Práctica 3 Algorítmica:

ALGORITMOS GREEDY

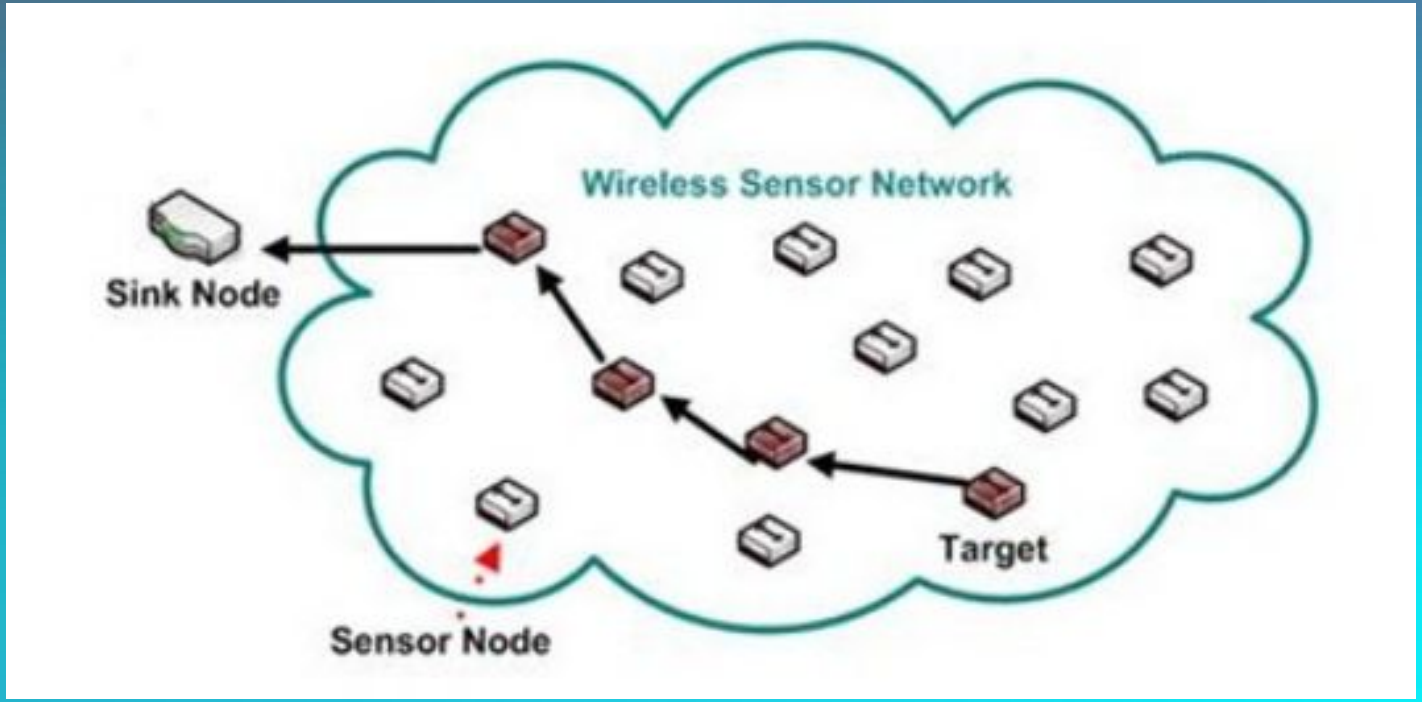
Por:

José A. Carmona Molina

Nuria Manzano Mata

Antonio Rodríguez Rodríguez

Algoritmo 1: Red de Sensores



Diseño de componentes:

- **Lista de candidatos:** Todos los posibles caminos desde el servidor al sensor objetivo.
- **Lista de candidatos usados:** Camino que minimiza el tiempo de transmisión (con los respectivos nodos por los que pasa).
- **Criterio de selección:** Camino que presente un tiempo mínimo de transmisión desde el sensor objetivo al servidor central.
- **Criterio de factibilidad:** Inserta todos los sensores que componen el camino cuyo tiempo de transmisión hasta llegar al servidor sea mínimo.
- **Función solución:** Recorrido mínimo posible hasta llegar al servidor central desde el sensor objetivo.
- **Función objetivo:** Minimizar el tiempo de transmisión entre servidor central y sensor objetivo.

