Universidad Nacional de Córdoba

Facultad de Ciencias Exactas Física y Naturales

Cátedra de Algoritmos y Estructura de Datos

2021

Trabajo Práctico Nº3

DEGIOANNI, Paula	**39968172**

Introducción

El objetivo que se planteó para este trabajo práctico fue el de planificar el vuelo de un dron para que aplique herbicida sobre un campo con malezas con el menor costo posible. Como entrada se recibe una imagen de un campo de siembra que fue previamente procesada por y representada con matriz binaria, donde 1 corresponde a que hay maleza y 0 que no. También en el archivo se representaban con una “x” barreras por donde no podía sobrevolar el dron. Solo se quiere aplicar herbicida sobre las malezas de tamaño significativo (tamaño 3x3 o mayor en la matriz de entrada).
Como salida se determina el camino más conveniente a realizar con su respectivo costo que refleja la distancia total recorrida. En cuanto a restricciones, es necesario que el dron pase solo una vez por cada una de las manchas y que al finalizar vuelva al punto de inicio.

Desarrollo del Trabajo Práctico

Descripción del algoritmo

Se dividió el problema a resolver en diferentes etapas, cada una con su respectiva entrada y salida, hasta llegar al resultado final. Debajo se detallan cada una de ellas.
La propuesta fue procesar la información del archivo de entrada y construir un grafo ponderado no dirigido con los datos, en el cual los vértices se corresponden a las manchas significativas y las aristas tienen un peso que representa la distancia euclidiana que separa las manchas entre sí. Una vez creado el grafo, se puede resolver el problema encontrando el ciclo hamiltoniano de costo mínimo, en este caso a partir de un algoritmo de búsqueda en amplitud.

1. Carga de las matrices desde el archivo de texto
   Se implementó con una función llamada cargarMatriz() que va leyendo linea por linea el archivo de entrada. Para cada línea, primero se filtran las barreras, que se setean en 1 en una matriz aparte y se dejan en 0 en la matriz original. De esta forma, la línea se puede agregar como fila de una matriz de bitsets. Se optó por usar bitsets en vez de variables enteras o booleanas para optimizar memoria y agilizar la carga y lectura de los datos. Cuando termina de procesar todo el archivo, devuelve una estructura definida como DATA con ambas matrices, la de malezas y las de las barreras.

2. Filtrado de la matriz para sólo conservar manchas significativas

Se implementó dentro de una función llamada filtrarMatriz() que recibe como parámetros la matriz a filtrar y el tamaño del filtro a aplicar. Se usó 3x3 tal como lo especificaba el enunciado.

Lo que hace esta función es crear una submatriz auxiliar, también de 3x3, que copia los valores de la matriz de malezas a medida que se desplaza de a una columna por vez y de a una fila cuando completa las columnas. Para cada posición, calcula la intersección con una matriz imaginaria de 3x3 de unos y, solo si la cantidad de unos en la submatriz es igual a la cantidad de unos del filtro, se conserva la mancha.

Cuando encuentra una mancha significativa, setea el bit correspondiente al centro del filtro (posición (1,1)) en una nueva matriz que luego retornará la función, la cual fue previamente inicializada con todos ceros. De esta forma, en la matriz filtrada de salida, sólo sobreviven las manchas significativas.

Notar que en el caso de que la mancha sea de tamaño mayor al filtro aplicado, en la matriz filtrada no se verá la ubicación puntual, sino dos o más unos adyacentes vertical u horizontalmente. En la siguiente etapa se resuelve este problema.

3. Asignación de coordenadas y creación de un vértice

   Mediante la función generarVertices(), primero se hace una limpieza en los lugares de la matriz donde hayan quedado unos adyacentes, tomando como única coordenada válida la que se ubique más arriba a la izquierda. El resto de los unos se ponen en 0.

