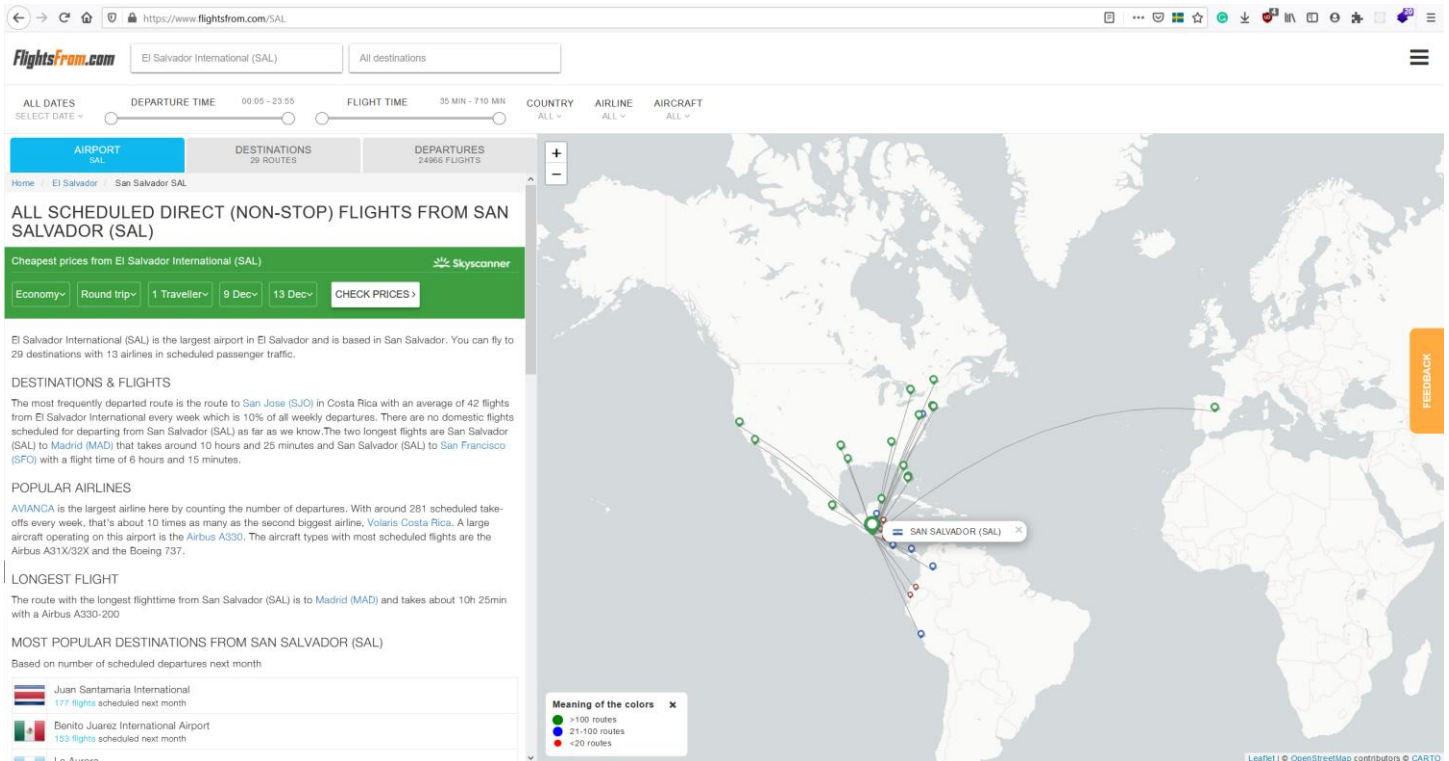


Según el sitio: [flightsfrom.com](https://www.flightsfrom.com)

El Aeropuerto Internacional De El Salvador vuela a 29 destinos con 13 aerolíneas. <https://www.flightsfrom.com/SAL>



Tarea:

Escriba el código de Digrafo.h, RutasBFS.h y RutasVuelos.cpp de acuerdo al contenido de este documento.

Complete el archivo ciudades.txt y vuelos.txt. Investigue si los destinos tienen vuelos directos de regreso a El Salvador, o si es mediante escala. Complete la información de acuerdo con sus resultados.

Agregue más destinos y rutas.

Tome capturas de pantalla de sus resultados. Suba su código y un documento con sus conclusiones y capturas.

