File: ./com/fiuba/algoritmos/InvalidParameterException.java

```
1  package com.fiuba.algoritmos;
2
3  /**
4   * Created by Nico on 19/6/2017.
5   */
6  public class InvalidParameterException extends RuntimeException {
7
8      private String razon;
9
10      InvalidParameterException(String razon) {
11          super();
12          this.razon = razon;
13      }
14  }
15
```

File: ./com/fiuba/algoritmos/Karger.java

```
1  package com.fiuba.algoritmos;
2
3  import com.fiuba.grafos.Arista;
4  import com.fiuba.grafos.Grafo;
5
6  import java.util.List;
7
8  public class Karger {
9
10      /**
11       * Aplica el Algoritmo de Karger para obtener el corte mínimo
12       * en un grafo conectado con una mejor probabilidad de éxito.
13       * - La cantidad de iteraciones de este método es de N = [n^2 *
(n- 1)]/2
14       */
15      public List<Arista> getCorteMinimo(Grafo grafo) {
16          int cantidadDeRepeticiones =
this.getCantidadDeRepeticiones(grafo);
17          List<Arista> corteMinimo = null;
18          for (int i = 0; i < cantidadDeRepeticiones; i++) {
19              List<Arista> nuevoCorteMinimo =
this.adivinarCorteMinimo(grafo);
20
21              if (corteMinimo == null || corteMinimo.size() >
nuevoCorteMinimo.size()) {
22                  corteMinimo = nuevoCorteMinimo;
23              }
24          }
25          return corteMinimo;
26      }
27
28      public List<Arista> adivinarCorteMinimo(Grafo grafo) {
29          Grafo grafoActual = grafo;
30          while (grafoActual.cantidadDeVertices() > 2) {
31              // Obtengo una arista de manera aleatoria -> {u,v}
32              Arista aristaAleatoria = grafo.getAristaAleatoria();
33
```

```
34              // Creo un nuevo grafo G2 en base a las aristas del
anterior
35              // pero con los siguientes cambios:
36              // - Un vértice menos --> v2 = v1 -1
37              // - Eliminando las aristas que vayan de 'u' a 'v' ->
{u,v}
38              // - Reemplazando los ejes de la forma {u,w} o {v,w} en
{uv, w}
39              Grafo nuevoGrafo = new
Grafo(grafoActual.cantidadDeVertices() - 1);
40              String verticeUnificado = aristaAleatoria.getInicio() +
aristaAleatoria.getFin();
41
42              grafoActual.getAristas().forEach(arista -> {
43
44                  // Agrego las aristas que no tienen que ver con la
nueva
45                  if
(!aristaAleatoria.getInicio().equals(arista.getInicio())
46                          &&
!aristaAleatoria.getInicio().equals(arista.getFin())
47                          &&
!aristaAleatoria.getFin().equals(arista.getInicio())
48                          &&
!aristaAleatoria.getFin().equals(arista.getFin())) {
49                          nuevoGrafo.agregarArista(arista.getInicio(),
arista.getFin());
50                          return;
51                  }
52
53                  // Reemplazando los ejes de la forma {u,w} o {v,w}
en {uv, w}
54                  if
(arista.getInicio().equals(aristaAleatoria.getInicio()) ||
arista.getInicio().equals(aristaAleatoria.getFin())) {
55                          nuevoGrafo.agregarArista(verticeUnificado,
arista.getFin());
56                          return;
57                  }
58
59                  // Reemplazando los ejes de la forma {w, u} o {w,v}
en {w, uv}
60                  if
(arista.getFin().equals(aristaAleatoria.getInicio()) ||
arista.getFin().equals(aristaAleatoria.getFin())) {
61                          nuevoGrafo.agregarArista(arista.getInicio(),
verticeUnificado);
62                          return;
63                  }
64              });
65
66              grafoActual = nuevoGrafo;
67          }
68          return grafoActual.getAristas();
69      }
70
71      private int getCantidadDeRepeticiones(Grafo grafo) {
72          double n = (double)grafo.cantidadDeVertices();
73          return (int) Math.ceil((Math.pow(n, 2) * (n-(double)1)) /
(double)2);
74      }
```

```
75  }
```

