北京郵電大學

数据结构实验报告

题目：随机生成图和传递闭包平均出度求解

姓　　名：魏生辉

学　　院：计算机学院（国家示范性软件学院）

专　　业：计算机类

班　　级：2023211307

学　　号：2023211075

指导教师：杨震

2024年 12月

目录

随机生成图和平均出度求解

1 需求分析

1.1 题目描述

本实验要求通过C语言编写一个程序，模拟并操作一个有向图。图的表示方式使用邻接矩阵，其中每个顶点的出度和可达性通过随机生成的边来建立。程序需要完成以下任务：

随机生成一个有向图，输入图的顶点数和边数。

生成后的图通过邻接矩阵表示，并输出该矩阵。

统计并计算图中每个顶点的可达性以及图的平均出度。

1.2 输入描述

输入包含两个整数 n 和 e，分别表示图中的顶点数和边数。

n 表示图的顶点数，要求 n > 0。

e 表示图的边数，要求 0 <= e <= n * (n - 1)，即边数不能超过顶点数之间可能形成的最大边数。

1.3 输出描述

输出包含图的邻接矩阵表示。

输出统计结果，包括每个顶点的出度以及图的平均出度。

1.4 样例输入输出

1.4.1 样例输入输出 1

```
请输入图的顶点数（n）：5
请输入图的边数（e）：12

----- 生成图的参数 -----
顶点数：5
边数：12
------------------------

----- 随机生成的图的邻接矩阵 -----
0 1 1 0 1
0 0 1 0 0
1 0 0 1 1
1 0 1 0 1
1 0 1 0 0

----- 统计结果 -----
Average outdegree: 4.00
```

1.4.2 样例输入输出 2

```
请输入图的顶点数（n）：5
请输入图的边数（e）：1

----- 生成图的参数 -----
顶点数: 5
边数: 1
------------------------

----- 随机生成的图的邻接矩阵 -----
0 0 0 0 0
0 0 0 0 0
0 1 0 0 0
0 0 0 0 0
0 0 0 0 0
```

1.4.3 样例输入输出 3

```
请输入图的顶点数（n）：2
请输入图的边数（e）：6
错误：顶点数必须大于0，边数必须在0到顶点数*(顶点数-1)之间。
```

1.4.4 样例输入输出 4

```
----- 生成图的参数 -----
顶点数: 20
边数: 51
------------------------

----- 随机生成的图的邻接矩阵 -----
0 0 0 0 1 1 0 1 0 0 0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 1 1 0 0 0 0 1 0 0 0 0
0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 1 0 1 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 1 0 1 1 0 0 0 0 0 0 0 1 0 0 0 0 0
0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0
0 0 1 0 1 0 0 0 0 0 0 0 0 0 1 1 0 0 0 1
0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 1
0 1 0 1 0 0 1 0 0 1 1 0 1 0 0 0 0 1 0 0

----- 统计结果 -----
Average outdegree: 17.15
```

1.4.4 样例输入输出 5

```
请输入图的顶点数（n）：22
请输入图的边数（e）：9999
错误：顶点数必须大于0，边数必须在0到顶点数*(顶点数-1)之间。
```

1.5 程序功能

该程序的主要功能是模拟一个有向图的生成、处理与分析，具体功能包括：

图的初始化：

用户输入顶点数 n 和边数 e，程序根据这些参数初始化一个有向图，使用邻接矩阵表示图的结构。初始化时，矩阵中的所有元素都设为0，表示没有边连接各顶点。

随机生成图：

程序通过随机方式生成图的边，确保生成的边数与用户输入的边数 e 相符。生成的边不会重复，并且不包括自环（即一个顶点到自身的边）。生成的边按照邻接矩阵表示图中的连接关系，1表示有边连接，0表示没有边连接。

打印邻接矩阵：

生成图后，程序将输出邻接矩阵，展示图的连接情况。每个顶点与其他顶点的连接关系通过矩阵中的1和0进行表示。

计算图的传递闭包：

程序使用 Floyd-Warshall算法 计算图的传递闭包。传递闭包用于判断任意两个顶点之间是否存在可达路径。如果存在一条路径从顶点 i 到顶点 j，则在传递闭包中 edges[i][j] 将被设置为1，表示从顶点 i 到顶点 j 可达。

