**中 国 矿 业 大 学**

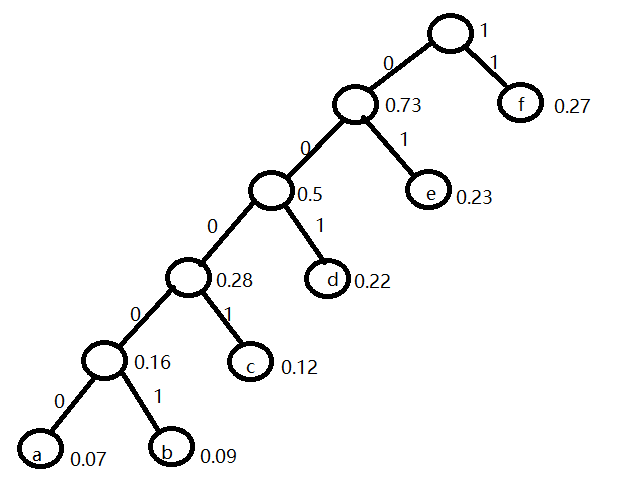
**2018 级《数据结构与算法分析》课程作业**

学生姓名 王茂凯

学 号 04181425

**中国矿业大学信控学院**

1. 假设字符a、b、c、d、e、f出现的概率分别为0.07、0.09、0.12、0.22、0.23、0.27，求最优Huffman编码，并画出Huffman树，试问编码的平均长度是多少？



a:00000

b:00001

c:0001

d:001

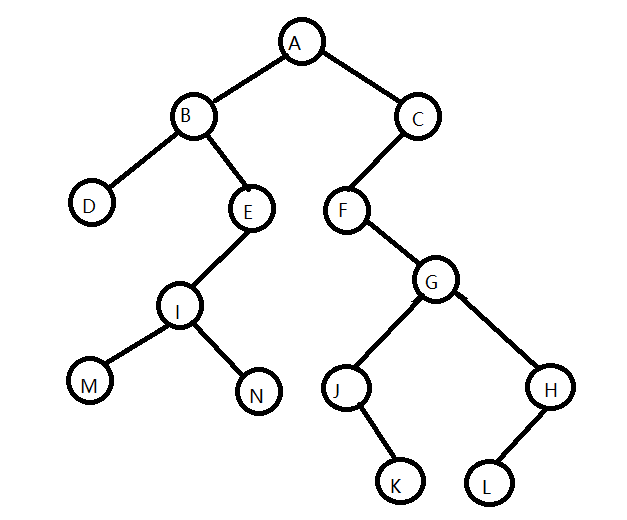
e:01

f:1

平均长度:20/6

2. 试将下图中所示的树转换为相应的二叉树。





3. 下图所示的有向图是强连通的吗？请列出所有简单路径，给出每个顶点的入度和出度，并给出其邻接矩阵、邻接表和逆邻接表。



是强连通图

简单路径:

2 3 1 4

4 3 1 2

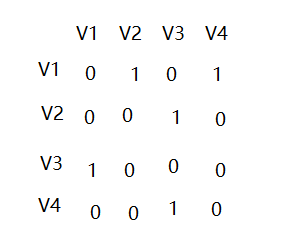
V1入度为1,出度为2

V2入度为1,出度为1

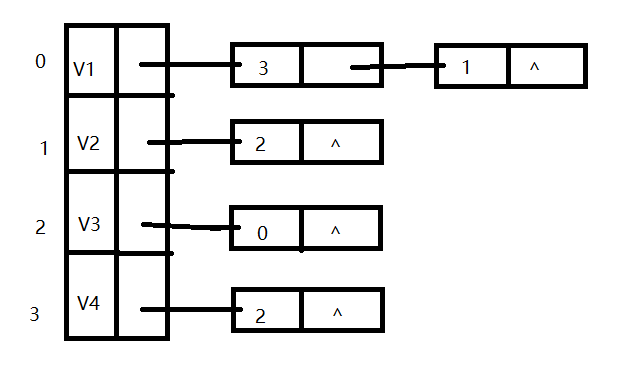
V3入度为2,出度为1

V4入度为1,出度为1

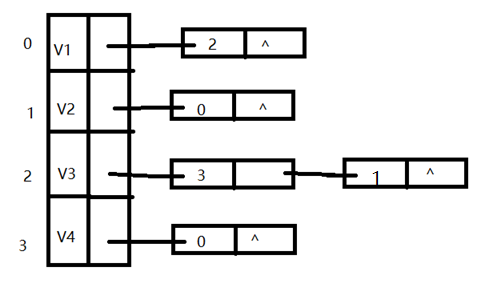
邻接矩阵:



