PREDA - UNED

Programación y Estructuras de Datos Avanzadas

Algoritmos voraces y el mantenimiento de la conectividad

- Hay una tupida red de conexiones c(i, j) entre los nodos i y j, siendo $i,j \in \{1,\ldots,8\}$ y con un coste $c(i, j) = (i \times j) \bmod 6$
- Se pretende mantener la conectividad de la red (grafo conexo) a un coste mínimo (optimización)
- ¿Estrategia?

Algoritmos voraces y el mantenimiento de la conectividad

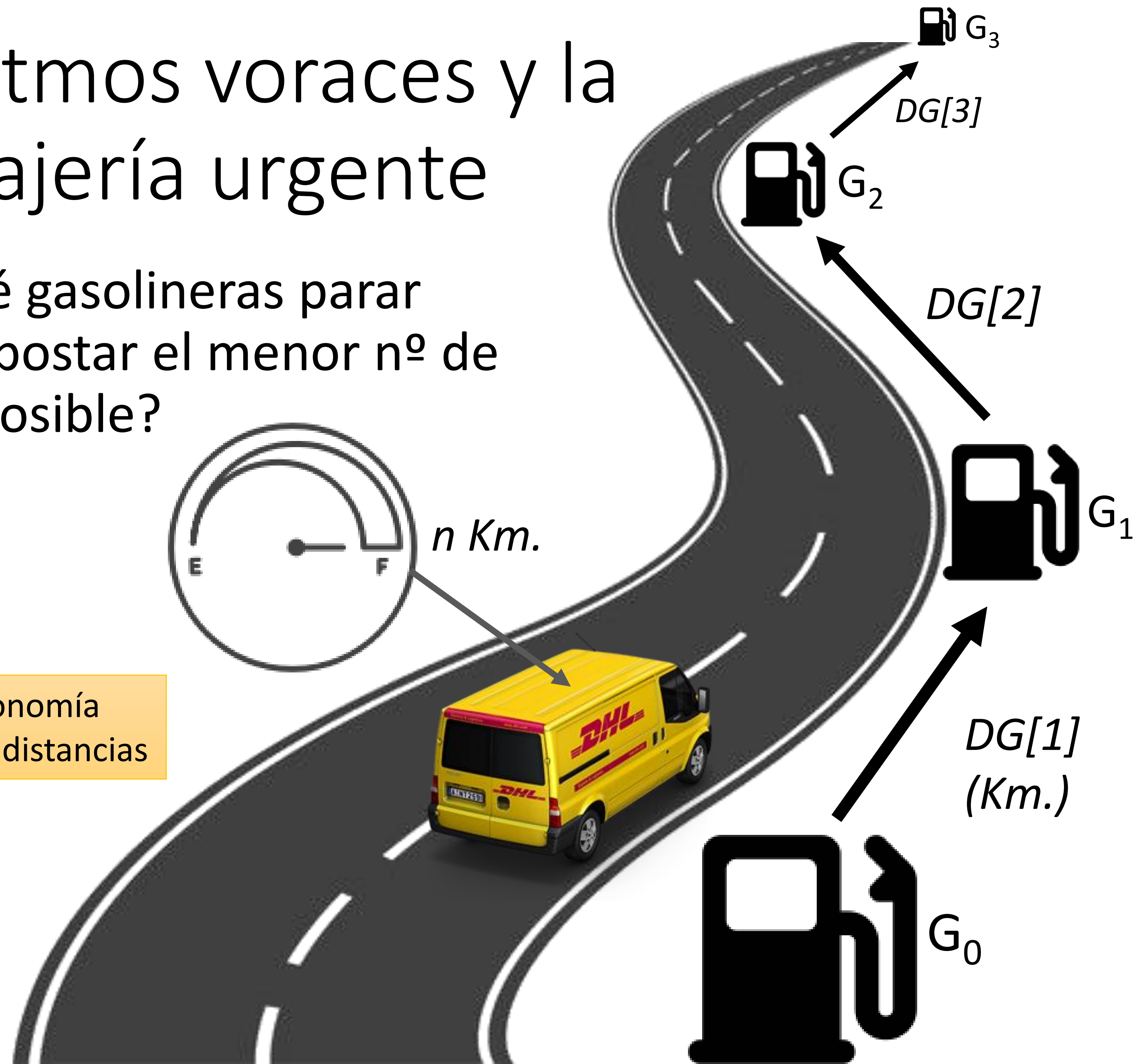
- Hay una tupida red de conexiones c(i, j) entre los nodos i y j, siendo $i,j \in \{1,\ldots,8\}$ y con un coste $c(i, j) = (i \times j) \bmod 6$
- Se pretende mantener la conectividad de la red (grafo conexo) a un coste mínimo (optimización)
- Estrategia:
  Asociar al grafo (la red) un árbol de recubrimiento de coste mínimo → Algoritmos de Prim o Kruskal
- ¿Qué estructura de datos usar para representar el grafo?