统计图的出度和计算平均出度：

程序会统计每个顶点的出度，即从该顶点出发的边的数量。同时，程序会计算图中所有顶点的平均出度，即所有顶点出度的总和除以顶点数 n。

输出结果：

最后，程序输出以下信息：

随机生成的图的邻接矩阵。

传递闭包计算结果（图中任意顶点的可达关系）。

图的平均出度值。

2 概要设计

2.1 问题解决思路概述

图的表示和初始化：

首先，我们使用邻接矩阵来表示图。邻接矩阵是一种适合表示稠密图的结构，通过二维数组 edges[i][j] 表示从顶点 i 到顶点 j 是否有边。在图初始化时，所有的矩阵元素都设置为0，表示默认情况下没有边。

随机生成边：

根据输入的顶点数 n 和边数 e，程序会随机生成 e 条边。为确保图的边没有重复，程序使用一个辅助数组 S 来标记已经生成的边。通过随机生成不重复的边，程序最终得到符合要求的有向图。

传递闭包的计算：

在有向图中，传递闭包表示图中所有顶点间的可达性。传递闭包的计算通过Floyd-Warshall算法实现。该算法的基本思想是：对于每一对顶点 i 和 j，如果存在一条路径从 i 到 j，则更新 edges[i][j] 为1，表示顶点 i 可以通过某些中间顶点到达顶点 j。Floyd-Warshall算法的时间复杂度是O(n^3)，适用于本实验中的图规模。

出度的统计与平均出度计算：

统计图中每个顶点的出度即该顶点的出边数。通过遍历邻接矩阵的每一行，可以统计每个顶点的出度。然后，通过将所有顶点的出度加总并除以顶点数 n，得到图的平均出度。

结果输出：

在程序运行完成后，首先输出随机生成的图的邻接矩阵，以便用户查看图的连接情况。接着，输出图的传递闭包，即计算得到的可达关系矩阵。最后，统计并输出图的平均出度，以便了解图中顶点的连接密度。

2.2 数据结构类型定义

```
typedef struct {
    int n; // 顶点个数
    int e; // 边的个数
    int edges[MAX_V][MAX_V]; // 邻接矩阵，表示图中顶点间的边
} Mgraph;
```

Mgraph 结构体成员说明：

n：表示图中顶点的个数（即图的维度）。这是一个整数值，用户输入的顶点数。

e：表示图中的边的数量。该值由用户输入，表示图中实际存在的边的数目。

edges[MAX_V][MAX_V]：一个二维数组，表示图的邻接矩阵。数组的大小是 MAX_V × MAX_V，其中 MAX_V 是预定义的最大顶点数（假设最大值为100）。邻接矩阵用于表示图中各顶点之间的边连接情况，edges[i][j] 为1表示从顶点 i 到顶点 j 有边，0表示没有边。

2.3 主程序的流程

程序的主要流程包括以下几个步骤：

用户输入：

程序首先提示用户输入图的顶点数 n 和边数 e。

用户输入后，程序检查输入是否有效：顶点数 n 必须大于0。

边数 e 必须满足 0 <= e <= n * (n - 1)，即边数不能超过最大可能的边数。

如果输入无效，程序会输出错误信息并终止执行。

图的初始化：

程序根据用户输入的顶点数 n 和边数 e 初始化图的结构体 Mgraph。

使用邻接矩阵 edges 来存储图的连接关系，矩阵中的所有元素初始化为0，表示图中没有边。

随机生成边：

程序通过调用 RandomCreateGraph() 函数来随机生成图的边。

RandomCreateGraph() 函数首先通过动态数组 S 记录哪些边已经生成过，确保没有重复生成边。

随机生成边的数量为用户输入的边数 e，并且生成的边不会包含自环（即不允许生成 edges[i][i] 为1的情况）。

打印邻接矩阵：

生成图后，程序通过 PrintGraph() 函数输出图的邻接矩阵，显示图中各顶点之间的连接情况。

计算传递闭包：

程序调用 Floyd_Warshall() 函数，使用Floyd-Warshall算法计算图的传递闭包。

该算法的核心思想是，针对每对顶点 (i, j)，通过中间顶点 k 检查是否存在从 i 到 j 的路径。如果存在路径，更新 edges[i][j] 为1，表示顶点 i 可以到达顶点 j。