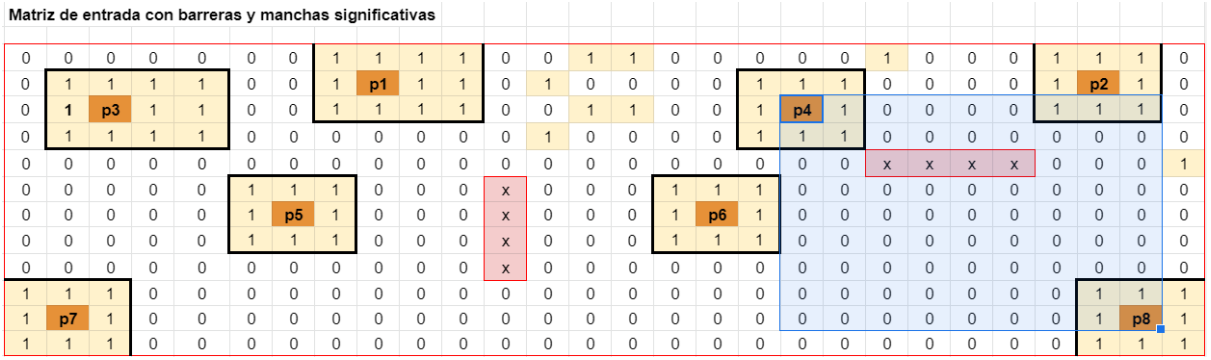
   Una vez que se tiene la coordenada de la mancha, se genera un vértice con las posiciones X e Y correspondientes y un número que hace las veces de identificador del vértice. Por último, se agrega el vértice recién creado a una lista para poder acceder a ellos luego.

   Esta función devuelve una matriz que tiene todos ceros y sólo tienen uno las posiciones que corresponden a las coordenadas de cada mancha significativa.

4. Enlazado de los vértices. Generación del grafo

   enlazarVerticesAlcanzables() es la función que lleva a cabo esta etapa, la cual a su vez tiene varios pasos. Mediante un ciclo while va tomando un vértice de la lista para enlazarlo con todos los demás vértices que sean aproximables desde él, esto es, que no atraviesen ninguna barrera en su camino. Cuando encuentra un vértice alcanzable se enlaza en ambos sentidos (v2 con v1 y v1 con v2) así cuando termina se puede eliminar de la lista sin peligro de que al final queden nodos sin enlazar. El ciclo while continúa hasta que no queden vértices en la lista.

Para cada vértice, el primer paso es determinar a cuáles otros vértices del resto de la lista puede llegar. Para eso, toma un vértice dentro de la lista de los no enlazados y genera una submatriz de la matriz de barreras que tenga el rango (x1,y1) : (x2,y2) donde, x1 e y1 son los valores más pequeños entre las coordenadas de ambos vértices y x2 e y2 los más grandes. Por ejemplo, si se tienen las coordenadas (2,9) y (3,3), la submatriz se toma en el rango (2,3) : (3,9). De esta forma queda un rectángulo con los vértices a los extremos. Si dentro de este rectángulo de la matriz de barreras hay algún 1, implica que entre los vértices que se están evaluando no hay camino posible. En el ejemplo de la figura, desde el vértice 4 no se puede llegar al vértice 8 pero sí al vértice 2.



Si se encuentra que un vértice sí es aproximable, se enlazan bidireccionalmente. Se calcula la distancia euclidiana entre ambos, que luego se agrega a la matriz de adyacencia. En la función también se agrega a una lista de vecinos propia del vértice. Esto está hecho así únicamente para poder visualizar los nodos enlazados en consola, pero no cumplen un rol significativo en el algoritmo. Para determinar el camino hamiltoniano, se utilizan los datos de la matriz de adyacencia.

Para calcular la distancia euclidiana, se hizo uso de una función aparte llamada calcularDistancia() que determina la distancia a partir de la fórmula:

$$d(P_1, P_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

$$d^2_{(i,j)} = \sum_{k=1}^{K} \left(x_{ik} - x_{jk} \right)^2$$

donde P1 y P2 son los vértices y (x1,y1), (x2,y2) sus respectivas coordenadas. Los nodos ya enlazados se eliminan de la lista y se van agregando en una cola que se utilizará en la siguiente etapa.

Finalmente, se imprimen en consola el listado de los vértices encontrados, sus vértices aproximables y la distancia que los separa de cada uno de ellos.