Diseño del Algoritmo:

Algoritmo de Dijkstra

```
FUNCION DIJKSTRA
    C = {2, 3, ..., N }
    PARA I = 2 HASTA N HACER D[I] = L [1, I]
    I = 2, ..., N
    P[I] = 1
    REPETIR N – 2 VECES
        w = algún elemento de C que minimice D[V]
        C = C – (w)
        PARA CADA v ∈ C HACER
            SI D[v] > D[w] + L[w,v] ENTONCES
            D[v] = D[w] + L[w,v]
            P[v] = w
    DEVOLVER D
```

Donde P es un vector que nos permite conocer por donde pasa cada camino de longitud minima desde el origen

Diseño del Algoritmo:

```
double Dijkstra(Problema p, int destino, int origen) {
    int n = p.getNumsensores();
    int sol[n] = {-1};
    double distancia[n], C[n];
    for (int i = 0; i < n; i++) {
        distancia[i] = p.gett_transmission(destino, i);
        C[i] = p.gett_transmission(destino, i);
        sol[i] = -1;
    }

    int cont = 1;
    while(cont < n) {
        int pos = 0;
        for (int i = 0; i < n; i++) {
            if (C[i] < distancia[cont]) {
                pos = i;
                C[pos] = INF;
                i = n;
            }
        }
        int aux = 0;
        for (int i = 0; i < n; i++) {
            if (distancia[i] > distancia[pos] + p.gett_transmission(pos, i)) {
                distancia[i] = distancia[pos] + p.gett_transmission(pos, i);
                if (aux != i) sol[aux] = -1;
                sol[i] = pos;
                aux = i;
            }
        }
        cont++;
    }
```

```
    cout << "Señal emitida desde: " << p.getSensor(origen) << endl;
    cout << "\nSensores intermedios: " << endl;
    int id = sol[origen];
    while (id != -1) {
        cout << p.getSensor(id) << endl;
        id = sol[id];
    }
    cout << endl;
    cout << "Sensor destino: " << p.getSensor(destino);

    return distancia[origen];
}
```

Estudio de Optimalidad:

```
s = Dijkstra(p, 0, 2);
```

```
antonio@antonio-IdeaPad-Gaming-3-15ARH05:~
os$ ./ejercicio1
Señal emitida desde: Ciencias

Sensores intermedios:
Medicina

Sensor destino: Etsiit
Tiempo transmisión: 60
```

```
s = Dijkstra(p, 2, 3);
```

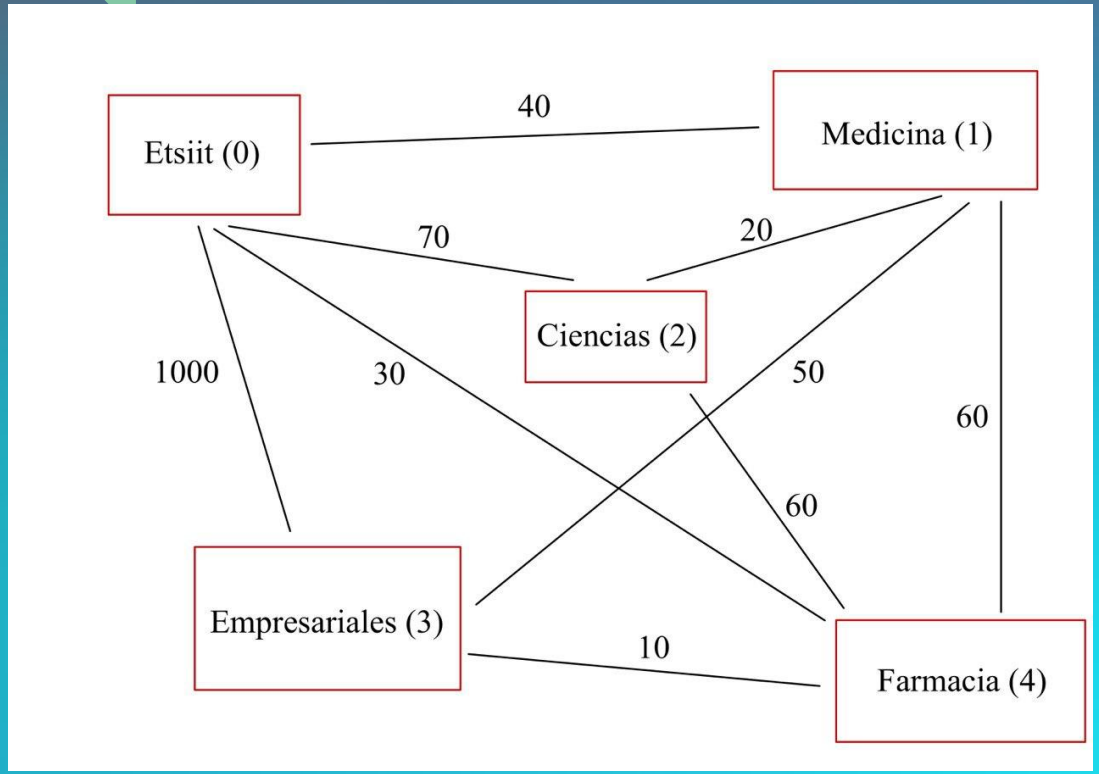
```
antonio@antonio-IdeaPad-Gaming-3-15ARH05:~/
os$ ./ejercicio1
Señal emitida desde: Empresariales(origen)

Sensores intermedios:
Medicina

Sensor destino: Ciencias
Tiempo transmisión: 70
```

