EDD



AYUDATÓN

PREPARACIÓN I3

Paula Grune - Joaquin Peralta - Alex Infanta - Steven Reynolds - Elías Ayaach



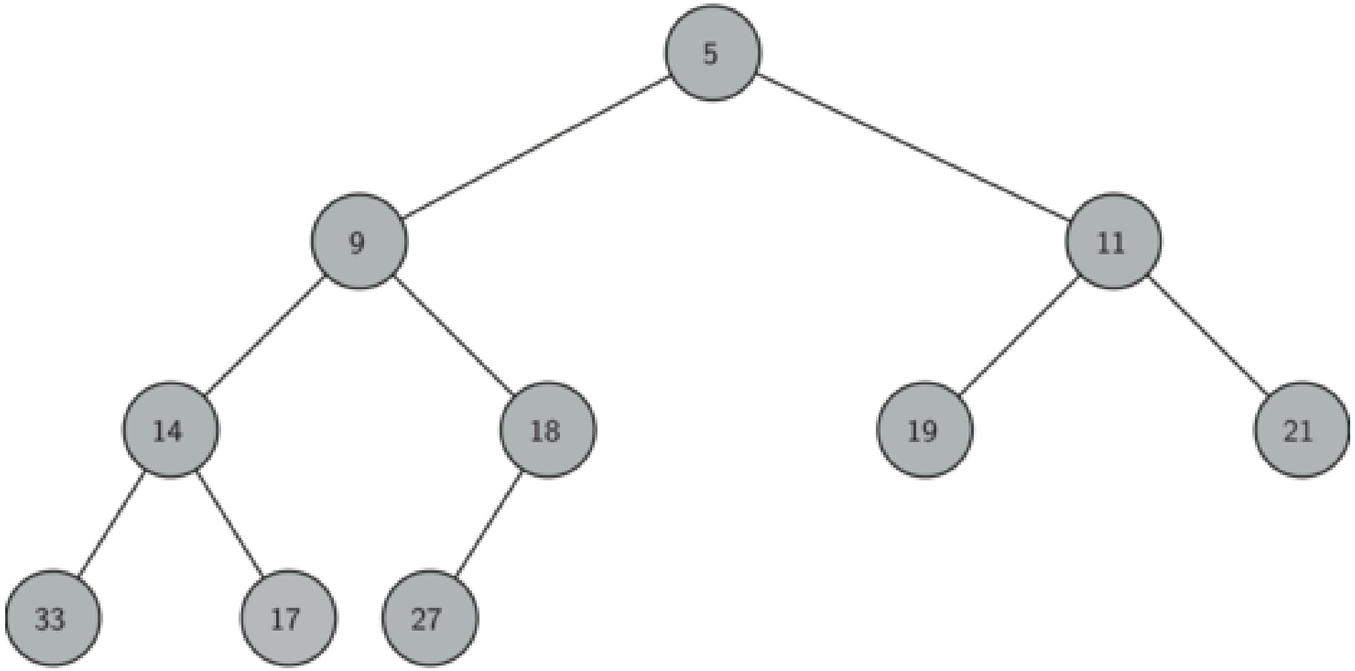








HEAPS



5	9	11	14	18	19	21	33	17	27
0	1	2	3	4	5	6	7	8	9

2025-1 I1 HEAPS



En una farmacia de barrio donde llegan a **lo más 99 clientes al día**, existe un tótem que entrega 3 categorías de turnos, estos son **A de atención normal**, **D de delivery y T de tercera edad**. Cada cliente, cuando llega, saca un ticket (turno) que correspondiente a su categoría con un número que va aumentando en cada categoría (Por ejemplo, A1, A2, T1, D1, A3, etc.). Existen 2 personas atendiendo (caja) y cada vez que terminan de atender, llaman al siguiente turno (Next) que se selecciona según su prioridad, donde T es más prioritario que D y el menor es A.

2025-1 I1 HEAPS

a) Escriba en Pseudocódigo la función Ticket → turno. Fíjese en que se debe seleccionar la categoría

2025-1 I1 HEAPS

a) Escriba en Pseudocódigo la función Ticket → turno. Fíjese en que se debe seleccionar la categoría

Idea a usar: Se implementa un **max heap**. Para determinar la prioridad se define las constantes T=1000, D=100 la prioridad del ticket queda como pT=T+t, pD=D+d y pA=a, con t,d,a los números de atención siguientes en cada categoría.

Solución: Se asume la estructura de UN max heap y la prioridad
a=0; d=0; t=0;

Algorithm 4: `Ticket(c)`

```
1  begin
2      if c=T then
3          t=t+1 Se entrega el próximo turno de la categoría
4          pt=1000+t Se calcula la prioridad
5          InsertHeap(pt) Se inserta en el heap
6          Return ct Se entrega el turno (p. ej. T1)
7      else
8          if c=D then
9              d=d+1
10             pd=100+d
11             Insertheap(pd)
12             Return cd
13         else
14             a=a+1
15             Insertheap(a)
16             Return ca
```

2025-1 I1 HEAPS

b) Escriba en pseudocódigo la función **Next(caja) → turno** donde caja es 1 o 2 que indica el cajero que lo atiende y entrega el número de turno para atención. El caso de borde es que alguna de las categorías no tenga turnos en espera

2025-1 I1 HEAPS

b) Escriba en pseudocódigo la función **Next(caja) → turno** donde caja es 1 o 2 que indica el cajero que lo atiende y entrega el número de turno para atención. El caso de borde es que alguna de las categorías no tenga turnos en espera

Algorithm 5: `Next` (sin parámetros)