Algoritmos voraces y el mantenimiento de la conectividad

- Hay una tupida red de conexiones c(i, j) entre los nodos i y j, siendo $i,j \in \{1,\ldots,8\}$ y con un coste $c(i, j) = (i \times j) \bmod 6$
- Se pretende mantener la conectividad de la red (grafo conexo) a un coste mínimo (optimización)
- Estrategia:
  Asociar al grafo (la red) un árbol de recubrimiento de coste mínimo → Algoritmos de Prim o Kruskal
- Al indicarse que todos los nodos están conectados entre sí, se trata de un grafo denso no dirigido, por lo que debe usarse una **matriz de adyacencia** simétrica

Algoritmos voraces y la mensajería urgente

- ¿En qué gasolineras parar para repostar el menor nº de veces posible?



n: Km. de autonomía
DG: vector de distancias

Algoritmos voraces y la me...

- ¿E... pa... ve...

n: Km...
DG: v...

```
tipo Vector = matriz[0..G-1] de booleano
fun EntregaExpresMinimasParadas (DG: Vector[1..G-1] de natural, n,G: natural):Vector
    var
        solucion: Vector
    fvar
    para i = 1 hasta G-1 hacer
        solucion[i] ← falso
    fpara
    i ← 0
    contKm ← 0
    repetir
        repetir
            i ← i + 1
            contKm ← contKm + DG[i]
        hasta (contKm > n) ∨ (i = G-1)
        si contKm > n entonces
            i ← i - 1
            solucion[i] ← cierto
            contKm ← 0
        fsi
    hasta i = G-1
    dev solucion[]
ffun
```

Estrategia:
Llegar hasta la gasolinera más lejana posible

G3
DG[3]
DG[2]
G1
DG[1] (Km.)
G0

Algoritmos voraces y la me...

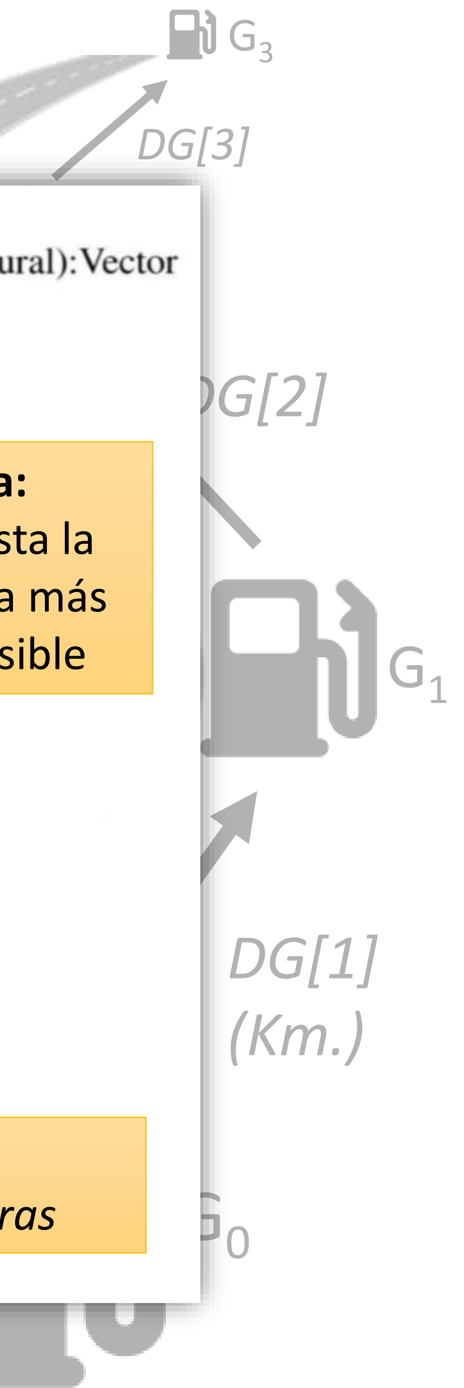
- ¿E... pa... ve...

n: Km...
DG: v...

```
tipo Vector = matriz[0..G-1] de booleano
fun EntregaExpresMinimasParadas (DG: Vector[1..G-1] de natural, n,G: natural):Vector
    var
        solucion: Vector
    fvar
    para i = 1 hasta G-1 hacer
        solucion[i] ← falso
    fpara
    i ← 0
    contKm ← 0
    repetir
        repetir
            i ← i + 1
            contKm ← contKm + DG[i]
        hasta (contKm > n) ∨ (i = G-1)
        si contKm > n entonces
            i ← i - 1
            solucion[i] ← cierto
            contKm ← 0
        fsi
    hasta i = G-1
    dev solucion[]
ffun
```

Estrategia:
Llegar hasta la gasolinera más lejana posible

Coste: O(G)
G = nº gasolineras



Algoritmos voraces y el robot desplazándose por un circuito

- Hay un robot que se tiene que desplazar desde un punto R a un punto S, y en el camino se puede encontrar con obstáculos infranqueables O
- El paso por una casilla franqueable supone un gasto de energía igual al valor que indica la casilla
- Objetivo:
  Llegar al punto S gastando el mínimo de energía
- ¿Estrategia?