Digrafo.h

```
Digrafo.h  RutaBFS.h  RutasVuelo.cpp
1  #ifndef DIGRAFO_H
2  #define DIGRAFO_H
3
4  #include <list>
5  #include <iterator>
6  #include <iostream>
7
8  using std::list;
9  using std::cout;
10 using std::endl;
11
12 class Digrafo {
13 private:
14     int vertices; // numero de vertices en este digrafo
15     int aristas; // numero de aristas en este digrafo
16     list<int> *adyacentes;
17 public:
18     Digrafo(); // constructor
19     Digrafo(const Digrafo &otro); // copy constructor
20     ~Digrafo(); // destructor
21     int getVertices() const; // obtener el numero de vertices
22     void setVertices(int vertices); // establecer el numero de vertices
23     int getAristas() const; // obtener el numero de aristas
24     list<int> getAdyacentes(int v) const; // obtener la lista de adyacentes
25     void agregarArista(int v, int w); // agregar la arista v-w
26     void imprimir(); // imprimir el grafo
27 };
28
29 Digrafo::Digrafo() : vertices(0), aristas(0), adyacentes(nullptr) {
30     // constructor vacio
31 }
32
33 Digrafo::Digrafo(const Digrafo &otro) {
34     vertices = otro.vertices;
35     aristas = otro.aristas;
36     if (vertices == 0) {
37         adyacentes = nullptr;
38     }
39     else {
40         adyacentes = new list<int>[vertices];
41         for (int i = 0; i < vertices; i++) {
42             list<int>::iterator it;
43             for (it = otro.adyacentes[i].begin(); it != otro.adyacentes[i].end(); it++)
44                 adyacentes[i].push_back(*it);
45         }
46     }
47 }
```

```
49 Digrafo::~Digrafo() {
50     if (adyacentes != nullptr) {
51         delete[] adyacentes;
52     }
53 }
54
55 int Digrafo::getVertices() const {
56     return vertices;
57 }
58
59 void Digrafo::setVertices(int vertices) {
60     if (this->vertices == 0) {
61         this->vertices = vertices;
62         adyacentes = new list<int>[vertices];
63     }
64 }
65
66 int Digrafo::getAristas() const {
67     return aristas;
68 }
69
70 void Digrafo::agregarArista(int v, int w) {
71     adyacentes[v].push_front(w);
72     aristas++;
73 }
74
75 list<int> Digrafo::getAdyacentes(int v) const {
76     return adyacentes[v];
77 }
78
79 void Digrafo::imprimir() {
80     for (int i = 0; i < vertices; i++) {
81         cout << i << ": ";
82         list<int>::iterator it;
83         for (it = adyacentes[i].begin(); it != adyacentes[i].end(); it++)
84             cout << *it << ' ';
85         cout << endl;
86     }
87 }
88
89 #endif // !DIGRAFO_H
```

RutasBFS.h

```
Digrafo.h  RutaBFS.h  RutasVuelo.cpp
1  #ifndef RUTA_BFS_H
2  #define RUTA_BFS_H
3  #include <limits>
4  #include <list>
5  #include <iterator>
6
7  #include "Digrafo.h"
8
9  using std::list;
10
11 class RutaBFS {
12 private:
13     const int INFINITO = std::numeric_limits<int>::max();
14     int vertices; // numero de vertices del grafo origen
15     bool *marcado; // marcado[v] = hay un camino entre s-v?
16     int *aristaHacia; // aristaHacia[v] = ultima arista en la ruta s-v mas corta
17     int *distanciaHacia; // distanciaHacia[v] = longitud de la ruta s-v mas corta
18     void bfs(const Digrafo &G, int s); // algoritmo BFS recursivo
19 public:
20     RutaBFS(const Digrafo &G, int s); // constructor
21     RutaBFS(const RutaBFS &otra); // copy constructor
22     ~RutaBFS(); // destructor
23     bool existeCaminoHacia(int v);
24     int getDistanciaHacia(int v);
25     list<int> getRutaHacia(int v);
26 };
27
28 RutaBFS::RutaBFS(const Digrafo &G, int s) {
29     vertices = G.getVertices();
30     marcado = new bool[vertices];
31     distanciaHacia = new int[vertices];
32     aristaHacia = new int[vertices];
33     for (int v = 0; v < vertices; v++) {
34         marcado[v] = false;
35         distanciaHacia[v] = INFINITO;
36     }
37     bfs(G, s);
38 }
39
40 void RutaBFS::bfs(const Digrafo &G, int s) {
41     list<int> q;
42     marcado[s] = true;
43     distanciaHacia[s] = 0;
44     q.push_back(s);
45     while (!q.empty()) {
46         int v = q.front();
47         q.pop_front();
48         list<int>::iterator it;
49         list<int> ady = G.getAdyacentes(v);
```



```
39
40 void RutaBFS::bfs(const Digrafo &G, int s) {
41     list<int> q;
42     marcado[s] = true;
43     distanciaHacia[s] = 0;
44     q.push_back(s);
45     while (!q.empty()) {
46         int v = q.front();
47         q.pop_front();
48         list<int>::iterator it;
49         list<int> ady = G.getAdyacentes(v);
50         for (it = ady.begin(); it != ady.end(); it++) {
51             int w = *it;
52             if (!marcado[w]) {
53                 aristaHacia[w] = v;
54                 distanciaHacia[w] = distanciaHacia[v] + 1;
55                 marcado[w] = true;
56                 q.push_back(w);
57             }
58         }
59     }
60 }
61
62 RutaBFS::RutaBFS(const RutaBFS &otra) {
63     vertices = otra.vertices;
64     marcado = new bool[vertices];
65     distanciaHacia = new int[vertices];
66     aristaHacia = new int[vertices];
67     for (int v = 0; v < vertices; v++) {
68         marcado[v] = otra.marcado[v];
69         distanciaHacia[v] = otra.distanciaHacia[v];
70         aristaHacia[v] = otra.aristaHacia[v];
71     }
72 }
73
74 RutaBFS::~~RutaBFS() {
75     delete[] marcado;
76     delete[] distanciaHacia;
77     delete[] aristaHacia;
78 }
79
80 bool RutaBFS::existeCaminoHacia(int v) {
81     return marcado[v];
82 }
83
```

```
83
84 = int RutaBFS::getDistanciaHacia(int v) {
85     return distanciaHacia[v];
86 }
87
88 = list<int> RutaBFS::getRutaHacia(int v) {
89     list<int> ruta;
90 =     if (existeCaminoHacia(v)) {
91         int x;
92         for (x = v; distanciaHacia[x] != 0; x = aristaHacia[x])
93             ruta.push_front(x);
94         ruta.push_front(x);
95     }
96     return ruta;
97 }
98
99 #endif // !RUTA_BFS_H
```

