```
#include <iostream>
#include <stdlib.h>

using namespace std;

struct nodo_grafo
{
        int id_nodo;
        struct nodo_arco* lista_arco;
        struct nodo_grafo* link;
};
typedef struct nodo_grafo NGrafo;

struct nodo_arco
{
        int id_arco;
        struct nodo_grafo* destino;
        struct nodo_arco* link;
};
typedef struct nodo_arco NArco;

void grafo_mostrar (NGrafo* lista_n);
void grafo_agregar_nodo (NGrafo* &lista_n, int id_nodo);
void grafo_agregar_arco (NGrafo* lista_n, int id_arco, int
id_nodo_origen, int id_nodo_destino);

void menu_opcion1 (NGrafo* lista_n);
void menu_opcion2 (NGrafo* &lista_n);
void menu_opcion3 (NGrafo* &lista_n);

int main (void)
{
        NGrafo* lista_n = NULL;

        int opcion = 0;
        do {
                cout << "************Menu de Opciones************\n";
                cout << endl;
                cout << "****** Grafos ******\n";
                cout << endl;
                cout << "1- Mostrar.\n";
                cout << "2- Insertar Nodo.\n";
                cout << "3- Insertar Arco.\n";
                cout << endl;
                cout << "    0- Salir\n";
                cout << endl;
                cout << "                          Ingrese opcion: ";
                cin >> opcion;
                cout << endl;
                cout << endl;

                switch(opcion)
                {
```

```
                case 1:
                        menu_opcion1 (lista_n);
                        break;
                case 2:
                        menu_opcion2 (lista_n);
                        break;
                case 3:
                        menu_opcion3 (lista_n);
                        break;
                }
        } while ( opcion != 0);

        return 0;
}


void menu_opcion1 (NGrafo* lista_n)
{
        grafo_mostrar (lista_n);
}

void menu_opcion2 (NGrafo* &lista_n)
{
        int id_nodo;

        cout << "Ingrese el id_nodo del nodo que desea incorporar: ";
        cin >> id_nodo;
        grafo_agregar_nodo (lista_n, id_nodo);
        cout << endl;
        cout << endl;
}

void menu_opcion3 (NGrafo* &lista_n)
{
        int id_arco, id_nodo_origen, id_nodo_destino;

        cout << "Ingrese el id_arco del arco que desea incorporar: ";
        cin >> id_arco;
        cout << "Ingrese el id_nodo del nodo desde donde sale el arco: ";
        cin >> id_nodo_origen;
        cout << "Ingrese el id_nodo del nodo a donde llega el arco: ";
        cin >> id_nodo_destino;

        grafo_agregar_arco (lista_n, id_arco, id_nodo_origen,
id_nodo_destino);

        cout << endl;
        cout << endl;
}


void grafo_mostrar (NGrafo* lista_n)
{
```

```
        cout << "Grafo:\n\n";

        while (lista_n != NULL)
        {
                cout << "Nodo " << lista_n->id_nodo << ":" << endl;

                NArco* aux_la = lista_n->lista_arco;
                while (aux_la != NULL)
                {
                        cout << "  Arco " << aux_la->id_arco << " -> Nodo
" << aux_la->destino->id_nodo << endl;
                        aux_la = aux_la->link;
                }

                lista_n = lista_n->link;
        }
        cout << endl;
        cout << endl;
}

void grafo_agregar_nodo (NGrafo* &lista_n, int id_nodo)
{
        NGrafo* aux = lista_n;

        while (aux != NULL)
        {
                if (aux->id_nodo == id_nodo)
                {
                        cout << "!!! Error: Ya existe un nodo con ese
id_nodo." << endl;
                        return;
                }
                aux = aux->link;
        }

        aux = new (NGrafo);
        aux->id_nodo = id_nodo;
        aux->lista_arco = NULL;
        aux->link = lista_n;
        lista_n = aux;
}

void grafo_agregar_arco (NGrafo* lista_n, int id_arco, int
id_nodo_origen, int id_nodo_destino)
{
        NGrafo* nodo_origen = lista_n;
        NGrafo* nodo_destino = lista_n;

        while (nodo_origen != NULL && nodo_origen->id_nodo !=