统计出度并计算平均出度：

程序通过调用 statistics() 函数统计图中每个顶点的出度（即从该顶点出发的边的数量）。

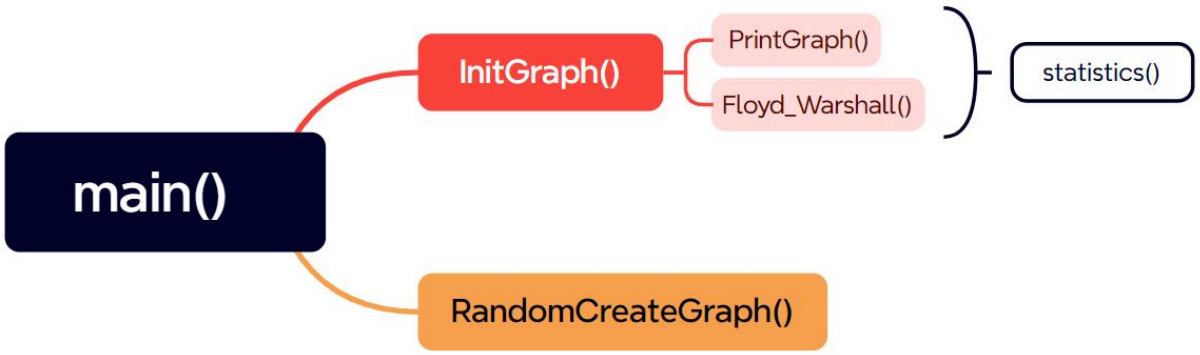
统计完每个顶点的出度后，计算所有顶点的平均出度，并将结果输出。

程序结束：

程序输出结果后，结束执行。

2.4 各程序模块之间的层次关系

函数模块层次关系图如下图。



3 详细设计

3.1 伪代码的设计

①main()：

```
// 获取用户输入的顶点数和边数
print("请输入图的顶点数（n）：")
n = input()
print("请输入图的边数（e）：")
e = input() // 输入校验
if n <= 0 or e < 0 or e > n * (n - 1):
print("错误：顶点数必须大于0，边数必须在0到顶点数*(顶点数-1)之间。")
 return
// 初始化图
G = createGraph(n, e)
// 随机生成图的边
RandomCreateGraph(G)
// 打印图的邻接矩阵
print("随机生成的图的邻接矩阵：")
PrintGraph(G)
// 计算图的传递闭包
Floyd_Warshall(G)
// 统计出度并计算平均出度
statistics(G)
```

② createGraph()

```
function createGraph(n, e):
  // 创建图结构并初始化
    G = new Graph
    G.n = n
    G.e = e

   // 初始化邻接矩阵为0, 表示没有边
    for i = 0 to n-1:
       for j = 0 to n-1:
          G.edges[i][j] = 0

    return G
```

③ RandomCreateGraph()

