Graph

What is Graph in data structure?

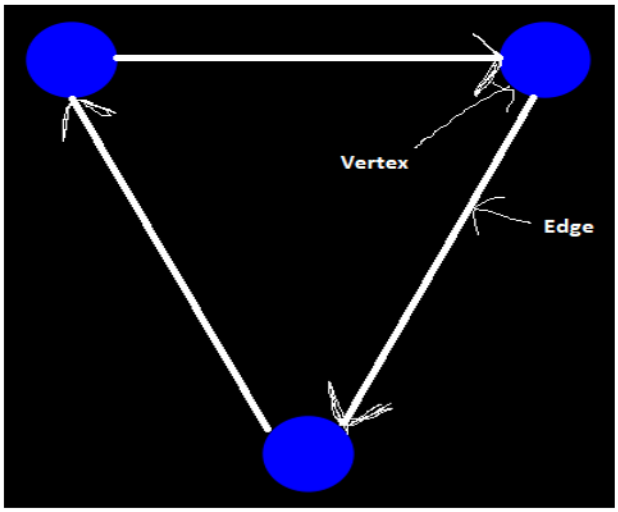
→ A graph is a representation of a collection of node that are linked together. Interconnected nodes are represented by points called vertices, and the connections connecting vertices are called edges.

Basic Terminology:

- Vertex- Each node is represented as a dot called Vertex.
- Edge- The path connecting two vertexes is called Edge.
- Adjacency- Two connected nodes/vertex are called adjacent.
- Path- The sequence of connections between different vertexes is called Path.

Can you think of real life applications of graph as a data structure?

It is widely used in path optimizing algorithm and even for solving very complicated electrical circuits.



- Adjacency Matrix- It is a way of representing a graph in form of a Boolean matrix. It there is a path between two node we represent it as 1 and if not then 0.

```c
#include<stdio.h>
#include<stdlib.h>

typedef struct Graph
{
    int V, E;
    int **adj;
} Graph;

Graph *adjMatrixOfGraph()
{
    int m, n, i;
    Graph *g = (Graph *) malloc(sizeof(Graph));
    if(!g)
    {
        printf("\nNo Memory allocated ");
        return NULL;
    }
    printf("Enter number of vertices and edges : ");
    scanf("%d %d", &g->V, &g->E);
    g->adj = (int**)malloc(sizeof(int *)*(g->V));
    for(i = 0; i < g->V;i++){
            g->adj[i] = (int*)malloc(sizeof(int)*(g->E));
    }
    for(m = 0; m <g->V; m++)
    {
        for(n = 0; n < g->E; n++)
        {
            g->adj[m][n] = 0;
        }
    }

    for(i = 0; i < g->E; i++)
    {
        printf("Enter node numbers in pair for an edges : ");
        scanf("%d %d", &m, &n);
        g->adj[m][n] = 1;
        g->adj[n][m] = 1;
    }
    return g;
}
void printGraph(Graph *g)
{
    int m, n;
    printf("\nAdjacency Matrix : \n");
    for(m = 0; m < g->V; m++)
    {
```

```
            for(n = 0; n < g->E; n++)
            {
                printf("%3d",g->adj[m][n]);
            }
            printf("\n");
        }
    }
    int main()
    {
        Graph * G = adjMatrixOfGraph();
        printGraph(G);
        return 0;
    }
```

- Adjacency List- It represents a graph in the form of linked list. The index number is the vertex.

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

struct node{
    int v;
    struct node* next;
};

struct graph{
    int nv;
    struct node** adjLists;
};

struct node* createNode(int v){
    struct node* nn = (struct node*)malloc(sizeof(struct node));
    nn->v = v;
    nn->next = NULL;
    return nn;
}

struct graph* createGraph(int v){
    struct graph* g = (struct graph*) malloc(sizeof(struct graph));
    g->nv = v;

    g->adjLists = malloc(v * sizeof(struct node*));

    int i;
    for(i = 0; i < v; i++){
        g->adjLists[i] = 0;
    }
    return g;
}
void addEdge(struct graph *g, int s, int d){
```

