PROYECTO FINAL – Hito II:

Programación Orientada a Objetos

Sección 1.09

Alumnos:

William Jarod Rojas Criollo

Carrera: Ciencias de la computación

201910294

Daniela Abril Vento Bustamante

Carrera: Ciencias de la computación

201910331

Profesor:

Marvin Abisrror Zarate

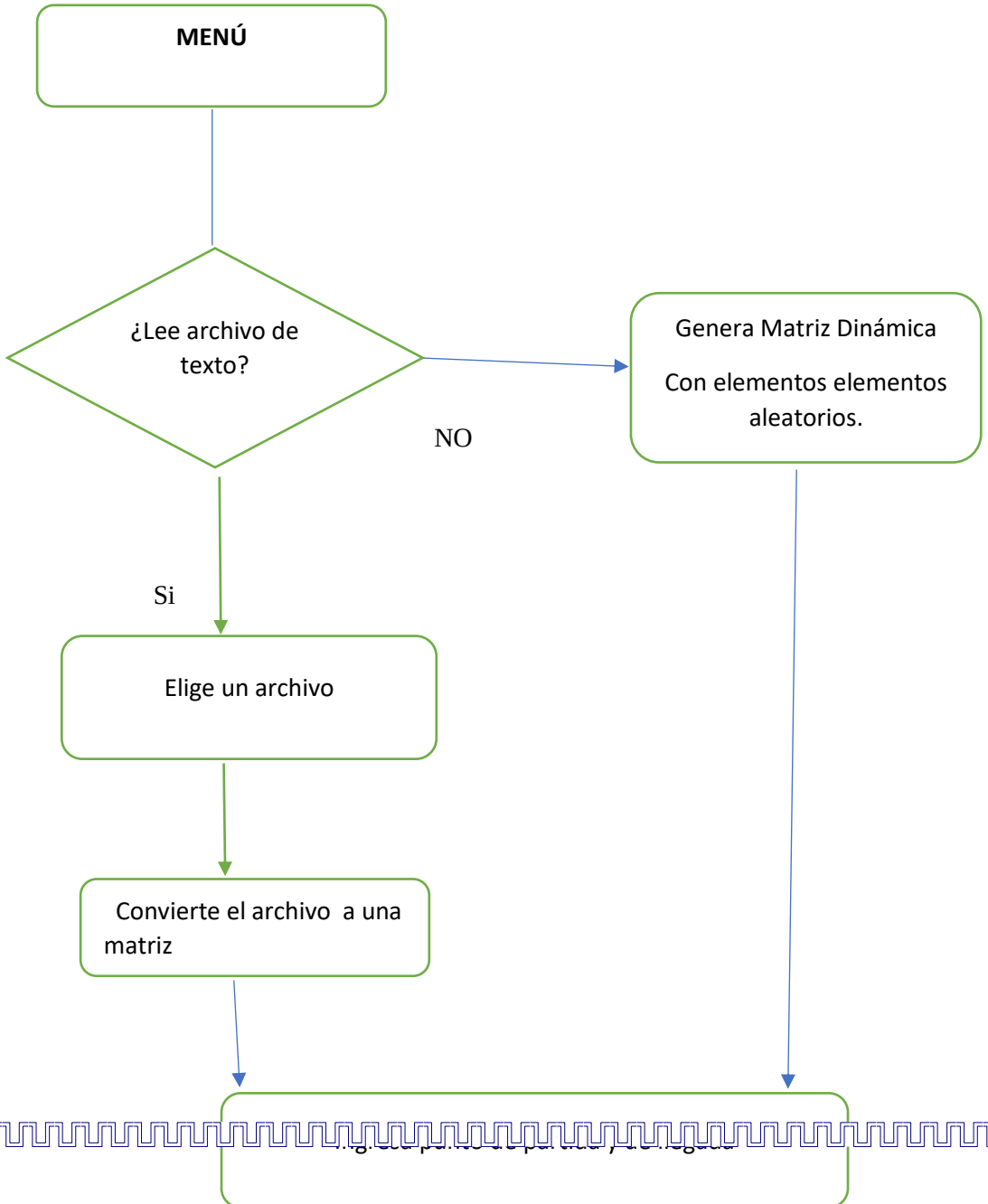
Lima, noviembre 2019

Índice:

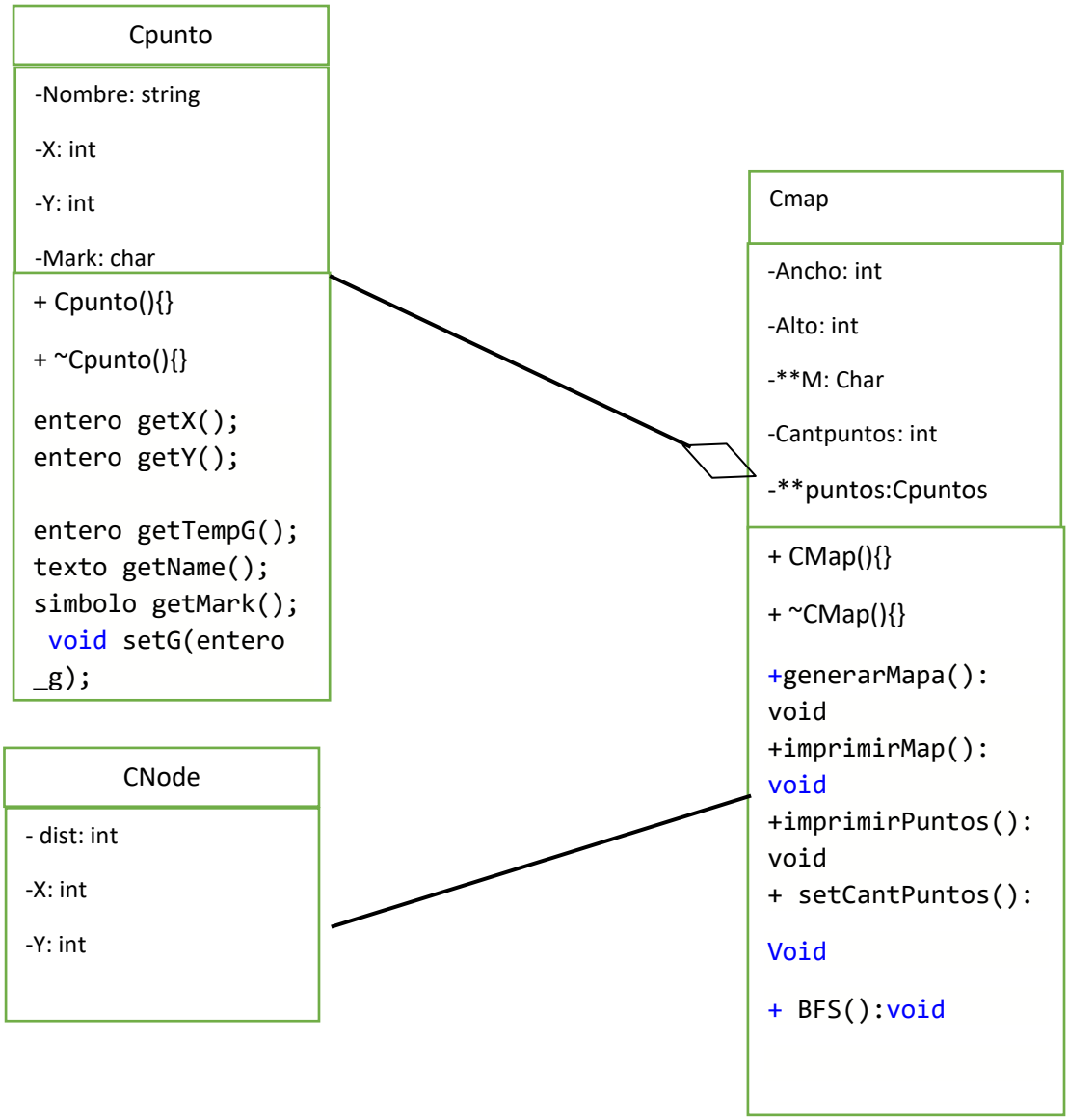
1.Diagramas.