Algoritmos voraces y el robot desplazándose por un circuito

O	S	O	2	1
3	1	3	1	1
1	6	6	O	6
1	2	O	**R**	4
7	1	1	2	6

- Hay un robot que se tiene que desplazar desde un punto R a un punto S, y en el camino se puede encontrar con obstáculos infranqueables O
- El paso por una casilla franqueable supone un gasto de energía igual al valor que indica la casilla
- Objetivo:
  Llegar al punto S gastando el mínimo de energía
- Estrategia:
  Camino de coste mínimo (Dijkstra)
  Si logramos representar el circuito como un grafo

Algoritmos voraces y el robot desplazándose por un circuito

- Representación del circuito

O	S	O	2	1
3	1	3	1	1
1	6	6	O	6
1	2	O	**R**	4
7	1	1	2	6

Algoritmos voraces y el robot desplazándose por un circuito

- Representación del circuito

De casillas a estados:

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

O	S	O	2	1
3	1	3	1	1
1	6	6	O	6
1	2	O	**R**	4
7	1	1	2	6

Matriz dispersa NO simétrica

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
1	-	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
2	∞	-	∞	∞	∞	3	1	3	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
3	∞	∞	-	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
4	∞	∞	∞	-	1	∞	∞	3	1	1	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
5	∞	∞	∞	2	-	∞	∞	∞	1	1	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
6	∞	0	∞	∞	∞	-	1	∞	∞	∞	1	6	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
7					∞	3	-	3							∞	∞	∞						∞	∞	
	∞	∞	∞								∞	∞	∞						0	-	∞				
21	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	1	2	∞	∞	∞	-	1	∞	∞	∞
22	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	1	2	∞	∞	∞	7	-	1	∞	∞
23	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	2	∞	0	∞	∞	1	-	2	∞
24	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	0	4	∞	∞	1	-	6
25	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	0	4	∞	∞	∞	2	-

Algor...
desp...

• Repres...

De casillas a...

...ersa NO simétrica

```
tipo VectorNat = matriz[0..n] de natural
fun Dijkstra (G = ⟨N,A⟩: grafo): VectorNat, VectorNat
    var
        especial, predecesor: VectorNat
        C: conjunto de nodos
    fvar
    Iniciar conjunto C con todos los nodos 1,2,3,..n excepto el 19
    para i ← 1 hasta n ∧ i ≠ 19 hacer
        especial[i] ← Distancia(19,i)
        predecesor[i] ← 19
    fpara
    mientras C contenga al nodo S hacer {el nodo 2 es S }
        v ← nodo ∈ C que minimiza especial[v]
        C ← C \{v}
        si v ≠ S entonces
            para cada w ∈ C hacer
                si especial[w] > especial[v] + Distancia(v,w) entonces
                    especial[w] ← especial[v] + Distancia(v,w)
                    predecesor[w] ← v
                fsi
            fpara
        fsi
    fmientras
    dev especial[],predecesor[]
ffun
```

Algoritmos voraces y la asistencia a incidencias

- Una empresa de software debe atender las incidencias de sus clientes con la máxima celeridad posible. Cada jornada laboral se planifican n salidas sabiendo de antemano el tiempo que va a llevar atender cada una de las incidencias.
- Objetivo:
  que los clientes esperen lo menos posible a que se resuelva su problema (minimizar el tiempo medio de espera)
- ¿Estrategia?

Algoritmos voraces y la asistencia a incidencias

- Una empresa de software debe atender las incidencias de sus clientes con la máxima celeridad posible. Cada jornada laboral se planifican n salidas sabiendo de antemano el tiempo que va a llevar atender cada una de las incidencias.
- Objetivo:
  que los clientes esperen lo menos posible a que se resuelva su problema (minimizar el tiempo medio de espera)
- Estrategia:
  Este problema es un ejemplo de "Minimización del tiempo en el sistema“ con 1 o más agentes

Ejercicio de examen

Problema (4 puntos). Tras unas lluvias torrenciales, las calles de una ciudad han quedado seriamente dañadas. La institución competente no puede arreglar todas las calles debido al elevado coste que ello supondría, por lo que han decidido volver a pavimentar solo aquellas que les permitan ir de una intersección a otra de la ciudad. Quieren gastarse lo menos posible en la pavimentación, teniendo en cuenta que el coste es directamente proporcional a la longitud de las calles que hay que pavimentar. Desarrolla un algoritmo que permita solucionar de forma óptima este problema con el menor coste.

