Cuaderno de prácticas.
Práctica 3. Greedy y Programacion Dinamica.

Enrique Pinazo Moreno

28 de mayo de 2025

Índice

1. Algoritmo Greedy (Algoritmo Voraz).

1.1. Diseño del Algoritmo Voraz.

- **Componentes del Diseño Greedy.**

  - Cometido: Para resolver el problema de encontrar el camino mas corto para atravesar una matriz de enteros, me ha basado en un algoritmo voraz el cual selecciona el camino con menos coste en cada paso, utilizando una matriz de enteros donde quedará registrado el mapa y el coste de cada celda, el uso de esta matriz radica en el metodo algortimoGreedy(), donde se inicializamos la suma de los costes y el numero con menor coste de esta columna, cuyo valor inicial segun la columna, es el ubicado en la posicion media de esta.

```
int suma = 0;
int numeroMenor = v[f/2][0];
```

  A continuacion se recorre la matriz de enteros, empezando desde la mitad de la columna j, y se aplica la heurística desarrollada para obtener el camino mas corto. En el recorrido, se compara como primer valor el numeroMenor inicializado previamente como el valor de la mitad de la columna + la suma de los costes acumulados, con el valor de la celda actual + la suma de los costes acumulados, si el numeroMenor es mayor o igual al valor de la celda actual, se actualiza el numeroMenor.
  El ultimo paso es añadir el numeroMenor al vector rGreedy, cuyo cometido es almacenar el camino mas corto en orden, y actualizar la suma de los costes acumulados.

```
for (int i=0; i < c; i++)
{
    numeroMenor = v[f/2][i];
    for (int j=0; j < f; j++)
    {
        if(numeroMenor+suma > v[j][i]+suma)
        {
            numeroMenor = v[j][i];
        }
    }
    rGreedy.push_back(numeroMenor);
    suma += numeroMenor;
}
```

- **Heuristica, Ejemplo y Demostración de Solución Óptima.**

  - Heurística: La heurística desarrollada en este algoritmo es seleccionar el camino con menor coste en cada paso, lo que significa que en cada iteración se elige la celda con el valor más bajo de la columna actual, sumando este valor al total acumulado. Este enfoque garantiza que sea minimo el coste total del camino a medida que se avanza por la matriz.
  Por ejemplo, si tenemos una matriz de enteros como la que vimos en el enunciado::

$$
\begin{array}{|cccccc|}
2 & 8 & 9 & 5 & 8 & 1 \\
4 & 4 & 6 & 2 & 3 & 3 \\
5 & 1 & 4 & 6 & 1 & 5 \\
3 & 2 & 5 & 4 & 8 & 2 \\
4 & 2 & 3 & 3 & 2 & 3
\end{array}
$$

  La heurística, selecciona desde la columna 0 pos [f/2,0] con el valor 5, hasta la columna c-1 pos [f/2,c-1] con el valor 5, se revisa si en su misma columna hay un valor menor, si lo hubiese, se actualiza la variable numeroMenor, si no, se mantiene dicho valor. Posteriormente, se sumaria el valor de la celda seleccionada al total acumulado y se añade al vector con el resultado rGreedy.
  En su esencia, sin importar el tamaño de la matriz, la heuristica de este algoritmo voraz seleccioanra el camino con menor coste sin importar el camino que se haya tomado anteriormente, siempre y cuando se mantenga el criterio de seleccionar el valor mas bajo en cada iteración , el resultado es obtener el verdadero camino con menos coste.

- Ejemplo: Supongamos que tenemos la siguiente matriz de enteros:

$$\begin{vmatrix} 2 & 8 & 9 & 5 & 8 & 1 \\ 4 & 4 & 6 & 2 & 3 & 3 \\ 5 & 1 & 4 & 6 & 1 & 5 \\ 3 & 2 & 5 & 4 & 8 & 2 \\ 4 & 2 & 3 & 3 & 2 & 3 \end{vmatrix}$$

Al aplicar el algoritmo voraz, comenzamos en la primera columna, con el valor inicial de su posicion en mitad "5". Luego, en cada iteración, es seleccioando el valor más bajo de la columna actual y lo sumamos al total acumulado. El resultado final sería un vector rGreedy que contiene los valores seleccionados en orden.
Primer numero de la columna <0>: 5
0+5 – 0+2
→ Hay un camino mas Factible: 2
0+2 – 0+4
0+2 – 0+5
0+2 – 0+3
0+2 – 0+4
Numero mas Factible: 2
Suma acumulada: 2

