Graph

Using map and list

type GraphNode struct {

Vertex map[int][]int

}

func (g \*GraphNode) Insert(vertex, edge int, isBidir bool) {

g.Vertex[vertex] = append(g.Vertex[vertex], edge)

if isBidir {

g.Vertex[edge] = append(g.Vertex[edge], vertex)

}

}

func (g \*GraphNode) Display() {

for vertex, edges := range g.Vertex {

fmt.Println(vertex, edges)

}

}

func (g \*GraphNode) RemoveEdge(v1, v2 int, isBi bool) {

arr := g.Vertex[v1]

for i, v := range arr {

if v == v2 {

g.Vertex[v1] = append(arr[:i], arr[i+1:]...)

break

}

}

if len(g.Vertex[v1]) == 0 {

delete(g.Vertex, v1)

}

if isBi {

arr := g.Vertex[v2]

for i, v := range arr {

if v == v1 {

g.Vertex[v2] = append(arr[:i], arr[i+1:]...)

break

}

}

if len(g.Vertex[v2]) == 0 {

delete(g.Vertex, v2)

}

}

}

func (g \*GraphNode) RemoveVertex(v int) {

delete(g.Vertex, v)

for key, arr := range g.Vertex {

for i, e := range arr {

if e == v {

g.Vertex[key] = append(arr[:i], arr[i+1:]...)

}

}

}

}

func (g \*GraphNode) BFS(value int) {

visited := make(map[int]bool)

visited[value] = true

arr := []int{value}

for len(arr) > 0 {

val := arr[0]

fmt.Println(val)

arr = arr[1:]

for \_, v := range g.Vertex[val] {

if !visited[v] {

visited[v] = true

arr = append(arr, v)

}

}

}

}

func (g \*GraphNode) DFS(value int) {

visited := make(map[int]bool)

visited[value] = true

stack := []int{value}

g.dfsHelper(value, stack, visited)

}

func (g \*GraphNode) dfsHelper(value int, stack []int, visited map[int]bool) {

if len(stack) == 0 {

return

}

val := stack[len(stack)-1]

fmt.Println(val)

stack = stack[:len(stack)-1]

for \_,v := range g.Vertex[val] {

if !visited[v] {

visited[v] = true

stack = append(stack, v)

g.dfsHelper(v,stack,visited)

}

}

}

func main() {

g := GraphNode{Vertex: make(map[int][]int)}

g.Insert(1, 2, false)

g.Insert(2, 3, false)

g.Insert(3, 4, true)

g.Insert(2, 5, false)

g.Insert(1, 5, false)

g.Insert(6, 7, false)

g.Insert(6, 2, false)

g.Insert(3, 2, false)

g.Insert(4, 5, true)

g.RemoveEdge(4, 5, true)

g.RemoveVertex(3)

g.Display()

g.DFS(2)

g.BFS(2)

}

Using 2D Array

package main

import "fmt"

type Graph struct {

vertices int

matrix [][]int

}

func (g \*Graph) NewGraph(vertices int) {

g.vertices = vertices

g.matrix = make([][]int, vertices)

for i := range g.matrix {

g.matrix[i] = make([]int, vertices)

}

}

func (g \*Graph) AddEdge(v1, v2 int) {

g.matrix[v1][v2] = 1

g.matrix[v2][v1] = 1

}

func (g \*Graph) Display() {

for i := range g.matrix {

fmt.Println(g.matrix[i])

}

}

func (g \*Graph) AddVertex() {

g.vertices++

for i := range g.matrix{

g.matrix[i] = append(g.matrix[i], 0)

}

g.matrix =append(g.matrix, make([]int, g.vertices))

}

func main() {

g := &Graph{}

g.NewGraph(4)

g.AddEdge(1,2)

g.AddEdge(0,2)

g.AddVertex()

g.Display()

}