En este punto mostraremos de forma gráfica, el comportamiento del código formulado para resolver el caso planteado para el proyecto.

1.



1.2 Diagrama de composición



2.Explicación del algoritmo BFS:

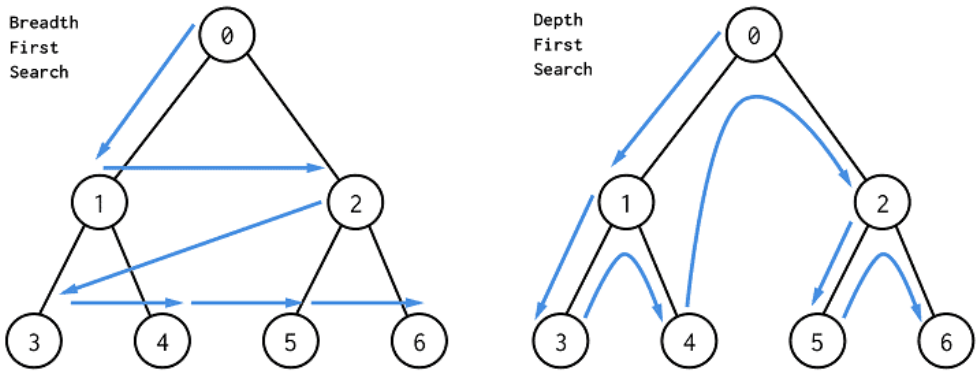
El algoritmo BFS(Breadth First Search) es una forma de encontrar todos los vértices alcanzables partiendo de un punto de origen. Este algoritmo consiste en la búsqueda de 3 tipos de vértices los cuales son: Vértices sin explorar, vértices explorados y el vértice final..

2.1 Elección del algoritmo:

La elección del algoritmo se debió a su fácil compresión por parte de los estudiantes que realizan el proyecto además de incluir conceptos vistos el ciclo pasado en la clase de introducción a las ciencias de la computación como el sistema de cola FIFO para implementar el algoritmo.

2.2 Funcionamiento:

El algoritmo funciona con la introducción de un nodo de origen, el cual será introducido a la cola junto con sus nodos adyacentes si se puede transitar por estos. Los procesos mencionados anteriormente son fundamentales para el correcto funcionamiento del algoritmo.



2.3 Implementación:

Para implementar con eficiencia el algoritmo utilizaremos una cola para la cual utilizaremos la librería <queue>,la cola a utilizar es el FIFO(First in First Out) en la cual introduciremos nodos a la cola y los analizaremos; si la cola está vacía todos los nodos posibles ya fueron analizados. Por lo tanto se implementara un serie de funciones en la cual se corroborará si el nodo adyacente es transitable o no ,si es transitable le pasaremos la dirección de memoria de la matriz dinámica para poder alterar los valores marcando por donde pasa el algoritmo mostrando al usuario el proceso de el algoritmo.

```
void CMap::BFS(entero i, entero j, entero x, entero y)
{
 int row[] = { -1, 0, 0, 1 };
 int col[] = { 0, -1, 1, 0 };
 char temp=M[x][y];
 booleano visited[20][20];



memset(visited, false, sizeof (visited));

 queue<CNode> q;

 visited[i][j] = true;
 q.push({i, j, 0});

 entero min_dist = INT_MAX;

 while (!q.empty())
 {
  CNode node = q.front();
  q.pop();
  entero i = node.x, j = node.y, dist = node.dist;
  if (i == x && j == y)
  {
   min_dist = dist;
   break;
  }

  for (int k = 0; k < 4; k++)
  {
   if (isValid(visited, i + row[k], j + col[k]) && M[i+row[k]][j+col[k]] != '1')
   {int n,m;
    visited[i + row[k]][j + col[k]] = true;

    M[i+row[k]][j+col[k]] = '-';
    q.push({ i + row[k], j + col[k], dist + 1 });
    n=i+row[k];
    m=j+col[k];
   }
  }
   M[x][y]=temp;
 }

 if (min_dist != INT_MAX)
  { imprimirMap();
    cout << "La distancia más corta tiene longitud "
     << min_dist<<"\n";}
 else
  cout << "No es posible calcular la ruta"<<"\n";
}
```