```
    struct node* n = createNode(d);
    n->next = g->adjLists[s];
    g->adjLists[s] = n;

    n = createNode(s);
    n->next = g->adjLists[d];
    g->adjLists[d]=n;
}
void printGraph(struct graph * g){
    int i;
    for(i = 0; i < g->nv; i++){
        struct node* t = g->adjLists[i];
        printf("\nVertex %d\n",i);
        while(t){
            printf("%d-> ", t->v);
            t = t->next;
        }
        printf("\n");
    }
}
int main()
{
    struct graph* g = createGraph(5);
    addEdge(g, 0, 1);
    addEdge(g, 0, 2);
    addEdge(g, 0, 3);
    addEdge(g, 1, 2);
    addEdge(g, 3, 4);
    addEdge(g, 4, 1);
    printGraph(g);
    return 0;
}
```

- Breadth First Search (BFS) - It follows a recursive algorithm to find all the vertex of a graph. We put each vertex in two categories; visited and not visited.

Algorithm:

- We start by putting any one vertex and put it in visited list.
- Then its adjacent vertices are pushed in a queue.
- Then the vertexes from the queue are moved to visited list as they are visited.
- The process is repeated until the queue is empty.

Implementation:

```c
// BFS algorithm in C

#include <stdio.h>
#include <stdlib.h>
#define SIZE 40

struct queue {
  int items[SIZE];
  int front;
  int rear;
};

struct queue* createQueue();
void enqueue(struct queue* q, int);
int dequeue(struct queue* q);
void display(struct queue* q);
int isEmpty(struct queue* q);
void printQueue(struct queue* q);

struct node {
  int vertex;
  struct node* next;
};

struct node* createNode(int);

struct Graph {
  int numVertices;
  struct node** adjLists;
  int* visited;
};

// BFS algorithm
void bfs(struct Graph* graph, int startVertex) {
  struct queue* q = createQueue();

  graph->visited[startVertex] = 1;
  enqueue(q, startVertex);

  while (!isEmpty(q)) {
    printQueue(q);
    int currentVertex = dequeue(q);
    printf("Visited %d\n", currentVertex);

    struct node* temp = graph->adjLists[currentVertex];

    while (temp) {
      int adjVertex = temp->vertex;

      if (graph->visited[adjVertex] == 0) {
```

```
        graph->visited[adjVertex] = 1;
        enqueue(q, adjVertex);
      }
      temp = temp->next;
    }
  }
}

// Creating a node
struct node* createNode(int v) {
  struct node* newNode = malloc(sizeof(struct node));
  newNode->vertex = v;
  newNode->next = NULL;
  return newNode;
}

// Creating a graph
struct Graph* createGraph(int vertices) {
  struct Graph* graph = malloc(sizeof(struct Graph));
  graph->numVertices = vertices;

  graph->adjLists = malloc(vertices * sizeof(struct node*));
  graph->visited = malloc(vertices * sizeof(int));

  int i;
  for (i = 0; i < vertices; i++) {
    graph->adjLists[i] = NULL;
    graph->visited[i] = 0;
  }

  return graph;
}

// Add edge
void addEdge(struct Graph* graph, int src, int dest) {
  // Add edge from src to dest
  struct node* newNode = createNode(dest);
  newNode->next = graph->adjLists[src];
  graph->adjLists[src] = newNode;

  // Add edge from dest to src
  newNode = createNode(src);
  newNode->next = graph->adjLists[dest];
  graph->adjLists[dest] = newNode;
}

// Create a queue
struct queue* createQueue() {
  struct queue* q = malloc(sizeof(struct queue));
  q->front = -1;
  q->rear = -1;
  return q;
}
```

```c
// Check if the queue is empty
int isEmpty(struct queue* q) {
  if (q->rear == -1)
    return 1;
  else
    return 0;
}

// Adding elements into queue
void enqueue(struct queue* q, int value) {
  if (q->rear == SIZE - 1)
    printf("\nQueue is Full!!");
  else {
    if (q->front == -1)
      q->front = 0;
    q->rear++;
    q->items[q->rear] = value;
  }
}

// Removing elements from queue
int dequeue(struct queue* q) {
  int item;
  if (isEmpty(q)) {
    printf("Queue is empty");
    item = -1;
  } else {
    item = q->items[q->front];
    q->front++;
    if (q->front > q->rear) {
      printf("Resetting queue ");
      q->front = q->rear = -1;
    }
  }
  return item;
}

// Print the queue
void printQueue(struct queue* q) {
  int i = q->front;

  if (isEmpty(q)) {
    printf("Queue is empty");
  } else {
    printf("\nQueue contains \n");
    for (i = q->front; i < q->rear + 1; i++) {
      printf("%d ", q->items[i]);
    }
  }
}

int main() {
```

```
  struct Graph* graph = createGraph(6);
  addEdge(graph, 0, 1);
  addEdge(graph, 0, 2);
  addEdge(graph, 1, 2);
  addEdge(graph, 1, 4);
  addEdge(graph, 1, 3);
  addEdge(graph, 2, 4);
  addEdge(graph, 3, 4);

  bfs(graph, 0);

  return 0;
}
```