Descripción del Algoritmo

En este documento se presenta la información necesaria para implementar el algoritmo de búsqueda de aplicaciones móviles más parecidas a otra, para esto se usó la estructura de datos espacial **k-dimensional tree**

K-dimensional Tree

Esta implementación se hizo en 2 pasos:

1. **Inserción**: Se leyó las filas del csv, y se guardaron en un dataframe que facilita la librería pandas de python. Luego se crea el árbol ordenándolo por las dimensiones identificadas y cada fila del dataframe se guardó en la "clase nodo"

1.1 Los datos identificados como dimensiones son los siguientes**:**

- **Id**
- **size_bytes**
- **Price**
- **rating_count_tot**
- **rating_count_ver**
- **user_rating**
- **user_rating_ver**
- **cont_rating**
- **prime_genre ,** transformo a numérico enumerándolo (el primer genero tendría el número 1, y así sucesivamente)
- **sup_devices.num**
- **ipadSc_urls.num**
- **lang.num**
- **vpp_lic**

2. **Búsqueda de vecinos más cercanos (KNN):** Es simplemente una variación de la búsqueda en profundidad de grafos, la distancia usada fue la euclidiana (comparando datos de una misma dimensión), se identificaba como un vecino cercano (Esto se repite hasta encontrar todos los vecinos pedidos) :
    - Si aún no se llena el conjunto de vecinos pedidos (10)
    - Si la distancia del nodo actual era más corta que la del peor nodo actual.
    - Luego se revisan los sub-árboles para evitar revisar todos los nodos.

A continuación se provee del código de ambas etapas.

```
def insertarRec(self,root,df_row,depth):genero
        global dimensions
        if(root == None):
            root = _node.create_node_charge_data(df_row,self.genre_dict,depth)
            return root
        cd = depth % dimensions
        df_app_ = self.fix_df(copy.deepcopy(df_row))
        if(self.special_case(df_app_.iloc[0][cd])  < self.special_case(root.df
_row.iloc[[0][0]][cd])):
            root.izq = self.insertarRec(root.izq,df_row,depth+1)
        else:
            root.der = self.insertarRec(root.der,df_row,depth+1)
        return root
```

Figura 1.1 inserción en K-d Tree

Aquí se da la cantidad de dimensiones, por medio de la variable global **dimensions**, y se usa la función **special_case** , para corregir problemas con el tipo de dato.

```
def knn(self,root,dato,n):
    global dimensions
    dimension_ = None
    puntos_ordenados = list()
    pila = list()
    pila.insert(0,root)
    while(len(pila) > 0):
        nodo = pila.pop(0)
        dimension_ = nodo.depth % dimensions
        if((len(puntos_ordenados) < n)):
            puntos_ordenados.insert(0,nodo)
            self.ordenar(puntos_ordenados,dato.iloc[0][dimension_],dimension_)
        elif((self.euclidian_distance(dato.iloc[0][dimension_],nodo.df_row.ilo
c[[0][0]][dimension_]) < self.euclidian_distance(dato.iloc[0][dimension_],punt
os_ordenados[0].df_row.iloc[[0][0][dimension_]))):
            puntos_ordenados.pop(0)
            puntos_ordenados.insert(0,nodo)
            self.ordenar(puntos_ordenados,dato.iloc[0][dimension_],dimension_)

        if(dato.iloc[0][dimension_] < nodo.df_row.iloc[[0][0]][dimension_]):
            pila.insert(0,nodo.der)
            pila.insert(0,nodo.izq)
        else:
            pila.insert(0,nodo.der)
            pila.insert(0,nodo.izq)
        pila = self.quitar_null(pila,root)
    return puntos_ordenados
```

Figura 2.1 Búsqueda de vecinos más cercanos

Igual que en la función pasada el número de dimensiones se da por la variable global, por otro lado para mantener el orden en la lista de los vecinos identificados más cercanos se ordena del peor al mejor (según distancia) por la función **ordenar**.