5. Búsqueda del camino óptimo

La propuesta del enunciado fue realizar la búsqueda del ciclo óptimo aplicando un algoritmo de búsqueda en amplitud. Lo que se hizo fue desarrollar una función que implemente la búsqueda en amplitud (busquedaEnAmplitud()) y luego desde otra función encontrarCircuito() se la fue llamando para que pruebe comenzar el camino desde cada uno de los vértices. encontrarCircuito() evalúa el camino que tenga el menor costo de entre todos los caminos posibles encontrados, y lo devuelve como resultado al main, concluyendo así la ejecución del programa.

busquedaEnAmplitud() recibe como argumento el vértice inicial del recorrido y a partir de ahí utiliza una cola como estructura auxiliar (aux) para realizar la búsqueda y una lista (camino) para ir agregando los vértices que conforman el camino final. Comienza agregando a camino el vértice inicial y encolando en aux todos los vértices aproximables desde él. Luego, dentro de un ciclo while, va desencolando y agregando al camino si se cumplen dos condiciones a la vez:

a. El vértice que se desencoló todavía no fue agregado al camino.
b. Hay una ruta posible entre el último vértice que se agregó al camino y el vértice que se desencoló.

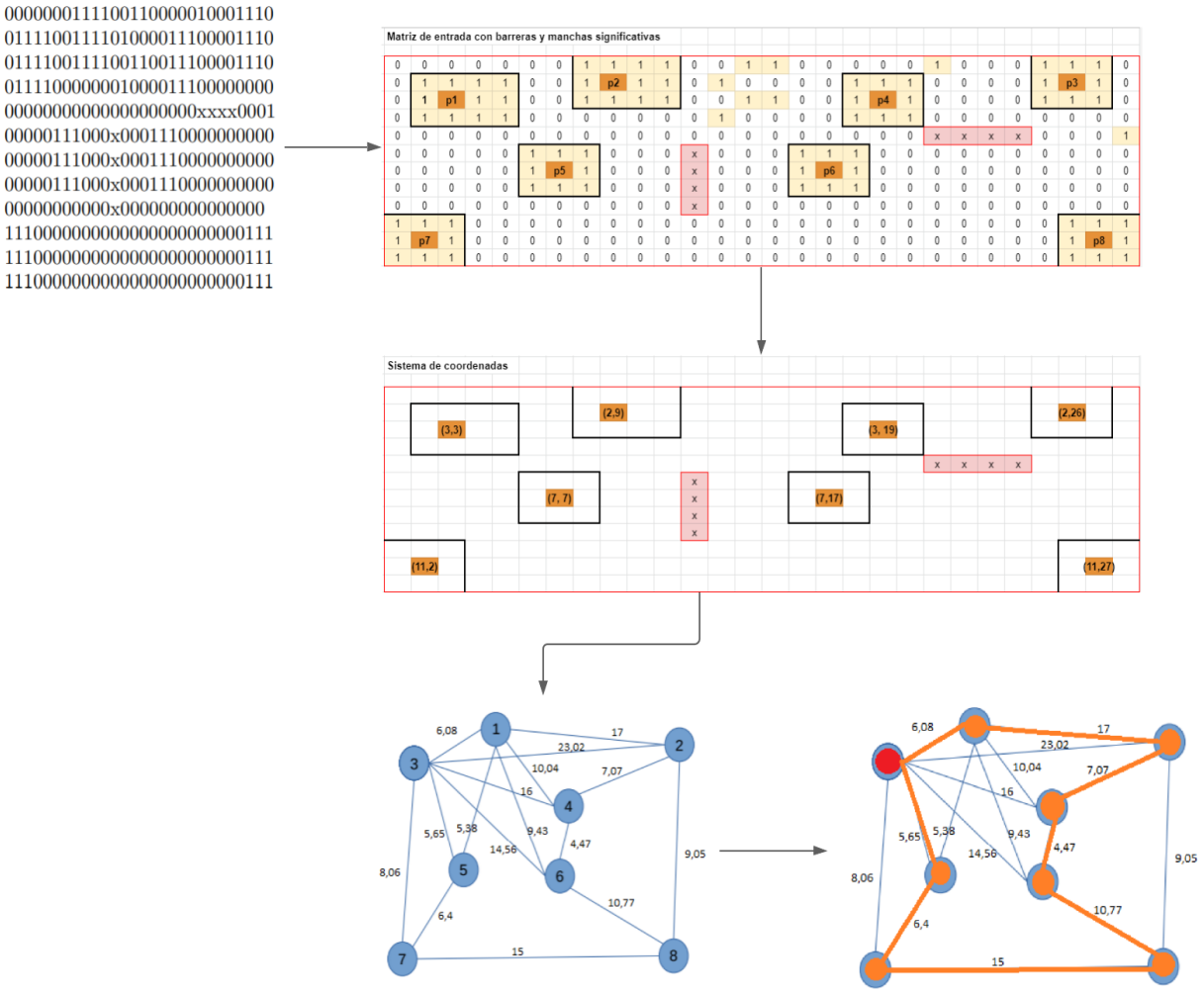
Si se cumplen ambas condiciones, se agrega el vértice al camino y en aux se encolan todos sus vértices vecinos.