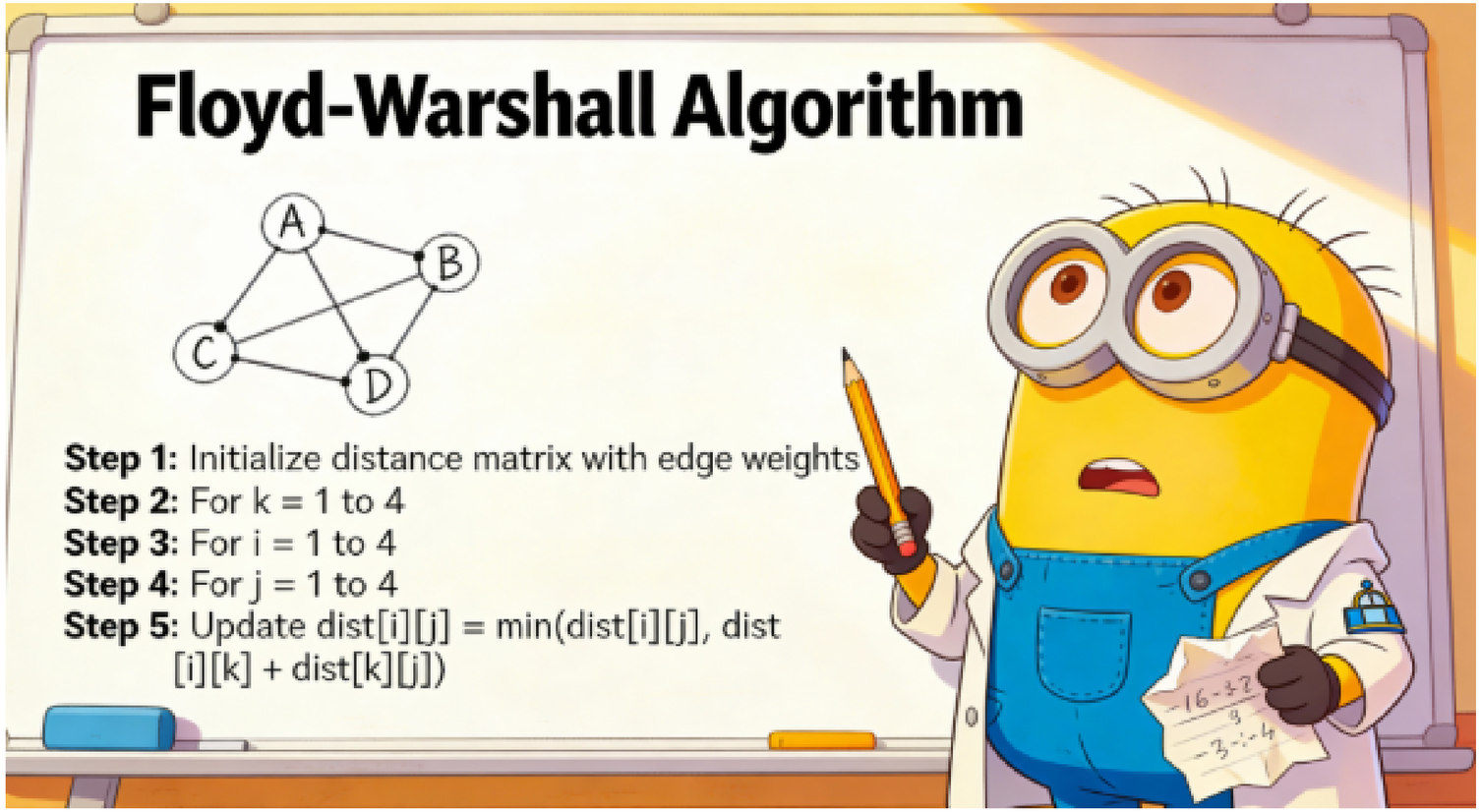
```
1  begin
2      t ← ExtractHeap(H) El heap global se llama H if t = null then
3          return null Si el heap está vacío retorna nulo (opcional)
4      else
5          if t > 1000 then
6              return T || (t/1000)
7          else
8              if 100 < t < 1000 then
9                  return D || (t/100)
10             else
11                 return A || (t)
```

FLOYD WARSHALL

```
FloydWarshall(W):
1     D^(0) ← W
2     for k = 1...n :
3         D^(k) ← (d_ij^(k)) una nueva matriz
4         for i = 1...n :
5             for j = 1...n :
6                 d_ij^(k) ← min( d_ij^(k-1),  d_ik^(k-1) + d_kj^(k-1) )
7     return D^(n)
```

FLOYD-WARSHALL

¿Cómo podemos usar el resultado del algoritmo de Floyd–Warshall para detectar la presencia de un ciclo con costo acumulado negativo (CCAN)? **Justifica**.



FLOYD-WARSHALL

¿Cómo podemos usar el resultado del algoritmo de Floyd–Warshall para detectar la presencia de un ciclo con costo acumulado negativo ? **Justifica**.

Una posibilidad es ejecutar una iteración adicional, con k = n + 1, y ver si alguno de los valores de la matriz D cambia. Si hay **ciclos con costo acumulado negativo**, entonces el costo de alguna ruta más corta deberá disminuir más allá del "**mínimo**" encontrado después de n iteraciones.
Lo otro es mirar los valores de la diagonal de D: hay un **ciclo con costo acumulado negativo** si y sólo si alguno de esos valores es negativo.

