Algorítmica
Práctica #3

Algoritmos Voraces o Greedy

José María Gómez García
Fernando Lojano Mayaguari
Valentino Lugli
Carlos Mulero Haro

TSP Cercanía

Componentes Greedy:

- **Lista de candidatos:** Nodos del grafo original
- **Lista de candidatos utilizados:** Aquellos nodos que ya han sido seleccionados para formar parte de la solución.
- **Función solución:** El número de nodos en la solución es el número de nodos del grafo.
- **Función selección:** Se selecciona el nodo v_0 tal que la distancia con el último nodo ya visitado v_i sea el mınimo.
- **Función de factibilidad:** No se pueden formar ciclos, esto siempre se cumple ya que siempre se escoge un nodo nuevo.
- **Funcíon objetivo:** Encontrar un ciclo hamiltoniano minimal del grafo.

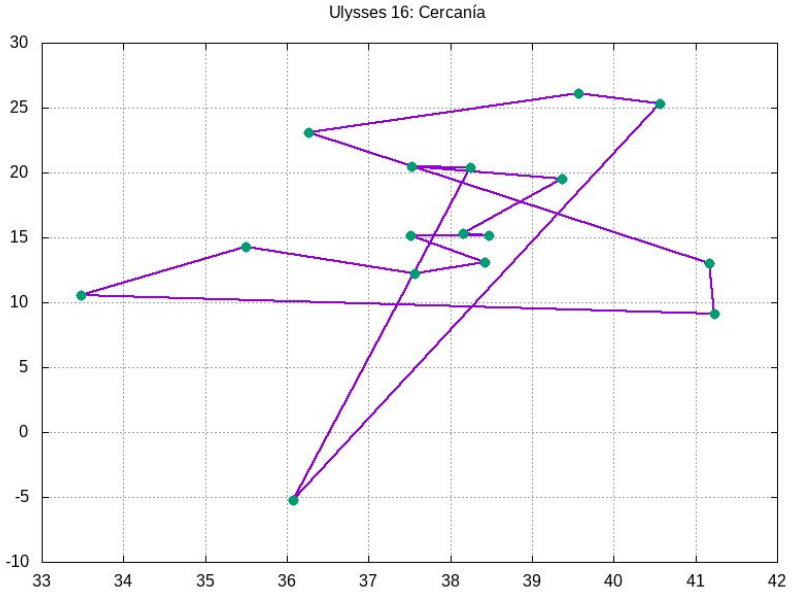
TSP Cercanía

Pseudocódigo:

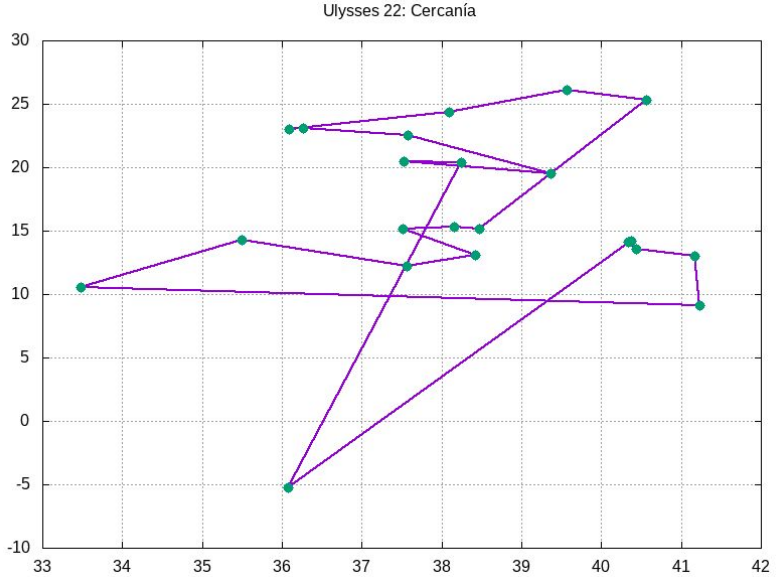
```
funcion cercaniaTSP(G=(C,A), T=(V,A))
inicio
      nIni := Nodo cualquiera de C
      nFin := nIni
      nVec

      C.eliminar(nIni) // C es ahora la lista de candidatos
      T.insertar(nIni) // Solucion a crear

      mientras !C.esVacio() hacer
            nVec := seleccionar nodo en C donde distancia=(nVec,nIni) sea menor
            T.insertar(nVec) // Insertar en solucion nodo mas cercano a nIni
            C.eliminar(nVec) // Quitarlo de los candidatos
            nIni := nVec
      fmientras
      T.insertar(nIni) // Cerrar el camino, insertando el nodo inicial y calculando distancia
entre el ultimo nodo candidato y el inicial.
ffunc
```

