```
1  class Arista:
2
3      def __init__(self, src, dst, weight = 1):
4          self.src = src
5          self.dst = dst
6          self.weight = weight
7
8      def getSource(self):
9          return self.src
10
11     def setSource(self, new):
12         self.src = new
13
14     def getDest(self):
15         return self.dst
16
17     def setDest(self, new):
18         self.dst = new
19
20     def getWeight(self):
21         return self.weight
```

```
1  from random import *
2
3  def generateInstance(path, cantidadVertices):
4      file = open(path, "w")
5      file.write(str(cantidadVertices) + "\n")
6      file.write(str(cantidadVertices * 2) + "\n")
7      aristas = []
8      noConectados = [ x for x in range(cantidadVertices)]
9      conectados = []
10
11     nodoInicio = choice(noConectados)
12     conectados.append(nodoInicio)
13     noConectados.remove(nodoInicio)
14     while len(noConectados) != 0:
15         src = choice(noConectados)
16         noConectados.remove(src)
17         dst = choice(conectados)
18         line = [src, dst]
19         aristas.append(line)
20         file.write(" ".join([str(x) for x in line]) + "\n")
21
22     while len(aristas) < cantidadVertices * 2:
23         while True:
24             src = randint(0, cantidadVertices - 1)
25             dst = randint(0, cantidadVertices - 1)
26             while src == dst:
27                 dst = randint(0, cantidadVertices - 1)
28             if [src, dst] not in aristas and [dst, src] not in aristas:
29                 break
30         line = [src, dst]
31         aristas.append(line)
32         file.write(" ".join([str(x) for x in line]) + "\n")
33
34     file.close()
35
```

```python
#!/usr/bin/env python
import random


def ejercicio_tres_punto_uno(n):
    array = []
    for i in range(0, n):
        array.append(random.randint(0, 10000))
    t = random.randint(0, 10000)
    return  {
        "array": array,
        "t": t
    }

def ejercicio_tres_punto_dos(array, t, e)
    n = len(array)
    Lists =[[0]]
    for i in range(1, n):
        auxList = merge_lists(Lists[i-1], [(x + array[i]) for x in Lists[i-1]])
        auxList = trim_list(auxList, e)
        Lists[i] = filter(lambda x: x < t, auxList)
    return max(Lists[n])

def merge_lists(first_list, second_list):
    return sort(first_list + list(set(second_list) - set(first_list)))

def trim_list(lista, e):
    m = len(lista)
    newList = [lista[0]]
    last = lista[m]
    for i in range(2, m):
        if lista[i] > (last * (1 + e)):
            newList.append(lista[i])
            last = lista[i]
    return newList
```

```
1  import random, sys
2
3  def generateInstance(path, days):
4      file = open(path, "w")
5      file.write(str(days) + "\n")
6      for i in xrange(days):
7          rand = random.randint(0, sys.maxint)
8          file.write(str(rand) + "\n")
9      file.close()
10 generateInstance("pru5.txt", 100)
```

```
1  from nodo import Nodo
2  from arista import Arista
3  from random import *
4
5  class Grafo:
6
7      def __init__(self, cantidadAristas):
8          self.vertices = [Nodo(str(x)) for x in range(cantidadAristas)]
9          self.aristas = []
10
11     def cantidadVertices(self):
12         return len(self.vertices)
13
14     def cantidadAristas(self):
15         return len(self.aristas)
16
17     def agregarArista(self, nodo1, nodo2, peso = 1):
18         arista = Arista(self.vertices[nodo1], self.vertices[nodo2], peso)
19         self.aristas.append(arista)
20
21     def contraerNodo(self, nodoSrc, nodoDst):
22         newNode = Nodo.merge(nodoSrc, nodoDst)
23         self.vertices.append(newNode)
24         self.vertices.remove(nodoSrc)
25         self.vertices.remove(nodoDst)
26
27         aristasAEliminar = []
28         for arista in self.aristas:
29             if arista.getSource() == nodoDst:
30                 arista.setSource(newNode)
31             if arista.getSource() == nodoSrc:
32                 arista.setSource(newNode)
33             if arista.getDest() == nodoDst:
34                 arista.setDest(newNode)
35             if arista.getDest() == nodoSrc:
36                 arista.setDest(newNode)
37             if arista.getSource() == arista.getDest():
38                 aristasAEliminar.append(arista)
39         self.aristas = [arista for arista in self.aristas if not arista in arist
   asAEliminar]
40
41     def getRandomEdge(self):
42         if len(self.aristas) == 0:
43             print "Se acabaron las aristas y hay %i vertices" %(self.cantidadVertices())
44         return choice(self.aristas)
45
46     def getVertice(self, nodo):
47         return self.vertices[nodo]
48
```

```
1  from readGraph import *
2  import random
3  from math import *
4
5  class Karger(object):
6
7      def __init__(self, path):
8          graph = createGraph(path, True)
9          self.minCut = (None, None, float("inf"))
10         iteraciones = factorial(graph.cantidadVertices()) / (factorial(2) * fact
   orial(graph.cantidadVertices() - 2)) * int(ceil(log(graph.cantidadVertices())))
11         for i in range(iteraciones):
12             # print "Iteracion %d/%d" %(i, iteraciones)
13             while graph.cantidadVertices() > 2:
14                 edge = graph.getRandomEdge()
15                 graph.contraerNodo(edge.getSource(), edge.getDest())
16             if graph.cantidadAristas() < self.getMinCut()[2]:
17                 self.minCut = (graph.getVertice(0).getId(), graph.getVertice(1).