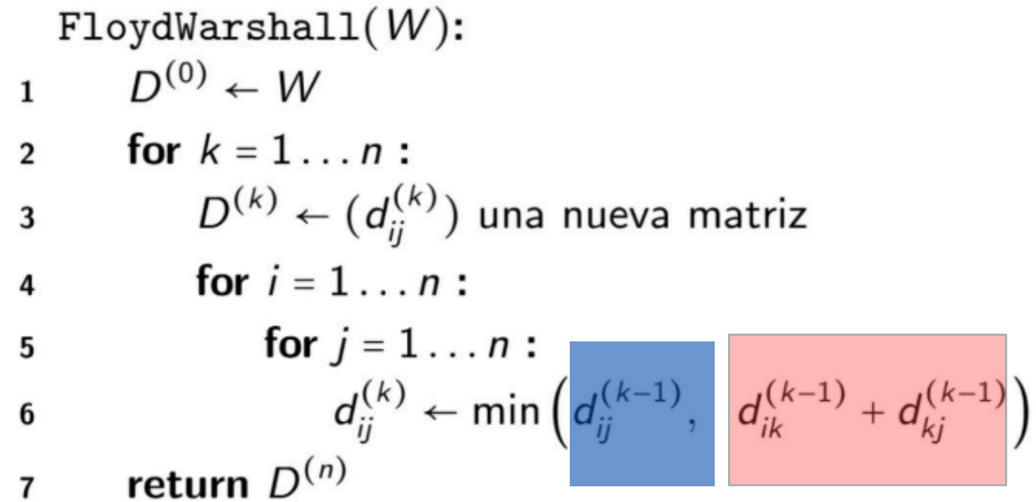
FLOYD-WARSHALL

Explica cómo Floyd–Warshall divide el problema en **subproblemas**

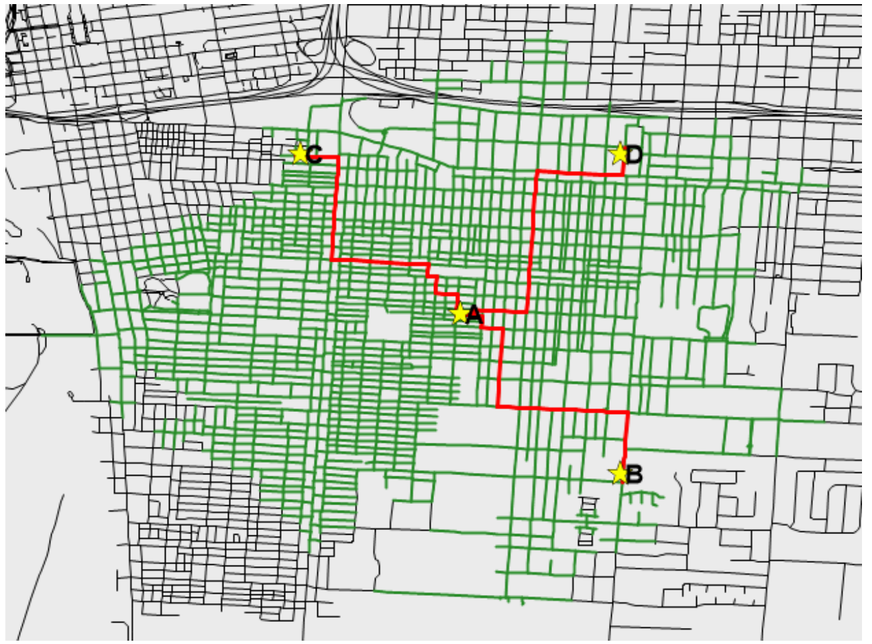
Floyd–Warshall calcula la ruta más corta entre i y j considerando solo los vértices {1,...,k} como intermedios. Para obtener D^(k)[i][j], divide el problema en dos subproblemas más pequeños:

1) **La ruta óptima no usa el vértice k:** Acá simplemente se reutiliza el subproblema anterior.

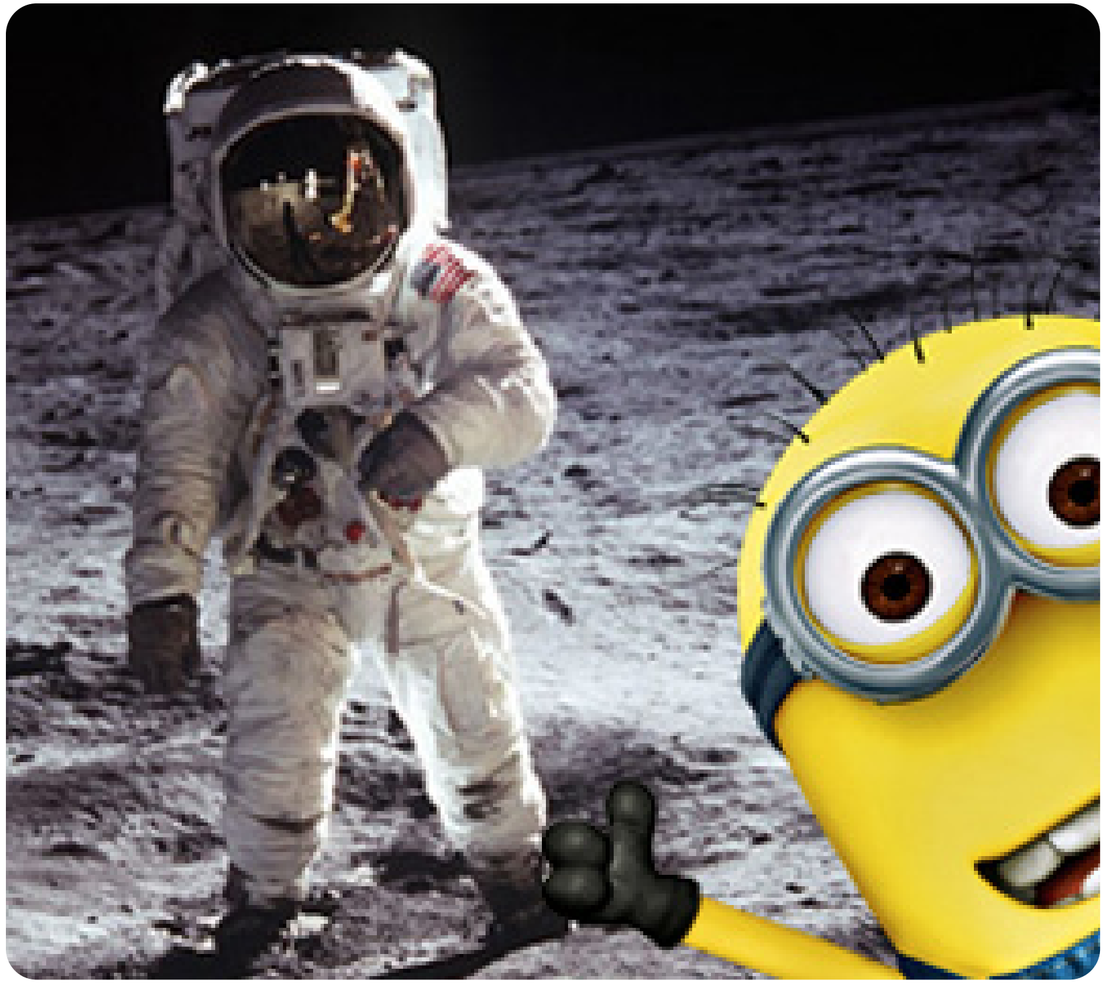
2) **La ruta óptima que usa el vértice k:** Acá se toma el camino de i a k que se asume como subproblema más corto precalculado y de k a j. La unión da el nuevo camino entre i y j



