

**GUÍA EXAMEN**

**Tercer Departamental**

**Instituto Politécnico Nacional**

**Escuela Superior de Cómputo**

**(ESCOM)**

**Grupo: 2CM1**

**Profesor: Tecla Parra Roberto**

**Materia: Algoritmos y Estructura de Datos**

**Alumno: Díaz Amaro Josué Ramsés**

**Fecha: 25/06/2025**

**3a Guía de Estructura de Datos**

1.-Mencione 3 tipos de grafos

|  |  |  |
| --- | --- | --- |
| 1.- | 2.- | 3.- |

2.-Mencione 2 formas de representar grafos en un programa

|  |  |
| --- | --- |
| 1.- | 2.- |

|  |  |
| --- | --- |
| A.-Dada la matriz de adyacencia de abajo dibujar el grafo | B.-Dada la matriz de adyacencia de abajo dibujar el grafo |
| |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  | *v1* | *v2* | *v3* | *v4* | *v5* | | *v1* | 1 | 1 | 1 | 1 | 1 | | *v2* | 1 | 1 | 1 | 1 | 1 | | *v3* | 1 | 1 | 1 | 1 | 1 | | *v4* | 1 | 1 | 1 | 1 | 1 | | *v5* | 1 | 1 | 1 | 1 | 1 | | |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  | *v1* | *v2* | *v3* | *v4* | *v5* | | *v1* | 1 | 1 | 1 | 0 | 0 | | *v2* | 1 | 1 | 1 | 1 | 0 | | *v3* | 1 | 1 | 1 | 1 | 1 | | *v4* | 0 | 1 | 1 | 1 | 0 | | *v5* | 0 | 0 | 1 | 0 | 1 | |
| C.-Dada la matriz de adyacencia de abajo dibujar el grafo | D.-Dada la matriz de adyacencia de abajo dibujar el grafo |
| |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  | *v1* | *v2* | *v3* | *v4* | *v5* | | *v1* | 1 | 0 | 1 | 1 | 1 | | *v2* | 0 | 1 | 0 | 1 | 1 | | *v3* | 1 | 0 | 1 | 0 | 1 | | *v4* | 1 | 1 | 0 | 1 | 0 | | *v5* | 1 | 1 | 1 | 0 | 1 | | |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  | *v1* | *v2* | *v3* | *v4* | *v5* | | *v1* | 1 | 1 | 0 | 0 | 1 | | *v2* | 1 | 1 | 1 | 0 | 0 | | *v3* | 0 | 1 | 1 | 1 | 0 | | *v4* | 0 | 0 | 1 | 1 | 1 | | *v5* | 1 | 0 | 0 | 1 | 1 | |
| A.-Dada la matriz de adyacencia de abajo dibujar el grafo y hallar su cerradura transitiva |  |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | |  | *v1* | *v2* | *v3* | *v4* | | *v1* | 0 | 1 | 1 | 0 | | *v2* | 1 | 0 | 0 | 1 | | *v3* | 1 | 0 | 0 | 1 | | *v4* | 0 | 1 | 1 | 0 | |  |

dibuje el grafo que corresponde a la sig lista de adyacencia

"A": ["C", "D", "F"],

"B": ["C", "E"],

"C": ["A", "B"],

"D": ["A", "G"],

"E": ["B"],

"F": ["A"],

"G": ["D"]

|  |  |
| --- | --- |
| .-Dado el grafo dibujar las **listas** de **adyacencia**. | .-Dado el grafo dibujar las **listas** de **adyacencia**. |
|  |  |

1.-Una matriz \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ es aquella donde la mayoría de sus elementos son cero

|  |  |
| --- | --- |
| a) identidad b) dispersa c) triangular d) densa | ( ) |

2.-Se represento en clase mediante un arreglo de listas

|  |  |
| --- | --- |
| a) lista simplemente ligada b) árbol c) grafo d) pila | ( ) |