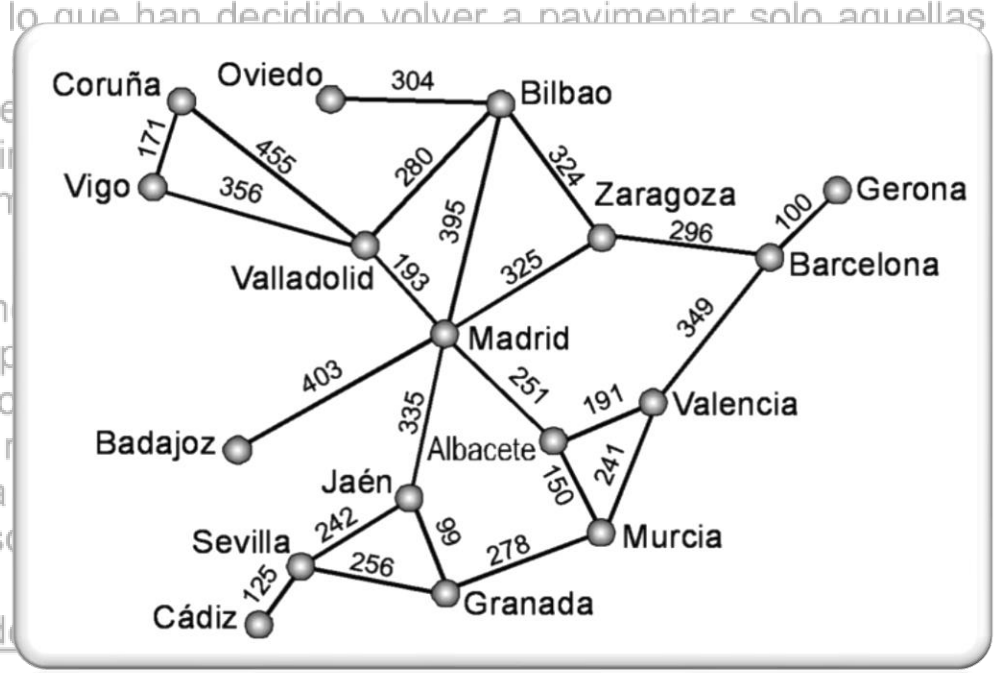
La resolución de este problema debe incluir, por este orden:

1. Elección del esquema más apropiado, el esquema general y explicación de su aplicación al problema (0,5 puntos).
2. Algoritmo <u>completo</u> a partir del refinamiento del esquema general (3 puntos solo si el punto 1 es correcto). Si se trata del esquema voraz debe hacerse la demostración de optimalidad. Si se trata del esquema de programación dinámica deben darse las ecuaciones de recurrencia.
3. Estudio del coste <u>del algoritmo desarrollado</u> (0.5 puntos solo si el punto 1 es correcto).

Ejercicio de examen

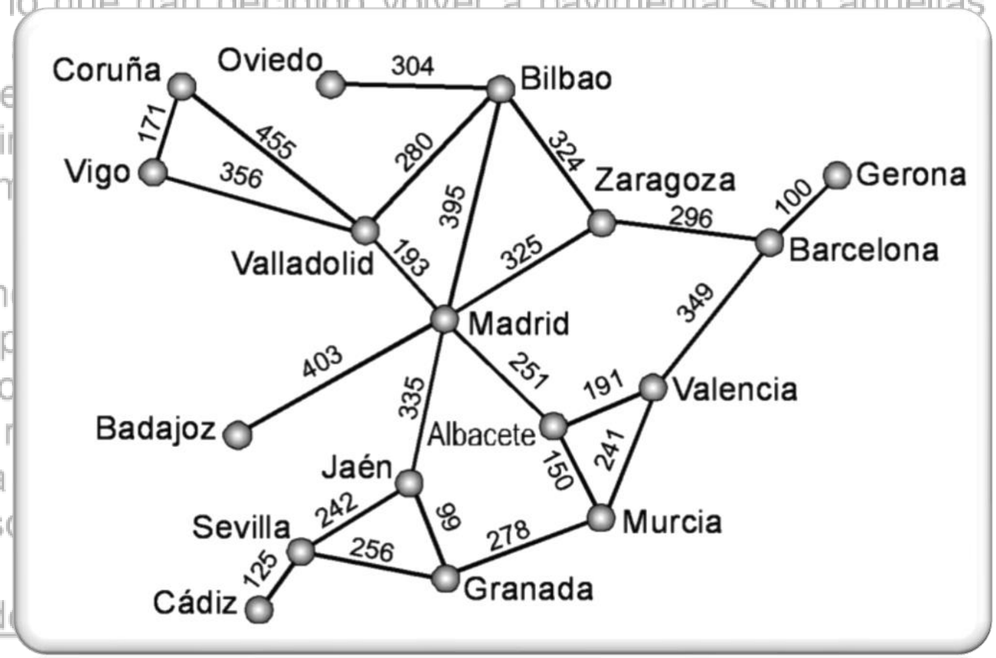
Problema (4 puntos). Tras unas lluvias torrenciales, las calles de una ciudad han quedado seriamente dañadas. La institución competente no puede arreglar todas las calles debido al
or lo que han decidido volver a pavimentar solo aquellas

