FACULTAD DE INGENIERÍA
Universidad de Buenos Aires

75.29 / 95.06 Teoría de Algoritmos

Trabajo Práctico Final

Teoría de Logaritmos

Fecha de Entrega: 19 de Enero de 2024

Integrantes

Alumno	Padrón
Joaquin Andresen	102707
Facundo Pareja	99719

Índice

Parte 1: Turnos en la fábrica

1. Expresar y explicar el problema como un problema de programación lineal.

El objetivo del problema es minimizar la cantidad de empleados asignados a los turnos, cumpliendo con los requerimientos de cada franja horaria.

Variables de decisión

x_i será el número de empleados que inician su turno en el horario i. Tendremos entonces n = 6 variables:

- x_1: empleados que inician su turno a las 00:00
- x_2: empleados que inician su turno a las 04:00
- x_3: empleados que inician su turno a las 08:00
- x_4: empleados que inician su turno a las 12:00
- x_5: empleados que inician su turno a las 16:00
- x_6: empleados que inician su turno a las 20:00

Función objetivo

Debemos minimizar la cantidad de empleados:

$$Minimizar\ Z = x_1 + x_2 + x_3 + x_4 + x_5 + x_6$$

Restricciones

Para definir las restricciones de nuestro problema debemos tener en cuenta dos cosas:

- Cada franja horaria debe contar con un mínimo número de empleados
- Los turnos son de 8 horas. Esto quiere decir que un empleado cubrirá dos franjas.

Por ejemplo, si queremos definir una restricción para la franja horaria que va de 00:00 a 04:00 tenemos que tener en cuenta a los empleados que ingresan a las 00:00 y trabajan hasta 08:00 pero también a los que ingresan a las 20:00 y trabajan hasta 04:00.

Entonces la restricción sería:

$$x_1 + x_6 \geq 15$$

Siguiendo el mismo razonamiento, las restricciones que tendríamos serán las siguientes:

$$x_1 + x_6 \geq 15 \text{ (00:00 a 04:00)}$$
$$x_1 + x_2 \geq 35 \text{ (04:00 a 08:00)}$$
$$x_2 + x_3 \geq 65 \text{ (08:00 a 12:00)}$$
$$x_3 + x_4 \geq 80 \text{ (12:00 a 16:00)}$$
$$x_4 + x_5 \geq 40 \text{ (16:00 a 20:00)}$$
$$x_5 + x_6 \geq 25 \text{ (20:00 a 00:00)}$$

Además, las variables deben ser positivas.

$$x_1, x_2, x_3, x_4, x_5, x_6 \geq 0$$

Forma estándar

Maximizar

$$(-Z) = -x_1 - x_2 - x_3 - x_4 - x_5 - x_6$$

Sujeto a

$$-x_1 - x_6 \leq -15$$
$$-x_1 - x_2 \leq -35$$
$$-x_2 - x_3 \leq -65$$
$$-x_3 - x_4 \leq -80$$
$$-x_4 - x_5 \leq -40$$
$$-x_5 - x_6 \leq -25$$
$$x_1, x_2, x_3, x_4, x_5, x_6 \geq 0$$

2. Determinar si el problema se puede resolver en tiempo polinomial. Justificar.

En teoría, el algoritmo Simplex utilizado para resolver este problema no puede garantizar que se obtenga una respuesta en tiempo polinomial (ver complejidad temporal obtenida en el punto 4).

En la práctica, la mayoría de los problemas se resuelven rápidamente utilizando Simplex.

Sin embargo, existen algoritmos que sí podrían resolverlo en tiempo polinomial, como el método elipsoide de Khachiyan o el método del punto interior de Karmarkar.

3. Explicar como resolver el problema utilizando Simplex. Brindar pseudocódigo y explicación de su propuesta. Expresar como se representa la solución.

Para resolver el problema necesitamos transformar el problema a la forma de holgura. Por lo tanto vamos a agregar las variables slack $x_7, x_8, x_9, x_{10}, x_{11}, x_{12}$.

Forma holgura

Maximizar

$$(-Z) = -x_1 - x_2 - x_3 - x_4 - x_5 - x_6$$

Sujeto a

$$x_7 = -15 + x_1 + x_6$$

$$x_8 = -35 + x_1 + x_2$$

$$x_9 = -65 + x_2 + x_3$$

$$x_{10} = -80 + x_3 + x_4$$

$$x_{11} = -40 + x_4 + x_5$$

$$x_{12} = -25 + x_5 + x_6$$

$$x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}, x_{12} \geq 0$$

Para obtener la solución básica las variables de decisión $x_1, x_2, x_3, x_4, x_5, x_6$ comenzaran en cero, nos quedará el siguiente vector:

(0,0,0,0,0,0,-15,-35,-65,-80,-40,-25)

Aquí encontramos un problema, ya que la solución (0,0,0,0,0,0) no es una solución factible del problema. Por ejemplo, si reemplazamos en la primera restricción veremos que no se cumple:

$$0 \leq -15$$

Por lo tanto, vamos a tener que crear un problema auxiliar que nos permita encontrar un punto inicial para realizar Simplex en nuestro problema.

Este problema auxiliar va a consistir en maximizar una nueva variable x_0 con coeficiente -1 y agregaremos esa variable a todas las restricciones.

Maximizar

$$Z = -x_0$$

Sujeto a

$$\begin{aligned}
-x_1 - x_6 - x_0 &\leq -15 \\
-x_1 - x_2 - x_0 &\leq -35 \\
-x_2 - x_3 - x_0 &\leq -65 \\
-x_3 - x_4 - x_0 &\leq -80 \\
-x_4 - x_5 - x_0 &\leq -40 \\
-x_5 - x_6 - x_0 &\leq -25 \\
x_0, x_1, x_2, x_3, x_4, x_5, x_6 &\geq 0
\end{aligned}$$

Pasamos a la forma de holgura

Maximizar

$$Z = -x_0$$

Sujeto a

$$x_7 = -15 + x_1 + x_6 + x_0$$

$$x_8 = -35 + x_1 + x_2 + x_0$$

$$x_9 = -65 + x_2 + x_3 + x_0$$

$$x_{10} = -80 + x_3 + x_4 + x_0$$

$$x_{11} = -40 + x_4 + x_5 + x_0$$

$$x_{12} = -25 + x_5 + x_6 + x_0$$

$$x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}, x_{12} \geq 0$$

Como no tenemos variables con coeficiente positivo en la función objetivo, vamos a tener que realizar un pivot especial antes de comenzar a aumentar variables. Se seleccionara como variable de salida a la que tenga el valor más negativo.