id_nodo_origen)
                nodo_origen = nodo_origen->link;
```

```
        while (nodo_destino != NULL && nodo_destino->id_nodo !=
id_nodo_destino)
                nodo_destino = nodo_destino->link;

        if (nodo_origen == NULL || nodo_destino == NULL)
        {
                cout << "!!! Error: Alguno de los nodos no existe." <<
endl;
                return;
        }

        NArco* arco_actual = nodo_origen->lista_arco;
        while(arco_actual != nullptr){

                if(arco_actual->id_arco == id_arco){
                        cout<<"El arco ya existe pa!!!"<<endl;
                        return;
                }

                arco_actual = arco_actual->link;
        }

        NArco* aux = new (NArco);
        aux->id_arco = id_arco;
        aux->destino = nodo_destino;
        aux->link = nodo_origen->lista_arco;
        nodo_origen->lista_arco = aux;
}
```

```
// Retorna el id de la habitacion de "entrada".
NGrafo* crear_laberinto (NGrafo* &l_nodo, int cant_habitacion, int maxfo,
int profundidad)
{
        // Generar nodos.
        for (int i=0; i< cant_habitacion; i++)
                grafo_agregar_nodo (l_nodo, i);

        int id_inicio = rand () % cant_habitacion;
        NGrafo* p_inicio = buscar_nodo (l_nodo, id_inicio);

    generar_conexiones (l_nodo, cant_habitacion, p_inicio, maxfo,
profundidad);

        return p_inicio;
}

void generar_conexiones (NGrafo* l_nodo, int cant_habitacion, NGrafo*
inicio, int maxfo, int profundidad)
{
        cout << "### EN HABITACION " << inicio->id_nodo << " ###." <<
endl;
        cout << "  La profundidad es de " << profundidad << "." << endl;
        cout << "  El maxfo es de " << maxfo << "." << endl;

        if ((profundidad <= 0) or (maxfo <= 0))
                return;

        int fo_aqui = (rand () % maxfo) + 1;
        cout << "  Se generan " << fo_aqui << " arcos." << endl;
        for (int i=0; i<fo_aqui; i++)
        {
                int id_nodo_destino = rand () % cant_habitacion;
                NGrafo* p_nodo_destino = buscar_nodo (l_nodo,
id_nodo_destino);
                // Deberia generar un id_arco razonable. NO esta bien que
todos los arcos tengan 1.
                grafo_agregar_arco (l_nodo, 1, inicio->id_nodo,
id_nodo_destino);

                cout << "Se genero un portal desde " << inicio->id_nodo <<
" hacia " << id_nodo_destino << "." << endl;

                generar_conexiones (l_nodo, cant_habitacion,
p_nodo_destino, fo_aqui, --profundidad);

                // Alternativa que genera grafos más chicos.
                // generar_conexiones (l_nodo, cant_habitacion,
p_nodo_destino, --fo_aqui, --profundidad);
        }
}
```

```
void grafo_eliminar_arco ( NGrafo* lista_n,
                           int id_nodo_origen,
                           int id_nodo_destino) {
      NGrafo* nodo_origen = lista_n;

      while ( nodo_origen != NULL &&
              nodo_origen->id_nodo != id_nodo_origen)
            nodo_origen = nodo_origen->link;

      // Verificamos si existe id_nodo_origen
      if (nodo_origen == NULL )
      {
            throw -1;        // Origen Inexistente
            return;
      }

      NArco*  anterior    = NULL;
      NArco*  actual    = nodo_origen->lista_arco;

      while (actual != NULL &&
              actual->destino->id_nodo != id_nodo_destino) {
            anterior = actual;
            actual = actual->link;
      }
      if (actual == NULL) {
            throw -2;        // Arco Inexistente
      } else {              // actual != NULL hay nodo a borrar
            if (anterior == NULL && actual != NULL) {
                  // quiere decir que es el 1er arco, hay que
                  // actualizar el puntero del nodo
                  nodo_origen->lista_arco = actual->link;
            } else {    // es un arco intermedio o el ultimo
                  anterior->link = actual->link;
            }
            delete actual;
      }
}
```