ón
cue
avi
el n
e in
rop
nto
el r
ata
esc
o d



Ejercicio de examen

Problema (4 puntos). Tras unas lluvias torrenciales, las calles de una ciudad han quedado seriamente dañadas. La institución competente no puede arreglar todas las calles debido al ... or lo que han decidido volver a pavimentar solo aquellas



Árbol de recubrimiento mínimo:
Prim o Kruskal

```
fun Voraz(c: conjuntoCandidatos): conjuntoCandidatos
    sol ← ∅
    mientras c ≠ ∅ ∧ ¬ solucion(sol) hacer
        x ← seleccionar(c)
        c ← c \ {x}
        si factible(sol ∪ {x}) entonces
            sol ← sol ∪ {x}
        fsi
    fmientras
    si solucion(sol) entonces devolver sol
        sino imprimir('no hay solución')
    fsi
ffun
```

Esquema general (Alg. Voraces)

```
fun Prim (G = ⟨N,A⟩: grafo): conjunto de aristas
    AR ← ∅
    NA ← {un nodo cualquiera de N}
    mientras NA ≠ N hacer
        Buscar {u,v} de coste mínimo tal que u ∈ NA y v ∈ N\NA
        AR ← AR ∪ {(u,v)}
        NA ← NA ∪ {v}
    fmientras
    dev AR
ffun
```

Esquema Prim


```
tipo VectorNat = matriz[0..n] de natural
tipo VectorEnt = matriz[0..n] de entero
fun PrimDetallado (G = ⟨N,A⟩: grafo): conjunto de aristas
    var
        nodoMinimo: VectorNat
        costeMinimo: VectorEnt
        AR: conjunto de aristas
    fvar
    AR ← ∅
    costeMinimo[1] ← -1
    para i ← 2 hasta n hacer
        nodoMinimo[i] ← 1
        costeMinimo[i] ← Distancia(1,i)
    fpara
    para i ← 1 hasta n-1 hacer
        min ← ∞
        para j ← 2 hasta n hacer
            si 0 ≤ costeMinimo[j] ∧ costeMinimo[j] < min entonces
                min ← costeMinimo[j]
                nodo ← j
            fsi
        fpara
        AR ← AR ∪ {(nodoMinimo[nodo], nodo)}
        costeMinimo[nodo] ← -1
        para j ← 2 hasta n hacer
            si Distancia(j,nodo) < costeMinimo[j] ∧ costeMinimo[j] ≠ -1 entonces
                costeMinimo[j] ← Distancia(j,nodo)
                nodoMinimo[j] ← nodo
            fsi
        fpara
    fpara
    dev AR
ffun
```

Código Prim

Árbol de recubrimiento mínimo:
Prim o Kruskal

https://algs4.cs.princeton.edu/43mst/PrimMST.java.html

Demostración de optimalidad

- Basada en '**conjunto prometedor**' (CP)
  - Conjunto de aristas factible (sin ciclos) que puede extenderse hasta solución óptima
  - El conjunto vacío lo es
  - Si es ya una solución, esa solución debe ser óptima
- Lema

  Sea **G = (N,A)** un grafo conexo, no dirigido, con pesos ≥ 0. Sea **NA** $\subset$ **N**. Sea **AR** $\subseteq$ **A un CP** de aristas tal que no haya ninguna arista de AR que sale de algún nodo de NA. Sea **(u, v)** la arista de menor peso que salga de NA (o una de ellas si hay varias con igual peso), entonces **AR** $\cup$ **{(u, v)} es un CP**.