Una vez que terminemos de realizar Simplex sobre el problema auxiliar, usaremos la solución obtenida como valor inicial para el problema original.

Pseudocódigo

```
función simplex(función objetivo, variables de decisión)
       Mientras exista un coeficiente positivo en la función objetivo
              Seleccionamos una variable de salida vs entre las básicas con
coeficiente positivo en la función objetivo

              Determinamos la variable de entrada ve cuya restricción ajusta a vs

              Si vs no se ajusta
                     retorno "es una problema ilimitado"

              Realizo pivot entre vs y ve

       Retornamos el valor de las variables de decisión ,el valor de la función
objetivo y las restricciones alteradas

función main()
       Transformar el problema original a la forma de holgura
       Generar problema auxiliar, variables auxiliares y restricciones auxiliares

       Elegimos xo como nuestra variable de salida vs
       Seleccionar variable ve que ajuste vs
       Realizar pivot entre vs y ve
       Realizó simplex(función auxiliar, variables auxiliares, restricciones
auxiliares)

       Elimino xo de las restricciones auxiliares retornadas y reacomodo la
función objetivo original
       Realizó simplex(función objetivo reacomodada, variables auxiliares,
restricciones auxiliares)

       Retornamos el resultado de simplex
```

4.Analizar la complejidad temporal y espacial de cada uno de los pasos de su solución.

Complejidad Temporal

- La transformación a forma holgada se puede realizar en O(n) para las m restricciones, por lo tanto nos queda O(mn)
- La búsqueda de coeficientes positivos se puede hacer en O(n)

- El cálculo para ajustar la variable se hace en O(1) y se debe hacer m veces, por lo tanto queda O(m)
- El proceso de pivot requiere O (n*m)
- El proceso de simplex se repite para $\binom{n+m}{n}$ puntos
- La eliminación de x_0 y el reacomodo de la función objetivo se puede realizar en O(n), lo cual no afecta la complejidad final

La complejidad temporal final en el peor de los casos será $O(2^n n * m)$

Complejidad Espacial

- En la transformación a forma holgada agregamos m variables, O(m)
- Las variables de entrada y salida requieren O(1)
- El registro de variables básicas y libres requiere O(m + n)
- Los coeficientes de las variables básicas se guardarán en una matriz que requiere O(m*n)

Por lo tanto, la complejidad espacial será de $O(m * n)$

5. Presente paso a paso la solución del problema.

Primero vamos a resolver el problema auxiliar.

Maximizar

$Z = -x_0$

Sujeto a

$x_7 = -15 + x_1 + x_6 + x_0$

$x_8 = -35 + x_1 + x_2 + x_0$

$x_9 = -65 + x_2 + x_3 + x_0$

$x_{10} = -80 + x_3 + x_4 + x_0$

$x_{11} = -40 + x_4 + x_5 + x_0$

$x_{12} = -25 + x_5 + x_6 + x_0$

$x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}, x_{12} \geq 0$

Primero debemos realizar un pívot, ya que no tenemos coeficientes positivos. Elegimos x_{10} ya que nos da el valor más negativo posible (-80). De esta forma nos quedaría:

Maximizar

$$Z = -80 + x_3 + x_4 - x_{10}$$

Sujeto a

$$x_0 = 80 - x_3 - x_4 - x_{10}$$

$$x_7 = 65 + x_1 - x_3 - x_4 + x_6 + x_{10}$$

$$x_8 = 45 + x_1 + x_2 - x_3 - x_4 + x_{10}$$

$$x_9 = 15 + x_2 - x_4 + x_{10}$$

$$x_{11} = 40 + x_5 - x_3 + x_{10}$$

$$x_{12} = 55 - x_3 - x_4 + x_5 + x_6 + x_{10}$$

$$x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}, x_{12} \geq 0$$

Continuamos el proceso de Simplex eligiendo la variable con mayor coeficiente positivo de la función objetivo, en este caso elijo x_4 y realizó pivot con x_9 , que es la variable que la ajusta.

Maximizar

$$Z = -65 + x_2 + x_3 - x_9$$

Sujeto a

$$x_0 = 65 - x_2 - x_3 + x_9$$

$$x_4 = 15 + x_2 - x_9 + x_{10}$$

$$x_7 = 50 + x_1 - x_2 - x_3 + x_6 + x_9$$

$$x_8 = 30 + x_1 - x_3 + x_9$$

$$x_{11} = 40 - x_3 + x_5 + x_{10}$$

$$x_{12} = 40 - x_2 - x_3 + x_5 + x_6 + x_9$$

$$x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}, x_{12} \geq 0$$

Repito los pasos hasta llegar a un punto donde no tenga más coeficientes positivos en la función objetivo. En nuestro ejemplo realizaremos los siguientes pivots:

- x_0 por x_{10} (realizado anteriormente)
- x_4 por x_9 (realizado anteriormente)
- x_2 por x_{12}
- x_5 por x_7
- x_1 por x_0

Una vez realizados los pivots, el problema auxiliar tendrá la siguiente forma:

Maximizar

$Z = -x_0$

Sujeto a

$$x_1 = 15 - x_0 + x_5 - x_6 + x_7$$

$$x_2 = 65 - x_0 - x_3 + x_9$$

$$x_4 = 80 - x_0 - x_3 - 2x_6 + x_{10}$$

$$x_5 = 25 - x_0 - x_6 + x_{12}$$

$$x_8 = 45 - x_0 - x_3 + x_5 - x_6 + x_7 + x_9$$

$$x_{11} = 65 - x_0 - x_3 - x_6 + x_{10} + x_{12}$$

$$x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}, x_{12} \geq 0$$

Nos queda como solución del problema auxiliar $(x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}, x_{12})$ = (0, 40, 65, 0, 80, 25, 0, 0, 70, 0,0, 65, 0). Esta vez si se cumplen las restricciones, por lo tanto tenemos una solución factible del problema original.

Para continuar, quitamos la variable x_0 de las restricciones y acomodamos la función objetivo original para que solo contenga variables básicas.

