UNIVERSIDAD NEBRIJA

Escuela Politécnica Superior

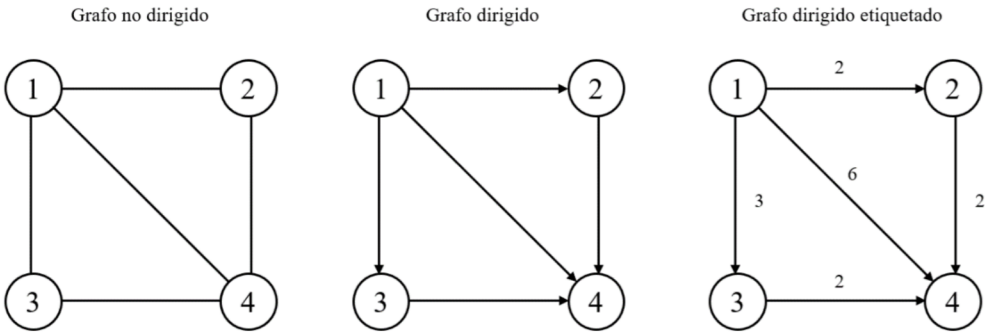
IIN151 – Estructuras de datos y algoritmos

Prácticas de laboratorio 5

Nombre: _____

Un grafo $G = (V, A)$ se define como un conjunto finito V de vértices y un conjunto A de aristas. Los vértices representan los nodos de un grafo y las aristas indican las uniones entre vértices. Si los vértices o las aristas tienen una etiqueta asociada, el grafo está etiquetado. Una etiqueta puede representar un nombre, un coste, una distancia o cualquier otro dato.

Un grafo no dirigido $G = (V, A)$ consta de un conjunto finito de vértices V y un conjunto de aristas A, donde cada arista es un par no ordenado de vértices. Un grafo dirigido $G = (V, A)$ consta de un conjunto finito de vértices V y un conjunto de aristas A, donde cada arista es un par ordenado de vértices.



Las aristas de un grafo dirigido son pares ordenados. La arista $v \rightarrow w$ se representa (v, w) y es distinta de (w, v).

La interfaz del TAD grafo.

```
inicializa(G, n) Inicializa el grafo G

  Precondición:  Ninguna
  Postcondición: Un grafo con n vértices

inserta(G, v, w, coste) Inserta la arista (v, w) con el coste indicado

  Precondición:  Ninguna
  Postcondición: El grafo incluye la arista (v, w)

imprime(G) Muestra la tabla de aristas del grafo

  Precondición:  Ninguna
  Postcondición: Ninguna
```