  Demostración:

  Sea ARM un árbol de recubrimiento mínimo de G tal que **AR** $\subseteq$ **ARM** (existe ya que AR es CP por hipótesis). <u>Si (u, v) $\in$ ARM no hay nada que demostrar</u>. <u>Si no, al añadir (u, v) a ARM se crea un ciclo</u>. En este ciclo, como (u, v) sale de NA existe necesariamente al menos otra arista (x,y) que también sale de NA o el ciclo no se cerraría. Si eliminamos (x,y) el ciclo desaparece y obtenemos un nuevo árbol ARM' de G. Como la longitud de (u, v), por definición, no es mayor que la longitud de (x,y), la longitud total de las aristas de ARM' no sobrepasa la longitud total de las aristas de ARM. Por lo tanto, ARM' es también un árbol de recubrimiento mínimo de G y contiene a (u, v). Como la arista que se ha eliminado sale de NA, no podría haber sido una arista de AR y se cumple que AR $\subseteq$ ARM'.
- Demostración de optimalidad por inducción:
  - Base: el conjunto vacío es prometedor.
  - Paso inductivo:

    suponemos que AR es un conjunto de aristas prometedor antes de que el algoritmo añada una nueva arista (u, v); NA es un subconjunto estricto de N ya que el algoritmo termina una vez que NA = N; la arista (u, v) es una de las de menor peso de las que salen de NA. Entonces podemos utilizar el lema ya que se cumplen sus condiciones y por lo tanto AR U { (u, v)} también es prometedor. Como AR es prometedor en todos los pasos del algoritmo, cuando el algoritmo se detenga AR contendrá una solución óptima

Demostración de optimalidad

- Basada en '**conjunto prometedor**' (CP)
  - Conjunto de aristas factible (sin ciclos) que puede extenderse hasta solución óptima
  - El conjunto vacío lo es
  - Si es ya una solución, esa solución debe ser óptima
- Lema

  Sea **G = (N,A)** un grafo conexo, no dirigido, con pesos ≥ 0. Sea **NA** $\subset$ **N**. Sea **AR** $\subseteq$ **A un CP** de aristas tal que no hay … la arista de menor peso que salg… … $\cup$ **{(u, v)} es un CP**.

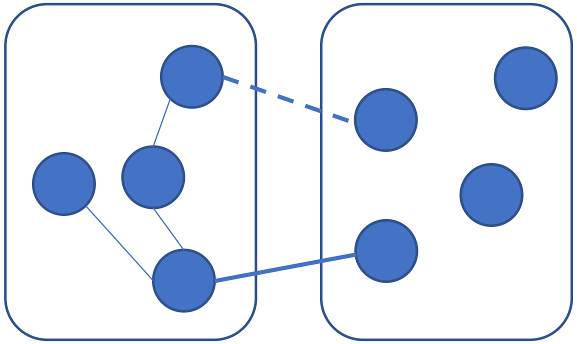
  Demostración:

  Sea ARM un á… … que AR es CP por hipótesis). Si … …ARM se crea un ciclo. En este ciclo, … …a (x,y) que también sale de NA o e… …tenemos un nuevo árbol ARM' de… … longitud de (x,y), la longitud total… …as de ARM. Por lo tanto, ARM' e… …u, v). Como la arista que se ha elim… …mple que AR $\subseteq$ ARM'.

- Demostración de c…
  - Base: el conjunto …
  - Paso inductivo:

  suponemos que AR es un conjunto de aristas prometedor antes de que el algoritmo añada una nueva arista (u, v); NA es un subconjunto estricto de N ya que el algoritmo termina una vez que NA = N; la arista (u, v) es una de las de menor peso de las que salen de NA. Entonces podemos utilizar el lema ya que se cumplen sus condiciones y por lo tanto AR U { (u, v)} también es prometedor. Como AR es prometedor en todos los pasos del algoritmo, cuando el algoritmo se detenga AR contendrá una solución óptima

Lo que estamos demostrando es que si (u, v) no es la arista a coger para formar ARM es porque hay otro ARM' que sí contiene a (u,v) y lo único que ha pasado es que para ARM hemos cogido otra arista (x, y) para formar ARM y es de igual peso que (u,v)



Ejercicio de examen

Problema (4 puntos). Tras unas lluvias torrenciales, las calles de una ciudad han quedado seriamente dañadas. La institución competente no puede arreglar todas las calles debido al elevado coste que ello supondría, por lo que han decidido volver a pavimentar solo aquellas que les permitan ir de una intersección a otra de la ciudad. Quieren gastarse lo menos posible en la pavimentación, teniendo en cuenta que el coste es directamente proporcional a la longitud de las calles que hay que pavimentar. Desarrolla un algoritmo que permita solucionar de forma óptima este problema con el menor coste.