Si alguna de estas condiciones no se cumple, se desencola aux hasta que se cumplan o hasta que aux quede vacía, en cuyo caso significaría que no se encontró un camino.

El proceso continúa hasta que se agregan todos los vértices una sola vez al camino, o bien hasta que se determina que no se encontró un camino. Cuando termina, la última comprobación es si desde el último vértice que se agregó, hay una ruta posible que vuelva al vértice de origen. Si no la hay, se desestima el camino. En consola se van imprimiendo todos los intentos de camino y el costo resultante cuando se encuentra un camino posible.

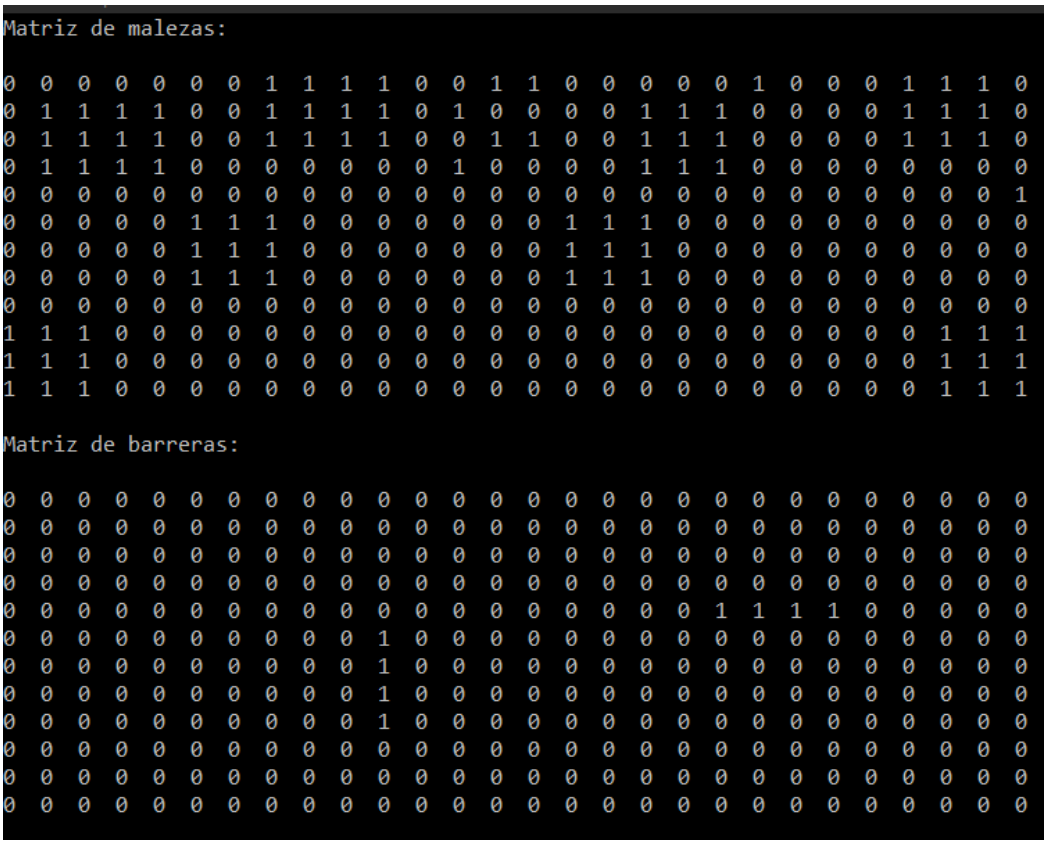
Resultados

Para ilustrar los resultados obtenidos, se utiliza como entrada la matriz que fue dada en la consigna. Esquemáticamente, lo que realiza el algoritmo es lo siguiente:

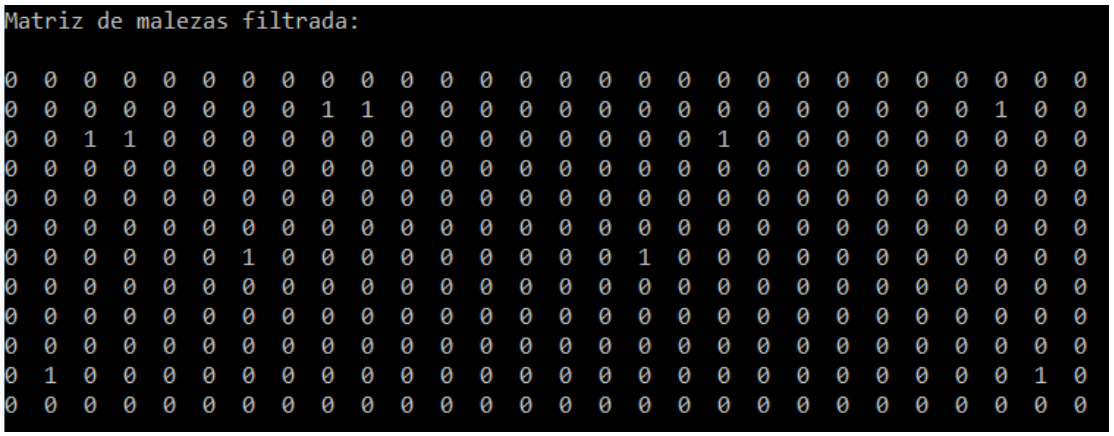


Se incluyen capturas de pantalla de salida por consola paso a paso.

1. Carga de las matrices desde el archivo de texto



2. Filtrado de la matriz para sólo conservar manchas significativas



3. Asignación de coordenadas y creación de un vértice

```
Ubicacion de las manchas significativas:

0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0
0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0



Vertices del grafo sin enlazar:
Vertice 1
        Coordenadas: (2, 9)
        Vertices alcanzables:

Vertice 2
        Coordenadas: (2, 26)
        Vertices alcanzables:

Vertice 3
        Coordenadas: (3, 3)
        Vertices alcanzables:

Vertice 4
        Coordenadas: (3, 19)
        Vertices alcanzables:

Vertice 5
        Coordenadas: (7, 7)
        Vertices alcanzables:

Vertice 6
        Coordenadas: (7, 17)
        Vertices alcanzables:

Vertice 7
        Coordenadas: (11, 2)
        Vertices alcanzables:

Vertice 8
        Coordenadas: (11, 27)
        Vertices alcanzables:
```