File: ./com/fiuba/algoritmos/MejorDiaAcciones.java

```
1   package com.fiuba.algoritmos;
2
3   /**
4    * Created by Nico on 19/6/2017.
5    */
6   public class MejorDiaAcciones {
7
8       private int diaCompra = 0;
9       private int diaVenta = 0;
10       private int montoGanancia = 0;
11
12       public MejorDiaAcciones(int[] valorDia) {
13
14           int posibleDiaCompraFuturo = 0;
15           if (valorDia.length < 1) {
16               throw new InvalidParameterException("Longitud de Dias
debe ser mayor a 0");
17           }
18           //Bucle FOR ejecuta n iteraciones, las operaciones internas
son todas O(1) => Bucle es O(n)
19           for (int i = 0; i < valorDia.length; i++) {
20               if (valorDia[i] < valorDia[posibleDiaCompraFuturo]) {
//Como el valor es menor al posible candidato a canjear, este se vuelve
un mejor candidato y lo reemplaza.
21                   posibleDiaCompraFuturo = i;
22               } else { //Si gano valor de venta o si el candidato me
hace mejorar la ganancia entonces conmuto los dias
23                   if (valorDia[i] > valorDia[diaVenta] ||
ganancia(posibleDiaCompraFuturo, i, valorDia) > montoGanancia) {
24                       diaCompra = posibleDiaCompraFuturo;
25                       diaVenta = i;
26                       montoGanancia = valorDia[diaVenta] -
valorDia[diaCompra];
27                   }
28               }
29           }
30       }
31
32       private int ganancia(int diaCompra, int diaVenta, int[]
valorDia) {
33           return valorDia[diaVenta] - valorDia[diaCompra];
34       }
35
36       public int getDiaCompra() {
37           return diaCompra;
38       }
39
40       public int getDiaVenta() {
41           return diaVenta;
42       }
43
44       public int getMontoGanancia() {
45           return montoGanancia;
46       }
```

```
47  }
```

File: ./com/fiuba/algoritmos/ResultadoSubsetSum.java

```
1  package com.fiuba.algoritmos;
2
3
4  public class ResultadoSubsetSum {
5      private int exactSubsetSum;
6      private int aproxSubsetSum;
7
8      public ResultadoSubsetSum(int exact, int aprox) {
9          this.exactSubsetSum = exact;
10          this.aproxSubsetSum = aprox;
11     }
12
13     public int getExactSubsetSum() { return this.exactSubsetSum; }
14     public int getAproxSubsetSum() { return this.aproxSubsetSum; }
15 }
```