Maximizar

$$Z = (15 + x_5 - x_6 + x_7) + (65 - x_3 + x_9) + x_3 + (80 - x_3 - 2x_6 + x_{10}) + (25 - x_6 + x_{12}) + x_6$$

$$Z = 185 - x_3 + x_5 - 3x_6 + x_7 + x_9 + x_{10} + x_{12}$$

Sujeto a

$$x_1 = 15 + x_5 - x_6 + x_7$$

$$x_2 = 65 - x_3 + x_9$$

$$x_4 = 80 - x_3 - 2x_6 + x_{10}$$

$$x_5 = 25 - x_6 + x_{12}$$

$$x_8 = 45 - x_3 + x_5 - x_6 + x_7 + x_9$$

$$x_{11} = 65 - x_3 - x_6 + x_{10} + x_{12}$$

$$x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}, x_{12} \geq 0$$

Ahora solo queda volver a realizar Simplex para obtener la solución óptima. En este caso, dependiendo de los pivots que realicemos, podremos encontrar múltiples soluciones óptimas. Por ejemplo, nosotros llegamos a ($x_1, x_2, x_3, x_4, x_5, x_6$) = (35,0,65,15,25,0), donde Z es igual a 140, pero el programa que desarrollamos encontró otra solución con el mismo valor objetivo.

6. Programe su propuesta. Puede utilizar librerías para Simplex. (Ejemplos: ALGLIB o PulP)

Para resolver el problema utilizamos la librería PulP. Esta nos permitió resolver el problema original de minimización, pero también nos permitió revisar los resultados del problema auxiliar y el original de maximización.

El programa **simplex.py** resuelve el problema original de maximización y al finalizar retorna el valor objetivo óptimo y los valores óptimos de cada variable de decisión. El output se ve así:

```
Valor óptimo de Z: 140.0
Valor óptimo de x1: 0.0
Valor óptimo de x2: 35.0
Valor óptimo de x3: 30.0
Valor óptimo de x4: 50.0
Valor óptimo de x5: 10.0
Valor óptimo de x6: 15.0
```

Si vemos las restricciones originales, podemos ver que todas se cumplen.

Parte 2: Profundización iterativa

1. Indicar que tipo de algoritmo es y para que se puede utilizar

Iterative Deepening Depth-First Search (IDDFS) [1]

Es un algoritmo de **búsqueda** para grafos, pues analiza cada nodo posible para un nivel de profundidad antes de incrementar el nivel, y repite esto hasta encontrar el nodo deseado. Este algoritmo está pensado como una mejora de BFS y DFS que evita dos problemas grandes de estos, a saber:

- Si bien BFS garantiza encontrar una solución, tiene una complejidad espacial exponencial
- DFS puede correr infinitamente en grafos infinitos o con ciclos.

Por lo tanto, conviene utilizar IDDFS para hallar las rutas más cortas entre dos nodos en grafos de gran tamaño (incluso infinitos) o con ciclos.

Iterative Deepening A*

Es un algoritmo de **búsqueda** para grafos pesados, resultante de combinar IDDFS con una heurística que permite decidir qué nodo conviene explorar mediante estimación. Este algoritmo está pensado como una mejora de A* que usa una cantidad mínima de memoria comparado con este, a cambio de potencialmente repetir exploración de nodos ya explorados. Se utiliza para hallar las rutas más cortas entre dos nodos en grafos **pesados** de gran tamaño (incluso infinitos) o con ciclos donde el requisito de memoria para aplicar el algoritmo A* es demasiado elevado.

2. Brindar pseudocódigo y explicación de cómo funciona.

Iterative Deepening Depth-First Search

```
Sea raíz el nodo de partida
Sea objetivo el nodo que se desea alcanzar

funcion busqueda_IDFS (raíz)
       para profundidad ∈ [0, inf]:
              encontrados, hay_mas_nodos = DLS(raíz, profundidad)
              si encontrados != vacio:
                     devolver encontrados
              sí no:
                     si hay_mas_nodos == Falso:
                            devolver vacio

funcion DLS(nodo, profundidad):
       si profundidad = 0:
              si nodo == objetivo:
                     devolver (nodo, Verdadero)
              si no:
```

```
                devolver (vacío, Verdadero)
    si profundidad > 0:
          restan_hijos = Falso
          para cada hijo del nodo actual:
                encontrados, hay_mas_nodos <- DLS(hijo, profundidad-1)
                si encontrados != vacio:
                      return (encontrados, Verdadero)
                si hay_mas_nodos == Verdadero:
                      restan_hijos = Verdadero
          return (vacio, restan_hijos)
```

El algoritmo IDDFS consiste básicamente en la aplicación repetida de una búsqueda en profundidad (DFS) aumentando progresivamente la profundidad máxima permitida (todos los nodos hasta el nivel 1, todos los nodos hasta el nivel 2, etc.). El pseudocódigo se explica analizando cada función por separado:

- La función `DLS` es una implementación recursiva y limitada en profundidad de DFS. Se puede considerar dividida en dos partes: una relacionada con la condición de corte y otra más general.
    - **Condición de corte**: para una profundidad = 0, o sea si se alcanzó el límite permitido para la profundidad, se verifica si se alcanzó el objetivo. En caso positivo se lo devuelve. En caso negativo, se devuelve un conjunto vacío. En ambos casos se indica que se puede continuar.
    - En caso de que no se haya alcanzado la condición de corte, se exploran los hijos recursivamente. Para cada hijo se llama a DLS nuevamente, pero con profundidad - 1. Esto sigue hasta que se llegue a la condición de corte eventualmente, y en caso de alcanzarse el objetivo, se lo devuelve. En caso contrario, solo se sigue iterando explorando si hay hijos. En caso de que no haya más nodos se lo indica a la función llamante.
- La función `busqueda_IDFS` llama a DLS para cada nivel posible de profundidad, de 0 a infinito. Si se encuentra el nodo objetivo, se lo devuelve, y en caso negativo, si además no quedan hijos posibles para analizar, se devuelve vacío. Esto último significa que no hay solución.

Iterative Deepening A* [2]

```
Sea camino el camino de búsqueda actual (se usa como una pila).
Sea g el costo para alcanzar el nodo actual.
Sea f el costo estimado del camino más barato raiz-objetivo.
Sea h(nodo) una función de estimación del camino más barato nodo-objetivo.
Sea costo(nodo,siguiente) una función que devuelve el costo nodo-siguiente.
Sea sucesores(nodo) una función que devuelve los hijos de un nodo ordenados por
valor de g + h (nodo).
Sea objetivo el nodo que se desea alcanzar.

funcion busqueda_A* (raiz):
      límite = h(raiz)
      camino = [raiz]
      repetir:
      t = buscar(camino, 0, limite)
      si t == ENCONTRADO:
```

```
                return (camino, límite)
        si t == INF:
                return NO_ENCONTRADO
        límite = t

funcion busqueda (camino, g, límite)
        nodo = camino.ultimo_elemento()
        si nodo == objetivo:
                devolver ENCONTRADO
        f = g + h(nodo)
        si f > límite:
                devolver f
        min = INF
        para cada sucesor en sucesores(nodo):
                sí sucesor no está en camino:
                        camino.agregar(hijo)
                        t = busqueda(camino, g + costo (nodo, sucesor), límite)
                        si t == ENCONTRADO:
                        return ENCONTRADO
si t < min:
min = t
camino.quitar_ultimo_elemento()
devolver minimo
```