Primer numero de la columna <1>: 1
2+1 – 2+8
2+1 – 2+4
2+1 – 2+1
2+1 – 2+2
2+1 – 2+2
Numero mas Fact
Suma acumulada: 3

Primer numero de la columna <2>: 4
3+4 – 3+9
3+4 – 3+6
3+4 – 3+4
3+4 – 3+5
3+4 – 3+3
→ Hay un camino mas Factible: 3
Numero mas Factible: 3
Suma acumulada: 6

Primer numero de la columna <3>: 6
6+6 – 6+5
→ Hay un camino mas Factible: 5
6+5 – 6+2
→ Hay un camino mas Factible: 2
6+2 – 6+6
6+2 – 6+4
6+2 – 6+3
Numero mas Factible: 2
Suma acumulada: 8

Primer numero de la columna <4>: 1
8+1 – 8+8
8+1 – 8+3
8+1 – 8+1
8+1 – 8+8
8+1 – 8+2
Numero mas Factible: 1
Suma acumulada: 9

Primer numero de la columna <5>: 5
9+5 – 9+1
→ Hay un camino mas Factible: 1
9+1 – 9+3
9+1 – 9+5
9+1 – 9+2
9+1 – 9+3
Numero mas Factible: 1
Suma acumulada: 10

Matriz de entrada:
|**2** 8 9 5 8 **1**|
|4 4 6 **2** 3 3|
|5 **1** 4 6 **1** 5|
|3 2 5 4 8 2|
|4 2 **3** 3 2 3|
Solución Greedy: [2],[1],[3],[2],[1],[1]
Suma total: 10

- **Semejanzas con el algoritmo Prim:**
  - Ambos algoritmos utilizan una estrategia voraz para construir soluciones incrementales.
  - En el algoritmo Greedy, se selecciona el número menor en cada columna para añadir a la suma total, mientras que en el algoritmo Prim se selecciona la arista de menor peso para añadir al árbol de expansión.
  - Ambos algoritmos construyen la solución de forma incremental, añadiendo un elemento (un número en Greedy y una arista en Prim) en cada paso hasta que la solución está completa.
- **¿Sería posible y conveniente abordarlo aplicándolo, o el de Kruskal:**
  - No sería conveniente aplicar el algoritmo de Kruskal en este caso, ya que Kruskal está diseñado para encontrar el árbol de expansión mínima en un grafo, mientras que el algoritmo greedy planteado es encontrar un camino con el menor coste en una matriz de enteros.
  - El algoritmo Greedy es más adecuado para este problema específico, ya que se centra en seleccionar el camino con menor coste en cada paso.
  - Además de que el algoritmo Kruskal requiere de una estructura de grafo al contrario de la estructura matricial del algorimto Greedy presentado.

- **Configuración de la ejecución del programa:**

```
./greedy_Voraz <f> <c> <elementos...>
```

Para que esto funcione, en el codigo se ha implementado un bucle que recorre los argumentos de la linea de comandos, desde el tercer argumento en adelante, se tranforman en enteros y se almacenan en un vector de vectores de enteros, la logica es la siguiente:

```
int f,c;
vector<vector<int>> v;
vector<int> rGreedy;

if(argc < 4){
    cout<<"Error: No se han introducido suficientes argumentos."<<endl;
    cout<<"Uso: "<<argv[0]<<" <f> <c> <elementos...>"<<endl;
    return 1;
}

f = atoi(argv[1]);
c = atoi(argv[2]);
v.resize(f, vector<int>(c));

for(int i = 0; i<f; i++){
    for(int j = 0; j<c; j++){
        v[i][j] = atoi(argv[3 + i*c + j]);
    }
}
```

- Ejemplo de ejecución y Tiempo de ejecución:

Ejemplo de ejecución del programa con una matriz de 3 filas y 3 columnas:

```
./greedy_Voraz 3 3 1 2 3 4 5 6 7 8 9
               --- -----------------
               f c elementos...
```

El tiempo de ejecucion del programa segun el tamaño mencionado en el enunciado, y los elementos de ella misma:

```
./greedy_Voraz 5 6
2 8 9 5 8 1
4 4 6 2 3 3
5 1 4 6 1 5
3 2 5 4 8 2
4 2 3 3 2 3
```

Salida:

```
Matriz de entrada:
 _ _ _ _ _ _
|2 8 9 5 8 1|
|4 4 6 2 3 3|
|5 1 4 6 1 5|
|3 2 5 4 8 2|
|4 2 3 3 2 3|
Solucion Greedy: [2],[1],[3],[2],[1],[1]
Tiempo de ejecucion: 5.3948e-05 segundos
Suma total: 10
```