File: ./com/fiuba/algoritmos/SubSetAprox.java

```
1  package com.fiuba.algoritmos;
2
3  import java.util.ArrayList;
4  import java.util.Collections;
5  import java.util.Iterator;
6  import java.util.Random;
7
8  public class SubSetAprox {
9
10     private int n;
11     private int t;
12     private ArrayList<Integer> S;
13     private float e;
14     private Random rand = new Random();
15
16     private int exactSubsetSum;
17     private int aproxSubsetSum;
18
19     public ResultadoSubsetSum getResultadoSubset(int n, int t) {
20         this.n = n;
21         this.t = t;
22         this.e=(float)0.40;
23
24         // Genero el conjunto S
25         this.S=this.genS(n);
26
27         this.exactSubsetSum = this.exactSubsetSum(S, t);
28         this.aproxSubsetSum = this.aproxSubsetSum(S, t, e);
29
30         return new ResultadoSubsetSum(exactSubsetSum,
aproxSubsetSum);
31     }
32
```

```
33      /*
34       * Genera una lista aleatoria de enteros positivos sin
duplicados
35       */
36      private  ArrayList<Integer> genS(int n){
37          this.n=n;
38          S= new ArrayList<>();
39          while (S.size() < n) {
40              int random = rand.nextInt(n+11);
41
42              // Contains tiene complejidad O(n)
43              if (!S.contains(random)) {
44                  S.add(random);
45
46              }
47          }
48
49          return S;
50      }
51
52      /*
53       *  Retorna una lista que será la union de l1 y l2 ordenandola
54       *  y deshaciendose de duplicados.
55       */
56      private ArrayList mergeList(ArrayList<Integer>
L1,ArrayList<Integer> L2 ){
57          Collections.sort(L2);
58          Collections.sort(L1);
59          ArrayList<Integer> L1c=L1;
60          L1c.removeAll(L2);
61          L1c.addAll(L2);
62          Collections.sort(L1c);
63          return L1c;
64      }
65
66      /*
67       * Suma un entero a cada elemento de una lista
68       */
69      public ArrayList<Integer> sumList(ArrayList<Integer> List, int
a){
70          ArrayList<Integer> sList=new ArrayList<>();
71          for (int i = 0; i < List.size(); i++) {
72              sList.add(List.get(i)+a);
73          }
74          return sList;
75      }
76
77      /*
78       * Implementación del algoritmo exactSubsetSum del libro,
79       * haciendo uso de mergelist y sumlist.
80       */
81      private int exactSubsetSum(ArrayList<Integer> S, int t){
82          this.S=S;
83          this.t=t;
84          int n= S.size();
85
86          String str="";
87          ArrayList<ArrayList<Integer>> L;
88          L=new ArrayList<ArrayList<Integer>>();
89          ArrayList<Integer> Laux=new ArrayList<Integer>();
90          Laux.add(0);
```

```
91          L.add(Laux);
92
93          for (int i = 1; i <= n; i++) {
94
95              L.add(mergeList(L.get(i-1), sumList(L.get(i-1), S.get(i-
1))));
96              Iterator<Integer> it = L.get(i-1).iterator();
97
98              // Nos deshacemos de las sumas mayores a t
99              while (it.hasNext()) {
100                  Integer integer = it.next();
101                  if (integer >t) {
102                      it.remove();
103                  }
104              }
105          }
106
107          return Collections.max(L.get(L.size()-1));
108      }
109
110      /*
111       * Recorta la lista suministrada de acuerdo a un parametro de
aproximación
112       */
113      private ArrayList<Integer> trimList(ArrayList<Integer>
long_list, float p){
114
115          int m=long_list.size();
116          ArrayList<Integer> t_list= new ArrayList<Integer>();
117
118          t_list.add(long_list.get(0));
119          int last= long_list.get(0);
120          for (int i = 2; i <= m; i++) {
121              if(long_list.get(i-1)>=(last*(1+p))){
122                  t_list.add(long_list.get(i-1));
123                  last=long_list.get(i-1);
124              }
125          }
126          return t_list;
127      }
128
129      /*
130       * Realiza el mismo procedimiento de exactSubsetSum, pero
implementa
131       * la funcion trim que recorta la lista deacuerdo a un
parametro de aproximación.
132       */
133      private int aproxSubsetSum(ArrayList<Integer> S, int t, float
e){
134          this.S=S;
135          this.e=e;
136          this.t=t;
137          int n= S.size();
138
139          ArrayList<ArrayList<Integer>> L;
140          L=new ArrayList<>();
141          ArrayList<Integer> Laux = new ArrayList<Integer>();
142          Laux.add(0);
143          L.add(Laux);
144
145          for (int i = 1; i <= n; i++) {
```

```
146
147             L.add(mergeList(L.get(i-1), sumList(L.get(i-1),
S.get(i-1))));
148
149             // Nos deshacemos de las sumas mayores a t
150             Iterator<Integer> it = L.get(i-1).iterator();
151             while (it.hasNext()) {
152                 Integer integer = it.next();
153                 if (integer >t) {
154                     it.remove();
155                 }
156             }
157
158             L.set(i, trimList(L.get(i), e/(2*n)));
159         }
160         return Collections.max(L.get(L.size()-1));
161     }
162
163     private String arrToString(ArrayList<Integer> a){
164         String str="";
165         for (int i = 0; i <= a.size()-1; i++) {
166             str+= a.get(i).toString() + " ";
167         }
168         return str;
169     }
170 }
```