El algoritmo IDA* consiste en la combinación de IDDFS con la heurística de A* que prioriza los nodos más promisorios. La idea es que las iteraciones no se corresponden con la profundidad como en IDDFS sino con los costos incrementales del camino. En el caso de A* este costo está compuesto de la suma de:

- El costo actual **g** para alcanzar el nodo
- El costo estimado **h** desde este nodo hasta el objetivo

En cada iteración se realiza una búsqueda en profundidad (DFS) y se detiene la exploración cuando el costo total g+h excede un determinado límite. Este límite comienza como el costo estimado de la raíz al objetivo y se incrementa para cada iteración siguiente del algoritmo. En cada iteración, el límite usado para la próxima iteración es el costo mínimo de todos los valores que excedan el límite actual.

El pseudocódigo se explica analizando cada función por separado:

- La función `busqueda_A*` inicializa el camino (que solo contiene a la raíz) y el límite (solo el costo estimado al nodo objetivo). Luego se realizan iteraciones hasta bien encontrar una solución o determinar que la misma no existe. En cada iteración se realiza una llamada a la función `búsqueda`. Si no se encuentra solución y el valor retornado es menor que un valor INF se incrementa el límite a este valor retornado, o sea el siguiente valor de límite mínimo que excede el límite actual. Si el valor retornado es INF, significa que no hay solución posible.
- La función `busqueda` recibe un camino, el costo hasta el momento y el límite. Si se alcanza el nodo buscado, se devuelve ENCONTRADO. Si la suma de g + h es mayor que el límite actual, se termina la evaluación de esa rama. En caso contrario, se evalúan los sucesores ordenados por menor a mayor costo de g+h. Para cada sucesor, si el mismo no está en el camino, se lo agrega y se llama recursivamente a la función. Si el costo devuelto es menor que el límite actual, se lo actualiza.

Cuando se agotan todos los sucesores y no hay solución, se devuelve el valor mínimo del costo para ese nodo.

3. Según los conceptos trabajados en clase, a que tipo de algoritmo corresponde? Justificar

Iterative Deepening Depth-First Search

Es un algoritmo de **fuerza bruta** pues analiza cada nodo posible para un nivel de profundidad antes de incrementar el nivel, y repite esto hasta encontrar el nodo deseado. Efectivamente se analizan todos los nodos posibles del grafo uno tras otro.

Iterative Deepening A*

Es un algoritmo de **búsqueda local** pues se busca resolver el problema explorando el conjunto de soluciones posibles de estado del problema (las distintas rutas posibles). Iniciando en un nodo raíz, que puede corresponder a una solución factible o una solución parcial, se evalúa un subconjunto de estados para buscar uno que mejore la situación actual.

4. Realizar análisis de su ejecución (Tener en cuenta su complejidad y optimalidad).

Una instancia de un problema cualquiera a resolver con estos algoritmo consta de:

- ☐ Un grafo G con V nodos y E aristas.
- ☐ Un nodo inicial.
- ☐ Un set de nodos a los que se desea llegar (objetivos).

El análisis de complejidad de este algoritmo requiere presentar algunos conceptos:

- ☐ La *profundidad* ***d*** de la solución a un problema es la longitud de la menor secuencia de nodos que va del nodo inicial a uno de los nodos objetivos.
- ☐ El *factor de ramificación de nodos* ***b*** se puede definir como el número promedio de nodos accesibles desde un nodo.
- ☐ El *factor de ramificación de ejes* ***e*** se puede definir como el número promedio de ejes salientes de un nodo.

Iterative Deepening Depth-First Search [2]

En primer lugar, establecemos la complejidad temporal de una búsqueda en profundidad (DFS) en términos de estos parámetros.

- Se observa que en el peor de los casos hay que explorar cada nodo hasta la profundidad de la solución. Esto comprende evaluar los nodos vecinos de cada nodo, y luego los vecinos de cada uno

de estos nodos vecinos, etc. Como hay en promedio una cantidad ***e*** de vecinos por nodo, resulta que es necesario realizar esto ***d*** veces. Luego su complejidad es $O(e * e * e * ...) = O(e^d)$

Finalmente se puede presentar un análisis de ambas complejidades:

- Complejidad temporal: se analiza específicamente **para un grafo donde *e*=*b*** (un árbol, por ejemplo). Observamos que para iteración en particular a una profundidad cualquiera, todos los nodos anteriores se visitan una vez. Entonces, la cantidad de nodos visitados en una profundidad d es

  $b^d + 2b^{d-1} + 3b^{d-2} + \ ... \ + db$

  Factorizando según b^d

  $b^d\,(1 + 2b^{-1} + 3b^{-2} + \ ... \ + db^{1-d})$

  Si se considera $x = \frac{1}{b}$

  $b^d\,(1 + 2x^1 + 3x^2 + \ ... \ + dx^{d-1})$

  Esto es trivialmente menor que la serie infinita

  $b^d\,(1 + 2x^1 + 3x^2 + 4x^3 + ...)$

  que converge a

  $b^d\,(1 - x)^{-2}$ para $|x| < 1$

  Como $(1 - x)^{-2} = (1 - \frac{1}{b})^{-2}$ es una constante que es independiente de d, si $b > 1$ la complejidad resulta $O(b^d)$.
- Complejidad espacial: como en todo momento el algoritmo está realizando una búsqueda en profundidad (DFS) que no excede *d*, su complejidad espacial es $O(d)$.