DJIKSTRA-BELLMAN FORD



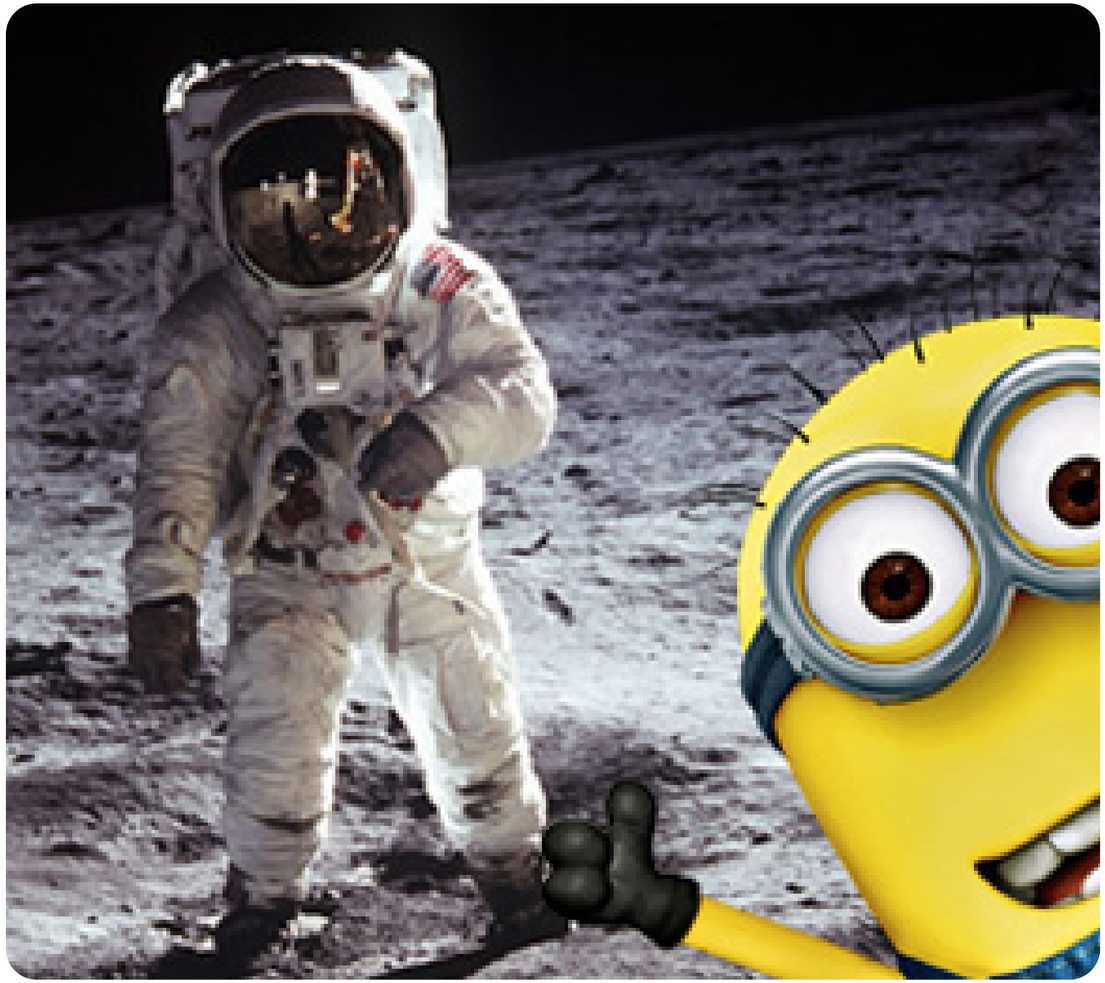
I3 2024-1 CONTEXTO



Un agujero de gusano es un túnel que atraviesa el espacio y el tiempo y conecta dos sistemas estelares. Los agujeros de gusano tienen algunas propiedades peculiares:

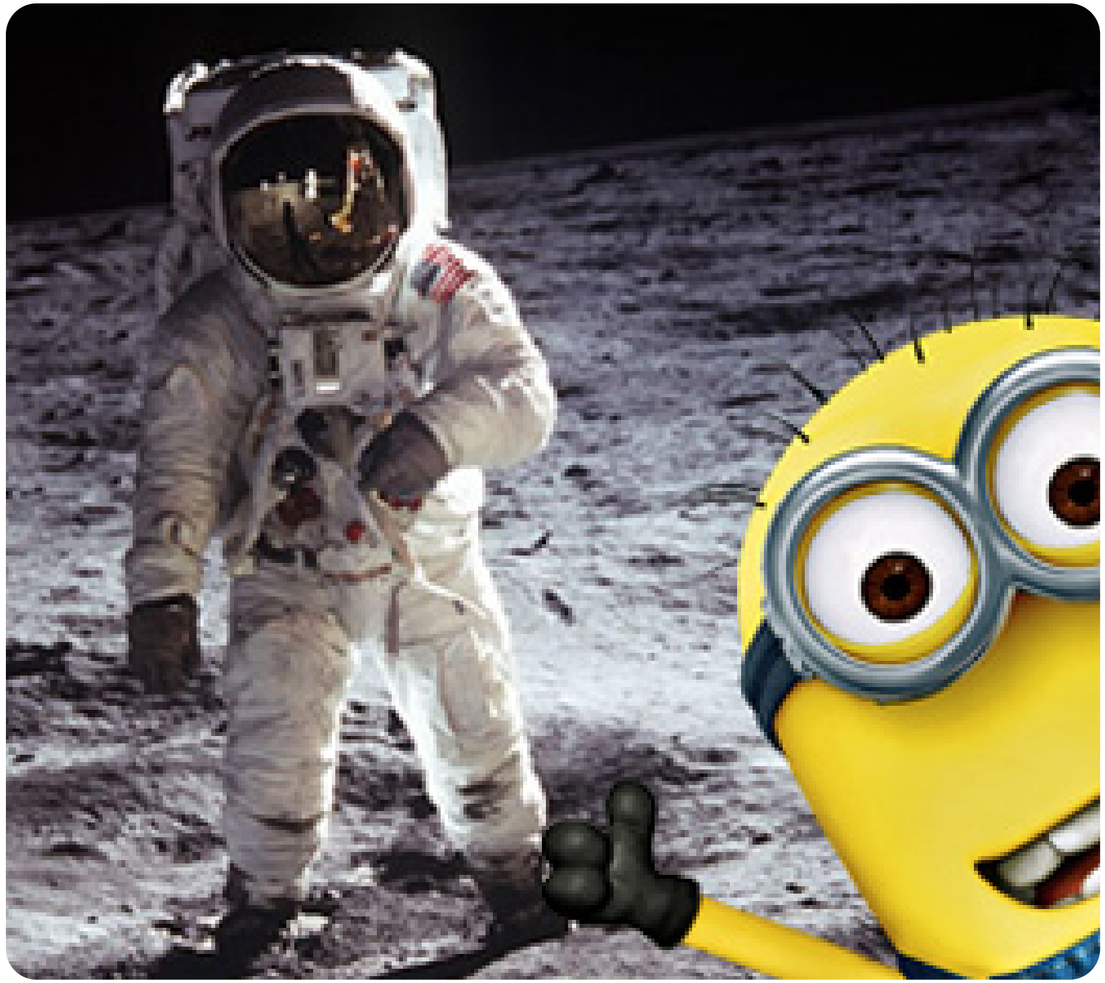
- Cada agujero de gusano es de un único sentido y tiene dos puntos extremos, uno llamado extremo origen (que es donde comienza el agujero de gusano) y el otro llamado extremo destino (que es donde termina el agujero de gusano). Ambos extremos están situados en sistemas estelares distintos.
- El tiempo que se tarda en atravesar un agujero de gusano es insignificante.

