

# Manual Técnico

En este encontrara una guía paso a paso con la explicación de la lógica de cómo funciona el programa “KAREL”.

`import random` Importar librería de generación de números aleatorios

```
def generarmapa (col,fil):  
    mapa = []  
    for i in range (fil):  
        mapa.append ([])  
        for j in range (col):  
            pared = random.randint (0,1)  
            mapa[i].append (pared)  
            ##print (mapa[i])  
    return mapa
```

Generación de mapa con numerosa aleatorios entre 0 y 1

```
SIMBOLOKAREL = '@'#chr (9786)  Asignación caracteres especiales código ASCII '☺' (9786) y '♥'  
SIMBOLOOBJETIVO = '$'#chr (9829) (9829)  
SIMBOLOCAMINO = 4
```

```
def ubicacionlibre (mapa):  
    casillalibre = False  
    col = len(mapa[0])  
    fil = len(mapa)  
    i=0  
    j=0  
    while not casillalibre:  
        i = random.randint (0,fil-1)  
        j = random.randint (0,col-1)  
        ##print (i,j)  
        if mapa [i][j]=='0':  
            casillalibre=True  
    return [i,j]
```

Encontrar una ubicación libre

```
def imprimirmapa (mapa):  
    col = len(mapa[0])  
    fil = len(mapa)  
    for i in range(fil):  
        fila = ''  
        for j in range(col):  
            fila+= str(mapa[i][j])  
        print (fila)
```

Impresión del contenido del mapa

```
def posibilidades (fil,col,mapa):  
    filaanterior = fil-1  
    filasiguiente = fil+1  
    columnaanterior = col-1  
    columnasiguiente = col+1  
    fila = len(mapa)  
    columna = len(mapa[0])  
    posi = []  
  
    #norte  
    if filaanterior>-1 and verificacionlibre(mapa[filaanterior][col]):  
        posi.append([filaanterior,col])  
    #norte oriente  
    if filaanterior>-1 and columnasiguiente<columna and verificacionlibre(mapa[filaanterior][columnasiguiente]):  
        posi.append([filaanterior,columnasiguiente])  
    #oriente  
    if columnasiguiente<columna and verificacionlibre(mapa[fil][columnasiguiente]):  
        posi.append([fil,columnasiguiente])  
    #sur oriente  
    if filasiguiente<fila and columnasiguiente<columna and verificacionlibre(mapa[filasiguiente][columnasiguiente]):  
        posi.append([filasiguiente,columnasiguiente])  
    #sur  
    if filasiguiente<fila and verificacionlibre(mapa[filasiguiente][col]):  
        posi.append([filasiguiente,col])  
    #sur occidente  
    if filasiguiente<fila and columnaanterior>-1 and verificacionlibre(mapa[filasiguiente][columnaanterior]):  
        posi.append([filasiguiente,columnaanterior])  
    #occidente  
    if columnaanterior>-1 and verificacionlibre(mapa[fil][columnaanterior]):  
        posi.append([fil,columnaanterior])  
    #norte occidente  
    if filaanterior>-1 and columnaanterior>-1 and verificacionlibre(mapa[filaanterior][columnaanterior]):  
        posi.append([filaanterior,columnaanterior])  
    return posi  
  
def verificacionlibre (posicion):  
    if posicion=='0' or posicion==SIMBOLOOBJETIVO:  
        return True  
    else:  
        return False
```

Posibilidad de movimiento y todas las direcciones

```
def llaveposicion(posicion):  
    llave = str (posicion[0])+',' + str (posicion[1])  
    return llave
```