   getId(), graph.cantidadAristas())
18
19     def getMinCut(self):
20         return self.minCut
21
```

```
1  class Nodo(object):
2
3      def __init__(self, id):
4          self.aristas = []
5          self.id = id
6
7      def addArista(self, arista):
8          self.aristas.append(arista)
9
10     def removeArista(self, arista):
11         self.aristas.remove(arista)
12
13     def addAristas(self, aristas):
14         self.aristas += aristas
15
16     def getId(self):
17         return self.id
18
19     def getAristas(self):
20         return self.aristas
21
22     @staticmethod
23     def merge(nodo1, nodo2):
24         newNode = Nodo(nodo1.getId() + "|" + nodo2.getId())
25         newNode.addAristas(nodo1.getAristas() + nodo2.getAristas())
26         return newNode
```

```
1  from grafo import *
2
3  def createGraph(path, dirigido):
4      file = open(path, "r")
5      cantidadVertices = file.readline()
6      grafo = Grafo(int(cantidadVertices))
7      cantidadAristas = int(file.readline())
8      for x in range(cantidadAristas):
9          linea = file.readline()
10         nodos = [int(x) for x in linea.split(" ")]
11         grafo.agregarArista(nodos[0], nodos[1])
12         if(not dirigido):
13             grafo.agregarArista(nodos[1], nodos[0])
14     return grafo
15
16 def createTransposeGraph(path, dirigido):
17     file = open(path, "r")
18     cantidadVertices = file.readline()
19     grafo = Grafo(int(cantidadVertices))
20     cantidadAristas = int(file.readline())
21     for x in range(cantidadAristas):
22         linea = file.readline()
23         nodos = [int(x) for x in linea.split(" ")]
24         grafo.agregarArista(nodos[1], nodos[0], nodos[2])
25         if(not dirigido):
26             grafo.agregarArista(nodos[0], nodos[1], nodos[2])
27     return grafo
```

```
1  from karger import Karger
2  from createInstanceKarger import generateInstance
3  from time import time
4
5  def run():
6      for i in range(1, 6):
7          cantidadVertices = i*1000
8          print "Creo la instancia de %d" %(cantidadVertices)
9          generateInstance("test" + str(i) + ".txt", cantidadVertices)
10         print "Empiezo el algoritmo"
11         inicio = time()
12         karg = Karger("test" + str(i) + ".txt")
13         fin = time()
14         print karg.getMinCut()[2]
15         print "Termine la ejecucion en %d segundos" %(fin - inicio)
16
17
```

```
1  import sys
2
3  class SellActions(object):
4      """docstring for SellAction"""
5      def __init__(self,path):
6          self.dateToBuyActual = -1
7          self.dateToSell = -1
8          self.dateToBuyFollowing = -1
9          self.path = path
10         file = open(self.path, "r")
11         self.days = int(file.readline())
12         file.close()
13         self.valueActions = [None] * self.days
14         self.benefits = [None] * self.days
15
16     def initialize(self):
17         if (self.valueActions[0]<self.valueActions[1]):
18             self.dateToBuyActual = 0
19             self.dateToSell = 1
20         else:
21             self.dateToBuyActual = 1
22             self.dateToSell = 1
23         self.benefits[0] = 0
24         self.benefits[1] = self.dateToBuyActual - self.dateToSell
25
26     def setValueAction(self, day, value):
27         self.valueActions[day] = value
28
29     def showSellAcctions(self):
30         print self.valueActions
31
32     def setBenefit(self, day):
33         newValue = self.valueActions[day]
34         valueNewisGreaterThanValueToSell = self.valueActions[self.dateToSell] <
   newValue
35         valueNewisLessThanValueToBuyActual = newValue < self.valueActions[self.d
   ateToBuyActual]
36         if (valueNewisGreaterThanValueToSell):
37             self.dateToSell = day
38             self.benefits[day] = newValue - self.valueActions[self.dateToBuyActu
   al]
39             existDateToBuyFollowing = self.dateToBuyFollowing != -1
40             if (existDateToBuyFollowing):
41                 benefitWithBuyFollowing = (self.valueActions[self.dateToSell] -
   self.valueActions[self.dateToBuyFollowing])
42                 existBenefitGreaterThanBefore = self.benefits[day] < benefitWith