File: ./com/fiuba/grafos/Arista.java

```
1  package com.fiuba.grafos;
2
3  /**
4   * Created by gatti2602 on 09/04/17.
5   * Arista Dirigida.
6   * Contiene campos src y dst inmutables al crear la arista.
7   */
8  public class Arista {
9
10     private String inicio, fin;
11     private int peso;
12
13     Arista(String inicio, String fin, int peso) {
14         this.inicio = inicio;
15         this.fin = fin;
16         this.peso = peso;
17     }
18
19     public String getInicio() {
20         return this.inicio;
21     }
22
23     public String getFin() {
24         return this.fin;
25     }
26
27     public Integer getPeso() {
28         return this.peso;
29     }
```

```
30
31      @Override
32      public String toString() {
33          return "{" + this.inicio + ", " + this.fin + "}";
34      }
35  }
```

File: ./com/fiuba/grafos/GeneradorDeGrafos.java

```
1   package com.fiuba.grafos;
2
3   import java.util.Random;
4
5   public class GeneradorDeGrafos {
6
7       /*
8        * Dados un parámetros n genere un grafo conexo no dirigido de n
vértices y 2n aristas.
9        */
10      public static Grafo generarMultigrafoConexo(int
numeroDeVertices) {
11          Grafo grafo = new Grafo(numeroDeVertices);
12          int numeroDeAristas = 2 * numeroDeVertices;
13
14          // Primero convertimos el grafo en conexo
15          // (existe un camino entre cualquier par de nodos)
16          // (conecto a todos los vértices entre si)
17          for (Integer i = 0; i < numeroDeVertices - 1; i++) {
18              grafo.agregarArista(i.toString(), new Integer(i +
1).toString());
19              numeroDeAristas--;
20          }
21
22          // Luego agregamos aristas de manera random
23          // hasta llegar a las 2*n aristas
24          Random random = new Random();
25          while (numeroDeAristas > 0) {
26              Integer nodoInicio = random.nextInt(numeroDeVertices);
27              Integer nodoFin = random.nextInt(numeroDeVertices);
28              grafo.agregarArista(nodoInicio.toString(),
nodoFin.toString());
29              numeroDeAristas--;
30          }
31
32          return grafo;
33      }
34  }
```

File: ./com/fiuba/grafos/Grafo.java

```
1   package com.fiuba.grafos;
2
3   import java.util.ArrayList;
4   import java.util.List;
5   import java.util.Random;
```

```
6
7  /**
8   * Grafo con las siguientes características:
9   * - No dirigido
10   * - No pesado
11   * - Multigrafo (múltiples aristas paralelas)
12   */
13  public class Grafo {
14
15      /**
16       * Cada elemento contiene la lista de arsitas del vertice
17       * referenciado
18       */
19      private ArrayList<Arista> aristas;
20      private int cantidadDeVertices;
21
22      public Grafo(int vertices) {
23          this.aristas = new ArrayList<>();
24          this.cantidadDeVertices = vertices;
25      }
26
27      public Integer cantidadDeAristas() {
28          return this.aristas.size();
29      }
30
31
32      public Integer cantidadDeVertices() {
33          return cantidadDeVertices;
34      }
35
36      public List<Arista> getAristas() {
37          return this.aristas;
38      }
39
40      /**
41       * Devuelve una arista del grafo de manera aleatoria
42       */
43      public Arista getAristaAleatoria() {
44          Integer numeroArista = new
Random().nextInt(this.cantidadDeVertices);
45          return this.aristas.get(numeroArista);
46      }
47
48      /**
49       * Agrega una Arista al grafo.
50       */
51      public void agregarArista(String nodoInicio, String nodoFin) {
52          Arista nuevaArista = new Arista(nodoInicio, nodoFin, 0);
53          this.aristas.add(nuevaArista);
54      }
55  }
```