Algoritmos y Estructuras de Datos

Teoría de Grafos

$G=(P,R)$ donde $P=\{x/x \text{ es un nodo}\}$ $R=\{(x,y) / x,y \in P \wedge xRy\}$

Def. por extensión y por comprensión

Funciones de Asignación

Cómo implementar un grafo. Estructuras estáticas y dinámicas. Diseño de celdas.

Paso $\rho(x,z)$ es la secuencia $<y_0, y_1, ..., y_n>$ $n\geq 0$ /

1. $x = y_0$; $z = y_n$ 2. $y_{i-1} \neq y_i$ 3. $(y_{i-1}, y_i) \in R$ $1\leq i\leq n$

$|\rho(x,z)|$ = nº de arcos entre x y z

Camino: $C(x,z)$ es la secuencia $<y_0, y_1, ..., y_n>$ $n\geq 0$ /

1. $x = y_0$; $z = y_n$ 2. $y_{i-1} \neq y_i$ 3. $(y_{i-1}, y_i) \in R \vee (y_i , y_{i-1}) \in R$ $1\leq i\leq n$

$|C(x,z)|$ = nº de conexiones entre x y z

Ciclo: $|\rho(x,x)|\geq 2$ Circuito: $|C(x,x)| \geq 2$ Loop: $|\rho(x,x)| = 0$

$L(x) = \{y/y\in P; (y,x) \in R\}$ $R(x) = \{z/z\in P; (x,z) \in R\}$

$\overline{L(x)} = \{y/y\in P; \exists \rho(y,x)\}$ $\overline{R(x)} = \{z/z\in P; \exists \rho(x,z)\}$

$|L(x)|$ = cantidad de arcos que llegan a x $|R(x)|$ = cantidad de arcos que salen de x

Minimal = $\{ x / x\in P , |L(x)| = 0 \}$ Maximal = $\{ z / z\in P , |R(z)| = 0 \}$

Mínimo = x es mín si $|L(x)| = 0 \wedge$ x es único. Máximo = z es máx si $|R(z)| = 0 \wedge$ z es único.

Grafo Básico: 1. Libre de loops. 2. $\forall x,y \in P$, si $\exists |\rho(x,y)|\geq 2 \Rightarrow (x,y) \notin R$

Grafos Isomorfos: dos grafos $G_1 = (P_1,R_1)$ $G_2 = (P_2,R_2)$ son isomorfos $G_1 \cong G_2$ si $\exists \varphi: P_1 \rightarrow P_2$ /

$\forall x,y \in P_1$: $(x,y) \in R_1 \Leftrightarrow (\varphi(x), \varphi(y)) \in R_2 \wedge \varphi(x) , \varphi(y) \in P_2$

Subgrafo: dado $G=(P,R)$ $G' = (P',R')$ será subgrafo de G si :

1. $P' \subseteq P$ 2. $R' = R_{|P}$

Definición de la Estructura En C++ - Alternativa Básica

```
#define M 6

int   A[M][M];
char etiqueta[M];
```

Definición de la Estructura En C++ - Alternativa Completa

```
#define M 6

int   A[M][M];
struct InfNodo datosNodo[M];
struct infArco  datosArco[M][M];
```

Características

- M se debe conocer de antemano, es decir se usan para grafos poco volátiles.
- Permiten ejecutar consultas rápidas.
- Estas matrices no se almacenan, se alojan en memoria.