4. Enlazado de los vértices. Generación del grafo

```
Vertices del grafo enlazados:
Vertice 1
        Coordenadas: (2, 9)
        Vertices alcanzables:
                <Vertice 2, distancia 17>
                <Vertice 3, distancia 6.08276>
                <Vertice 4, distancia 10.0499>
                <Vertice 5, distancia 5.38516>
                <Vertice 7, distancia 11.4018>
Vertice 2
        Coordenadas: (2, 26)
        Vertices alcanzables:
                <Vertice 1, distancia 17>
                <Vertice 3, distancia 23.0217>
                <Vertice 4, distancia 7.07107>
                <Vertice 8, distancia 9.05539>
Vertice 3
        Coordenadas: (3, 3)
        Vertices alcanzables:
                <Vertice 1, distancia 6.08276>
                <Vertice 2, distancia 23.0217>
                <Vertice 4, distancia 16>
                <Vertice 5, distancia 5.65685>
                <Vertice 7, distancia 8.06226>
Vertice 4
        Coordenadas: (3, 19)
        Vertices alcanzables:
                <Vertice 1, distancia 10.0499>
                <Vertice 2, distancia 7.07107>
                <Vertice 3, distancia 16>
                <Vertice 6, distancia 4.47214>
Vertice 5
        Coordenadas: (7, 7)
        Vertices alcanzables:
                <Vertice 1, distancia 5.38516>
                <Vertice 3, distancia 5.65685>
                <Vertice 7, distancia 6.40312>
Vertice 6
        Coordenadas: (7, 17)
        Vertices alcanzables:
                <Vertice 4, distancia 4.47214>
                <Vertice 8, distancia 10.7703>
Vertice 7
        Coordenadas: (11, 2)
        Vertices alcanzables:
                <Vertice 1, distancia 11.4018>
                <Vertice 3, distancia 8.06226>
                <Vertice 5, distancia 6.40312>
                <Vertice 8, distancia 25>
Vertice 8
        Coordenadas: (11, 27)
        Vertices alcanzables:
                <Vertice 2, distancia 9.05539>
                <Vertice 6, distancia 10.7703>
                <Vertice 7, distancia 25>

Cantidad total de vertices: 8
```

```
Aristas:

0       17      6.08276 10.0499 5.38516 0       11.4018 0       0       0       0       0       0       0       0
17      0       23.0217 7.07107 0       0       0       9.05539 0       0       0       0       0       0       0
6.08276 23.0217 0       16      5.65685 0       8.06226 0       0       0       0       0       0       0       0
10.0499 7.07107 16      0       0       4.47214 0       0       0       0       0       0       0       0       0
5.38516 0       5.65685 0       0       0       6.40312 0       0       0       0       0       0       0       0
0       0       0       4.47214 0       0       0       10.7703 0       0       0       0       0       0       0
11.4018 0       8.06226 0       6.40312 0       0       25      0       0       0       0       0       0       0
0       9.05539 0       0       0       10.7703 25      0       0       0       0       0       0       0       0
0       0       0       0       0       0       0       0       0       0       0       0       0       0       0
0       0       0       0       0       0       0       0       0       0       0       0       0       0       0
0       0       0       0       0       0       0       0       0       0       0       0       0       0       0
0       0       0       0       0       0       0       0       0       0       0       0       0       0       0
0       0       0       0       0       0       0       0       0       0       0       0       0       0       0
0       0       0       0       0       0       0       0       0       0       0       0       0       0       0
0       0       0       0       0       0       0       0       0       0       0       0       0       0       0
```

5. Búsqueda del camino óptimo

```
Buscando camino a partir de vertice 1
1 2 3 4 6 8 7 5
Posible camino encontrado: 1 2 3 4 6 8 7 5 1
Costo: 108.052

Buscando camino a partir de vertice 2
2 1 3 4 6 8 7 5
Camino no encontrado

Buscando camino a partir de vertice 3
3 1 2 4 6 8 7 5
Posible camino encontrado: 3 1 2 4 6 8 7 5 3
Costo: 82.4563

Buscando camino a partir de vertice 4
4 1 2 3 5 7 8 6
Posible camino encontrado: 4 1 2 3 5 7 8 6 4
Costo: 102.374

Buscando camino a partir de vertice 5
5 1 3 7 8 2 4 6
Camino no encontrado

Buscando camino a partir de vertice 6
6 4 1 2 3 5 7 8
Posible camino encontrado: 6 4 1 2 3 5 7 8 6
Costo: 102.374

Buscando camino a partir de vertice 7
7 1 3 5
Camino no encontrado

Buscando camino a partir de vertice 8
8 2 1 3 4 6
```