TSP Cercanía

Recorridos:



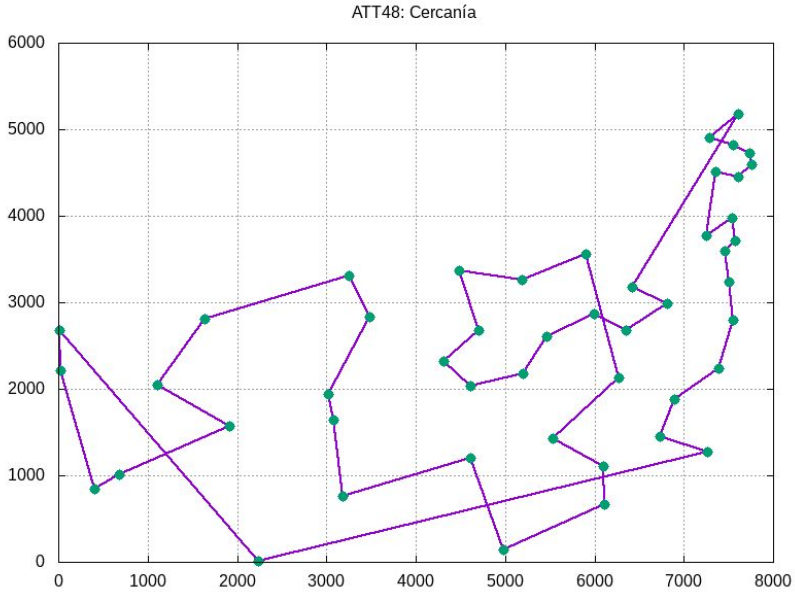
41,1% más largo



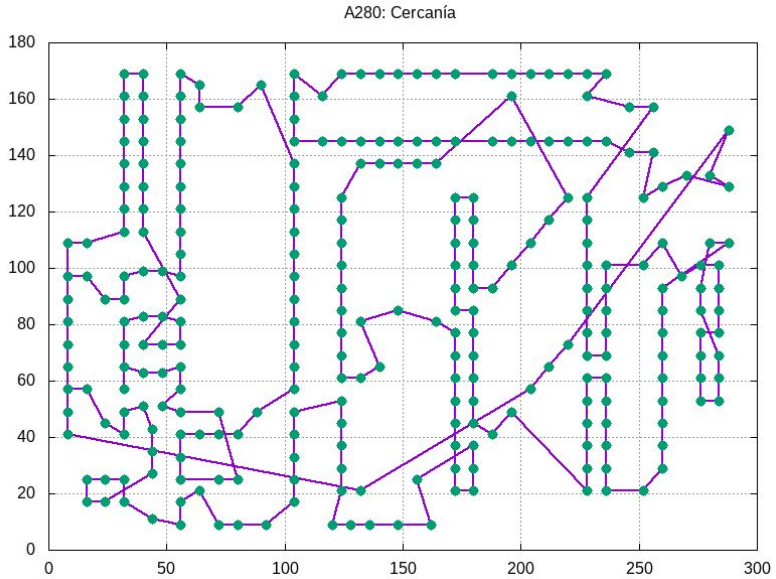
25,68% más largo

TSP Cercanía

Recorridos:



21,06% más largo



22,41% más largo

TSP Inserción

Componentes Greedy:

- **Lista de candidatos:** Nodos del grafo original
- **Lista de candidatos utilizados:** Aquellos nodos que ya han sido seleccionados para formar parte de la solución.
- **Función solución:** El número de nodos en la solución es el número de nodos del grafo.
- **Función selección:** Devuelve el nodo a insertar en la siguiente iteración y el nodo delante del cual debe insertarse
- **Función de factibilidad:** No se pueden formar ciclos, ya que el nodo seleccionado para ser insertado en la solución se elimina de los candidatos.
- **Funcíon objetivo:** Encontrar un ciclo hamiltoniano minimal del grafo.