RutasVuelo.cpp

```
Digrafo.h  RutaBFS.h  RutasVuelo.cpp
1  #include <iostream>      // entrada y salida
2  #include <fstream>      // procesar archivos de texto
3  #include <string>       // cadenas de caracteres
4  #include <sstream>      // flujos de cadenas de caracteres
5  #include <vector>       // vectores (similares a los arreglos, pero pueden crecer dinamicamente)
6  #include <list>        // implementacion de listas de C++
7  #include <map>         // mapas (funcionan como los arboles binarios vistos en clase)
8  #include <iterator>    // sirve para recorrer los elementos de listas y mapas
9
10 #include "Digrafo.h"    // Grafo dirigido
11 #include "RutaBFS.h"    // Algoritmo de busqueda en anchura (BFS)
12
13 using std::ifstream;    // sirve para procesar archivos de texto
14 using std::string;      // cadenas de caracteres
15 using std::cout;        // salida en pantalla
16 using std::cin;         // leer entradas del teclado
17 using std::stoi;        // convertir cadenas a enteros
18 using std::endl;        // caracter de fin de linea
19 using std::stringstream; // flujo de caracteres de entrada
20 using std::vector;       // vectores
21 using std::map;          // mapas
22 using std::pair;         // pares clave-valor de un mapa
23 using std::list;         // listas
24 using std::iterator;     // iteradores
25
26 map<string, int> codigosCiudades;
27 vector<string> ciudades;
28
29 Digrafo rutas;
30
31 void procesarArchivoCiudades(string nombreArchivo) {
32     cout << "Abriendo archivo de ciudades: " << nombreArchivo << "..." << endl;
33     ifstream archivo(nombreArchivo);
34     string ciudad;
35
36     if (!archivo.is_open()) {
37         cout << "No se pudo abrir el archivo de ciudades.\n";
38         return;
39     }
40
41     cout << "\nProcesando archivo de ciudades..." << endl;
42 }
```

```
40
41     cout << "\nProcesando archivo de ciudades..." << endl;
42
43     vector<string> fila;
44     int numeroVertices = 0;
45     while(getline(archivo, ciudad)) {
46         codigosCiudades.insert(pair<string, int>(ciudad, numeroVertices++));
47         ciudades.push_back(ciudad);
48         cout << "Se agrego la ciudad " << ciudad << endl;
49     }
50
51     cout << "Cerrando archivo de ciudades" << nombreArchivo << "..." << endl;
52     archivo.close();
53
54     cout << endl;
55 }
56
57 void procesarArchivoVuelos(string nombreArchivo) {
58     cout << "Abriendo archivo de vuelos " << nombreArchivo << "..." << endl;
59     ifstream archivo(nombreArchivo);
60     string linea, token;
61     string origen, destino;
62
63     if (!archivo.is_open()) {
64         cout << "No se pudo abrir el archivo de vuelos.\n";
65         return;
66     }
67
68     cout << "\nProcesando archivo de vuelos..." << endl;
69
70     // Crear el grafo
71     // Obtener el numero de aristas
72     rutas.setVertices(ciudades.size());
73
74     vector<string> fila;
75     while (getline(archivo, linea)) {
76         fila.clear();
77         istringstream ss(linea);
78         while (getline(ss, token, ','))
79             fila.push_back(token);
80         origen = fila[0];
81         destino = fila[1];
82         rutas.agregarArista(codigosCiudades[origen], codigosCiudades[destino]);
83         cout << "Se agrego el vuelo " << origen << " -> " << destino << endl;
84     }
85
86     cout << "Cerrando archivo de vuelos" << nombreArchivo << "..." << endl;
87     archivo.close();

