第8次作业

## 第6章习题

6.6已知有向图G用邻接矩阵存储，设计算法以分别求解顶点vi的入度、出度和度。

6.7已知图G用邻接矩阵存储，设计算法以分别实现函数firstadj（G，v）和nextadj(G,v,w)。

6.8设图G用邻接矩阵A[n+1,n+1]表示，设计算法以判断G是否是无向图。

6.9已知图G用邻接表存储，设计算法输出其所有边或弧。（假设各表头指针在数组A[n+1]中）

6.10对下列图，分别执行dfs(1)和dfs(5)，写出遍历序列，并构造出相应的dfs生成树。

图6-1 题6.10图

**1**

**2**

**4**

**5**

**3**

**7**

**6**

**8**

**3**

**1**

**2**

**5**

**4**

**6**

**7**

(a)

(b)

6.11对下图的邻接表，不用还原出原图，请执行dfs(1)，写出遍历序列，并构造出相应的dfs生成树。

图6-27 例6-6的邻接表图

info ptr

4

1

2

3

4

5

6

7

8

**∧**

3

2

**∧**

5

4

3

**∧**

7

**∧**

4

**∧**

8

**∧**

5

5

**∧**

8

4

**∧**

6

6.6

#include <iostream>

#include <vector>

using namespace std;

vector<int> get\_degree(vector<vector<int>>& adj\_matrix, int i) {

int out\_degree = 0, in\_degree = 0;

for (int j = 0; j < adj\_matrix.size(); j++) {

if (adj\_matrix[i][j] != 0) // 统计出度

out\_degree++;

if (adj\_matrix[j][i] != 0) // 统计入度

in\_degree++;

}

int degree = out\_degree + in\_degree;

return { in\_degree, out\_degree, degree };

}

int main() {

// 测试代码

vector<vector<int>> adj\_matrix = { {0, 1, 1, 0, 0},

{0, 0, 0, 1, 1},

{0, 0, 0, 1, 0},

{0, 0, 0, 0, 1},

{0, 0, 0, 0, 0} };

vector<int> degree = get\_degree(adj\_matrix, 2);

cout << "Vertex 2: In-Degree = " << degree[0] << ", Out-Degree = " << degree[1] << ", Degree = " << degree[2] << endl;

return 0;

}

6.7

#include <iostream>

#include <vector>

using namespace std;

int firstadj(vector<vector<int>>& adj\_matrix, int v) {

for (int i = 0; i < adj\_matrix.size(); i++) {

if (adj\_matrix[v][i] > 0) {

return i;

}

}

return -1; // 顶点v没有邻接顶点

}

int nextadj(vector<vector<int>>& adj\_matrix, int v, int w) {

for (int i = w + 1; i < adj\_matrix.size(); i++) {

if (adj\_matrix[v][i] > 0) {

return i;

}

}

return -1; // 顶点v在邻接表中没有下一个邻接顶点

}

int main() {

// 测试代码

vector<vector<int>> adj\_matrix = { {0, 1, 1, 0, 0},

{0, 0, 0, 1, 1},

{0, 0, 0, 1, 0},

{0, 0, 0, 0, 1},

{0, 0, 0, 0, 0} };

cout << "First Adjacent Vertex of Vertex 0: " << firstadj(adj\_matrix, 0) << endl;

cout << "Next Adjacent Vertex of Vertex 1 after Vertex 3: " << nextadj(adj\_matrix, 1, 3) << endl;

return 0;

}

6.8

#include <iostream>

#include <vector>

using namespace std;

bool is\_undirected(vector<vector<int>>& adj\_matrix) {

for (int i = 0; i < adj\_matrix.size(); i++) {

for (int j = i + 1; j < adj\_matrix.size(); j++) {

if (adj\_matrix[i][j] != adj\_matrix[j][i]) { // 判断是否对称

return false;

}

}

}

return true;

}

int main() {

// 测试代码

vector<vector<int>> adj\_matrix = { {0, 1, 1, 0, 0},

{1, 0, 0, 1, 1},

{1, 0, 0, 1, 0},

{0, 1, 1, 0, 1},

{0, 1, 0, 1, 0} };

if (is\_undirected(adj\_matrix)) {

cout << "The Graph is Undirected." << endl;

}

else {

cout << "The Graph is Directed." << endl;

}

return 0;

}

6.9

#include <iostream>

#include <vector>

using namespace std;

void print\_edges(vector<vector<int>>& adj\_list) {

for (int i = 0; i < adj\_list.size(); i++) {

for (int j = 0; j < adj\_list[i].size(); j++) {

if (i < adj\_list[i][j]) { // 避免重复输出有向图中的边或弧

cout << i << " -> " << adj\_list[i][j] << endl;

}

}

}

}

int main() {

// 测试代码

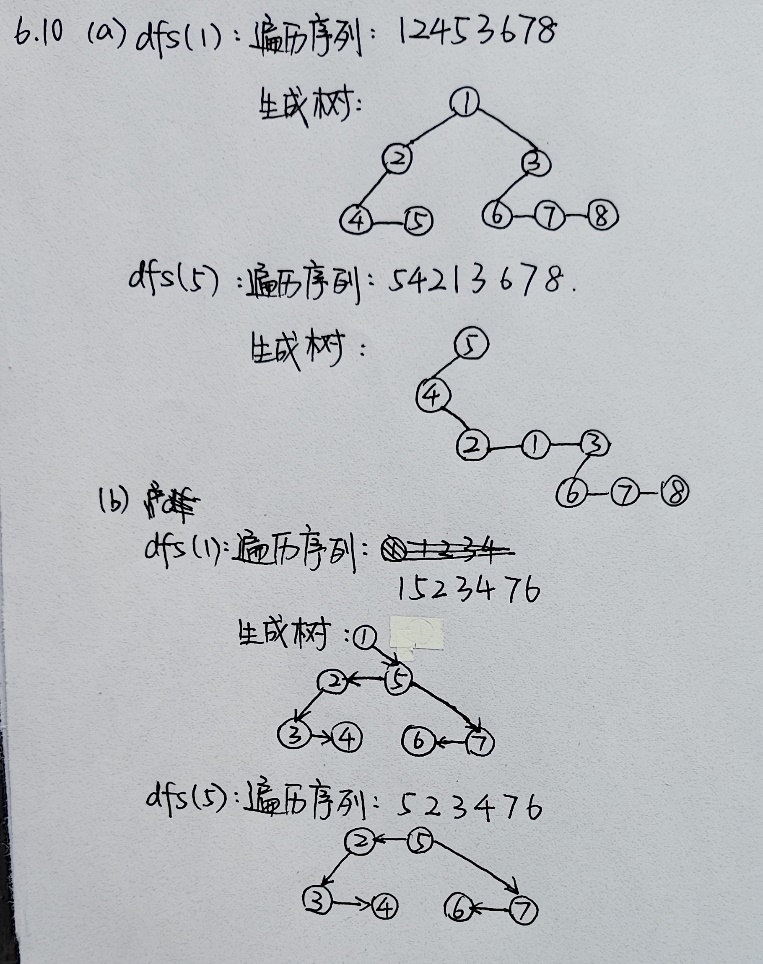
vector<vector<int>> adj\_list = { {1, 2}, {2, 3, 4}, {3}, {4}, {} };

print\_edges(adj\_list);

return 0;

}

6.10



6.11