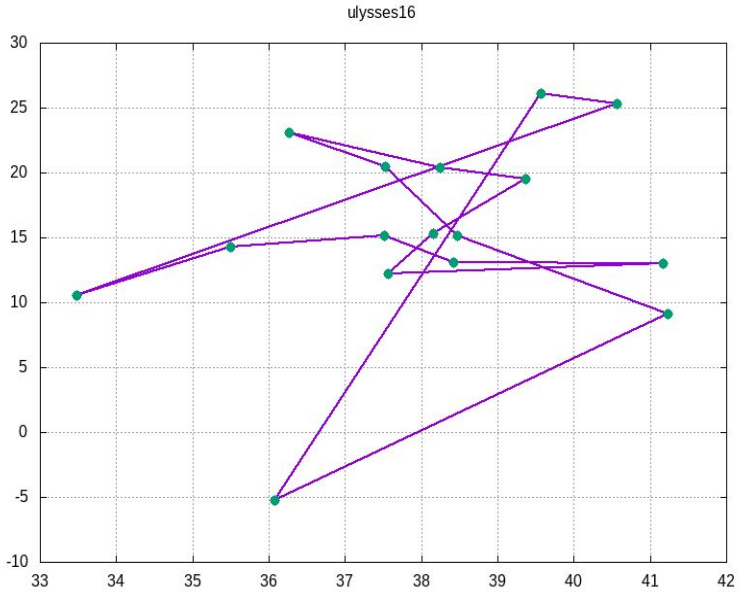
TSP Inserción

Pseudocódigo:

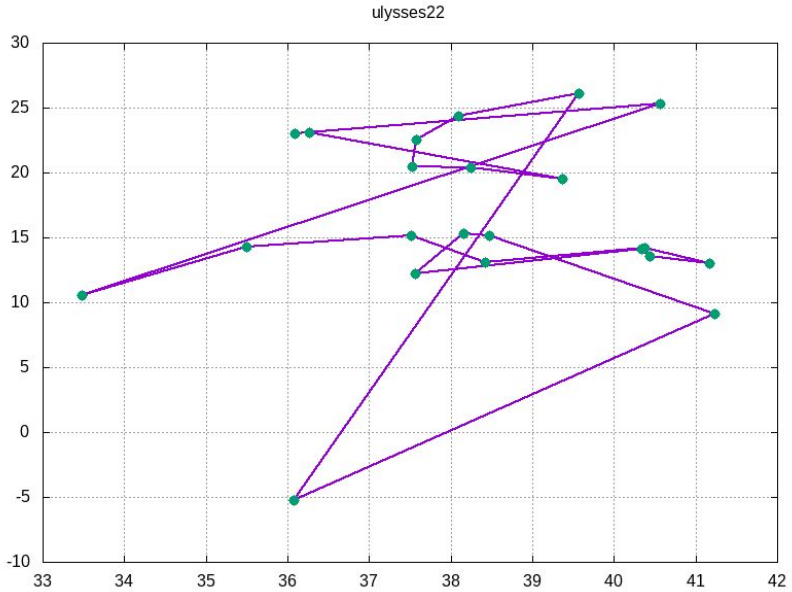
```
funcion insercionTSP(G, dist)
inicio
    dist := G.nodoNorte().distancia(G.nodoEste())+G.nodoEste().distancia(G.nodoOeste())
    g_res //Grafo donde se recoge la solucion

    g_res = G.recorridoParcial() //recorrido parcial con los tres nodos inic
    G.eliminar(G.nodoEste())
    G.eliminar(G.nodoOeste())
    G.eliminar(G.nodoNorte())
    aux  //par de nodos vacío al principio

    mientras !G.vacio() hacer
        aux := seleccionar nodo en C donde distancia sea menor
        g_res.insertar(aux.first, aux.second)
        C.eliminar(aux.second)    // Quitarlo de los candidatos
        dist+=aux.first.distancia(aux.second)
    fmientras

    T.insertar(g_res.primero)
    devolver g_res
ffunc
```

