 **Bharatiya Vidya Bhavan’s**

**SARDAR PATEL INSTITUTE OF TECHNOLOGY**

(Autonomous Institute Affiliated to University of Mumbai)

Munshi Nagar, Andheri (W), Mumbai – 400 058.

COMPS Department

|  |  |
| --- | --- |
| **Experiment** | 8 |
| **Aim** | **Implement Breadth Frst Search and Depth First Search Traversal for given Graph. Graph should be dynamic , means it should accept number of vertices and edges , dynamically. ( It should not be fixed in program )** |
| **Name** | David Daniels |
| **UID** | 2023300038 |
| **Class** | Div -A |
| **Batch** | C |
| **Date of Submission** | 17-10-24 |

|  |  |
| --- | --- |
| **Theory** |  |
| **Algorithm** |  |
| **Problem Solving** |  |
| **Program(Code)** | #include <stdio.h>  #include <stdlib.h>  #include <stdbool.h>  typedef struct graph  {      int no\_of\_edges;      int no\_of\_vertices;      //array of adjacency graphs      int \*\* arr;  }graph;  graph\* expand\_graph(graph \* g ,int vertices) {    int old\_vertices\_no= g->no\_of\_vertices;    g->no\_of\_vertices=vertices;      g->arr= realloc(g->arr,g->no\_of\_vertices \* sizeof(int\*));      for(int i = 0; i < g->no\_of\_vertices; i++)      {          g->arr[i] = realloc(g->arr[i],g->no\_of\_vertices \* sizeof(int));      }    //set new nodes to null    for (int i = old\_vertices\_no; i <vertices ; i++)    {      for (int j = 0; j < vertices; j++)      {          g->arr[i][j]=0;          g->arr[j][i]=0;      }    }    return g;  }  graph \* add\_edge (graph \* g ,int e1,int e2)  {      //printf("ADDING %d , %d \n",e1,e2);      g->no\_of\_edges++;      g->arr[e1][e2]=1;      g->arr[e2][e1]=1;      return g;  }  typedef struct queue{      int\* items;      int front;      int rear;      int size;  } queue;  bool isEmpty(queue\* q) { return (q->front == q->rear - 1); }  bool isFull(queue\* q) { return (q->rear == q->size); }  void enqueue(queue\* q, int value)  {      if (isFull(q)) {          printf("queue is full\n");          return;      }      q->items[q->rear] = value;      q->rear++;  }  int dequeue(queue\* q)  {      if (isEmpty(q)) {          printf("queue is empty\n");          return -1;      }      q->front++;      return q->items[q->front];  }  int \* init\_visited(graph  \* g)  {  int \* visited = (int \*) malloc(g->no\_of\_vertices \* sizeof(int));      for (int i = 0; i < g->no\_of\_vertices; i++)      {          visited[i]=0;      }  return visited;  }  void bfs\_with\_level(graph \* g,int start)  {      int \* visited = init\_visited(g);      queue \* q = (queue \* )malloc(sizeof(queue));      int size\_o\_q=10;      q->size=size\_o\_q;      q->items=(int \*)malloc(size\_o\_q\*sizeof(int));      q->front = -1;      q->rear = 0;      int level=0;      //i IS SAME AS STARTING VERTEX      int i=start;      printf("%d ,Level= %d\n",i,level);        enqueue(q,i);      printf("Enqueuing %d \n",i);      visited[i]=1;      while (!isEmpty(q))      {          int node = dequeue(q);          printf("Dequeuing %d \n",node);          level++;          for (int j = 0; j < g->no\_of\_vertices; j++)          {              //if edge is 1 and node not visited              if (g->arr[node][j]==1 && visited[j]==0)              {                  //printf("%d \n",j);                  printf("%d ,Level= %d\n",j,level);                  visited[j]=1;                  enqueue(q,j);                  printf("Enqueuing %d \n",j);              }            }          level++;          i++;        }      free(q->items);      free(q);  }  void bfs(graph \* g,int start)  {      int \* visited = init\_visited(g);      queue \* q = (queue \* )malloc(sizeof(queue));      int size\_o\_q=10;      q->size=size\_o\_q;      q->items=(int \*)malloc(size\_o\_q\*sizeof(int));      q->front = -1;      q->rear = 0;      //i IS SAME AS STARTING VERTEX      int i=start;      printf("%d ",i);      enqueue(q,i);      visited[i]=1;      while (!isEmpty(q))      {          int node = dequeue(q);            for (int j = 0; j < g->no\_of\_vertices; j++)          {              //if edge is 1 and node not visited              if (g->arr[node][j]==1 && visited[j]==0)              {                  printf("%d ",j);                  visited[j]=1;                  enqueue(q,j);              }            }          i++;          }      free(q->items);      free(q);  }  int time = 0;  int \*start\_time;  int \*end\_time;  void dfs\_with\_time(graph \*g, int start, int \*visited) {      if (start\_time == NULL) {          start\_time = (int \*)malloc(sizeof(int) \* g->no\_of\_vertices);          