I3 2024-1 CONTEXTO



- Un sistema estelar puede ser el punto extremo (origen o destino) de más de un agujero de gusano.
- Por alguna razón desconocida, a partir de nuestro sistema solar, siempre es posible terminar en cualquier sistema estelar siguiendo una secuencia de agujeros de gusano (quizás la Tierra sea el centro del universo).
- Entre cualquier par de sistemas estelares, hay como máximo un agujero de gusano en cualquier dirección.

I3 2024-1 CONTEXTO



Todos los agujeros de gusano tienen una diferencia de tiempo constante entre sus puntos extremos, medido en años usando un valor entero. Por ejemplo, un agujero de gusano específico puede hacer que la persona que lo atraviese viaje **X** años en el futuro. Otro agujero de gusano puede hacer que la persona viaje **Y** años en el pasado. Dado que se asume que el universo se originó con el **Big Bang**, diremos que ése es el tiempo **0. Ningún agujero de gusano puede hacernos retroceder a un instante anterior al Big Bang: si lo intentamos, quedaremos en el tiempo 0.** Alice, una brillante doctora en física, quiere utilizar agujeros de gusano para viajar en el tiempo y estudiar diversos fenómenos.

I3 2024-1 PROBLEMA

Escriba un algoritmo eficiente que, a partir de la representación de un conjunto de sistemas estelares y sus correspondientes agujeros de gusano, le permita a Alice determinar si es posible (o no) viajar a ver el Big Bang. Además, en caso afirmativo, su algoritmo debe indicar al menos una secuencia de agujeros de gusano tal que si se los atravieza, permitan ese viaje.

I3 2024-1 PROBLEMA

(a) Escriba un algoritmo eficiente que, a partir de la representación de un conjunto de sistemas estelares y sus correspondientes agujeros de gusano, le permita a Alice determinar si es posible (o no) viajar a ver el Big Bang. Además, en caso afirmativo, su algoritmo debe indicar al menos una secuencia de agujeros de gusano tal que si se los atravieza, permitan ese viaje.

Idea a Usar: Deben modelar la situación usando un grafo dirigido. Los nodos son los sistemas estelares, las aristas dirigidas son los agujeros de gusano. Las aristas deben tener peso representado por un número entero, indicando la diferencia en años que se produce al seguir un agujero de gusano. El valor del entero será positivo si el agujero de gusano permite viajar en el futuro, o negativo si se viaja al pasado.

Con lo anterior en mente la idea sería determinar si el grafo tiene un ciclo negativo o no, usando el algoritmo de Bellman-Ford, modificado como a continuación para poder obtener el ciclo negativo:

VIENDO EN NUESTRO TORPEDO PARA LA I3...

```
BellmanFord(s):
1    for u ∈ V :
2        d[u] ← ∞;   π[u] ← ∅
3    d[s] ← 0
4    for k = 1 . . . |V| − 1 :
5        for (u, v) ∈ E :
6            if d[v] > d[u] + cost(u, v) :
7                d[v] ← d[u] + cost(u, v)
8                π[v] ← u
```

```
ValidBellmanFord(s):
1    for u ∈ V :
2        d[u] ← ∞;   π[u] ← ∅
3    d[s] ← 0
4    for k = 1 . . . |V| − 1 :
5        for (u, v) ∈ E :
6            if d[v] > d[u] + cost(u, v) :
7                d[v] ← d[u] + cost(u, v)
8                π[v] ← u
9    for (u, v) ∈ E :
10       if d[v] > d[u] + cost(u, v) :
11           return false
12   return true
```

CON LO ANTERIOR EN MENTE LA IDEA SERÍA DETERMINAR SI EL GRAFO TIENE UN CICLO NEGATIVO O NO, USANDO EL ALGORITMO DE BELLMAN-FORD, MODIFICADO COMO A CONTINUACIÓN PARA PODER OBTENER EL CICLO NEGATIVO:

I3 2024-1 PROBLEMA



```
BigBang(s):
    for u ∈ V :
        d[u] ← ∞;   π[u] ← ∅
    d[s] ← 0
    for k = 1 ... |V| − 1 :
        for (u, v) ∈ E :
            if d[v] > d[u] + c(u, v) then
                d[v] ← d[u] + c(u, v)
                π[v] ← u
    for (u, v) ∈ E :
        if d[v] > d[u] + c(u, v) then
            v' ← u
            L ← ∅
            while v' ≠ v :
                L.append(v')
                v' ← π[v']
            return (true, L)
    return (false, ∅)
```