TSP Inserción

Recorridos:



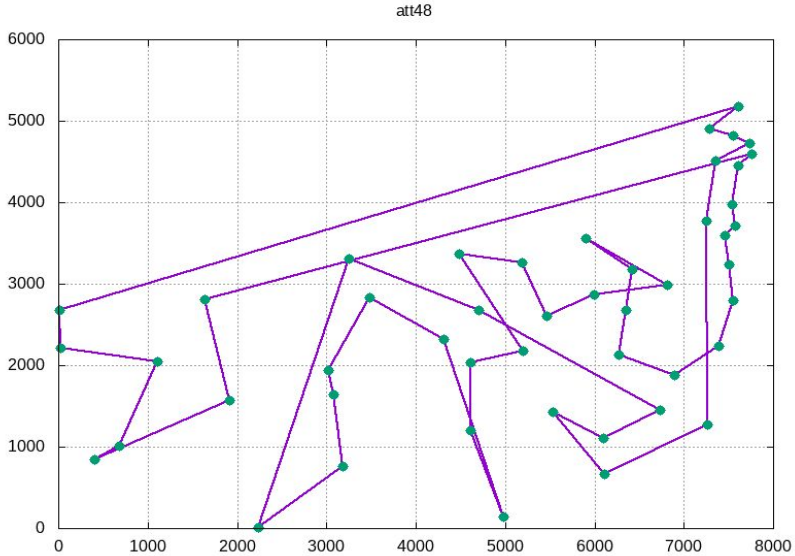
43,83% más largo



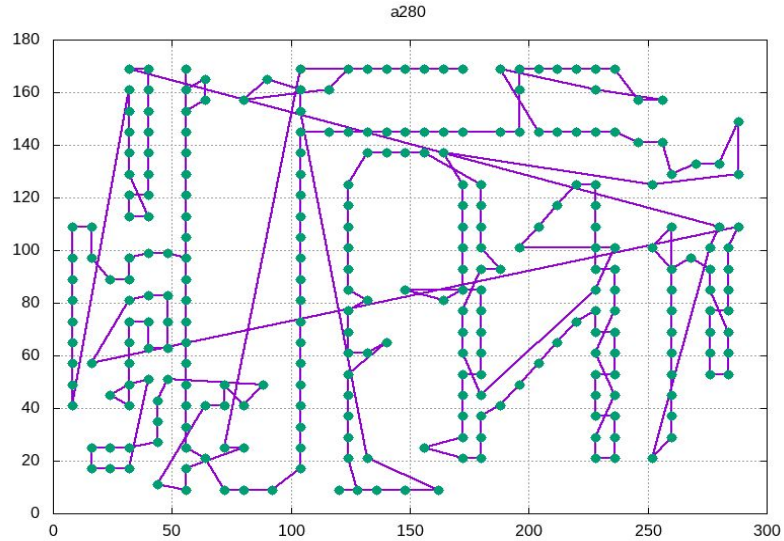
44,59% más largo

TSP Inserción

Recorridos:



67,00% más largo



62,04% más largo

TSP Aristas

Componentes Greedy:

- **Lista de candidatos:** Todas las parejas de nodos del grafo original posibles (distancia entre los nodos, la cual llamaremos "arista").
- **Lista de candidatos utilizados:** Todos los nodos que forman parte de las parejas seleccionadas como solución.
- **Función solución:** El número de nodos de nuestra solución es el mismo que el número de nodos original; siendo el último igual que el primero para cerrar el camino.
- **Función selección:** Se selecciona la pareja de nodos v_0 tal que la distancia entre ellos sea la mínima y cualquiera de sus nodos se pueda unir a la solución.
- **Función de factibilidad:** No se forman ciclos ya que las parejas disponibles que podrían crearlos son eliminadas al insertar un nuevo nodo a la solución.
- **Funcíon objetivo:** Encontrar un ciclo hamiltoniano minimal del grafo.