Implemente el TAD grafo utilizando el siguiente código.

```
const int INFINITO = 9999;

struct TGrafo {
   int vertices;
   int **aristas;
};

void inicializa(struct TGrafo *G, int vertices) {
   G->vertices = vertices;
   G->aristas = new int* [G->vertices];

   for (int i = 0; i < G->vertices; i++)
      for (int j = 0; j < G->vertices; j++)
         G->aristas[i] = new int[G->vertices];

   for (int i = 0; i < G->vertices; i++)
      for (int j = 0; j < G->vertices; j++)
         G->aristas[i][j] = INFINITO;
}

void inserta(struct TGrafo *G, int v, int w, int coste) {
   if (v >= 1 && v <= G->vertices && w >= 1 && w <= G->vertices)
      G->aristas[v - 1][w - 1] = coste;
}
```

```
void imprime(struct TGrafo *G, const char s[]) {
   std::cout << "\n" << s << "\n";

   for (int i = 0; i < G->vertices; i++) {
      std::cout << "\n[" << (i + 1) << "] ";

      for (int j = 0; j < G->vertices; j++)
         if (G->aristas[i][j] == INFINITO)
            std::cout << " -- ";
         else {
            if (G->aristas[i][j] < 10)
               std::cout << "  " << G->aristas[i][j] << " ";
            else
               if (G->aristas[i][j] < 100)
                  std::cout << " " << G->aristas[i][j] << " ";
               else
                  std::cout << G->aristas[i][j] << " ";

         }
   }

   std::cout << "\n";
}

void imprimeVector(int *v, int vertices, const char s[]) {
   std::cout << "\n" << s << " { ";

   for (int i = 1; i < vertices; i++)
      if (v[i] == INFINITO)
         std::cout << " -- ";
      else {
         if (v[i] < 10)
            std::cout << "  " << v[i] << " ";
         else
            if (v[i] < 100)
               std::cout << " " << v[i] << " ";
            else
               std::cout << v[i] << " ";
      }

   std::cout << "}\n";
}
```

```
void imprimeTabla(int **m, int vertices, const char s[]) {
   std::cout << "\n" << s << "\n";

   for (int i = 0; i < vertices; i++) {
      std::cout << "\n[" << (i + 1) << "] ";

      for (int j = 0; j < vertices; j++)
         if (m[i][j] == INFINITO)
            std::cout << " -- ";
         else {
            if (m[i][j] < 10)
               std::cout << "  " << m[i][j] << " ";
            else
               if (m[i][j] < 100)
                  std::cout << " " << m[i][j] << " ";
               else
                  std::cout << m[i][j] << " ";
         }
   }

   std::cout << "\n";
}

void dfs(struct TGrafo *G, int v, bool *&visitado) {

   // recorrido en profundidad de un grafo

}

void profundidad(struct TGrafo *G, int inicio) {

   // recorrido en profundidad de un grafo

}

void dijkstra(struct TGrafo *G, int vertices, int *&D) {

   // algoritmo de Dijkstra para resolver el problema del camino
   // más corto desde un origen

}

void floyd(struct TGrafo *G, int vertices, int **&A, int **&P) {

   // algoritmo de Floyd para resolver el problema del camino
   // más corto entre todos los pares de vértices

}
```

```
void costeMinimoArista(struct TGrafo *G,
                       Conjunto U, Conjunto W, int &u, int &w) {
   int min = INFINITO;

   for (int i = 1; i <= G->vertices; i++)
      if (U.pertenece(i))
         for (int j = 1; j <= G->vertices; j++)
            if (W.pertenece(j))
               if (coste(G, i, j) < min) {
                  min = coste(G, i, j);
                  u = i;
                  w = j;
               }
}
void prim(struct TGrafo *G, int vertices, struct TGrafo *&T) {

   // algoritmo de Prim para calcular el árbol de recubrimiento
   // mínimo

}
```

El programa de prueba del TAD grafo.

```
int main() {

   int *C;
   struct TGrafo *G1 = new TGrafo;

   inicializa(G1, 5);

   inserta(G1, 1, 2, 10);
   inserta(G1, 1, 4, 30);
   inserta(G1, 1, 5, 100);
   inserta(G1, 2, 3, 50);
   inserta(G1, 3, 5, 10);
   inserta(G1, 4, 3, 20);
   inserta(G1, 4, 5, 60);

   imprime(G1, "Grafo para Dijkstra");
   dijkstra(G1, 5, C);

   imprimeVector(C, 5, "Costes minimos Dijkstra");
```

```
    int **A, **P;
    struct TGrafo *G2 = new TGrafo;

    inicializa(G2, 3);

    inserta(G2, 1, 1, 2);
    inserta(G2, 1, 2, 8);
    inserta(G2, 1, 3, 5);
    inserta(G2, 2, 1, 3);
    inserta(G2, 3, 2, 2);

    imprime(G2, "Grafo para Floyd");

    floyd(G2, 3, A, P);

    imprimeTabla(A, 3, "Costes minimos Floyd");
    imprimeTabla(P, 3, "Aristas Floyd");

  struct TGrafo *G3 = new TGrafo;

    inicializa(G3, 6);

    inserta(G3, 1, 2, 6);
    inserta(G3, 2, 1, 6);
    inserta(G3, 1, 3, 1);
    inserta(G3, 3, 1, 1);
    inserta(G3, 1, 4, 5);
    inserta(G3, 4, 1, 5);
    inserta(G3, 2, 3, 5);
    inserta(G3, 3, 2, 5);
    inserta(G3, 2, 5, 3);
    inserta(G3, 5, 2, 3);
    inserta(G3, 3, 4, 5);
    inserta(G3, 4, 3, 5);
    inserta(G3, 3, 5, 6);
    inserta(G3, 5, 3, 6);
    inserta(G3, 3, 6, 4);
    inserta(G3, 6, 3, 4);
    inserta(G3, 4, 6, 2);
    inserta(G3, 6, 4, 2);
    inserta(G3, 5, 6, 6);
    inserta(G3, 6, 5, 6);

    imprime(G3, "Grafo G para Prim");
```

```
    struct TGrafo *T = new TGrafo;

    prim(G3, 6, T);

    imprime(T, "Arbol de recubrimiento con coste minimo");

    imprime(G1, "Grafo para recorrido en profundidad");

    profundidad(G1, 1);

}
```