   BuyFollowing
43                 if (existBenefitGreaterThanBefore):
44                     self.benefits[day] = benefitWithBuyFollowing
45                     self.dateToBuyActual = self.dateToBuyFollowing
46                     self.dateToBuyFollowing = -1
47         elif (valueNewisLessThanValueToBuyActual):
48             self.dateToBuyFollowing = day
49             self.benefits[day] = self.benefits[day-1]
50         else:
51             self.benefits[day] = self.benefits[day-1]
52
53     def findTheDaysToBuyAndSellAcctions(self):
54         file = open(self.path, "r")
55         self.days = int(file.readline())
56         for self.day in xrange(0, self.days):
57             value = int(file.readline())
58             self.setValueAction(self.day, value)
59             if (self.day == 1):
60                 self.initialize()
61             elif (self.day > 1):
62                 self.setBenefit(self.day)
63         file.close()
64         print "Day to buy is " + str(self.dateToBuyActual)
65         print "Day to sell is " + str(self.dateToSell)
```

```
66              print "The benefit is " + str(self.benefits[self.days-1])
```

```
1  from testUtils import *
2  from sellActions import *
3  from generateInstanceToSellActions import *
4  from time import time
5
6  def run():
7      sa = SellActions("test.txt")
8      sa.findTheDaysToBuyAndSellAcctions()
9      test("Day to buy = 6", sa.dateToBuyActual, 6)
10     test("Day to sell = 7", sa.dateToSell, 7)
11     test("The benefit is = 100", sa.benefits[len(sa.benefits)-1], 100)
12
13     sa = SellActions("test2.txt")
14     sa.findTheDaysToBuyAndSellAcctions()
15     test("Day to buy = 1", sa.dateToBuyActual, 1)
16     test("Day to sell = 2", sa.dateToSell, 2)
17     test("The benefit is = 4", sa.benefits[len(sa.benefits)-1], 4)
18
19 def timeTests():
20     """
21   print "Generate instances to buying and selling"
22   generateInstance("testSA100.txt", 100)
23   print "Find the optimal days to buying and selling"
24   inicio = time()
25   sa = SellActions("testSA100.txt")
26   sa.findTheDaysToBuyAndSellAcctions()
27   fin = time()
28   print "Execution time with 100 days: %f" %(fin – inicio)
29
30   print "Generate instances to buying and selling"
31   generateInstance("testSA1000.txt", 1000)
32   print "Find the optimal days to buying and selling"
33   inicio = time()
34   sa = SellActions("testSA1000.txt")
35   sa.findTheDaysToBuyAndSellAcctions()
36   fin = time()
37   print "Execution time with 1000 days: %f" %(fin – inicio)
38
39   print "Generate instances to buying and selling"
40   generateInstance("testSA10000.txt", 10000)
41   print "Find the optimal days to buying and selling"
42   inicio = time()
43   sa = SellActions("testSA10000.txt")
44   sa.findTheDaysToBuyAndSellAcctions()
45   fin = time()
46   print "Execution time with 10000 days: %f" %(fin – inicio)
47
48   print "Generate instances to buying and selling"
49   generateInstance("testSA100000.txt", 100000)
50   print "Find the optimal days to buying and selling"
51   inicio = time()
52   sa = SellActions("testSA100000.txt")
53   sa.findTheDaysToBuyAndSellAcctions()
54   fin = time()
55   print "Execution time with 100000 days: %f" %(fin – inicio)
56
57   print "Generate instances to buying and selling"
58   generateInstance("testSA1000000.txt", 1000000)
59   print "Find the optimal days to buying and selling"
60   inicio = time()
61   sa = SellActions("testSA1000000.txt")
62   sa.findTheDaysToBuyAndSellAcctions()
63   fin = time()
64   print "Execution time with 1000000 days: %f" %(fin – inicio)
65
66   print "Generate instances to buying and selling"
67   generateInstance("testSA10000000.txt", 10000000)
68   print "Find the optimal days to buying and selling"
69   inicio = time()
70   sa = SellActions("testSA10000000.txt")
```

```
71      sa.findTheDaysToBuyAndSellAcctions()
72      fin = time()
73      print "Execution time with 10000000 days: %f" %(fin − inicio)
74  """
75  run()
76  timeTests()
```

```
1  def test(name, result, expected):
2      print "TEST %s: %s" %(name, "OK" if result == expected else str(result))
```

Table of Contents