邻接表:

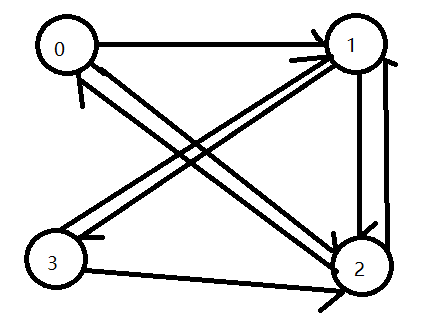


逆邻接表:



4. 已知一个有向图的邻接表，试编写一个算法，计算各顶点的入度（请附上所基于的有向图）。

已知邻接表求入度,在输入邻接表时统计每个顶点被输入的个数



#include <iostream>

using namespace std;

/\*

0--first-->1--next-->2--next-->nullptr

1--first-->2--next-->3--next-->nullptr

2--first-->1--next-->0--next-->nullptr

3--first-->2--next-->1--next-->nullptr

\*/

class graphtest

{

public:

    graphtest(int n) : \_n(n), graphhead(new head[\_n]()) //构造函数

    {

    }

    ~graphtest() //析构函数

    {

        for (int i = 0; i < \_n; ++i)

        {

            listnode\* temp = graphhead[i].first;

            while (temp != nullptr) //链表删除方式进行delete

            {

                temp = graphhead[i].first->next;

                delete graphhead[i].first;

                graphhead[i].first = temp;

            }

        }

        delete[] graphhead; //delete数组

    }

    void insertuv(const int& u, const int& v) //边的插入

    {

        listnode\* temp = new listnode(); //链表头结点插入方式

        temp->next = graphhead[u].first;

        temp->to = v;

        graphhead[u].first = temp;

    }

    void printgraph() //打印邻接表

    {

        for (int i = 0; i < \_n; ++i)

        {

            listnode\* temp = graphhead[i].first;

            cout << i << "--first-->";

            while (temp != nullptr)

            {

                cout << temp->to << "--next--> ";

                temp = temp->next;

            }

            cout << "nullptr" << endl;

        }

    }

    void solution(); //解题函数

private:

    struct listnode //邻接点结构体

    {

        int to;

        listnode\* next;

    };

    struct head //顶点数组结构体

    {

        //int data;

        listnode\* first;

    };

    int \_n;          //顶点数

    head\* graphhead; //顶点数组

};

int main()

{

    int n = 0;

    int e = 0;

    cout << "input n and e" << endl; //n为顶点数,e为边数

    cin >> n >> e;

    graphtest test(n);

    int u = 0;

    int v = 0;

    for (int i = 0; i < e; ++i)

    {

        cout << "input u v" << endl; //u-->v

        cin >> u >> v;

        test.insertuv(u, v);

    }

    test.printgraph();

    test.solution();

    return 0;

}

void graphtest::solution()

{

    int\* arry = new int[\_n]();              //存放入度的数组

    for (int i = 0; i < \_n; ++i) //遍历邻接表

    {

        listnode\* temp = (listnode\*)graphhead[i].first;

        while (temp != nullptr)

        {

            int t = temp->to;

            arry[t]++; //t顶点入度加1

            temp = temp->next;

        }

    }

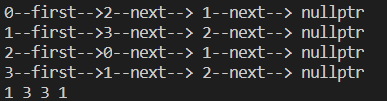
    for (int i = 0; i < \_n; ++i) //打印结果

        cout << arry[i] << " ";

    cout << endl;

    delete[]arry;

}



即

0顶点入度为1

1顶点入度为3

2顶点入度为3

3顶点入度为1