Algoritmo utilizado Dijkstra → Nos da el camino menor entre dos puntos de un grafo.

Siempre da la mejor solución.

Ejemplo paso a paso:

```
s = Dijkstra(p, 0, 3);
```



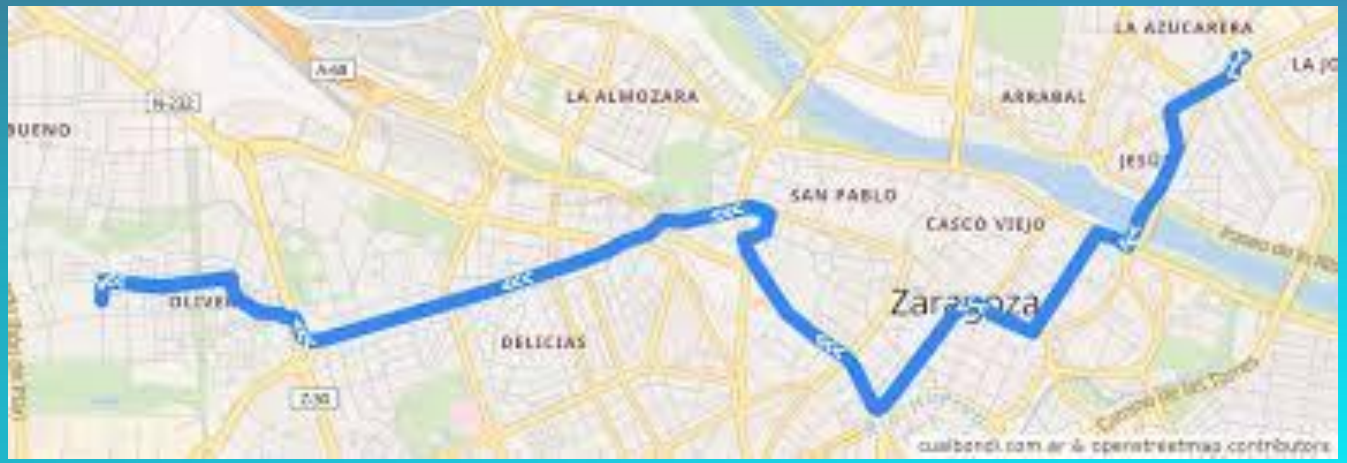

```
5
Etsiit
Medicina
Ciencias
Empresariales(origen)
Farmacia
-1
40 Etsiit-Medicina
70 Etsiit-Ciencias
1000 Etsiit-Empresariales(origen)
30 Etsiit-Farmacia
40 Etsiit-Medicina
-1
20 Medicina-campo
50 Medicina-Empresariales(origen)
60 Medicina-Farmacia
70 Etsiit-Ciencias
20 Medicina-campo
-1
-1
60 Ciencias-Farmacia
1000 Etsiit-Empresariales(origen)
50 Medicina-Empresariales(origen)
-1
-1
10 Empresariales(origen)-Farmacia
30 Etsiit-Farmacia
60 Medicina-Farmacia
60 Ciencias-Farmacia
10 Empresariales(origen)-Farmacia
-1