File: ./com/fiuba/Main.java

```
1  package com.fiuba;
2
3  import com.fiuba.algoritmos.Karger;
4  import com.fiuba.algoritmos.MejorDiaAcciones;
```

```java
import com.fiuba.algoritmos.ResultadoSubsetSum;
import com.fiuba.algoritmos.SubSetAprox;
import com.fiuba.grafos.Arista;
import com.fiuba.grafos.GeneradorDeGrafos;
import com.fiuba.grafos.Grafo;

import java.util.Arrays;
import java.util.List;
import java.util.Random;
import java.util.Scanner;
import java.util.concurrent.TimeUnit;

public class Main {

    public static void main(String[] args) {

        System.out.println("-------------------------------------");
        System.out.println("\uD83D\uDE80 \uD83D\uDE80 TP3 Teoría de Algoritmos \uD83D\uDE80 \uD83D\uDE80");
        System.out.println("-------------------------------------");
        System.out.println();

        int algoritmoElegido = elegirAlgoritmo();
        System.out.println();

        switch (algoritmoElegido) {
            case 1:
                System.out.println("SubsetAprox");
                System.out.println("-----------");
                resolveSubsetAprox();
                break;
            case 2:
                System.out.println("Karger");
                System.out.println("------");
                resolveKarger();
                break;
            case 3:
            default:
                System.out.println("Acciones");
                System.out.println("--------");
                resolveAcciones();
                break;

        }

        System.out.println();
        System.out.println("\uD83D\uDC7E \uD83D\uDC7E Fin! \uD83D\uDC7E \uD83D\uDC7E");
    }

    private static void resolveAcciones() {
        System.out.println("Elige la cantidad de dias a simular");
        Scanner scanner = new Scanner(System.in);

        int dias = Integer.parseInt(scanner.nextLine());
        int[] valores = new int[dias];
        Random rand = new Random();

        for (int i = 0; i < dias; i++)
            valores[i] = rand.nextInt(1000);

```

```
64          System.out.println("Simulacion con " +
Integer.toString(dias) + " dias:");
65          System.out.println();
66          long tiempoDeInicio = System.nanoTime();
67          MejorDiaAcciones mejorDia = new MejorDiaAcciones(valores);
68          long tiempoDelAlgoritmo = System.nanoTime() -
tiempoDeInicio;
69          System.out.println("Mejor Dia de Compra: " +
mejorDia.getDiaCompra());
70          System.out.println("Mejor Dia de Venta: " +
mejorDia.getDiaVenta());
71          System.out.println("Ganancia: " +
mejorDia.getMontoGanancia());
72          System.out.println("Tiempo de Ejecucion: " +
TimeUnit.NANOSECONDS.toMillis(tiempoDelAlgoritmo) + " mSeg.");
73
74
75      }
76
77      public static int elegirAlgoritmo() {
78          System.out.println("Elige el algoritmo a resolver");
79          System.out.println();
80          System.out.println("1. SubsetAprox");
81          System.out.println("2. Karger");
82          System.out.println("3. Acciones");
83          System.out.println();
84
85          Scanner scanner = new Scanner(System.in);
86          return Integer.parseInt(scanner.nextLine());
87      }
88
89      public static void resolveSubsetAprox() {
90          Scanner scanner = new Scanner(System.in);
91
92          System.out.println("Ingrese el valor de n:");
93          System.out.println();
94          int n = Integer.parseInt(scanner.nextLine());
95          System.out.println();
96
97          System.out.println("Ingrese el valor de t:");
98          System.out.println();
99          int t = Integer.parseInt(scanner.nextLine());
100          System.out.println();
101
102          SubSetAprox algoritmo= new SubSetAprox();
103
104          long tiempoDeInicio = System.nanoTime();
105          ResultadoSubsetSum resultado =
algoritmo.getResultadoSubset(n, t);
106          long tiempoDelAlgoritmo = System.nanoTime() -
tiempoDeInicio;
107
108          System.out.println(
109              "SubsetAprox con n=" + n + " y t=" + t +
110              ". Resultado: (aprox) " + resultado.getAproxSubsetSum()
+ " (exacto) " + resultado.getExactSubsetSum() +
111              ". Tiempo de Ejecucion: " +
TimeUnit.NANOSECONDS.toMillis(tiempoDelAlgoritmo) + " mSeg."
112          );
113      }
114
```

```
115      public static void resolveKarger() {
116          System.out.println("Elige el número de vértices del
grafo:");
117          System.out.println();
118
119          Scanner scanner = new Scanner(System.in);
120          int numeroDeVertices =
Integer.parseInt(scanner.nextLine());
121          System.out.println();
122
123          Grafo grafo =
GeneradorDeGrafos.generarMultigrafoConexo(numeroDeVertices);
124
125          System.out.println("Generado un grafo con " +
grafo.cantidadDeVertices() + " vertices y " + grafo.cantidadDeAristas()
+ " aristas.");
126          System.out.println();
127
128          Karger algoritmo = new Karger();
129
130          long tiempoDeInicio = System.nanoTime();
131          List<Arista> corteMinimo = algoritmo.getCorteMinimo(grafo);
132          long tiempoDelAlgoritmo = System.nanoTime() -
tiempoDeInicio;
133
134          System.out.println(
135              "Karger con n=" + grafo.cantidadDeVertices() +
136              ". Resultado: " +
Arrays.toString(corteMinimo.toArray()) +
137              ". Tiempo de Ejecucion: " +
TimeUnit.NANOSECONDS.toMillis(tiempoDelAlgoritmo) + " mSeg."
138          );
139      }
140  }
```