end\_time = (int \*)malloc(sizeof(int) \* g->no\_of\_vertices);          for (int i = 0; i < g->no\_of\_vertices; i++) {              start\_time[i] = 0;              end\_time[i] = 0;          }      }        time++;      visited[start] = 1;      start\_time[start] = time;      printf("Node %d: Entering Time = %d\n", start, start\_time[start]);      for (int j = 0; j < g->no\_of\_vertices; j++) {          if (g->arr[start][j] == 1 && !visited[j]) {              dfs\_with\_time(g, j, visited);          }      }      time++;      end\_time[start] = time;      printf("Node %d: Exiting Time = %d\n", start, end\_time[start]);  }  void show\_table(graph \*g) {      printf("\nNode\tStart Time\tEnd Time\n");      for (int i = 0; i < g->no\_of\_vertices; i++) {          printf("%d\t%d\t\t%d\n", i, start\_time[i], end\_time[i]);      }  }  void dfs(graph \*g, int start, int \* visited) {      int i=start;      printf("%d ",i);      visited[i]=1;      for (int j = 0; j < g->no\_of\_vertices; j++)      {          if (g->arr[i][j]==1 && !visited[j])          {              dfs(g,j,visited);          }        }    }  void display\_graph(graph \* g)  {      for (int i = 0; i < g->no\_of\_vertices; i++)      {          for (int j = 0; j < g->no\_of\_vertices; j++)          {              printf("%d ",g->arr[i][j]);          }          printf("\n");      }    }  void choice(graph \* g)  {      int choice=0,num=0;      printf(" Enter 1 To add vertex \n Enter 2 To add edge \n Enter 3 for BFS \n Enter 4 for DFS \n Enter 5 to print Matrix \n Enter 0 to Exit \n");      while (true)      {        printf("Enter choice \n");          scanf("%d",&choice);          if (choice==0)          {              printf("\nGoodbye\n");              break;          }      switch (choice)      {      case 1:          {              printf("Enter new number of vertices: \n");              scanf("%d", &num);              expand\_graph(g,num);              printf("\n\n");              break;          }      case 2:          {              printf("Enter Edge to Add: \n");              scanf("%d", &num);              int num1=num;              scanf("%d", &num);              int num2=num;              add\_edge(g,num1,num2);              printf("\n\n");              break;          }      case 3:          {              printf("Enter Start Vertice: \n");              scanf("%d", &num);              printf("BFS is: \n");              bfs(g,num);              bfs\_with\_level(g,num);              printf("\n\n");              break;          }      case 4:          {              printf("Enter Start Vertice: \n");              scanf("%d", &num);              int \* visited=init\_visited(g);              printf("DFS is: \n");              dfs\_with\_time(g,num,visited);              show\_table(g);              printf("\n\n\n");              visited=init\_visited(g);              dfs(g,num,visited);              printf("\n\n\n");              break;          }      case 5:          {              printf("Graph Matrix is: \n");              display\_graph(g);              break;          }      default:          {              printf("Error");              break;          }      }  }  }  int main(int argc, char const \*argv[])  {      graph \* g = (graph \*) malloc(sizeof(graph));      printf("Enter Number Initial Vertices: ");      int num=0;      scanf("%d",&num);      g->no\_of\_vertices=num;      g->no\_of\_edges=0;      // make 2d array      g->arr= malloc(g->no\_of\_vertices \* sizeof(int\*));      for(int i = 0; i < g->no\_of\_vertices; i++)      {          g->arr[i] = malloc(g->no\_of\_vertices \* sizeof(int));      }        for (int i = 0; i < g->no\_of\_vertices; i++)      {          for (int j = 0; j < g->no\_of\_vertices; j++)          {              g->arr[i][j]=0;          }      }      choice(g);      for(int i = 0; i < g->no\_of\_vertices; i++)      {      free(g->arr[i]);      }      free(g->arr);      free(g);        return 0;  } |
| **Output** |  |
| **Conclusion** | Thus, we implemented dynamic Breadth-First Search (BFS) and Depth-First Search (DFS) traversals for a graph that allows the user to specify the number of vertices and edges at runtime. This flexibility enhances our understanding of graph structures and algorithms, showcasing their adaptability to various scenarios. By experimenting with different graph configurations, we observed how BFS and DFS yield distinct traversal paths, highlighting their unique characteristics and applications in graph theory. |