

En este código en lenguaje C++ se encarga de leer las instancias modificadas por el equipo de trabajo para posteriormente se cree un archivo .txt que contenga la información de la instancia y más la ruta optima, la distancia recorrida y el beneficio obtenido. La ruta es creada por medio de una función “nodos” donde por medio de varios condicionales y ciclos se obtiene el vector de nodos. A medida que avanza el código se encuentran algunas descripciones de las funciones implementadas.

```

1  //Encabezado del programa
2  #include <algorithm>
3  #include <iostream>
4  #include <fstream>
5  #include <string>
6  #include<vector>
7  #include<locale>
8  #include<sstream>
9  #include <random>
10 #include <numeric>
11
12 using namespace std;
13 random_device rd;
14 mt19937 generator(rd());
15
16 vector<int> nodos(vector<int>& vulne, vector<int>& vDem, int& nmerca, vector
    <vector<int>>& mCluster, vector<int>& vMinDem) {
17     vector<int> aux = vulne;          // Vector nivel de vulnerabilidad
18     vector<int> vOrden;              // Vector de resultante (ruta)
19     vector<int> aux1 = vDem;         // Vector de demandas de los nodos
20
21     bool salir; // Booleano para cerrar el ciclo después de encontrar el número de
        clúster.
22     int sum = 10; // Valor de inicialización. Suma de los elementos del vector
        vulnerabilidad (aux)
23     int a; // Máximo valor de vulnerabilidad de todos los nodos en el vector aux
24     int nAnt = 23; // Número del clúster anterior que se visitó
25     int nClus = 3; // Número de clúster actual visitado
26     int g = 0;
27     vector<vector<int>> mcVulne(mCluster.size());
28
29
30     while (nmerca > 0 && sum > 0) {
31         a = *max_element(aux.begin(), aux.end()); // Máximo nivel de
            vulnerabilidad, se guarda en a
32         int ind = max_element(aux.begin(), aux.end()) - aux.begin(); // Valor
            del iterador o posición del valor máximo a
33         int k = ind + 1; // Indicador mas 1 (Para ubicar el nodo correcto en la
            instancia)
34         int sumMinDem = accumulate(vMinDem.begin(), vMinDem.end(), 0);
35
36         salir = false;
37         for (int i = 0; i < mCluster.size(); i++) { // Ciclo para identificar en
            que clúster está el nodo con mayor vulnerabilidad
38             for (int j = 0; j < mCluster[i].size(); j++) {
39                 if (k == mCluster[i][j]) {
40                     nClus = i;

```

Anexo 5. SolucionActual

```
41         salir = true;  
42         break;  
43     }  
44 }  
45 if (salir == true) break; // Después de identificar el clúster se
```

```

termina el ciclo
46     }
47
48     if (a == g) {
49         nClus = nAnt;
50         int ind1 = max_element(mcVulne[nClus].begin(), mcVulne[nClus].end())
           - mcVulne[nClus].begin();
51         ind = mCluster[nClus][ind1] - 1;
52     }
53
54     if (nClus != nAnt) {          // Identificar si se cambió de clúster al visitar
           el nuevo nodo
55         if (aux1[ind] <= nmerca && nmerca - aux1[ind] >= sumMinDem - vMinDem
           [nClus]) { // Regla lógica = Sí la demanda del nodo es menor o
           igual al número de mercados
56             // Y el número de mercados disponibles menos la demanda del nodo
           es mayor igual a la demanda mínima total de todos los clústeres
           menos la demanda mínima de ese clúster
57             nmerca -= aux1[ind]; // Se descuenta la demanda del nodo a el
           número de mercados disponibles
58             vOrden.push_back(ind); // Se pone el nodo en el vector de orden
           de la ruta
59             aux[ind] = 0; // Se revisó el nodo y con el fin de no volver a
           revisarlo se pone el valor 0
60             vMinDem[nClus] = 0; // Cómo ya se atendió el clúster se actualiza
           el valor de la demanda mínima de ese clúster como 0
61         }
62         else {
63             aux[ind] = 0; // No cumple con la regla lógica anterior entonces
           no se puede visitar el nodo
64         }
65     }
66     else {
67         if (aux1[ind] <= nmerca && nmerca - aux1[ind] >= sumMinDem) { //
           Regla lógica = Sí la demanda del nodo es menor igual al número de
           mercados
68             // Y el número de mercados disponibles menos la demanda del nodo
           es mayor igual a la demanda mínima total de todos los clústeres
69             nmerca -= aux1[ind]; // se descuenta la demanda del nodo a el
           numero de mercados disponibles
70             vOrden.push_back(ind); // Se pone el nodo en el vector de orden
           de la ruta
71             aux[ind] = 0; // Se revisó el nodo y con el fin de no volver a
           revisarlo se pone el valor 0
72             vMinDem[nClus] = 0; // Cómo ya se atendió el clúster se actualiza
           el valor de la demanda mínima de ese clúster como 0
73         }
74         else {
75             aux[ind] = 0; // No cumple con la regla lógica anterior entonces
           no se puede visitar el nodo
76         }
77     }