```

Main y Ejecuciones:

```
antonio@antonio-IdeaPad-Gaming-3-15ARH05:~/Descargas/Greedy/Ejercicios_definitiv
os$ ./ejercicio1
Señal emitida desde: Empresariales(origen)

Sensores intermedios:
Farmacia

Sensor destino: Etsiit
Tiempo transmisión: 40
```

```
int main()
{
   Problema p;
   double s;

   p.leerFichero("sensores.dat");
   s = Dijkstra(p, 0, 3);

   cout << endl
        << "Tiempo: " << s << endl;

   return 0;
}
```

Algoritmo 2: Viaje en bus



Diseño de componentes:

- **Lista de candidatos:** Todas las gasolineras en las que se puede realizar una parada.
- **Lista de candidatos usados:** Gasolineras donde hacemos parada.
- **Criterio de selección:** Seleccionaremos las gasolineras que permitan hacer una parada antes de superar los kilómetros de autonomía del autobús y tratando que la distancia entre parada y parada sea máxima.
- **Criterio de factibilidad:** Se inserta en la solución si la siguiente gasolinera está más lejos de lo que el autobús puede avanzar.
- **Función solución:** Lista de gasolineras en las que se deberá parar, intentando hacer el mínimo número de paradas.
- **Función objetivo:** Minimizar el número de paradas hasta llegar a su destino.

Diseño del Algoritmo:

```cpp
void OrdenaBurbuja(vector<pair<string,double>> & v, int n) {

    int i, j, aux;
    string id;
    bool haycambios= true;

    i= 0;
    while (haycambios) {
        haycambios=false; // Suponemos vector ya ordenado
        for (j= n-1; j>i; j--) { // Recorremos vector de final a i

            if (v[j-1].second>v[j].second) { // Dos elementos consecutivos mal ordenados
              aux = v[j].second; // Los intercambiamos
              id = v[j].first;
              v[j].second = v[j-1].second;
              v[j].first = v[j-1].first;
              v[j-1].second = aux;
              v[j-1].first = id;
              haycambios= true; // Al intercambiar, hay cambio
            }
        }
        i++;
    }
}
```

```cpp
void Minimas_paradas(vector<pair<string,double>> gasolineras, double autonomia){
    OrdenaBurbuja(gasolineras, gasolineras.size());
    cout << "Salimos de: " << gasolineras.front().first << endl << "Pararemos en: \n";
    int cont = 0;
    for(int n = 0; n < gasolineras.size();) {
        int distancia = gasolineras[n].second - gasolineras[cont].second;
        if(distancia <= autonomia){
            n++;
        } else {
            if (gasolineras[n-1] != gasolineras[cont]){
                cout << "Gasolinera " << gasolineras[n-1].first << " en el km " << gasolineras[n-1].second << endl;
                cont = n-1;
            } else {
                cout << "No es posible completar el recorrido" << endl;
                exit(-1);
            }
        }
    }
    cout << gasolineras.back().first << " en el kilometro " << gasolineras.back().second << endl;
}
```

Estudio de Optimalidad:

```
antonio@antonio-IdeaPad-Gaming-3-15ARH05:~/Descargas/Greedy/Ejercicios_definitiv
os$ ./ejercicio2

Error: El programa se debe ejecutar de la siguiente forma.

./ejercicio2 Autonomía gasolinera1 dist_origen1 identificador2 gasolinera2 ... g
asolineraN dist_origenN

antonio@antonio-IdeaPad-Gaming-3-15ARH05:~/Descargas/Greedy/Ejercicios_definitiv
os$ ./ejercicio2 5 Origen 2 b 4 c 9 Destino 14
Salimos de: Origen
Pararemos en:
Gasolinera b en el km 4
Gasolinera c en el km 9
Destino en el kilometro 14
antonio@antonio-IdeaPad-Gaming-3-15ARH05:~/Descargas/Greedy/Ejercicios_definitiv
os$ ./ejercicio2 5 Origen 2 b 4 c 9 Destino 15
Salimos de: Origen
Pararemos en:
Gasolinera b en el km 4
Gasolinera c en el km 9
No es posible completar el recorrido
```

Como vemos siempre obtenemos el camino con menor número de paradas realizadas → Optimizado.