La resolución de este problema debe incluir, por este orden:

1. Elección del esquema más apropiado, el esquema general y explicación de su aplicación al problema (0,5 puntos).
2. Algoritmo completo a partir del refinamiento del esquema general (3 puntos solo si el punto 1 es correcto). Si se trata del esquema voraz debe hacerse la demostración de optimalidad. Si se trata del esquema de programación dinámica deben darse las ecuaciones de recurrencia.
3. Estudio del coste del algoritmo desarrollado (0.5 puntos solo si el punto 1 es correcto).

Coste Alg. Prim:

- Matriz adyacencia: $O(n^2)$
- Listas adyacencia + montículo para candidatos: $O(a \log n)$
  (mejor solo con grafos dispersos: $O(n \log n)$ y no densos: $O(n^2 \log n)$)

Ejercicio de examen

Problema

En la compleja red de metro de Tokyo, la cantidad que se paga por un billete es proporcional a la distancia que se recorre. Por tanto es necesario instalar en cada estación un panel informativo que indique el precio del billete a cualquier otra estación de la red. Describir el algoritmo más eficiente que calcule la información de todos esos paneles, de forma que el viajero se trasladará de una estación a otra por el camino más corto.

Se pide:

a) Elección del esquema **más apropiado**, el esquema general y explicación de su aplicación al problema (0,5 puntos).
b) Descripción de las estructuras de datos necesarias (0,5 puntos solo si el punto 1 es correcto).
c) Algoritmo completo a partir del refinamiento del esquema general (2,5 puntos solo si el punto 1 es correcto). Indicar demostración de optimalidad si es voraz. Indicar las ecuaciones de recurrencia en caso de programación dinámica.
d) Estudio del coste del algoritmo <u>desarrollado</u> (0,5 puntos solo si el punto 1 es correcto).

Ejercicio de examen

Problema

En la compleja red de metro de Tokyo, la cantidad que se paga por un billete es proporcional a la distancia que se recorre. Por tanto es necesario instalar en cada estación un panel informativo que indique el precio del billete a cualquier otra estación de la red. Describir el algoritmo más eficiente que calcule la información de todos esos paneles, de forma que el viajero se trasladará de una estación a otra por el camino más corto.

Se pide:

a) Elección del esquema **más apropiado**, el esquema general y explicación de su aplicación al problema (0,5 puntos).
b) Descripción de las estructuras de datos necesarias (0,5 puntos solo si el punto 1 es correcto).
c) Algoritmo completo a partir del refinamiento del esquema general (2,5 puntos solo si el punto 1 es correcto). Indicar demostración de optimalidad si es voraz. Indicar las ecuaciones de recurrencia en caso de programación dinámica.
d) Estudio del coste del algoritmo desarrollado (0,5 puntos solo si el punto 1 es correcto).

Camino más corto entre cada par de nodos:
Dijkstra n veces (veremos una alternativa con PD: Alg. De Floyd)

tipo VectorNat = matriz[0..n] de natural
fun Dijkstra ($G = \langle N, A \rangle$: grafo): VectorNat
var
especial: VectorNat
C: conjunto de nodos
fvar
$C = \{2, 3, \ldots, n\}$
para i ← 2 **hasta** n **hacer**
especial[i] ← Distancia(1,i)
fpara
mientras C contenga más de 1 nodo **hacer**
v ← nodo ∈ C que minimiza especial[v]
C ← C \ {v}
para cada w ∈ C **hacer**
especial[w] ← min(especial[w], especial[v] + Distancia(v,w))
fpara
fmientras
dev especial[]
ffun

Demostración de optimalidad

La demostración de que el algoritmo de Dijkstra calcula los caminos de menor coste o longitud desde un nodo tomado como origen a los demás nodos del grafo se realiza por inducción. Se trata de demostrar que:

1. Si un nodo $i \neq 1$ está en S, entonces $especial[i]$ almacena la longitud del camino más corto desde el origen, 1, hasta el nodo i.
2. Si un nodo i no está en S, entonces $especial[i]$ almacena la longitud del camino especial más corto desde el origen, 1, hasta el nodo i.

Ver pág. 74

...na estación a

...y explicación

Coste Dijkstra: $O(n^2)$ → n veces: $O(n^3)$

Dijkstra con montículo de mínimos: $O((n + a) \log n)$

- Con grafo disperso: $O(n \log n)$
- Con grafo denso: $O(n^2 \log n)$

si es voraz. Indicar las ecuacion... dinámica.

d) Estudio del coste del algoritmo d... correcto).