3. Explicación del código.

El código desarrollado posee como cualidad principal el hecho de leer matrices por archivo o generarlas de forma aleatoria dependiendo del usuario; por otro lado, el programa cuenta con 12 librerías provenientes de el mismo lenguaje c++ y la declaración de 4 tipos para evitar trabajar con grupos nativos, el código utiliza principalmente una matriz dinámica para poder ser alterada por el algoritmo BFS, también cuenta con la implementación de la clase nodo y la clase punto. El programa fue desarrollado para poder trabajar con mas de dos nodos calculando la ruta de dos puntos pasando por un tercero o la cantidad de punto adicionales que se pidan por parte del usuario.

3.1 Descripción de las clases y la elección de los métodos

En este punto se dará a conocer con mayor detalle las clases implementadas en el proyecto.

3.1.1. Clase Punto

La clase punto tiene como partes privadas la posición en x e y siendo ambas de tipo entero un símbolo de tipo char para poder reconocerlo dentro de la matriz y un nombre propio , se pude notar que en la parte publica hay ciertos setter y getter con variables no reconocidas como G,F entre otros ya que esta clase también pude trabajarse con el algoritmo de búsqueda llamada A*.

```
class CPunto {

private:
    entero x;
    entero y;
    simbolo mark;
    texto name;

public:
    CPunto (texto _name, simbolo _mark, entero _x, entero _y);
    CPunto (const CPunto &otroPunto);
    entero getX();
    entero getY();
    entero getG();
    entero getF();
    entero getH();
    entero getTempG();
    texto getName();
    simbolo getMark();
    ~CPunto ();
    void setG(entero _g);
    void setF(entero _f);
    void setH(entero _h);
    void setTempG(entero _tg);
    void setX(entero _x);
    void setY(entero _y);
    void setName(texto _name);
    void setMark(simbolo _mark);
    void imprimirXY();
};
```

3.1.2 Clase Mapa

La clase mapa tiene la relación de agregación con la clase punto ya que la clase mapa puede existir sin necesidad de la clase punto ,para poder agregar los puntos se crea un puntero a un puntero para poder almacenar los puntos que se agregaran, además, la clase también posee como atributos ancho, alto y a cantidad de puntos siendo de tipo entero. En esta clase también se encuentra el algoritmo BFS para hallar la ruta optima.

```
class CMap {

private:
  entero ancho;
  entero alto;
  simbolo** M;
  entero cantPuntos;
  CPunto** puntos = NULL;
public:
  CMap();
  CMap (entero _ancho, entero _alto);
  void generarMapa();
  void addPunto(CPunto* punto, entero i);
  CPunto* removerPunto(texto _punto);
  CPunto* buscarPunto(texto _punto);
  void updateList();
  void modificarPunto(CPunto* _punto);
  void updateMap();
  booleano verificarNombre(string nombre);
  void imprimirMap();
  void imprimirPuntos();
  void setCantPuntos(entero n) {cantPuntos = n;}
  entero getAncho();
  entero getAlto();
  entero getCantPuntos();
  simbolo** getMap();
  booleano isValid(booleano visited[20][20], entero row, entero col);
  void BFS(entero startX, entero startY, entero finishX, entero finishY);
  ~CMap ();
};
```

3.1.3 Clase Nodo

Esta clase en el momento de la planeación pude ser reemplazada por un Struct debido a que todos sus atributos son públicos. La razón de ser de esta clase es solo para facilitar la implementación de el BFS cuando se utilice la cola FIFO.

```
class CNode
{
 public:
 int x, y, dist;
};
```