I3 2024-1 PROBLEMA

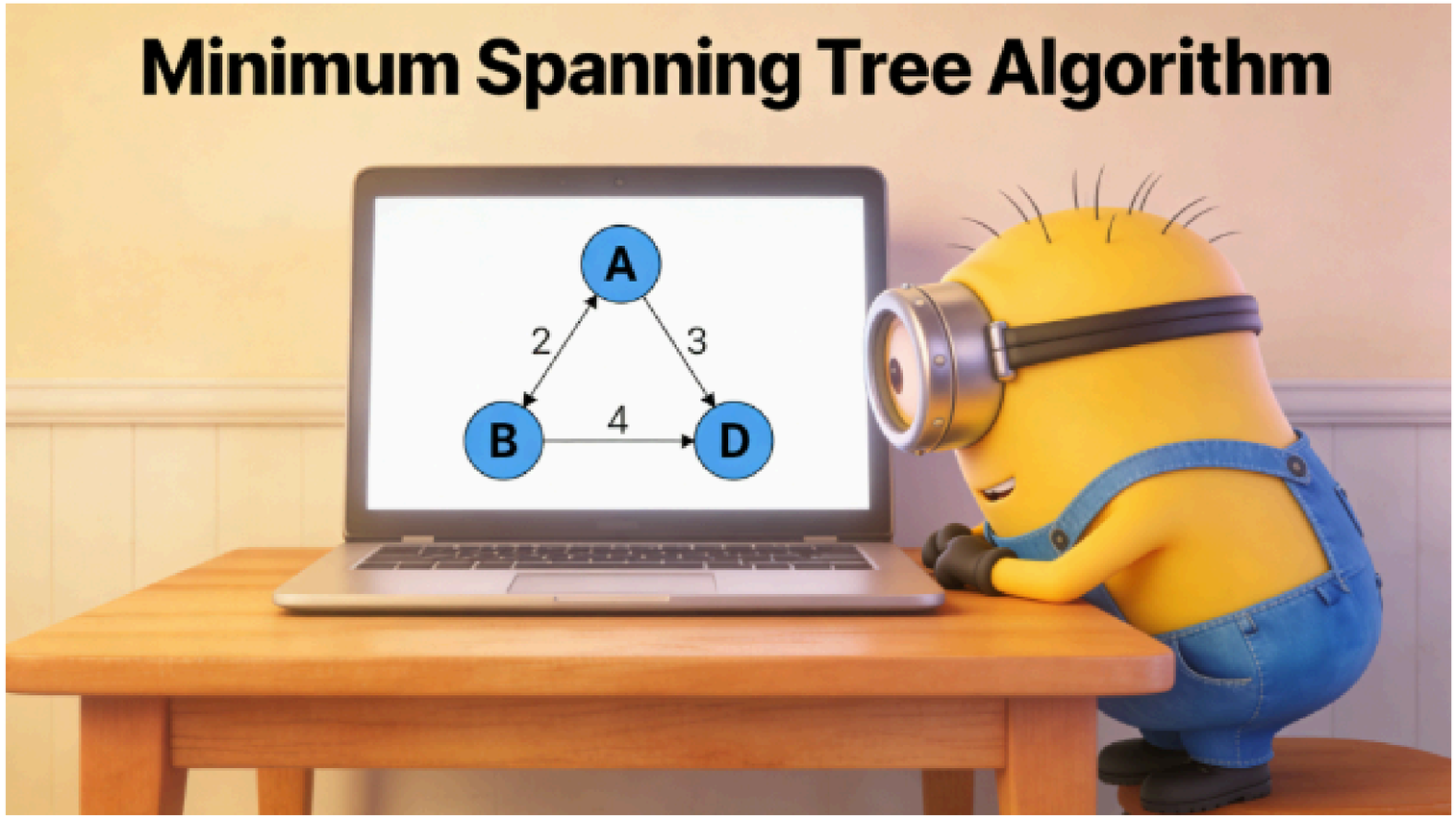
¿Cómo modificaría lo hecho en los items anteriores para obtener un algoritmo que permita determinar si es posible o no avanzar de forma indefinida en el futuro? Indique los detalles (aunque no es necesario escribir pseudocódigo).

I3 2024-1 PROBLEMA

¿Cómo modificaría lo hecho en los items anteriores para obtener un algoritmo que permita determinar si es posible o no avanzar de forma indefinida en el futuro? Indique los detalles (aunque no es necesario escribir pseudocódigo).

Bellman-Ford no sirve en este caso, porque lo que hay que hacer es detectar un ciclo positivo. Sin embargo, se pueden modificar los pesos de las aristas, multiplicando los valores originales por -1 , y así aplicar el algoritmo del item (b).

MST



I3 2023-2 MST

Se propone el siguiente algoritmo con la intención de obtener un MST de un grafo G. Observe que la línea 2 toma aristas de E en orden arbitrario.

(i) Describa qué estructuras de datos utilizar para implementar este algoritmo de forma eficiente. Indique la complejidad de CasiKruskal al usar dichas estructuras.

```
CasiKruskal(G):
1    T ← lista vacía
2    for e ∈ E :
3        if Agregar e a T no forma ciclo :
4            T ← T ∪ {e}
5    return T
```

I3 2023-2 MST

(i) Describa qué estructuras de datos utilizar para implementar este algoritmo de forma eficiente. Indique la complejidad de CasiKruskal al usar dichas estructuras.