TSP Aristas

Pseudocódigo:

```
funcion aristasTSP(x[], y[])
inicio
    dimension := x.size() // Numero de nodos candidatos

    // Calculamos las distancias entre todas las aristas
    para i:=0 hasta dimension-1 hacer
        para j:=i hasta dimension-1 hacer
            si i != j entonces
                distancias.push(calculardistancia(x[i],x[j],y[i],x[j])
                aristas_aux.push(i)
                aristas_aux.push(j)
            fsi
        fpara
    fpara
```

TSP Aristas

Pseudocódigo:

```
// Ordenamos las distancias de menor a mayor distancia
    mientras aristas_sol.size() sea menor que (dimension-1)*dimension hacer

        indicemenordistancia := posicion del menor elemento de distancias[]

        aristas_ord.push(aristas_aux[indicemenordistancia*2])
        aristas_ord.push(aristas_aux[indicemenordistancia*2+1])
        distancias.eliminar(indicemenordistancia)
        aristas_aux.eliminar(indicemenordistancia*2,indicemenordistancia*2+1)
    fmientras

// Incluimos la primera arista a nuestra solucion (la pareja de nodos cuya distancia es la
menor se encuentra la primera ya que ordenamos previamente las distancias)
    solucion.push(aristas_ord[0])
    solucion.push(aristas_ord[1])
    aristas_ord.eliminar(0,1)
```

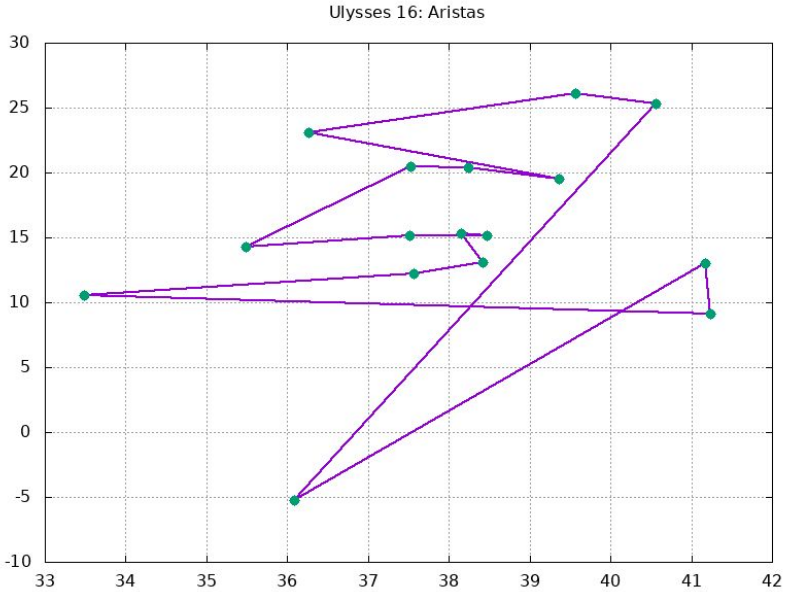
TSP Aristas

Pseudocódigo:

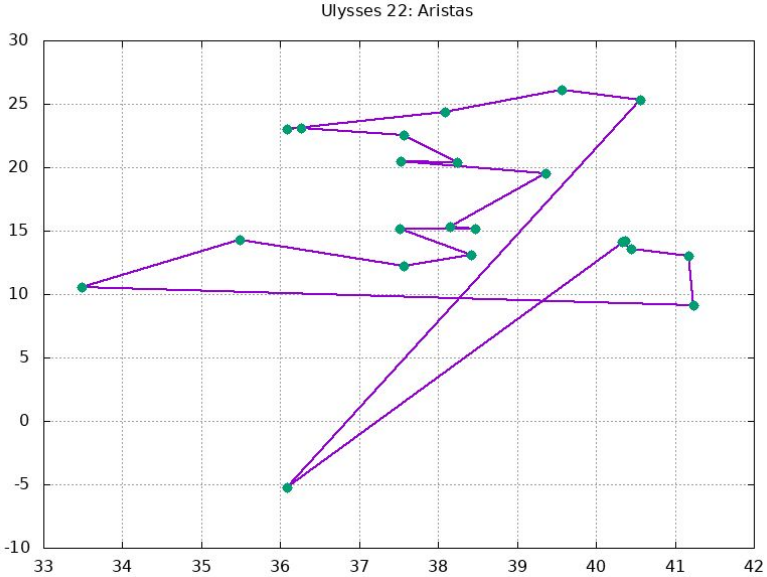
```
mientras solucion.size() sea menor que dimension hacer

        para i:=0 hasta dimension-1 y mientras que !encontrado  con paso 2 hacer
            si cualquier componente de la pareja de nodos puede insertarse en solucion entonces
                encontrado := true
                nodo_union := nodo de la pareja de nodos que ya forma parte de la solucion
                solucion.insert(el nuevo nodo que forma parte de la solucion)
            fsi
        fpara
        para i:=0 hasta aristas_ord.size() con paso 2 hacer
            si aristas_ord[i] es igual a nodo_union o aristas_ord[i+1] es igual a nodo_union entonces
                aristas_ord.eliminar(i,i+1)
            fsi
        fpara
    fmientras

    // Cerramos el camino uniendo el final con el principio
    solucion.push(solucion[0])

ffuncion
```