Ejemplo paso a paso:

```
void Minimas_paradas(vector<pair<string,double>> gasolineras, double autonomia){
    OrdenaBurbuja(gasolineras, gasolineras.size());
    cout << "Salimos de: " << gasolineras.front().first << endl << "Pararemos en: \n";
    int cont = 0;
    for(int n = 0; n < gasolineras.size();) {
        int distancia = gasolineras[n].second - gasolineras[cont].second;
        if(distancia <= autonomia){
            n++;
        } else {
            if (gasolineras[n-1] != gasolineras[cont]){
                cout << "Gasolinera " << gasolineras[n-1].first << " en el km " << gasolineras[n-1].second << endl;
                cont = n-1;
            } else {
                cout << "No es posible completar el recorrido" << endl;
                exit(-1);
            }
        }
    }
    cout << gasolineras.back().first << " en el kilometro " << gasolineras.back().second << endl;
}
```

```
antonio@antonio-IdeaPad-Gaming-3-15ARH05:~/Descargas/Greedy/Ejercicios_definitiv
os$ ./ejercicio2 5 Origen 0 b 3 c 6 Destino 10
Salimos de: Origen
Pararemos en:
Gasolinera b en el km 3
Gasolinera c en el km 6
Destino en el kilometro 10
```

Main y Ejecuciones:

```
antonio@antonio-IdeaPad-Gaming-3-15ARH05:~/Descargas/Greedy/Ejercicios_definitiv
os$ ./ejercicio2 6 Origen 0 b 3 c 6 Destino 10
Salimos de: Origen
Pararemos en:
Gasolinera c en el km 6
Destino en el kilometro 10
antonio@antonio-IdeaPad-Gaming-3-15ARH05:~/Descargas/Greedy/Ejercicios_definitiv
os$ ./ejercicio2 5 Origen 0 b 3 c 6 Destino 10
Salimos de: Origen
Pararemos en:
Gasolinera b en el km 3
Gasolinera c en el km 6
Destino en el kilometro 10
antonio@antonio-IdeaPad-Gaming-3-15ARH05:~/Descargas/Greedy/Ejercicios_definitiv
os$ ./ejercicio2 5 Origen 0 b 3 c 9 Destino 10
Salimos de: Origen
Pararemos en:
Gasolinera b en el km 3
No es posible completar el recorrido
```

```
int main(int argc, char** argv) {
    if (argc <= 1) {
        cerr<<"\nError: El programa se debe ejecutar de la siguiente forma.\n\n";
        cerr<<argv[0]<<" Autonomia gasolinera1 dist_origen1 identificador2 gasolinera2 ... gasolineraN dist_origenN\n\n";
        return 0;
    }
    vector<pair<string,double>> gasolineras;
    for (int i=2; i<argc; i+=2) {
        string gasolinera = argv[i];
        double dist = stod(argv[i+1]);
        pair<string, double> input;
        input.first = gasolinera;
        input.second = dist;
        gasolineras.push_back(input);
    }
    int autonomia = stod(argv[1]);
    Minimas_paradas(gasolineras, autonomia);
```

Main y Ejecuciones:

```
int main(int argc, char** argv) {
    if (argc <= 1) {
        cerr<<"\nError: El programa se debe ejecutar de la siguiente forma.\n\n";
        cerr<<argv[0]<<" Autonomia gasolinera1 dist_origen1 identificador2 gasolinera2 ... gasolineraN dist_origenN\n\n";
        return 0;
    }
    vector<pair<string,double>> gasolineras;
    for (int i=2; i<argc; i+=2) {
        string gasolinera = argv[i];
        double dist = stod(argv[i+1]);
        pair<string, double> input;
        input.first = gasolinera;
        input.second = dist;
        gasolineras.push_back(input);
    }
    int autonomia = stod(argv[1]);
    Minimas_paradas(gasolineras, autonomia);
```

Algoritmo 3: Buque Mercante

Diseño de componentes:

- **Lista de candidatos:** Todos los contenedores que podrían meterse en el barco.
- **Lista de candidatos usados:** Contenedores que han sido cargados en el barco.
- **Criterio de selección:** Seleccionar los contenedores que más llenan el barco sin sobrepasar su capacidad (los más grandes).
- **Criterio de factibilidad:** Se inserta en el barco la solución si al sumarlo no supera el peso máximo del barco.
- **Función solución:** Lista de contenedores que se han insertado en el barco.
- **Función objetivo:** Maximizar el peso de los contenedores que se meterán en el barco.