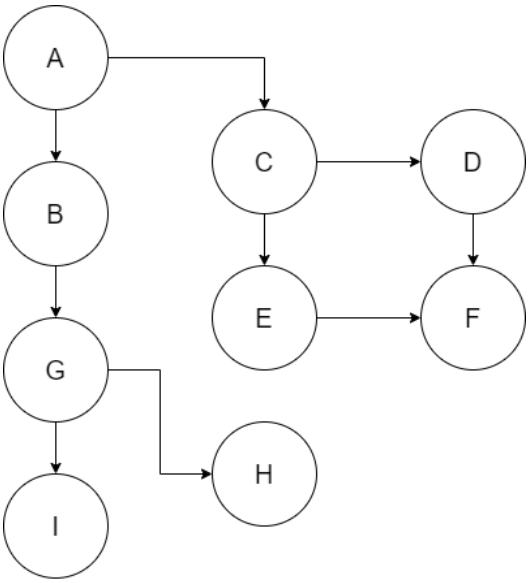
Iterative Deepening A* [2] [3]

- Para analizar la complejidad espacial, se considera únicamente la iteración final, o sea la que encuentra la solución. Esta debe expandir todos los nodos sucesores del estado inicial con valores mayores o iguales que la estimación de costo inicial y menores que la solución de costo óptimo. Si se emplea la misma regla para desempatar entre costos iguales que en A*, los nodos explorados de ambos algoritmos resultan los mismos, y por lo tanto la complejidad temporal es $O(b^d)$.
- La complejidad espacial es menor que la de A* debido a que no se almacenan los nodos de cada iteración. Al igual que IDDFS, como en todo momento el algoritmo está realizando una búsqueda en profundidad (DFS) que no excede *d*, su complejidad espacial es $O(d)$.

5. Realizar un ejemplo paso a paso de su utilización.

Iterative Deepening Depth-First Search

Considérese el siguiente grafo, donde se desea encontrar una ruta de A a H



Partiendo de A, se aplica DFS iterativamente, incrementando progresivamente la profundidad máxima permitida. Para cada nuevo nivel de profundidad se detallan los nodos hijos visitados.

Profundidad 0

- Se comienza en A.
- Nodos visitados: [A].
- No se alcanzó el objetivo.

Profundidad 1

- Se exploran hijos de A = [B, C].
- Nodos visitados: [A, B, C].
- No se alcanzó el objetivo.

Profundidad 2

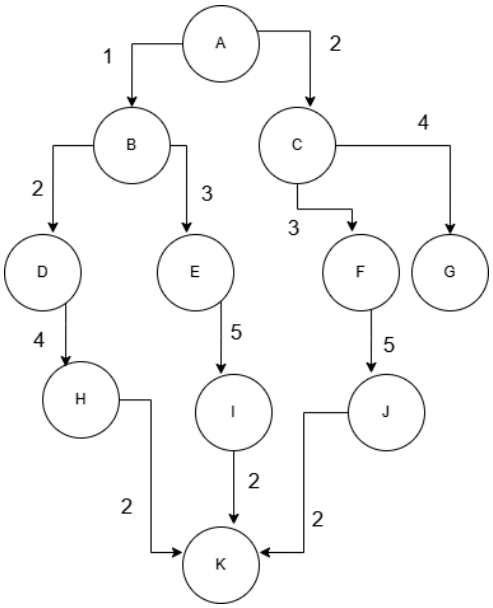
- Se exploran hijos de B = [G].
- Se exploran hijos de C = [D, E].
- Nodos visitados: [A, B, C, G, D, E].
- No se alcanzó el objetivo

Profundidad 3

- Se exploran hijos de G = [I, H].
- Se exploran hijos de D = ∅.
- Se exploran hijos de E = [F].
- Nodos visitados: [A, B, C, G, D, E, E, I, H, F].
- Se alcanzó el objetivo H

Iterative Deepening A*

Considérese el siguiente grafo, donde se desea encontrar una ruta de A a K. Los costos se indican sobre los enlaces.



Inicialmente se parte solo considerando el costo h. La heurística elegida es

$saltos\ restantes\ *\ mínimo\ costo\ de\ arista$

Resulta

Nodo	g(n)	Nodo	g(n)
A	4	G	2
B	3	H	1
C	3	I	1
D	2	J	1
E	2	K	0
F	2	-	-

El límite inicial resulta $f(A) = g(A) + h(A) = 0 + 4 = 4$. Cada iteración aumenta el límite y se describen los nuevos nodos expandidos.

Iteración 1

- Se comienza en A.:
  - $f(B) = 1 + 3 = 4$
  - $f(C) = 2 + 3 = 5$ ->excede el límite
- Se expande B:
  - $f(D) = 3 + 2 = 5$ -> excede el límite
  - $f(E) = 4 + 2 = 6$ -> excede el límite
- No se alcanzó el objetivo. Como el mínimo valor que excede el límite es 5, ese es el nuevo límite.

Iteración 2

- Se expande C:
  - $f(F) = 5 + 2 = 7$-> excede el límite
  - $f(G) = 6 + 2 = 8$ -> excede el límite
- Se expande D:
  - $f(H) = 7 + 1 = 8$-> excede el límite
- No se alcanzó el objetivo. Como el mínimo valor que excede el límite es 6 (ver iteración 1), ese es el nuevo límite.

Iteración 3

- Se expande E
  - $f(I) = 9 + 1 = 10$ -> excede el límite
- No se alcanzó el objetivo. Como el mínimo valor que excede el límite es 7 (ver iteración 2), ese es el nuevo límite.

Iteración 4

- Se expande F
  - $f(J) = 7 + 1 = 8$ -> excede el límite
- No se alcanzó el objetivo. Como el mínimo valor que excede el límite es 8 (ver iteración 2), ese es el nuevo límite.

Iteración 5

- Se expande J
  - $f(K) = 8 + 2 = 10$ -> excede el límite
- Se expande H:
  - $f(H) = 7 + 1 = 10$-> excede el límite
- Se expande I:
  - $f(K) = 7 + 1 = 8$-> excede el límite
- G no tiene nodos nuevos que se puedan alcanzar desde el mismo.
- No se alcanzó el objetivo. Como el mínimo valor que excede el límite es 10, ese es el nuevo límite.

Iteración 6

- En esta iteración se alcanza K.

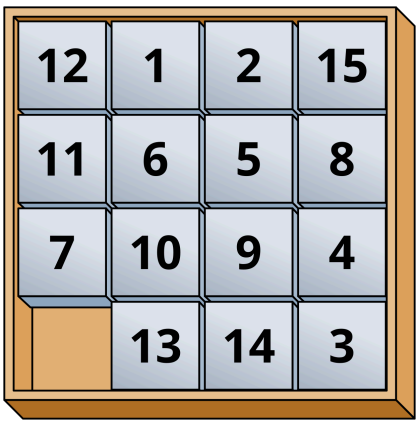
6. Brindar un enunciado de un problema que se puede resolver con este algoritmo.

Iterative Deepening Depth-First Search

Una red social consta de una gran cantidad de usuarios, que pueden ser o no amigos entre sí. Para un evento de fin de año, los desarrolladores precisan obtener una cadena de amigos lo más corta posible entre dos usuarios cualesquiera, o bien saber que no hay solución. Si bien en las máquinas de la empresa se dispone de gran capacidad de procesamiento para esta tarea, la memoria escasea. ¿Qué algoritmo propondría?