Generación de llave correspondiente a la posición,  
para usar en el diccionario

```
def bfs(graph, start, memoria):  
    visited, queue = [], [start]  
    memoria[llaveposicion(start)] = []  
    optimo = None  
    while queue and optimo is None:  
        #print ('cola', queue)  
        vertex = queue.pop(0)  
        #print ('posicion actual', vertex)  
        if vertex not in visited:  
            visited.append(vertex)  
            casillasporvisitar = posibilidades(vertex[0], vertex[1], graph)  
            #print ('casillas', casillasporvisitar)  
            for casilla in casillasporvisitar:  
                if casilla not in queue and casilla not in visited:  
                    queue.append(casilla)  
                    caminoanterior = memoria[llaveposicion(vertex)]  
                    memoria[llaveposicion(casilla)] = caminoanterior + [vertex]  
                    if graph[casilla[0]][casilla[1]] == SIMBOLOOBJETIVO:  
                        optimo = memoria[llaveposicion(casilla)] + [casilla]  
                        break  
            #print ('casillas agregadas a la cola', casillasporvisitar)  
    return visited, optimo
```

de los vecinos se exploran sus respectivos vecinos adyacentes, y así hasta que se recorra todo el árbol.

## BFS

### Breadth first search

Es un algoritmo para recorrer o buscar elementos en un grafo (usado frecuentemente sobre árboles).

Intuitivamente, se comienza en la raíz (eligiendo algún nodo como elemento raíz en el caso de un grafo) y se exploran todos los vecinos de este nodo. A continuación para cada uno

```
def imprimiropoptimo(mapa, optimo):  
    mapalocal = list(mapa)  
    if optimo is None:  
        return  
    for casilla in optimo:  
        if not mapalocal[casilla[0]][casilla[1]] == SIMBOLOKAREL and not mapalocal[casilla[0]][casilla[1]] == SIMBOLOOBJETIVO:  
            mapalocal[casilla[0]][casilla[1]] = SIMBOLOCAMINO  
    imprimirmapa(mapalocal)
```

Impresión camino para llegar al objetivo, y  
cambiar los caracteres '0' por '4'

```
def escribirmapa(mapa, ruta):  
    archivo = open(ruta, 'w')  
    for fila in mapa:  
        for columna in fila:  
            archivo.write(str(columna))  
        archivo.write('\n')  
    archivo.close()
```

Escribir mapa en un archivo de texto

```
def leermapa(ruta):  
    archivo = open(ruta, 'r')  
    mapa = []  
    for linea in archivo:  
        mapa.append([])  
        linea = linea.strip()  
        for caracter in linea:  
            mapa[-1].append(caracter)  
    return mapa
```

Leer mapa desde un archivo de texto

```
7% karel1.py - C:\Users\USER\Desktop\karel1.py
File Edit Format Run Options Windows Help

def main ():

    memoria = {}
    #tamano de la matriz a generar
    n=int(input('Fila '))
    m=int(input('Columna '))
    print('')
    ruta='mapa.txt'
    #generacion aleatoria de mapa
    mapagenerado = generarmapa(n,m)
    #generacion ubicacion inicial de karel
    ubicacioninicial = ubicacionlibre(mapagenerado)
    mapagenerado[ubicacioninicial[0]][ubicacioninicial[1]]=SIMBOLOKAREL
    #generacion ubicacion del objetivo
    ubicacionobjetivo = ubicacionlibre(mapagenerado)
    mapagenerado[ubicacionobjetivo[0]][ubicacionobjetivo[1]]=SIMBOLOOBJETIVO
    #exportar mapa generado
    escribirmapa(mapagenerado,ruta)
    #no comentar
    mapagenerado=leermapa(ruta)
    #busqueda de salida
    camino, optimo = bfs(mapagenerado,ubicacioninicial,memoria)
    #print(camino)
    #print('memoria',memoria)
    #print('optimo',optimo)
    imprimirmapa(mapagenerado)
    if optimo is None:
        print('No hay solucion')
    else:
        print('')
        print('La solucion es: ')
        print('')
        imprimiroptimo(mapagenerado, optimo)
        #exportar mapa con solucion
        escribirmapa(mapagenerado,'solucion.txt')

if __name__ == '__main__':
    main()
```

Ln: 175 Col: 0

Llamado de las funciones y  
ejecución del programa