TSP Aristas

Recorridos:



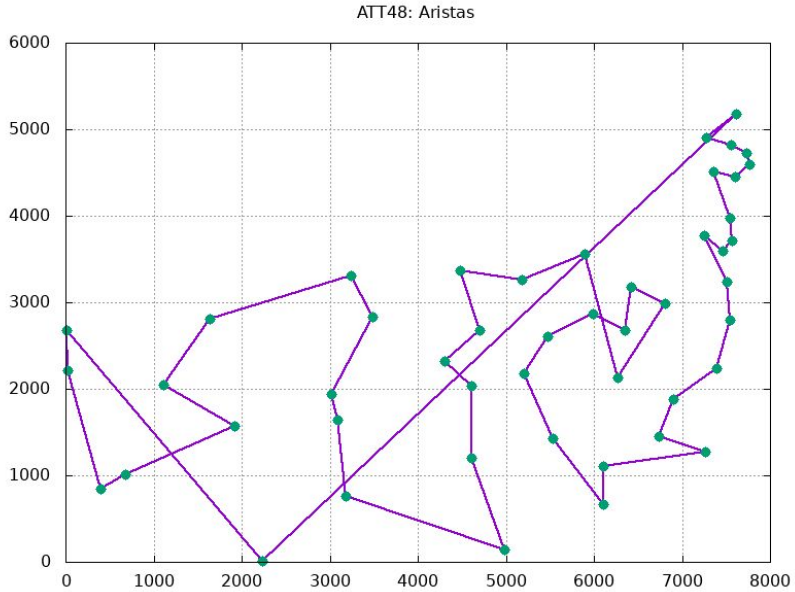
23,28% más largo



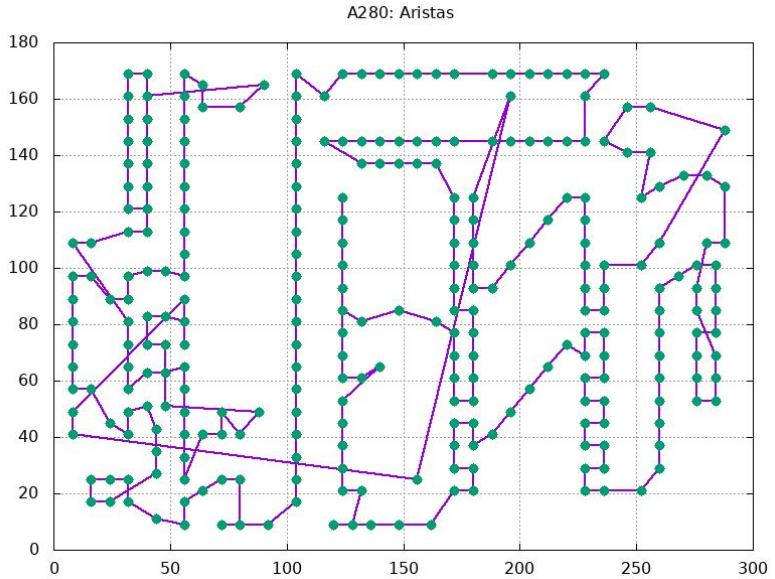
22,97% más largo

TSP Aristas

Recorridos:



25,75% más largo



24,04% más largo

TSP Comparación

TSP	Óptimo	Cercanía		Inserción		Aristas	
		Camino	Diferencia	Camino	Diferencia	Camino	Diferencia
A280	2579	3157	22,41%	4179	62,04%	3199	24,04%
ATT48	33522	40583	21,06%	55982	67,00%	42155	25,75%
Ulysses 16	73	103	41,1%	105	43,83%	90	23,28%
Ulysses 22	74	93	25,68%	107	25,68%	91	22,97%

Asignación de Tareas

Componentes Greedy:

- **Lista de candidatos:** Conjunto de tareas propuestas para su ejecución.
- **Lista de candidatos utilizados:** Conjunto de tareas que ya han sido elegidas para ser ejecutadas.
- **Función solución:** Obtener el conjunto de tareas que otorguen el mayor beneficio.
- **Función selección:** Escoge la tarea que más beneficio otorgue.
- **Función de factibilidad:** La tarea seleccionada debe estar dentro de su plazo límite de ejecución.
- **Función objetivo:** Obtener el conjunto de tareas que otorguen mayor beneficio al ser ejecutadas.

Asignación de Tareas

Pseudocódigo:

```
funcion AsignacionTareas(priority_queue Q)
inicio
    TIEMPO_CONSUMIDO := 0
    BENEFICIO_TOTAL := 0
    lFinal := Cola vacia de tareas
    nTarea := Objeto tarea local

    mientras !Q.Vacio() hacer
        nTarea := Q.top()
        Q.pop()

        si TIEMPO_CONSUMIDO es menor o igual que tiempo de nTarea entonces
            BENEFICIO_TOTAL := BENEFICIO_TOTAL + nTarea.Beneficio()
            TIEMPO_CONSUMIDO := TIEMPO_CONSUMIDO + 1
            lFinal.insertar(nTarea)

        si no
            Se descarta la tarea
    fmientras
ffuncion
```

Asignación de Tareas

Caso Base

```
> ./main 5
El numero de tareas generadas es: 5
Identificador: ID_324  Plazo limite: 5  Beneficio: 256
Identificador: ID_79  Plazo limite: 2  Beneficio: 185
Identificador: ID_330  Plazo limite: 5  Beneficio: 161
Identificador: ID_312  Plazo limite: 4  Beneficio: 143
Identificador: ID_31  Plazo limite: 5  Beneficio: 17

ID de tarea ejecutada: ID_324
Beneficio actual: 256

ID de tarea ejecutada: ID_79
Beneficio actual: 441

ID de tarea ejecutada: ID_330
Beneficio actual: 602

ID de tarea ejecutada: ID_312
Beneficio actual: 745

ID de tarea ejecutada: ID_31
Beneficio actual: 762

El conjunto final de tareas obtenidas es:
Identificador: ID_324  Plazo limite: 5  Beneficio: 256
Identificador: ID_79  Plazo limite: 2  Beneficio: 185
Identificador: ID_330  Plazo limite: 5  Beneficio: 161
Identificador: ID_312  Plazo limite: 4  Beneficio: 143
Identificador: ID_31  Plazo limite: 5  Beneficio: 17

Con un total de Beneficio: 762
Unidades de tiempo consumidas: 5
>
```

Caso Ideal

```
> ./main 5
El numero de tareas generadas es: 5
Identificador: ID_516  Plazo limite: 5  Beneficio: 384
Identificador: ID_223  Plazo limite: 5  Beneficio: 266
Identificador: ID_334  Plazo limite: 1  Beneficio: 229
Identificador: ID_46  Plazo limite: 1  Beneficio: 177
Identificador: ID_17  Plazo limite: 1  Beneficio: 24

ID de tarea ejecutada: ID_516
Beneficio actual: 384

ID de tarea ejecutada: ID_223
Beneficio actual: 650

Se omite esta tarea con identificador ID_334 por haber superado el limite de tiempo.

Se omite esta tarea con identificador ID_46 por haber superado el limite de tiempo.

Se omite esta tarea con identificador ID_17 por haber superado el limite de tiempo.

El conjunto final de tareas obtenidas es:
Identificador: ID_516  Plazo limite: 5  Beneficio: 384
Identificador: ID_223  Plazo limite: 5  Beneficio: 266

Con un total de Beneficio: 650
Unidades de tiempo consumidas: 2
>
```