Iterative Deepening A*

Se dispone de una grilla cuadricular 4x4 con 15 fichas que se pueden deslizar en las cuatro direcciones cardinales.



El objetivo es dejar las fichas ordenadas de izquierda a derecha y de abajo para arriba. No se está limitado en capacidad de procesamiento pero si en memoria disponible. Proponer un algoritmo que indique los movimientos a realizar para llegar a una solución.

Parte 3: Autómatas finitos

1. Construir un autómata finito determinista (AFD) de forma formal para cada uno de los lenguajes simples.

En primer lugar definimos los lenguajes simples que forman ambos lenguajes. **Considerando a = 0 y b = 1**:
Lenguaje 1: {w | w tiene un número impar de 1}
Lenguaje 2: {w | w tiene al menos dos 0s y tres 1s}
La intersección de estos dos forma el **lenguaje A**.
Lenguaje 3: {w | w tiene un número par de 0}
Lenguaje 4: {w | w termina con dos 1}
La intersección de estos dos forma el **lenguaje B**.
Para cada lenguaje definimos su correspondiente autómata:

$$M = (Q, \Sigma, \delta, q_0, F)$$

donde Q es el conjunto de estados, Σ el alfabeto, δ la funcion de transicion, q_0 el estado inicial y F el conjunto de estados de aceptación. La función de transición se indica en forma de tabla en todos los casos.

AFD de Lenguaje 1
Se puede implementar mediante solo dos estados, forzando a que se requiera al menos una entrada 1 para alcanzar el estado final, y una vez ahí se requiera una cantidad de 1s equivalente a múltiplos de 2 para regresar.

$$Q = \{S_0, S_1\}$$
$$\Sigma = \{0, 1\}$$
$$q_0 = S_0$$
$$F = \{S_1\}$$

Q	0	1
S_0	S_0	S_1
S_1	S_1	S_0

AFD de Lenguaje 2

Se puede implementar considerando que cada estado representa un estado de cuenta de 1s y 0s. Por ejemplo, el estado inicial representará (0,0) y una entrada de 1 resultará en el estado (1,0) y una entrada de 0 resultará en (0,1). Una vez que se alcanza el límite de una de las cuentas, por ejemplo (3,0), cada 1 recibido posteriormente resultará en el mismo estado. El estado final representa (3,2) y cualquier entrada vuelve al mismo.

$$Q = \{S_0, S_1, S_2, S_3, S_4, S_5, S_6, S_7, S_8, S_9, S_{10}, S_{11}\}$$

$$\Sigma = \{0, 1\}$$

$$q_0 = S_0$$

$$F = \{S_{11}\}$$

Q	0	1	Q	0	1
S_0	S_1	S_2	S_6	S_7	S_9
S_1	S_4	S_3	S_7	S_8	S_{10}
S_2	S_3	S_6	S_8	S_8	S_{11}
S_3	S_5	S_7	S_9	S_{10}	S_9
S_4	S_4	S_5	S_{10}	S_{11}	S_{10}
S_5	S_5	S_8	S_{11}	S_{11}	S_{11}

Para mayor claridad se presentan los estados de cuenta que representa cada estado:

- $S_0 = (0, 0)$
- $S_1 = (0, 1)$
- $S_2 = (1, 0)$
- $S_3 = (1, 1)$
- $S_4 = (0, 2)$
- $S_5 = (1, 2)$
- $S_6 = (2, 0)$
- $S_7 = (2, 1)$
- $S_8 = (2, 2)$
- $S_9 = (3, 0)$
- $S_{10} = (3, 1)$
- $S_{11} = (3, 2)$

AFD de Lenguaje 3

Se puede implementar de forma casi idéntica al AFD del lenguaje 1, pero estableciendo el estado inicial como el estado final, y forzando a que se requiera una cantidad de 0s equivalente a múltiplos de 2 para volver.

$$Q = \{S_0, S_1\}$$

$$\Sigma = \{0, 1\}$$

$$q_0 = S_0$$

$$F = \{S_0\}$$

Q	0	1
S_0	S_1	S_0
S_1	S_0	S_1

AFD de Lenguaje 4

Se puede implementar mediante 3 estados, forzando a que la única forma de acceder al estado anteúltimo y último sea mediante 1, y que la entrada 0 resetee el progreso al estado inicial.

$$Q = \{S_0, S_1, S_2\}$$

$$\Sigma = \{0, 1\}$$

$$q_0 = S_0$$

$$F = \{S_2\}$$

δ_4	0	1
S_0	S_0	S_1
S_1	S_0	S_2
S_2	S_0	S_2

2. Basándose en los autómatas construidos generar de forma formal los dos autómatas finitos determinísticos de los lenguajes más complejos.

AFD de Lenguaje A

El AFD del lenguaje simple 2 requiere al menos 3 veces la entrada 1. Para permitir solo entradas impares, como requiere el lenguaje 1, basta con que la entrada 1 en el estado final resulte en el estado (2,2), efectivamente "restando" el valor. De esta forma se requiere una cantidad de 1s equivalente a múltiplos de 2 para volver, y se garantiza que la cantidad sea siempre impar.

$$Q = \{S_0, S_1, S_2, S_3, S_4, S_5, S_6, S_7, S_8, S_9, S_{10}, S_{11}\}$$

$$\Sigma = \{0, 1\}$$

$$q_0 = S_0$$

$$F = \{S_{11}\}$$

Q	0	1	Q	0	1
S_0	S_1	S_2	S_6	S_7	S_9
S_1	S_4	S_3	S_7	S_8	S_{10}
S_2	S_3	S_6	S_8	S_8	S_{11}
S_3	S_5	S_7	S_9	S_{10}	S_9
S_4	S_4	S_5	S_{10}	S_{11}	S_{10}
S_5	S_5	S_8	S_{11}	S_{11}	S_8

AFD de Lenguaje B

Partiendo del AFD del lenguaje 4 como base, es fácil ver que se puede requerir una cantidad par de 0s si se crea un cuarto estado, tal que una entrada de 0 en cualquier otro estado nos lleva a este, y a su vez una entrada de 0 en este nos lleva al estado inicial. Para impedir salir de este estado sin un 0 como entrada, se hace que 1 lleve al mismo estado. De esta forma, cualquier cadena aceptada requiere una cantidad par de 0s.

