Laboratorio - 6

Alumno: Jackson Fernando Merma Portocarrero

Escuela Profesional: Ingeniería de Sistemas

CUI: 20202143

Correo: jmermap@unsa.edu.pe

Generalidades

- Estos problemas tienen entrada de datos (se especificará en cada uno de ellos).
- En la carpeta **binary_ejercices**, se encuentran los ejercicios de búsqueda binaria.
- El problema de DFS, se encuentra a la altura de la carpeta principal (**clase_06**).

Parte 1: Ejercicios de búsqueda binaria

- Búsqueda binaria básica: **"*basic_binary_search.cpp*"**

  En este básicamente se implementó una búsqueda binaria en un arreglo, el método principal es *"search()"*, donde se realiza la búsqueda y retorna la posición si es encontrado, de lo contrario retorna -1.

```
int search(int *arr,int len,int num){
   int a=0,b=len-1;

   //buscando
   while(a<=b){
      int middle=(a+b)/2;

      //si lo encuentra, devuelve la posicion
      if(arr[middle]==num) return middle;

      if(arr[middle]>num){
         b=middle-1;
      }else{
         a=middle+1;
      }
   }
   return -1;
}
```

```
Input:
7
1 2 3 4 5 6 7
2

Output:
Encontrado en la posicion: 2
```

- Número cuadrático: "***sqrt_binary_function.cpp***"

  En este problema, se implementó una búsqueda binaria decimal, además de no reducir o aumentar a los límites en la búsqueda (**left=middle OR right=middle**), esto implica que si no se encuentra una raíz, se forma en bucle infinito en: **left < right +1**, es por ello que el bucle termina en: **left < right - 1** en caso de no existir raíz perfecta.

```
int search_root(int number){
   float ans;
   int a=0,b=number;

   while(a<b-1){
      ans=(a+b)/2;
      if(ans*ans==number) return (int)ans;
      if(ans*ans>number){
         b=ans;
      }else{
         a=ans;
      }
   }
   return -1;
}
```

```
Input:
100

Output:
Raiz de 100: 10
```

- Búsqueda del mayor o igual: "***mayor_o_igual.cpp***"

  Este trabaja exactamente igual a la búsqueda típica binaria, sin embargo, si no se encuentra, se devuelve la siguiente posición en caso de existir.

```
int search_equal_or_next(int *arr,int len,int found){
   int a=0,b=len-1;
   int middle;

   while(a<=  b){
      middle=(a+b)/2;
      if(arr[middle]==found) return arr[middle];
```

```
        if(arr[middle]>found){
            b=middle-1;
        }else{
            a=middle+1;
        }
    }
    cout<<"dont"<<endl;
    //la posicion acta
    if(found>arr[len-1]) return -1;//no existe
    return arr[middle+1];//posicion siguiente
}
```

```
Input:
7
1  2  3  5  6  7  8
4

Output:
5
```

- Menor elemento en un arreglo ordenado y rotado:
  "***menor_elemento_arreglo_rotado.cpp***"

  Para su implementación se trabajó con el dato de referencia en la posición 1 del arreglo; de esta forma se puede interpretar lo siguiente:
  - Si el dato en la posición k es menor que el dato 1, entonces hay que buscar por la izquierda, tomando al dato en k también (el mínimo no puede ser el dato 1).
  - De lo contrario, el menor debe estar en la derecha (se está en la parte no rotada) y no se toma al dato en k (el mínimo puede ser el dato 1)

    Finalmente, se compara entre el primero y el dato en la posición final, esto para prevenir el caso de un arreglo no rotado.

```
int searchRotatedArr(int *arr, int len){
    int a=0,b=len-1,m=(a+b)/2;

    while(a<b){
        //si arr[m] es menor, quiere decir que se a podido rotar y el menor este a la izquierda
        if(arr[m]<arr[0]){
            b=m;
        }else{//de lo contrario se a podido rotar pero sigue en la derecha
            a=m+1;
        }
        m=(a+b)/2;
    }
    return min(arr[m],arr[0]);//si esta ordenado
}
```

```
Input:
5
```

```
5 1 2 3 4
Output:
1
```

Parte 2: DFS en un arreglo

- Entrada de datos: Se ha considerado tomar a '-1' como el valor de # y los 0 trabajan de igual forma.

  La entrada de datos consta de:
  - La primera línea lee la dimensión de la matriz (**n*n**)
  - Las siguientes '**n**' líneas leen '**n**' números separados por espacios que serán los valores almacenados en la matriz
  - En la siguiente línea, lee dos números separados por espacio **(x,y)** que representan las coordenadas del punto (se tomará en cuenta que **0 < x,y ≤ n**), aunque de todas formas el algoritmo reconoce entradas inválidas.
  - Finalmente se lee un color representado por un número

- Recorrido DFS: En el recorrido, se envían más parámetros del modelo (matriz, tamaño, coordenada x, coordenada y, color), además que se devuelve una matriz. Luego, respecto al procedimiento, este inicia verificando los límites, el color y las paredes, si pasa esta condición, entonces se pinta y se recorre a los hijos de este punto, los cuales fueron definidos en dos arreglos de tamaño 4, estos trabajan con el desplazamiento a las 4 coordenadas mediante valore como 1, -1 y 0.

- Salida de datos: Se muestra un antes y un después del recorrido imprimiendo ambas matrices.

```
//recorrido -> arriba, abajo, izquierda, derecha
int son_fil[4]={-1,1,0,0};
int son_col[4]={0,0,-1,1};

vector<vector<int>> dfs_matrix_painter(vector<vector<int>> matrix, int fil, int col, int color){
   int len=matrix.size();

   //verificacion de limites, pintado y pared
   if((fil<0||fil>=len||col<0||col>=len) || matrix[fil][col]==-1 || matrix[fil][col]==color)
      return matrix;

   matrix[fil][col]=color;//se pinta

   //recorrido de los 4 posibles hijos
   for(int i=0;i<4;i++){
      matrix = dfs_matrix_painter(matrix,fil+son_fil[i],col+son_col[i],color);
   }
   return matrix;
}
```

Input:

```
10
-1 -1  0  0 -1  0  0  0  0  0
-1 -1 -1 -1 -1 -1 -1  0 -1  0
-1  0  0 -1  0  0 -1  0  0 -1
-1  0 -1  0 -1  0 -1 -1 -1  0
-1 -1  0 -1 -1  0 -1 -1 -1  0
-1  0  0 -1 -1 -1 -1 -1 -1  0
 0 -1  0  0  0  0  0  0  0  0
 0 -1  0 -1  0  0  0 -1 -1  0
-1 -1  0  0  0  0  0  0 -1  0
 0 -1  0 -1 -1  0  0  0 -1 -1
5 3
7
```

Output:

```
Before...
# # 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 # #
After...
# # 0 0 # 0 0 0 0 0
# # # # # # # 0 # 0
# 0 0 # 0 0 # 0 0 #
# 0 # 0 # 0 # # # 7
# # 7 # # 0 # # # 7
# 7 7 # # # # # # 7
0 # 7 7 7 7 7 7 7 7
0 # 7 # 7 7 7 # # 7
# # 7 7 7 7 7 7 # 7
0 # 7 # # 7 7 7 # #
```