Diseño del Algoritmo:

```cpp
void OrdenaBurbuja(vector<pair<string,double>> & v, int n) {

    int i, j, aux;
    string id;
    bool haycambios= true;

    i= 0;
    while (haycambios) {
        haycambios=false; // Suponemos vector ya ordenado
        for (j= n-1; j>i; j--) { // Recorremos vector de final a i

            if (v[j-1].second>v[j].second) { // Dos elementos consecutivos mal ordenados
              aux = v[j].second; // Los intercambiamos
              id = v[j].first;
              v[j].second = v[j-1].second;
              v[j].first = v[j-1].first;
              v[j-1].second = aux;
              v[j-1].first = id;
              haycambios= true; // Al intercambiar, hay cambio
            }
        }
        i++;
    }
}
```

```cpp
// ALGORITMO MAXIMIZACION DEL PESO TIPO GREDDY.
void maximizacion_peso(vector<pair<string, double>> & pesoContenedores, double cargaMaxima) {
    int n = pesoContenedores.size();
    OrdenaBurbuja(pesoContenedores, n);
    double sumaPesos = 0;
    cout << "Carga maxima: " << cargaMaxima << endl;
    cout << "Contenedores cargados: ";
    while (sumaPesos < cargaMaxima && !pesoContenedores.empty()) {
        int aux = pesoContenedores.back().second;
        if (sumaPesos + aux <= cargaMaxima) {
            sumaPesos += aux;
            cout << pesoContenedores.back().first << ":" << aux << "T ";
        }
        pesoContenedores.pop_back();
    }
    cout << endl << "Peso total: " << sumaPesos << endl;
}
```

Estudio de Optimalidad:

- Contraejemplo

```
antonio@antonio-IdeaPad-Gaming-3-15ARH05:~/Descargas/Greedy/Ejercicios_definitiv
os$ ./ejercicio3 30 a 11 b 9 c 22 d 10
Carga máxima: 30
Contenedores cargados: c:22T
Peso total: 22
```

La mejor solución sería a,b y d; pero sólo selecciona c por ser el mayor.

Esto ocurre al estar ante un algoritmo del tipo mochila fraccionada, en el que no podemos fraccionar (al no poder dividir el peso de los contenedores) con lo que no siempre obtendremos la solución más óptima, aunque sí la más rápida.

Ejemplo paso a paso:

```
antonio@antonio-IdeaPad-Gaming-3-15ARH05:~/Descargas/Greedy/Ejercicios_definitiv
os$ ./ejercicio3

Error: El programa se debe ejecutar de la siguiente forma.

./ejercicio3 cargaMaxima identificador1 pesoContenedores1 identificador2 pesoCon
tenedores2 ... identificadorN pesoContenedoresN

antonio@antonio-IdeaPad-Gaming-3-15ARH05:~/Descargas/Greedy/Ejercicios_definitiv
os$ ./ejercicio3 30 a 11 b 9 c 7 d 10
Carga máxima: 30
Contenedores cargados: a:11T d:10T b:9T
Peso total: 30
```

Main y Ejecuciones:

```cpp
int main (int argc, char *argv[]) {
    if (argc <= 1) {
        cerr<<"\nError: El programa se debe ejecutar de la siguiente forma.\n\n";
        cerr<<argv[0]<<" cargaMaxima identificador1 pesoContenedores1 identificador2 pesoContenedores2 ... identificadorN pesoContenedoresN\n\n";
        return 0;
    }
    vector<pair<string,double>> pesoContenedores;
    for (int i=2; i<argc; i+=2) {
        string id = argv[i];
        double aux = stod(argv[i+1]);
        pair<string, double> input;
        input.first = id;
        input.second = aux;
        pesoContenedores.push_back(input);
    }
    maximizacion_peso(pesoContenedores, stod(argv[1]));
}
```

```
antonio@antonio-IdeaPad-Gaming-3-15ARH05:~/Descargas/Greedy/Ejercicios_definitiv
os$ ./ejercicio3 100 a 30 b 35 c 120 d 37 e 28
Carga máxima: 100
Contenedores cargados: d:37T b:35T e:28T
Peso total: 100
antonio@antonio-IdeaPad-Gaming-3-15ARH05:~/Descargas/Greedy/Ejercicios_definitiv
os$ ./ejercicio3 100 a 30 b 35 c 100 d 37 e 28
Carga máxima: 100
Contenedores cargados: c:100T
Peso total: 100
antonio@antonio-IdeaPad-Gaming-3-15ARH05:~/Descargas/Greedy/Ejercicios_definitiv
os$ ./ejercicio3 20 a 30 b 35 c 100 d 37 e 28
Carga máxima: 20
Contenedores cargados:
Peso total: 0
```