山东大学 软件学院 学院

软件测试 课程实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| 学号：201800301202 | 姓名：李成 | | 班级： 2018级软件4班 |
| 实验题目：图的操作 | | | |
| 实验学时：4 | | 实验日期2019-12-03 | |
| 实验目的:  掌握无向图的创建、遍历方法。 | | | |
| 硬件环境：  1、笔记本PC | | | |
| 软件环境：   1. Windows10 2. Eclipse 3. Java | | | |
| 实验步骤与内容：  实验内容：  1、创建图类，存储结构使用邻接矩阵。  2、输入图的节点数 n（不超过 10 个）、边数 m，节点分别用 1-n 代表。  3、采用“起始节点，终止节点，权值”输入图的 m 条边，创建图。  4、输出从节点 1 开始的 BFS 遍历，在遍历过程中，如有多个可以选择  的节点，则优先选择编号较小的节点。  5、输出从节点 1 开始的 DFS 遍历，在遍历过程中，如有多个可以选择  的节点，则优先选择编号较小的节点。  6、输出从第 1 节点到第 n 节点最短路径的长度，如果没有路经，输出 0。  实验步骤：   1. 需要用到队列来实现广度优先。   class queue{    private:    int begin;  int end;  int capacity;  int \*num;    public:    queue(int n){    num = new int[n];    capacity = n;    begin = 0;    end = 0;    }    void push(int n){    end = (end+1)%capacity;    num[end] = n;    }  int read(){    return num[(begin+1)%capacity];    }    int pop(){    begin = (begin+1)%capacity;    }    bool empty(){    return begin==end;  }      };   1. 创建一个图的类。其中比较重要的是广度优先和深度优先的算法设计。   class graph{    int \*\*matrix;  int number = 0;  int edge=0;  int n1 = 1;    public:    graph(int n ,int m , int \*\*d){    number = n;    edge = m ;    matrix = new int \*[number+1];    for(int i = 0 ; i < number+1 ; i++){    matrix[i] = new int[number+1];    }    for(int i = 0 ; i < n+1 ; i++){    for(int j = 0 ; j < n+1 ; j++){    matrix[i][j] = 999;    }  }      for(int i = 1 ; i < m+1 ; i++){    matrix[d[i-1][0]][d[i-1][1]]=d[i-1][2];    matrix[d[i-1][1]][d[i-1][0]] = d[i-1][2];    }    }    void bfs(int label){  int n=0 ;  int reach[number+1];  queue q(10);  int result[number];    for(int i = 0 ; i <= number ; i++){    reach[i] = 0;    }  for(int i = 0 ; i < number ; i++){    result[i] = 0;    }    reach[1] = label;    q.push(1);  while(!q.empty()){    int w = q.read();    for(int i = 0 ; i < number; i++){    if(result[i]==w){    break;    }    if(result[i]==0){    result[i] = w;    break;    }  }    q.pop();    for(int i = 1 ; i <= number ; i++){    if(matrix[w][i]!=999&&reach[i]==0){    reach[i] = label;  q.push(i);    }  }  }    cout<<result[0];    for(int i = 1; i < number ; i++){    if(result[i]!=0){    cout<<","<<result[i];  }  }  }  void rDfs(int v,int label, int \*reach2,int \*result2){    reach2[v] = label;    for(int i = 1 ; i <= number ;i++){    if(matrix[v][i]!=999&&reach2[i]!=label){    result2[n1] = i;  n1++;  rDfs(i,label,reach2,result2);    }  }  }  void dfs(int label){    int reach1[number+1];  int result1[number];    for(int i = 0 ; i <= number ; i++){    reach1[i] = 0;    }  for(int i = 0 ; i < number ; i++){    result1[i] = 0;    }    rDfs(1,label,reach1,result1);    result1[0] = 1;    cout<<endl;    cout<<result1[0];    for(int i = 1 ; i < number ; i++){    if(result1[i]!=0)  cout<<","<<result1[i];  }  }  bool empty(int \*a){    for(int i = 0 ; i <= number ; i++){    if(a[i]!=0)    return false;    }    return true;    }    int shortest(){    int distance[number+1];  int predecessor[number+1];  int reachable[number+1];  int n = 0;  int v;    for(int i = 0 ; i <= number ; i++){    reachable[i] = 0;    }    for(int i = 1 ; i <= number ;i++){    distance[i]=matrix[1][i];  if(distance[i]==999){    predecessor[i] = -1;  }    else{    predecessor[i] = 1;  reachable[n] = i;  n++;  }  }    distance[1] = 0;  predecessor[1] = 0;    while(!empty(reachable)){    int min;  int j = 0;      for(int i = 0 ; i <= number ; i++){    if(reachable[i]!=0){    min = distance[reachable[i]];  v = reachable[i];  j = i;  break;  }  }    for(int i = 0 ; i <= number ; i++){    if(reachable[i]!=0&&min>distance[reachable[i]]){    min = distance[reachable[i]];    v = reachable[i];  j = i;    }  }    reachable[j] = 0;    for(int i = 1 ; i <= number ; i++){    if(matrix[v][i]!=999&&(predecessor[i] == -1 || distance[i] > distance[v]+matrix[v][i])){    distance[i] = distance[v]+matrix[v][i];    if(predecessor[i]==-1){    for(int w = 0 ; w <= number ;w++){    if(reachable[w]==0){    reachable[w] = i;    break;  }    predecessor[i] = v;    }  }  }  }  }  if(distance[number]==999)  return 0;  return distance[number];    }  };   1. 有一个getNum方法用来获取输入数据中的数字。   void getNum(char\* str, int\* num){    int t;  int len = strlen(str);  int index=0;    for(int i = 0 ; i < len ; i++){    while(!(str[i]>='0'&&str[i]<='9')||str[i]==NULL){    i++;    }    while(str[i]>='0'&&str[i]<='9'){    t = str[i]-'0';    num[index] = num[index]\*10+t;    i++;  }    index++;    }  }   1. main()用来输入数据和规范结果。   int main(){    int n;  int m;    string str;    cout<<"Input"<<endl;    cin>>str;    char dotEdge[str.length()];    strcpy(dotEdge,str.c\_str());    int dotEdge1[2];    getNum(dotEdge,dotEdge1);    n = dotEdge1[0];  m = dotEdge1[1];  int \*\*data;  char \*\*data2;  string str2;    data = new int \*[m];  data2 = new char \*[m];    for(int i = 0 ; i < m ;i++){    data[i] = new int[3];    }    for(int i = 0 ; i < m ; i++){    for(int j = 0 ; j < 3 ; j++){    data[i][j]=0;    }  }    for(int i = 0 ; i < m ; i++){    cin>>str2;  data2[i] = new char[str2.length()];  strcpy(data2[i],str2.c\_str());    getNum(data2[i],data[i]);    }    graph G(n,m,data);    cout<<"Output"<<endl;    G.bfs(999);  G.dfs(999);  cout<<endl;  cout<<G.shortest()<<endl;    cout<<"End";    return 0;    } | | | |
| 结论分析与体会：  这次实验的关键在于广度优先和深度优先的算法以及最短路问题的解决，通过这次实验，让我深入地体会到了两种不同遍历方式的过程，还有就是迪杰斯特拉算法的具体内容。 | | | |