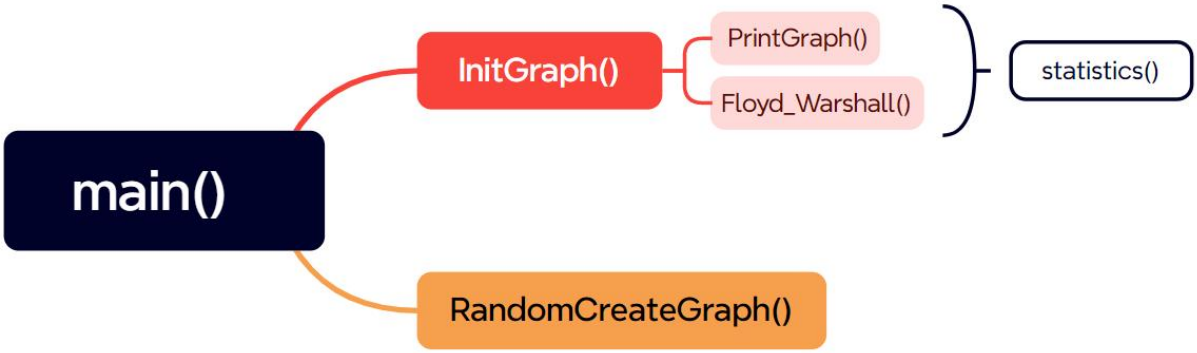
```
RandomCreateGraph(G):
  size = G.n * (G.n - 1)
  S = new array of size

 // 初始化辅助数组S为0, 表示没有生成边
for i = 0 to size-1:
    S[i] = 0

 // 随机生成e条边
 cnt = 0
 while cnt < G.e:
    index = random(0, size-1)
    if S[index] == 0:
        S[index] = 1
        cnt += 1

// 将S中的值赋给邻接矩阵
cnt = 0
for i = 0 to G.n-1:
    for j = 0 to G.n-1:
        if i != j:
            G.edges[i][j] = S[cnt]
            cnt += 1
```

3.2 函数的调用关系图



4 调试分析报告

4.1 调试过程中遇到的问题和解决方法

在调试过程中，遇到了一些问题，经过分析和调整，成功解决了这些问题。以下是主要的问题及解决方法：

①输入

描述：在用户输入图的顶点数和边数时，程序有时没有正确处理输入的边数和顶点数的合法性。例如，输入的顶点数为负数或者边数超出了合法范围时，程序没有及时给出提示，导致后续操作出错。

解决方法：增加了输入验证逻辑，确保用户输入的 n（顶点数）大于0，且 e（边数）在 0 和 n*(n-1) 之间。如果输入无效，则输出错误提示并终止程序。

②邻接矩阵未正确初始化

描述：在使用邻接矩阵表示图时，发现邻接矩阵在初始化时没有将所有元素正确设为0，导致在生成图时产生错误的连接关系。

解决方法：检查并确认了在 InitGraph() 函数中，邻接矩阵 edges 在图初始化时必须全部设为0。增加了循环来保证每个元素都被初始化为0。

③生成边时存在重复

描述：在生成随机图时，由于邻接矩阵是通过 S 数组来记录生成的边，发现有时会重复生成相同的边，导致边数超过用户指定的 e。

解决方法：通过检查数组 S，确保每一条边在随机选择时都是唯一的。对数组 S 增加了判重机制，确保每次生成的边没有重复。

4.2 设计实现的回顾讨论

以下是对设计与实现过程的回顾与讨论：

1. 图的表示与数据结构选择

设计思路：我们选择使用 邻接矩阵 来表示图。这是因为邻接矩阵对于表示稠密图（即边较多的图）更为高效，同时可以在常数时间内查找任意两顶点之间是否有边（O(1)）。

讨论：邻接矩阵的优点是简单且访问效率高，但当图较为稀疏时，邻接矩阵的空间复杂度较高，存储了大量的零值。在实际应用中，如果图的边数较少，可能更适合使用 邻接表，以节省存储空间。但考虑到本实验是为了测试和模拟简单的图操作，邻接矩阵的实现足够高效。

2. 图的初始化与边的生成

设计思路：在程序设计中，首先通过 InitGraph() 函数对图进行初始化，确保所有顶点和边的相关数据结构被正确设置为初值。随后，使用 RandomCreateGraph() 来随机生成图的边。这一过程中，使用一个辅助数组 S 来记录已生成的边，从而避免重复边的生成。

讨论：随机生成图的思路是通过一个随机数生成器来选择边，这种方法有效避免了手动指定边的繁琐，同时可以模拟多种不同的图结构。然而，生成边时需要特别小心边的重复性，以避免出现错误的图结构。在这里，我们采取了简单的标记法 S[index] = 1 来保证每一条边的唯一性，但如果在实际应用中需要生成更复杂的图结构，可能需要更高效的算法来避免重复边。