3,-Se represento en clase mediante un arreglo de listas

|  |  |
| --- | --- |
| a) cola b) árbol c) lista simplemente ligada d) tabla hash | ( ) |

4.-En un(a) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_el 1er elemento en entrar es el primero en salir

|  |  |
| --- | --- |
| a) cola b) árbol c) cola de prioridad d) tabla hash | ( ) |

5.-En una *c*ola de prioridad siempre se elimina el elemento de

|  |  |  |
| --- | --- | --- |
| a) el principio | b) el final | ( ) |
| c) el de mayor (menor) prioridad | d) ninguna de las anteriores |  |

6.-Un grafo G es \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ si existe un camino en G entre cualesquiera par de vértices de G .

|  |  |
| --- | --- |
| a) disconexo b) ciclico c) conexo d) dirigido | ( ) |

7.-Un grafo G es \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ si existe un vertice v tal que partiendo de dicho vértice v se puede volver a este mismo vertice v.

|  |  |
| --- | --- |
| a) disconexo b) ciclico c) conexo d) dirigido | ( ) |

8.-Son aquellos grafos en los que las aristas que conectan los nodos son de sentido único.

|  |  |
| --- | --- |
| a) disconexo b) ciclico c) conexo d) dirigido | ( ) |

9.-Una relación es \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ si ∀x ∈ A|(x, x) ∈ R

|  |  |
| --- | --- |
| a) de equivalencia b) simétrica c) reflexiva d) transitiva | ( ) |

10.-Una relación es \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ si (x, y) ∈ R → (y, x) ∈ R

|  |  |
| --- | --- |
| a) de equivalencia b) simétrica c) reflexiva d) transitiva | ( ) |

11.-Una relación es \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ si [(x, y) ∈ R ∧ (y, z) ∈ R]

|  |  |
| --- | --- |
| a) de equivalencia b) simétrica c) reflexiva d) transitiva | ( ) |

12,-Una relación es \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ si es simétrica, reflexiva y transitiva

|  |  |
| --- | --- |
| a) de equivalencia b) simétrica c) reflexiva d) transitiva | ( ) |

-Escriba una funcion que diga si una matriz dispersa es la matriz identidad.

-Escriba funciones para : crear, comparar igualdad, sumar , restar, multiplicar, e imprimir matrices dispersas

**Códigos:**

typedef struct Nodo {

    int fila, columna;

    int valor;

    struct Nodo\* siguiente;

} Nodo;

typedef struct {

    int filas, columnas;

    Nodo\* cabeza;

} MatrizDispersa;

MatrizDispersa\* crearMatriz(int *filas*, int *columnas*) {

    MatrizDispersa\* matriz = (MatrizDispersa\*)malloc(sizeof(MatrizDispersa));

    matriz->filas = *filas*;

    matriz->columnas = *columnas*;

    matriz->cabeza = NULL;

    return matriz;

}

void insertarElemento(MatrizDispersa\* *matriz*, int *fila*, int *columna*, int *valor*) {

    if (*valor* == 0) return;

    Nodo\* nuevo = (Nodo\*)malloc(sizeof(Nodo));

    nuevo->fila = *fila*;

    nuevo->columna = *columna*;

    nuevo->valor = *valor*;

    nuevo->siguiente = *matriz*->cabeza;

*matriz*->cabeza = nuevo;

}

void imprimirMatriz(MatrizDispersa\* *matriz*) {

    Nodo\* actual = *matriz*->cabeza;

    printf("Elementos no nulos:\n");

    while (actual != NULL) {

        printf("(*%d*, *%d*) = *%d*\n", actual->fila, actual->columna, actual->valor);

        actual = actual->siguiente;

    }

}

int sonIguales(MatrizDispersa\* *A*, MatrizDispersa\* *B*) {

    if (*A*->filas != *B*->filas || *A*->columnas != *B*->columnas) return 0;

    Nodo\* a = *A*->cabeza;

    Nodo\* b = *B*->cabeza;

    while (a != NULL && b != NULL) {

        if (a->fila != b->fila || a->columna != b->columna || a->valor != b->valor) {

            return 0;