```



```

Digrafo.h  RutaBFS.h  RutasVuelo.cpp
85
86     cout << "Cerrando archivo de vuelos" << nombreArchivo << "..." << endl;
87     archivo.close();
88
89     cout << endl;
90
91     cout << "Se creo el siguiente grafo:" << endl;
92
93     rutas.imprimir();
94
95     cout << endl;
96 }
97
98 void procesarSolicitudVuelo() {
99     string ciudadOrigen, ciudadDestino;
100
101     cout << "Ciudad origen: ";
102     getline(cin, ciudadOrigen);
103
104     if (codigosCiudades.count(ciudadOrigen) == 0) {
105         cout << "Lo siento, no volamos desde esa ciudad." << endl;
106         return;
107     }
108
109     cout << "Ciudad destino: ";
110     getline(cin, ciudadDestino);
111
112     if (codigosCiudades.count(ciudadDestino) == 0) {
113         cout << "Lo siento, no volamos hacia esa ciudad." << endl;
114         return;
115     }
116
117     RutaBFS itinerario(rutas, codigosCiudades[ciudadOrigen]);
118
119     if (!itinerario.existeCaminoHacia(codigosCiudades[ciudadDestino])) {
120         cout << "No se pudo encontrar un itinerario de vuelo." << endl;
121     }
122     else {
123         cout << "Itinerario sugerido: " << endl;
124         list<int>::iterator it;
125         list<int> conexiones = itinerario.getRutaHacia(codigosCiudades[ciudadDestino]);
126         for (it = conexiones.begin(); it != conexiones.end(); it++) {
127             int codigo = *it;
128             if (codigo == codigosCiudades[ciudadDestino])
129                 cout << ciudadDestino << endl;
130             else
131                 cout << ciudades[codigo] << " -> ";
132         }
133     }

```

```
Digrafo.h RutaBFS.h RutasVuelo.cpp
113     cout << "Lo siento, no volamos hacia esa ciudad." << endl;
114     return;
115 }
116
117 RutaBFS itinerario(rutas, codigosCiudades[ciudadOrigen]);
118
119 if (!itinerario.existeCaminoHacia(codigosCiudades[ciudadDestino])) {
120     cout << "No se pudo encontrar un itinerario de vuelo." << endl;
121 }
122 else {
123     cout << "Itinerario sugerido: " << endl;
124     list<int>::iterator it;
125     list<int> conexiones = itinerario.getRutaHacia(codigosCiudades[ciudadDestino]);
126     for (it = conexiones.begin(); it != conexiones.end(); it++) {
127         int codigo = *it;
128         if (codigo == codigosCiudades[ciudadDestino])
129             cout << ciudadDestino << endl;
130         else
131             cout << ciudades[codigo] << " -> ";
132     }
133 }
134 }
135
136 int mostrarMenu() {
137     string eleccion;
138     cout << "\nMenu principal" << endl;
139     cout << "1- Buscar un vuelo" << endl;
140     cout << "2- Salir" << endl;
141     cout << "Su eleccion: ";
142     getline(cin, eleccion);
143     cout << endl;
144     return stoi(eleccion);
145 }
146
147 int main()
148 {
149     procesarArchivoCiudades("ciudades.txt");
150     procesarArchivoVuelos("vuelos.txt");
151
152     int opcion = mostrarMenu();
153
154     while (opcion != 2) {
155         procesarSolicitudVuelo();
156         opcion = mostrarMenu();
157     }
158
159     return 0;
160 }
```

ciudades.txt

San Salvador
San Francisco
Miami
Hong Kong
Paris
Bogota
Madrid
Guatemala

vuelos.txt

San Salvador,Miami
Miami,Paris
Paris,Hong Kong
San Salvador,Bogota
Bogota,Paris
Miami,San Francisco
San Salvador,Madrid
Madrid,Guatemala
Guatemala,San Salvador