

Anexo 12. RUTASOL

Este programa se encarga de resolver los casos aplicados a través del archivo .mod del modelo y del archivo .dat resultante del programa DATOS1SOL del Anexo 11. Las líneas de código mostradas a continuación, se muestran el proceso de resolución y de generación del nuevo archivo .txt, con los correspondientes valores de las variables de decisión y los valores de la función multiobjetivo y las funciones individuales.

```
1 using namespace std;
2
3 #include <algorithm>
4 #include <iostream>
5 #include <fstream>
6 #include <string>
7 #include <vector>
8 #include <locale>
9 #include <sstream>
10 #include <numeric>
11 #include <filesystem>
12 #include "ampl/ampl.h"
13
14 void dat(string name) {
15
16     string nombre = name;
17     double z;
18
19     ofstream f("C:\\Users\\User\\Escritorio\\PD C++\\RUTAS\\" + nombre +      ↗
20         "_Ruta.txt"); //Direccion donde se ubicaran los resultados
21
22     try {
23         ampl::Environment env("D:\\Documentos\\PROGRAMAS\\AMPL\\      ↗
24             \\ampl_mswin64"); //Dirección del "ampl.exe"
25         ampl::AMPL ampl(env);
26
27         //Leer el modelo y los archivos de datos
28         ampl.read("C:\\Users\\User\\Escritorio\\PD C++\\Modelos\\MODELO.mod");// ↗
29             Ubicación donde esta el modelo
30         ampl.readData("C:\\Users\\User\\Escritorio\\PD C++\\Instancias2 .dat\\" + ↗
31             name + ".dat");//Ubicación donde estan los .dat
32         ampl.setOption("solver", "cplex");
33         ampl.setOption("cplex_options", "outlev 1 timelimit 3600 return_mipgap 3 ↗
34             bestbound");
35         //Resolver
36         ampl.solve();
37         cout << "AMPL fin." << endl << endl;
38
39         //Estado
40         ampl::DataFrame num = ampl.getData("solve_result_num");
41         f << "solve_result_num: " << (*num.begin())[0].toString() << endl;
42
43         ampl::DataFrame num1 = ampl.getData("_solve_elapsed_time");
44         f << "_solve_elapsed_time: " << (*num1.begin())[0].toString() << endl;
45
46         ampl::DataFrame num2 = ampl.getData("TotCost.relmipgap");
47         f << "GAP: " << (*num2.begin())[0].toString() << endl;
48
49         //ampl::DataFrame numvar = ampl.getData("");
50         //f << "Numero de Variables: " << (*numvar)//
51
52         //Obtener el valor objetivo del modelo
```

```

48     ampl::Objective FO = ampl.getObjective("TotCost");
49     double z = FO.value();
50     f << "Función Costo Total: " << z << endl;
51     f << endl;
52
53     //Conseguir el valor de las variables y traerlas a c++
54     //Funciones objetivo Costo y Vulne
55     ampl::Variable Costo_ampl = ampl.getVariable("Costo");
56     ampl::DataFrame df_Costo = Costo_ampl.getValues();
57     f << "Costo-Distancia: " << (*df_Costo.begin())[0].toString() << endl;
58
59     ampl::Variable Vulne_ampl = ampl.getVariable("Vulne");
60     ampl::DataFrame df_Vulne = Vulne_ampl.getValues();
61     f << "Vulnerabilidad: " << (*df_Vulne.begin())[0].toString() << endl << ↵
        endl;
62
63     //Variable X
64     ampl::Variable x_ampl = ampl.getVariable("X");
65     ampl::DataFrame df_x = x_ampl.getValues();
66
67     ampl::DataFrame::iterator it1 = df_x.begin();
68     f << "Variable Uso de Arcos (i,j): " << endl;
69     for (; it1 != df_x.end(); it1++) {
70         int indx1 = int((*it1)[0].dbl());
71         int indx2 = int((*it1)[1].dbl());
72         int var_value = int((*it1)[2].dbl());
73
74         if (var_value != 0) {
75             f << "X" << int((*it1)[0].dbl()) << "-" << int((*it1)[1].dbl()) ↵
                << "=" << (*it1)[2].dbl() << endl;
76         }
77     }
78     f << endl;
79
80     //Variable Y
81     ampl::Variable y_ampl = ampl.getVariable("Y");
82     ampl::DataFrame df_y = y_ampl.getValues();
83
84     ampl::DataFrame::iterator it2 = df_y.begin();
85     f << "Variable Uso de Nodo:" << endl;
86     for (; it2 != df_y.end(); it2++) {
87         int indx1 = int((*it2)[0].dbl());
88         int var_value = int((*it2)[1].dbl());
89
90         if (var_value != 0) {
91             f << "Y" << int((*it2)[0].dbl()) << "=" << (*it2)[1].dbl() << ↵
                endl;
92         }
93     }
94     f << endl;
95
96     //Variables u

```

```

97     ampl::Variable u_ampl = ampl.getVariable("u");
98     ampl::DataFrame df_u = u_ampl.getValues();
99
100    ampl::DataFrame::iterator it4 = df_u.begin();
101    f << "Orden de visita de los nodos:" << endl;
102    for (; it4 != df_u.end(); it4++) {
103        int indx1 = int((*it4)[0].dbl());
104        int var_value = int((*it4)[1].dbl());
105
106        if (var_value != 0) {
107            f << "u" << int((*it4)[0].dbl()) << "=" << (*it4)[1].dbl() << ➤
            endl;
108        }
109    }
110    f << endl;
111
112    //Variables Z
113    ampl::Variable Z_ampl = ampl.getVariable("Z");
114    ampl::DataFrame df_Z = Z_ampl.getValues();
115
116    ampl::DataFrame::iterator it5 = df_Z.begin();
117    f << "Variable Nodo de Inicio" << endl;
118    for (; it5 != df_Z.end(); it5++) {
119        int indx1 = int((*it5)[0].dbl());
120        int var_value = int((*it5)[1].dbl());
121
122        if (var_value != 0) {
123            f << "Z" << int((*it5)[0].dbl()) << "=" << (*it5)[1].dbl() << ➤
            endl;
124        }
125    }
126
127    f << endl;
128
129    ampl.close();
130 }
131
132 catch (const exception& e) {
133     cout << "Error: " << e.what() << "\n";
134     cin.get();
135 }
136
137 f.close();
138
139 }
140
141 #include "Header.h"
142
143 int main() {
144     vector<string> vName;    // Vector de strings que será lleno con los nombres ➤
145                             de las instancias
146     string aux;

```

```
146     fstream f("C:\\Users\\User\\Escritorio\\PD C++\\nombres2.txt", fstream::in); ↗
        // Se lee el .txt con los nombres de cada una de las instancias
147     while (f >> aux) {
148         vName.push_back(aux); // Lee todos los nombres de las instancias y se ↗
            van poniendo en el vector de strings vName.
149     }
150
151     for (int i = 82; i < vName.size(); i++) {
152         dat(vName[i]); // Llama la acción "crearDatos" para cada uno de los ↗
            nombres presentes en el vector de strings vName.size()
153     }
154
155     cout << "fin." << endl; // Muestra "fin." en pantalla cuando haya terminado ↗
        de ejecutarse el programa.
156     cin.get();
157     return 0;
158 }
```