        }

        a = a->siguiente;

        b = b->siguiente;

    }

    return (a == NULL && b == NULL);

}

MatrizDispersa\* sumar(MatrizDispersa\* *A*, MatrizDispersa\* *B*) {

    if (*A*->filas != *B*->filas || *A*->columnas != *B*->columnas) {

        printf("No se pueden sumar: dimensiones incompatibles.\n");

        return NULL;

    }

    MatrizDispersa\* resultado = crearMatriz(*A*->filas, *A*->columnas);

    Nodo\* a = *A*->cabeza;

    while (a != NULL) {

        insertarElemento(resultado, a->fila, a->columna, a->valor);

        a = a->siguiente;

    }

    Nodo\* b = *B*->cabeza;

    while (b != NULL) {

        Nodo\* r = resultado->cabeza;

        while (r != NULL) {

            if (r->fila == b->fila && r->columna == b->columna) {

                r->valor += b->valor;

                break;

            }

            r = r->siguiente;

        }

        if (r == NULL) {

            insertarElemento(resultado, b->fila, b->columna, b->valor);

        }

        b = b->siguiente;

    }

    return resultado;

}

MatrizDispersa\* restar(MatrizDispersa\* *A*, MatrizDispersa\* *B*) {

    if (*A*->filas != *B*->filas || *A*->columnas != *B*->columnas) {

        printf("No se pueden restar: dimensiones incompatibles.\n");

        return NULL;

    }

    MatrizDispersa\* resultado = crearMatriz(*A*->filas, *A*->columnas);

    Nodo\* a = *A*->cabeza;

    while (a != NULL) {

        insertarElemento(resultado, a->fila, a->columna, a->valor);

        a = a->siguiente;

    }

    Nodo\* b = *B*->cabeza;

    while (b != NULL) {

        Nodo\* r = resultado->cabeza;

        while (r != NULL) {

            if (r->fila == b->fila && r->columna == b->columna) {

                r->valor -= b->valor;

                break;

            }

            r = r->siguiente;

        }

        if (r == NULL) {

            insertarElemento(resultado, b->fila, b->columna, -b->valor);

        }

        b = b->siguiente;

    }

    return resultado;

}

MatrizDispersa\* multiplicar(MatrizDispersa\* *A*, MatrizDispersa\* *B*) {

    if (*A*->columnas != *B*->filas) {

        printf("No se pueden multiplicar: dimensiones incompatibles.\n");

        return NULL;

    }

    MatrizDispersa\* resultado = crearMatriz(*A*->filas, *B*->columnas);

    for (Nodo\* a = *A*->cabeza; a != NULL; a = a->siguiente) {

        for (Nodo\* b = *B*->cabeza; b != NULL; b = b->siguiente) {

            if (a->columna == b->fila) {

                int fila = a->fila;

                int columna = b->columna;

                int valor = a->valor \* b->valor;

                Nodo\* r = resultado->cabeza;

                while (r != NULL) {

                    if (r->fila == fila && r->columna == columna) {

                        r->valor += valor;

                        break;

                    }

                    r = r->siguiente;

                }

                if (r == NULL) {

                    insertarElemento(resultado, fila, columna, valor);

                }

            }

        }

    }

    return resultado;

}

int esIdentidad(MatrizDispersa\* *matriz*) {

    if (*matriz*->filas != *matriz*->columnas) return 0;

    for (int i = 0; i < *matriz*->filas; i++) {

        for (int j = 0; j < *matriz*->columnas; j++) {

            int encontrado = 0;

            Nodo\* actual = *matriz*->cabeza;

            while (actual != NULL) {

                if (actual->fila == i && actual->columna == j) {

                    if (i == j && actual->valor != 1) return 0;

                    if (i != j && actual->valor != 0) return 0;

                    encontrado = 1;

                    break;

                }

                actual = actual->siguiente;

            }

            if (!encontrado && i == j) return 0;

        }

    }

    return 1;

}