Camino más corto entre cada par de nodos:
Dijkstra n veces (veremos una alternativa con PD: Alg. De Floyd)

Ejercicio de examen

Problema (4 puntos). Un proveedor distribuye *n* productos perecederos. Cada producto *i* en el caso de que se venda antes de su fecha de caducidad f_i le permite obtener un beneficio b_i, siendo siempre $b_i > 0$. En un determinado instante, el distribuidor solo puede estar vendiendo uno de los productos. El proveedor quiere saber la forma óptima de organizar la distribución de los productos (orden en el que venderlos) para que el beneficio total sea máximo.

La resolución de este problema debe incluir, por este orden:

1. Elección razonada del esquema más apropiado de entre los siguientes: Voraz, Divide y Vencerás, Vuelta atrás o Ramificación y Poda. Escriba la estructura general de dicho esquema e indique como se aplica al problema (0,5 puntos).
2. Descripción de las estructuras de datos necesarias (0,5 puntos solo si el punto 1 es correcto).
3. Algoritmo completo a partir del refinamiento del esquema general (2.5 puntos sólo si el punto 1 es correcto). Si se trata del esquema voraz debe hacerse la demostración de optimalidad.
4. Estudio del coste del algoritmo desarrollado (0,5 puntos solo si el punto 1 es correcto).

Ejercicio de examen

Problema (4 puntos). Un proveedor distribuye *n* productos perecederos. Cada producto *i* en el caso de que se venda antes de su fecha de caducidad f_i le permite obtener un beneficio b_i, siendo siempre $b_i > 0$. En un determinado instante, el distribuidor solo puede estar vendiendo uno de los productos. El proveedor quiere saber la forma óptima de organizar la distribución de los productos (orden en el que venderlos) para que el beneficio total sea máximo.

La resolución de este problema debe incluir, por este orden:

1. Elección razonada del esquema más apropiado de entre los siguientes: Voraz, Divide y Vencerás, Vuelta atrás o Ramificación y Poda. Escriba la estructura general de dicho esquema e indique como se aplica al problema (0,5 puntos).
2. Descripción de las estructuras de datos necesarias (0,5 puntos solo si el punto 1 es correcto).
3. Algoritmo completo a partir del refinamiento del esquema general (2.5 puntos sólo si el punto 1 es correcto). Si se trata del esquema voraz debe hacerse la demostración de optimalidad.
4. Estudio del coste del algoritmo desarrollado (0,5 puntos solo si el punto 1 es correcto).

Planificación con plazos
(Si el tiempo de ejecución de cada tarea es diferente hay que usar VA o RyP)

Ejercicio de examen

```
fun PlanificacionPlazos (F[1..n], n): Vector[1..n] de natural
    S ← {1}
    para i = 2 hasta n hacer
        si los trabajos en S ∪ {i} constituyen una secuencia factible entonces
            S ← S ∪ {i}
        fsi
    fpara
    dev S
ffun
```

El array $F[\,]$ almacena las fechas tope de realización de los n trabajos ordenados en orden decreciente de beneficios.

Ver algoritmo detallado ($O(n^2)$) y versión mejorada ($O(n \log n)$)

. Cada producto *i* en el
obtener un beneficio b_i,
puede estar vendiendo
anizar la distribución de
máximo.

uientes: Voraz, Divide y
ctura general de dicho

2. Descripción de las estructuras de datos necesarias (0,5 puntos solo si el punto 1 es correcto).
3. Algoritmo com
punto 1 es co
optimalidad.
4. Estudio del cos

Planificación c
(Si el tiempo d

Demostración de optimalidad

Se trata de demostrar que el algoritmo voraz que considera los trabajos en orden decreciente de beneficios, siempre que el conjunto de trabajos sea una solución factible, encuentra una planificación óptima.

S_I :	g	h	i	k	l		
S_J :	l	m	o	g	p	q	i

S_I'' :	k	h		g	l		i
S_J'' :	p	m	o	g	l	q	i

Ver pág. 80

Resumen de ejemplos

- Cambio de monedas (mínimo nº monedas)
  - Ordenar (↓) por valores (deben ser potencias consecutivas de m)
- Árbol de recubrimiento de distancia o coste mínimo
  - Algoritmos de Prim y Kruskal
- Camino de coste mínimo entre un nodo y los demás
  - Algoritmo de Dijkstra
- Minimización del tiempo en el sistema
  - Ordenar (↑) por tiempos de servicio. Generalización para a agentes: reparto “circular”
- Planificación con plazos
  - Ordenar (↓) por beneficios comprobando si es factible
- Almacenamiento óptimo (tiempo de acceso desde el principio + lectura)
  - Ordenar (↑) por longitudes. Generalización para m soportes: reparto “circular”
- Mochila con objetos fraccionables
  - Ordenar (↓) por su ratio valor/peso
- Mantenimiento de la conectividad
- Mensajería urgente
  - Llegar hasta la gasolinera más lejana posible
- Robot por circuito
- Asistencia a incidencias