```

```

78
79     sum = accumulate(aux.begin(), aux.end(), 0);           // Suma de los valores
    del vector vulnerabilidad
80     nAnt = nClus; // Actualización del número de clúster anterior para la
    nuerva iteración
81
82     for (int j = 0; j < mCluster.size(); j++) {
83         for (int t = 0; t < mCluster[j].size(); t++) {
84             mcVulne[j].erase(mcVulne[j].begin(), mcVulne[j].end());
85         }
86     }
87
88     for (int j = 0; j < mCluster.size(); j++) {
89         for (int t = 0; t < mCluster[j].size(); t++) {
90             mcVulne[j].push_back(aux[mCluster[j][t] - 1]);
91         }
92     }
93
94     g = *max_element(mcVulne[nAnt].begin(), mcVulne[nAnt].end());
95 }
96 return vOrden; // Retorno del vector de orden de ruta
97 }
98
99 vector <int> vulnerabilidad(vector <int>& vOrden, vector <int>& vulne) { //
    Función que determina el vector de vulnerabilidad satisfecha
100     vector <int> j;
101
102     for (int i = 1; i < vOrden.size(); i++) {
103         j.push_back(vulne[vOrden[i] - 1]);
104     }
105     return j;
106 }
107
108 vector <int> distancia(vector <int>& vOrden, vector <vector <int>>& mDistancia) {
    // Función que determina el vector de distancia de recorrido
109     // entre nodos en la ruta hallada
110     vector <int> k;
111     double distanciaT = 0;
112
113     for (int i = 1; i < vOrden.size(); i++) { // Se empieza en 1 para empezar en
    el nodo segundo nodo en el orden de ruta Vorden
114         k.push_back(mDistancia[vOrden[i] - 1][vOrden[i]]); // Se le resta uno
    para encontrar la distancia de ir al nodo anterior al actual acorde con la ruta
    dada por el vector vOrden
115     }
116     return k;
117 }
118
119 vector<int> quitarChar(string s) { // Función que cambia el vector leído de la instancia
    de string a integer
120     locale loc;
121     for (int i = 0; i < s.size(); i++) {

```

```

122     if (isdigit(s[i], loc) == false) {
123         s[i] = ' ';
124     }
125 }
126 vector<int> v;           // es un gen.
127 int aux;
128 stringstream daniel(s);
129 while (daniel >> aux) {
130     v.push_back(aux);
131 }
132 return v;
133 }
134
135 void mostrar(const vector<int>& v, ofstream& f) {
136     if (!v.empty()) {
137         f << v.front();
138         for (int i = 1; i < v.size(); i++) {
139             f << " " << v[i];
140         }
141     }
142 }
143
144 void leer(int& N, int& M, vector<vector<int>>& mCluster, vector<vector<int>>&
mDistance, vector<vector<int>>& mIndi, string& nombre, int& nmerca) {
145
146     fstream f;
147     string dire = "C:\\Users\\User\\Escritorio\\InstancesPD\\Nuevas\\" + nombre + ".txt";
148
149     f.open(dire, fstream::in);
150     if (f.is_open() == false) {
151         cout << "No se puede abrir el archivo de lectura.\n";
152         cout << dire << endl;
153     }
154
155     string aux;
156
157     f >> N;
158     f >> M;
159     getline(f, aux);
160
161     vector<int> v;
162     for (int i = 0; i < M; i++) {
163         getline(f, aux);
164         v = quitarChar(aux);
165         mCluster.push_back(v);
166     }
167
168     vector<int> v1;
169     for (int i = 0; i < N; i++) {
170         getline(f, aux);
171         v1 = quitarChar(aux);
172         mDistance.push_back(v1);