$$Q = \{S_0, S_1, S_2, S_3\}$$

$$\Sigma = \{0, 1\}$$

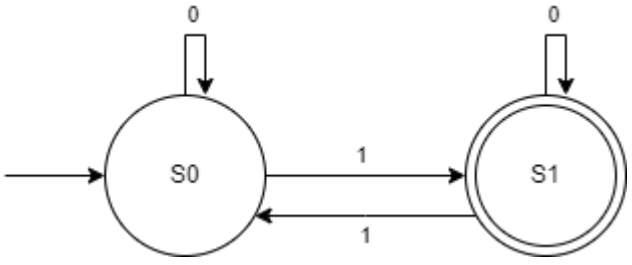
$$q_0 = S_0$$

$$F = \{S_2\}$$

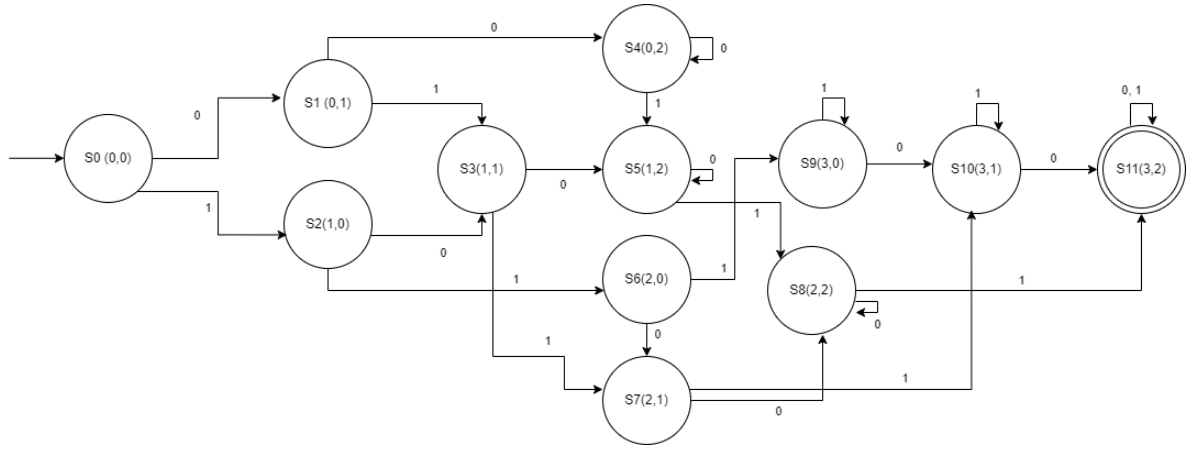
Q	0	1
S_0	S_3	S_1
S_1	S_3	S_2
S_2	S_3	S_2
S_3	S_3	S_3

3. Representar cada uno de los AFD mediante su representación gráfica

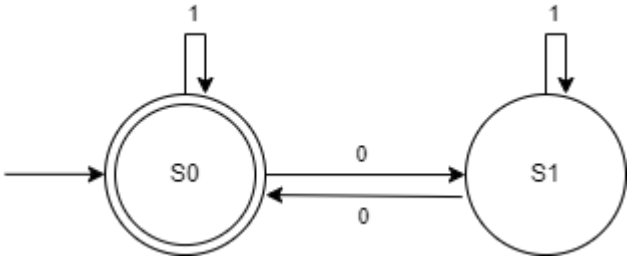
Lenguaje 1



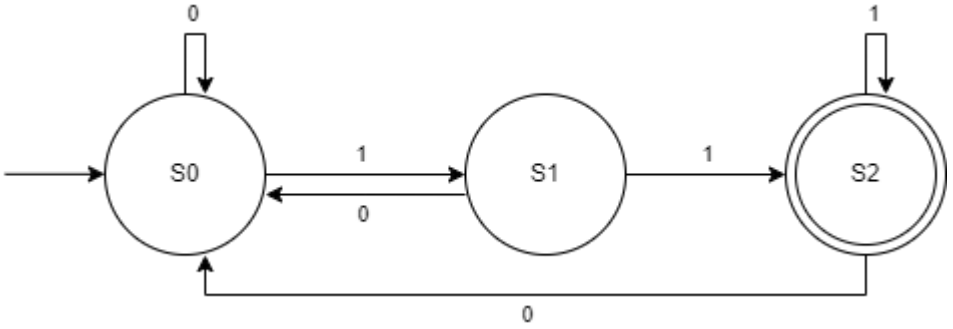
Lenguaje 2



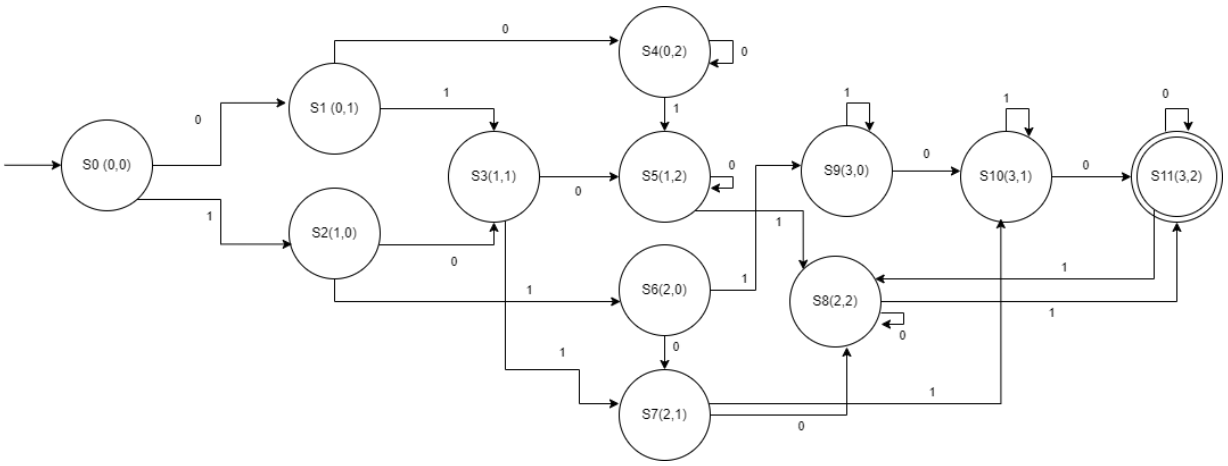
Lenguaje 3



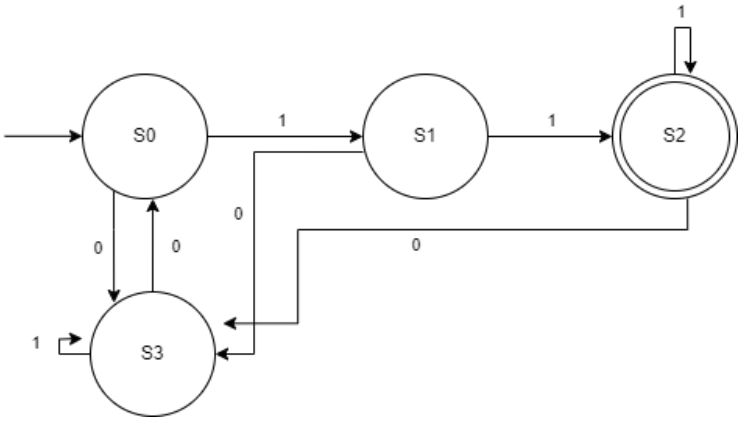
Lenguaje 4



Lenguaje A



Lenguaje B



4. Para cada AFD Brindar un ejemplo de una cadena que acepta y otra que rechaza. Explicarlos paso a paso.

Para cada cadena se provee el resultado de cada estado con su correspondiente entrada.