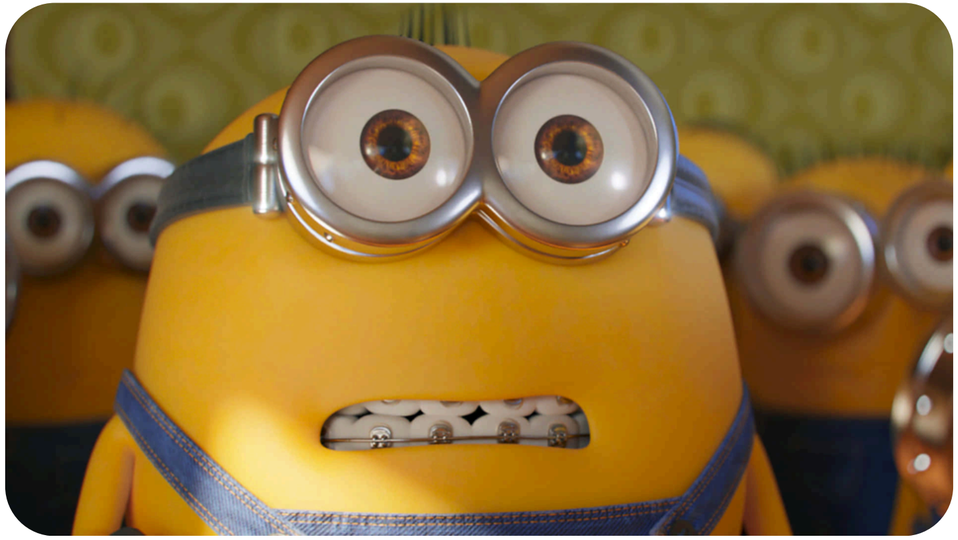
```
CasiKruskal(G):
1     T ← lista vacía
2     for e ∈ E :
3         if Agregar e a T no forma ciclo :
4             T ← T ∪ {e}
5     return T
```

La línea 3 corresponde a la operación más costosa de este algoritmo. **Para determinar eficientemente si agregar una arista forma o no un ciclo**, podemos usar la estructura de conjunto disjunto para almacenar los conjuntos de nodos que están conectados, tal como en Kruskal.

I3 2023-2 MST

(ii) Demuestre que CasiKruskal no retorna un MST para todo grafo G.

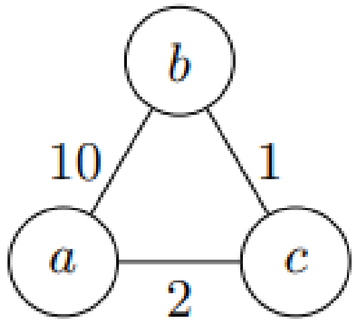
```
CasiKruskal(G):
1     T ← lista vacía
2     for e ∈ E :
3         if Agregar e a T no forma ciclo :
4             T ← T ∪ {e}
5     return T
```



I3 2023-2 MST

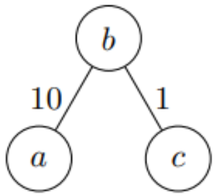
(ii) Demuestre que CasiKruskal no retorna un MST para todo grafo G.

Consideremos el grafo $G = (\{a, b, c\}, \{\{a, b\}, \{b, c\}, \{a, c\}\})$ con los siguientes costos



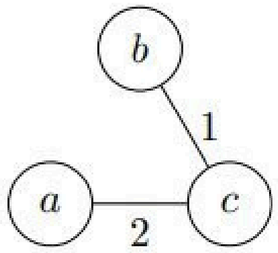
I3 2023-2 MST

Si el orden de iteración de aristas en la linea 2 del algoritmo es {a, b}, {b, c}, {a, c} el árbol obtenido por CasiKruskal es



costo 11

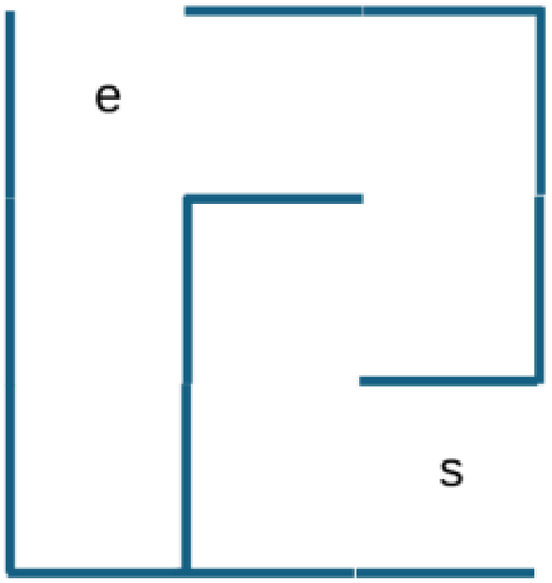
Pero sabemos que el único árbol de cobertura de costo minimo para G es

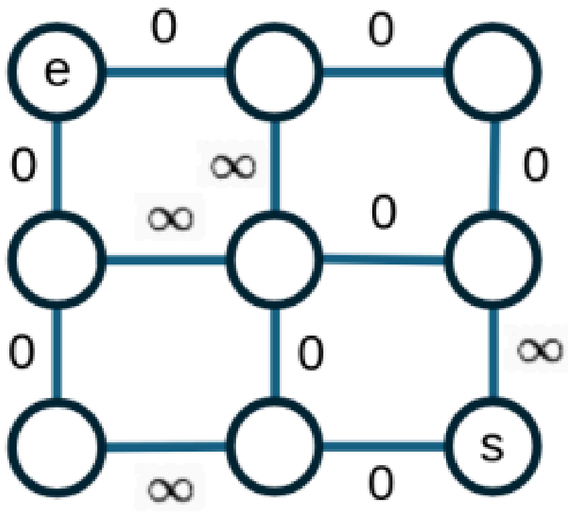


costo 3

I3-2024-1 PRIM

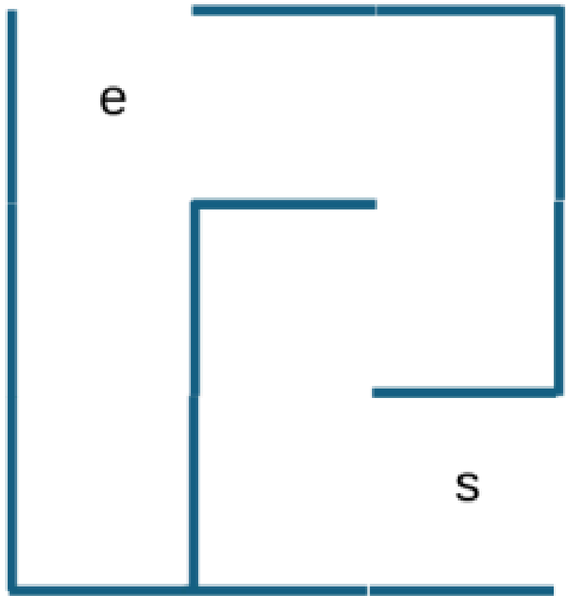
Un laberinto se puede representar como un grafo conexo en que los nodos representan las celdas y las aristas conectan a las celdas con sus vecinas, dos celdas conectadas en el laberinto son unidas por una arista de costo 0 y un muro entre dos celdas es representado por una arista de costo ∞. Por ejemplo:

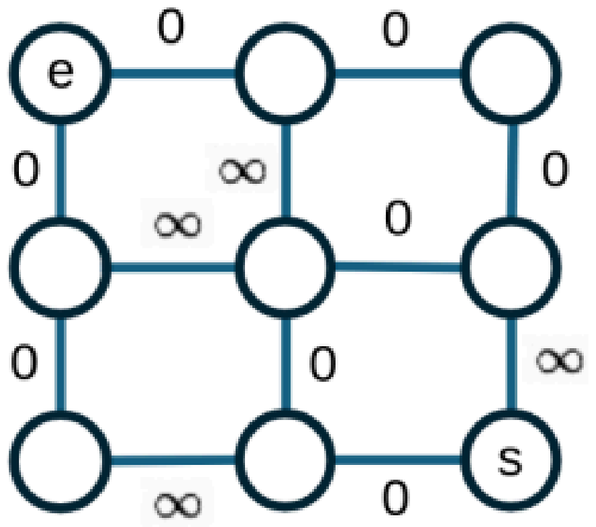




I3-2024-1 PRIM

(a) En este escenario, ¿es posible utilizar el **algoritmo de Prim** para determinar el camino más corto desde un nodo de partida e hasta el nodo de salida s del laberinto? Argumente su respuesta.

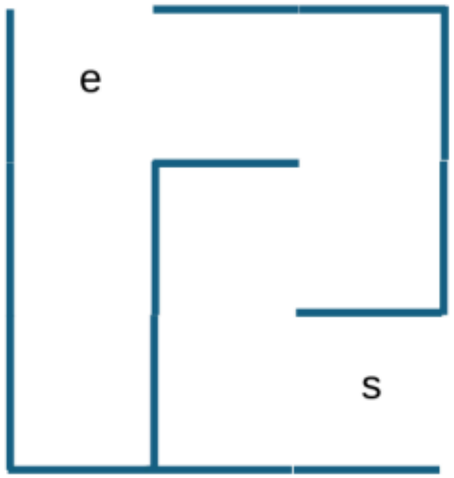


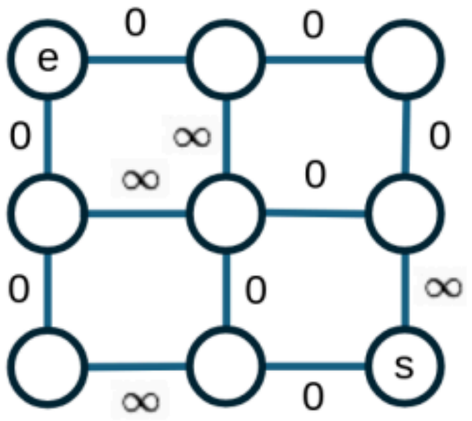


I3-2024-1 PRIM

(a) En este escenario, ¿es posible utilizar el algoritmo de Prim para determinar el camino más corto desde un nodo de partida e hasta el nodo de salida s del laberinto? Argumente su respuesta.

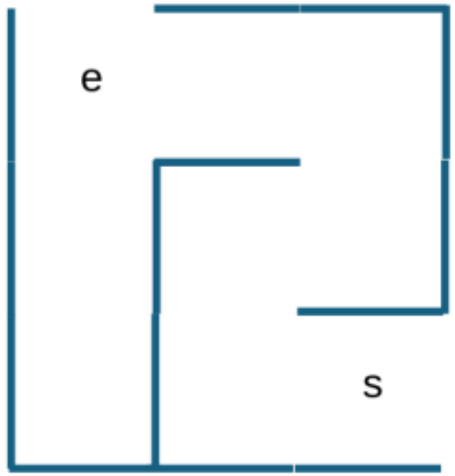
Solución: Si, es posible. Dado que Prim permite **determinar el MST de un grafo conexo**, en particular este MST conecta a costo mínimo a e y s, utilizando las aristas de costo cero.

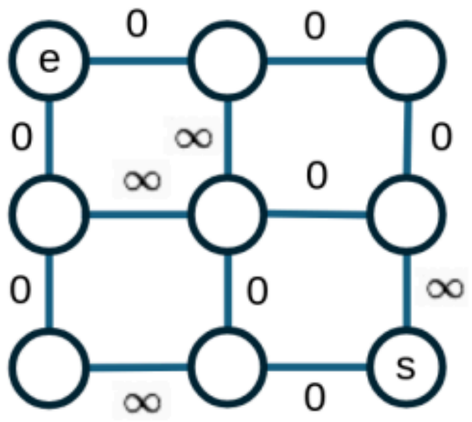




I3-2024-1 PRIM

b) Algunos juegos requieren crear laberintos de manera dinámica. Modifique el algoritmo de Prim (entregue el pseudocódigo) para crear un laberinto sin ciclos a partir de un grafo G con un nodo de partida e y un nodo de salida s. Hint: Puede suponer que inicialmente todas las celdas del grafo G están rodeadas por muros.





I3-2024-1 PRIM

Solución: La solución asume que el grafo tiene inicialmente todas las aristas de costo ∞.

```
   Prim(s):
1      Q ← cola de prioridades con s
2      T ← lista vacía
3      x.key ← 0;   x.parent ← ∅
4      while Q no está vacía :
5          u ←Extract(Q);  u.color ← negro
6          if u.parent ≠ ∅ then
7              T ← T ∪ {(u.parent, u)}
8          for v ∈ α[u] ∧ v.color ≠ negro :
9              if v ∉ Q then
10                 Insert(Q, v)
11                 v.key ← 0
12                 v.parent ← u
13                 DecreaseKey(Q, v, v.key)
14     return T
```

Los nodos en Q son aquellos que ya fueron conectados (linea 5). Luego en la línea 9 los que no estaban en Q son nodos que no generan ciclos, entonces los agregamos Q y botamos el muro que lo separa de u en la linea 11 y lo conectamos, ajustando su prioridad en el heap en la linea 13.


i3
Wish You
Success!