File: ./TDATP3.iml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <module type="JAVA_MODULE" version="4">
3    <component name="NewModuleRootManager" inherit-compiler-
output="true">
4      <exclude-output />
5      <content url="file://$MODULE_DIR$">
6        <sourceFolder url="file://$MODULE_DIR$/src"
isTestSource="false" />
7      </content>
8      <orderEntry type="inheritedJdk" />
9      <orderEntry type="sourceFolder" forTests="false" />
10      <orderEntry type="module-library">
11        <library>
12          <CLASSES>
13            <root url="jar://$APPLICATION_HOME_DIR$/lib/groovy-all-
2.4.6.jar!/" />
14          </CLASSES>
15          <JAVADOC />
16          <SOURCES />
17        </library>
18      </orderEntry>
```

```
19      <orderEntry type="module-library">
20        <library name="JUnit4">
21          <CLASSES>
22            <root url="jar://$APPLICATION_HOME_DIR$/lib/junit-
4.12.jar!/" />
23            <root url="jar://$APPLICATION_HOME_DIR$/lib/hamcrest-core-
1.3.jar!/" />
24          </CLASSES>
25          <JAVADOC />
26          <SOURCES />
27        </library>
28      </orderEntry>
29    </component>
30  </module>
```

File: ./TP3.iml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <module type="JAVA_MODULE" version="4">
3    <component name="NewModuleRootManager" inherit-compiler-
output="true">
4      <exclude-output />
5      <content url="file://$MODULE_DIR$">
6        <sourceFolder url="file://$MODULE_DIR$/src"
isTestSource="false" />
7      </content>
8      <orderEntry type="inheritedJdk" />
9      <orderEntry type="sourceFolder" forTests="false" />
10      <orderEntry type="module-library">
11        <library name="JUnit4">
12          <CLASSES>
13            <root url="jar://$APPLICATION_HOME_DIR$/lib/junit-
4.12.jar!/" />
14            <root url="jar://$APPLICATION_HOME_DIR$/lib/hamcrest-core-
1.3.jar!/" />
15          </CLASSES>
16          <JAVADOC />
17          <SOURCES />
18        </library>
19      </orderEntry>
20    </component>
21  </module>
```