Lenguaje 1

Se propone la siguiente cadena **rechazada**: 011 (observamos que tiene una cantidad par de 1s)

$\delta(S_0, 0) = S_0 \rightarrow \delta(S_0, 1) = S_1 \rightarrow \delta(S_1, 1) = S_0$

y no se alcanza el estado final.

Se propone la siguiente cadena **aceptada**: 0111 (observamos que tiene una cantidad impar de 1s)

$\delta(S_0, 0) = S_0 \rightarrow \delta(S_0, 1) = S_1 \rightarrow \delta(S_1, 1) = S_0 \rightarrow \delta(S_0, 1) = S_1$

y se alcanza el estado final.

Lenguaje 2

En este caso se agrega a cada estado el valor de la suma para cada valor para mayor claridad.
Se propone la siguiente cadena **rechazada**: 1010 (observamos que no tiene la cantidad requerida de 1s)

$$\delta(S_0, 1) = S_2(1,0) \rightarrow \delta(S_2, 0) = S_3(1,1) \rightarrow \delta(S_3, 1) = S_7(2,1) \rightarrow \delta(S_7, 0) = S_8(2,2)$$

y no se alcanza el estado final.
Se propone la siguiente cadena **aceptada**: 01011 (observamos que tiene las cantidades requeridas de ambos números)

$$\delta(S_0, 0) = S_1(0,1) \rightarrow \delta(S_1, 1) = S_3(1,1) \rightarrow \delta(S_3, 0) = S_5(1,2) \rightarrow \delta(S_5, 1) = S_8(2,2)$$

$$\delta(S_8, 1) = S_{11}(3,2)$$

y se alcanza el estado final.

Lenguaje 3

Se propone la siguiente cadena **rechazada**: 0100 (observamos que tiene una cantidad impar de 0s)

$$\delta(S_0, 0) = S_1 \rightarrow \delta(S_1, 1) = S_1 \rightarrow \delta(S_1, 0) = S_0 \rightarrow \delta(S_0, 0) = S_1$$

y no se alcanza el estado final.
Se propone la siguiente cadena **aceptada**: 010 (observamos que tiene una cantidad par de 0s)

$$\delta(S_0, 0) = S_1 \rightarrow \delta(S_1, 1) = S_1 \rightarrow \delta(S_1, 0) = S_0$$

y se alcanza el estado final.

Lenguaje 4

Se propone la siguiente cadena **rechazada**: 0110 (observamos que termina en 10)

$$\delta(S_0, 0) = S_0 \rightarrow \delta(S_0, 1) = S_1 \rightarrow \delta(S_1, 1) = S_2 \rightarrow \delta(S_2, 0) = S_0$$

y no se alcanza el estado final.
Se propone la siguiente cadena **aceptada**: 01011 (observamos que termina en 11)

$$\delta(S_0, 0) = S_0 \rightarrow \delta(S_0, 1) = S_1 \rightarrow \delta(S_1, 0) = S_0 \rightarrow \delta(S_0, 1) = S_1 \rightarrow \delta(S_1, 1) = S_2$$

y se alcanza el estado final.

Lenguaje A

En este caso se agrega a cada estado el valor de la suma para cada valor para mayor claridad.
Se propone la siguiente cadena **rechazada**: 101011 (observamos que tiene una cantidad par de 1s)

$$\delta(S_0, 1) = S_2(1,0) \rightarrow \delta(S_2, 0) = S_3(1,1) \rightarrow \delta(S_3, 1) = S_7(2,1) \rightarrow \delta(S_7, 0) = S_8(2,2)$$

$$\delta(S_8, 1) = S_{11}(3,2) \rightarrow \delta(S_{11}, 1) = S_8(2,2)$$

y no se alcanza el estado final.
Se propone la siguiente cadena **aceptada**: 01011110 (observamos que tiene las cantidades requeridas de ambos números y una cantidad impar de 1s)

$\delta(S_0, 0) = S_1(0,1) \rightarrow \delta(S_1, 1) = S_3(1,1) \rightarrow \delta(S_3, 0) = S_5(1,2) \rightarrow \delta(S_5, 1) = S_8(2,2)$
$\delta(S_8, 1) = S_{11}(3,2) \rightarrow \delta(S_{11}, 1) = S_8(2,2) \rightarrow \delta(S_8, 1) = S_{11}(3,2) \rightarrow \delta(S_{11}, 0) = S_{11}(3,2)$
y se alcanza el estado final.

Lenguaje B

Se propone la siguiente cadena **rechazada**: 010011 (observamos que termina en 11 pero con una cantidad impar de 0s)
$\delta(S_0, 0) = S_3 \rightarrow \delta(S_3, 1) = S_3 \rightarrow \delta(S_3, 0) = S_0 \rightarrow \delta(S_0, 0) = S_3 \rightarrow \delta(S_3, 1) = S_3$
$\delta(S_3, 1) = S_3$
y no se alcanza el estado final.
Se propone la siguiente cadena **aceptada**: 01011 (observamos que termina en 11 y con una cantidad par de 0s)
$\delta(S_0, 0) = S_3 \rightarrow \delta(S_3, 1) = S_3 \rightarrow \delta(S_3, 0) = S_0 \rightarrow \delta(S_0, 1) = S_1 \rightarrow \delta(S_1, 1) = S_2$
y se alcanza el estado final.

Referencias

[1] Poole, D. L., & Mackworth, A. K. (2023). *Artificial Intelligence: Foundations of Computational Agents,* (3rd ed.). Cambridge University Press.

[2] Korf, R E. (1985). Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence, 27*(1), 97-109. https://doi.org/10.1016/0004-3702

[3] Russell, S., & Norvig, P. (2009). *Artificial Intelligence: A Modern Approach,* (3rd ed.). Pearson.