```

```

172     }
173
174     vector<int> v2;
175     for (int i = 0; i < 2; i++) {
176         getline(f, aux);
177         v2 = quitarChar(aux);
178         mIIndi.push_back(v2);
179     }
180
181     f >> nmerca;
182
183     f.close();
184 }
185
186 void solucion(string& nombre, int& pdist, int& ben) {
187
188     string nombre2 = nombre;
189     nombre.pop_back(); nombre.pop_back(); nombre.pop_back(); nombre.pop_back();
190
191     ofstream f("C:\\Users\\User\\Escritorio\\InstancesPD\\NuevasSolucion\\" + nombre + "_Sol.txt");
192
193     int N, M, nmerca, nmerca_Inicial;
194     vector <vector <int>> mDistancia;
195     vector <vector <int>> mCluster;
196     vector <vector <int>> mIIndi;
197
198     leer(N, M, mCluster, mDistancia, mIIndi, nombre, nmerca);
199     f << N << endl;
200     f << M << endl;
201     int num = 4;
202     nmerca_Inicial = nmerca;
203
204     vector <int> vulne = mIIndi[1];
205     vector <int> vDem = mIIndi[0];
206
207     vector <vector <int>> mClusDem;
208     vector <int> p;
209     vector <int> vMinDem;
210
211     for (int i = 0; i < mCluster.size(); i++) {
212         p.clear();
213         ; for (int j = 0; j < mCluster[i].size(); j++) {
214             p.push_back(vDem[mCluster[i][j] - 1]);
215         }
216         mClusDem.push_back(p);
217     }
218
219     int b = 0;
220
221     for (int i = 0; i < mClusDem.size(); i++) {
222         int a = *min_element(mClusDem[i].begin(), mClusDem[i].end());

```

```
223     b += a;
224     vMinDem.push_back(a);
225 }
226
227 vector <int> OrdenNodos = nodos(vulne, vDem, nmerca, mCluster, vMinDem);
228 OrdenNodos.insert(OrdenNodos.begin(), 0);
229 OrdenNodos.push_back(0);
230
231 for (int i = 0; i < M + N; i++) {
232     if (i < M) {
233         mostrar(mCluster[i], f);
234         f << endl;
235     }
236     else {
237         mostrar(mDistancia[i - M], f);
238         f << endl;
239     }
240 }
241
242 mostrar(vDem, f);
243 f << endl;
244 f << endl;
245 mostrar(vulne, f);
246 f << endl;
247 f << endl;
248
249 f << nmerca_Inicial << endl;
250 f << endl;
251
252 for (int i = 0; i < OrdenNodos.size(); i++) {
253     f << OrdenNodos[i] << " ";
254 }
255 f << endl;
256 f << endl;
257
258 vector <int> k = distancia(OrdenNodos, mDistancia);
259
260 for (int i = 0; i < k.size(); i++) {
261     f << k[i] << " ";
262 }
263 f << endl;
264 f << endl;
265
266 double sumDist = accumulate(k.begin(), k.end(), 0);
267
268 f << sumDist << endl;
269 f << endl;
270
271 pdist = sumDist;
272
273 vector <int> j = vulnerabilidad(OrdenNodos, vulne);
274
```

...\\User\\Escritorio\\C++2\\P23 Mejorado\\P23 Mejorado\\Header1.h

7

```
275     for (int i = 0; i < j.size(); i++) {
276         f << j[i] << " ";
277     }
278     f << endl;
279     f << endl;
280
281     double sumVulne = accumulate(j.begin(), j.end(), 0);
282     f << sumVulne << endl;
283     ben = sumVulne;
284
285 }
286
287 #include "Header1.h"
288
289 int main() {
290     vector<string> vName;
291     string aux;
292     int promedioD = 0;
293     int pdist = 0;
294     int ben = 0;
295     int vulneC = 0;
296
297     fstream f("C:\\Users\\User\\Escritorio\\InstancesPD\\nombres2.txt",
298             fstream::in);
299
300     while (f >> aux) {
301         vName.push_back(aux);
302     }
303     for (int i = 0; i < vName.size(); i++) {
304         solucion(vName[i], pdist, ben);
305         promedioD += pdist;
306         vulneC += ben;
307     }
308
309     cout << "Distancia de todas las instancias es " << promedioD / 88 << endl;
310     cout << " La vulnerabilidad cubierta (beneficio) es " << vulneC / 88 << endl;
311     cin.get();
312     cout << "fin." << endl;
313     cin.get();
314     return 0;
315 }
```