else:
               return final

         elif len(a_copy) > 1 and len(b_copy) > 1:
            x1 = [a_copy, b_copy]
            while True:
               if type(x1) is list:
```

```
                while True:
                    x = dict(x1[0])
                    c, d = obj.spilt(x)
                    c1, c2 = codes(c, d)

                    if len(c1) + len(c2) > 2:
                        return checklength(c1, c2)
                    while True:
                        y = dict(x1[1])
                        a1, b1 = obj.spilt(y)
                        ca1, ca2 = codes(a1, b1)
                        if len(ca1) + len(ca2) > 2:
                            return checklength(ca1, ca2)
                        elif len(ca1) + len(ca2) == 2 :
                            return final
                        else:
                            break
                    break
                break
            return final

        elif len(a_copy) < 1 and len(b_copy) < 1:
            return final

        elif len(a_copy) == 1 and len(b_copy) == 1:
            return final

    if len(a_copy) + len(b_copy) >2:
        return checklength(a_copy, b_copy)
    else:
        return final


def merging(final):
    res = {}
    for dict in final:
        for list in dict:
            if list in res:
                res[list] += (dict[list])
            else:
                res[list] = dict[list]
    return res


def entropy(eg):
    x = []
    for k, v in eg.items():
        eg[k] = float(v)
        x.append(eg[k] * math.log2(1 / eg[k]))
    entropy = sum(x)
    return round(entropy,3)

def codelength(code,prob):
    x=[]
    y=[]
    z=[]
    for k, v in code.items():
        code[k] = len(v)
        x.append(code[k])
    for k, v in prob.items():
        prob[k] = float(v)
        y.append(prob[k])
    for i in range(0,len(code)):
        z.append(x[i]*y[i])
    length = sum(z)
    return float(round(length,3))

def code_rate(u,v):
    eff = u/v
    return round(eff, 3)
```

```
def display(dict1):
    print("{:<5} {}".format('SYMBOLS', 'CODES'))
    for key, value in dict1.items():
        symbols = key
        codes = value
        print("{:<5} {}".format(symbols, codes))


obj = shano_fanno()
descend = dict(obj.descendingorder(a))
print('Descending order of the sequence :-\n', descend)
x = obj.check(descend)
a, b = obj.spilt(x)
a1 = obj.shano(a, b)
shannon_code = merging(a1)
print('The shannon-fano coding :-')
display(shannon_code)
ent = entropy(sequence)
print('The entropy = ', ent)
L = codelength(shannon_code, descend)
print('The Average code length =', L)
rate = code_rate(ent, L)
print('The code rate is', rate)
print('The efficiency is', rate*100, '%')
print('The code redundancy is', round((1-rate)*100, 3), '%')
print('Made by Varad Patil 120A2036')
```

```
Enter the numbers of the symbols:-4
Enter symbols and Probabilities: a 0.3
Enter symbols and Probabilities: b 0.2
Enter symbols and Probabilities: c 0.2
Enter symbols and Probabilities: d 0.3
our Input sequence is {'a': 0.3, 'b': 0.2, 'c': 0.2, 'd': 0.3}
Descending order of the sequence :-
 {'a': 0.3, 'd': 0.3, 'b': 0.2, 'c': 0.2}
The shannon-fano coding :-
SYMBOLS CODES
a     ['0', '0']
d     ['0', '1']
b     ['1', '0']
c     ['1', '1']
The entropy =  1.971
The Average code length = 2.0
The code rate is 0.986
The efficiency is 98.6 %
The code redundancy is 1.4 %
Made by Varad Patil 120A2036

Process finished with exit code 0
```