3. 传递闭包计算（Floyd-Warshall算法）

设计思路：我们使用 Floyd-Warshall 算法来计算图的传递闭包。该算法基于动态规划思想，逐步更新邻接矩阵中每对顶点之间的最短路径信息，最终得到传递闭包。

讨论：Floyd-Warshall算法的时间复杂度是 O(n^3)，当图的顶点数量较大时，该算法的效率会有所下降。不过，由于本实验的图规模（最多100个顶点）相对较小，使用该算法不会引发性能问题。如果图的规模更大，可以考虑使用其他图算法（如深度优先搜索（DFS）或广度优先搜索（BFS））来优化计算过程。

4. 出度统计与平均出度计算

设计思路：在统计出度时，我们通过遍历图的每个顶点并计算其从该顶点出发的边的数量，最终计算出所有顶点的平均出度。

讨论：计算平均出度的思路较为简单直观，但需要注意的是，这种计算只关注 出度（即顶点向外的边），而不考虑 入度 或其他图的性质。如果需要进一步分析图的性质（如强连通性、图的可达性等），可能需要更复杂的统计方法。此外，出度的统计方法假设每条边都有明确的方向，并且是从一个顶点指向另一个顶点，这对于有向图是非常合适的。

4.3 算法复杂度分析

1·时间复杂度

程序的总体时间复杂度是 **O(n^3)**，其中 n 是图的顶点数。

这个复杂度主要由 Floyd-Warshall算法 引起，适用于较小规模的图。

2·空间复杂度

InitGraph: O(n^2)

RandomCreateGraph: O(n^2)

Floyd-Warshall: O(n^2)

statistics: O(1)（常量空间）

因此，程序的总体空间复杂度为：**O(n^2)**

4.4 改进设想的经验和体会

4.4.1 改进 1--替换邻接矩阵为邻接表

问题：邻接矩阵的空间复杂度是 O(n^2)，当图的边数较少（稀疏图）时，会浪费大量内存。

改进：使用 邻接表 代替邻接矩阵，邻接表的空间复杂度是 O(n + e)，适合存储稀疏图。用链表或动 态数组存储每个顶点的出边。

修改图的初始化、随机生成边和传递闭包算法以适应邻接表。

```
typedef struct {
    int n; // 顶点个数
    int e; // 边的个数
    int* adj[MAX_V]; // 邻接表 }
Mgraph;
```

4.4.2 改进2 —用DFS替换Floyd-Warshall算法

```
void DFS(Mgraph* G, int v, int visited[]) {
    visited[v] = 1; // 标记当前顶点为已访问
      for (int i = 0; i < G->n; i++)
         {
            if (G->adj[v][i] == 1 && !visited[i])
           { DFS(G, i, visited); // 递归访问所有可达的顶点}
          }
       }
// 计算图的传递闭包（使用DFS）
 void Floyd_Warshall_DFS(Mgraph* G) {
// 使用DFS计算每个顶点可到达的所有顶点
 for (int i = 0; i < G->n; i++)
    { int visited[MAX_V] = {0}; // 用来标记访问过的顶点
      DFS(G, i, visited); // 从顶点i开始DFS
        for (int j = 0; j < G->n; j++)
        { if (visited[j])
          { G->adj[i][j] = 1; // 如果j是可达的，则更新邻接表 }
        }
      }
    }
```

Floyd-Warshall算法的时间复杂度是 O(n^3)，当顶点数较多时，效率较低。
改进：对于传递闭包，可以使用 深度优先搜索（DFS） 或 广度优先搜索（BFS） 进行逐个顶点的

5 用户使用说明

1.使用 gcc 编译生成可执行文件。

gcc -o main -std=c11 main.c

2.执行可执行文件：

在在 Linux或macOS 环境下： ***./main***

- 在 Windows cmd 环境下： ***main***

3.输入数据

执行程序后，用户需要通过标准输入提供两个整数。

4.输出参照1.3部分

6 测试结果

测试部分划分为如下环节。

6.1 测试1--基础样例测试

进行1.4 节的样例测试。

例如

```
请输入图的顶点数（n）：5
请输入图的边数（e）：12

----- 生成图的参数 -----
顶点数: 5
边数: 12
------------------------

----- 随机生成的图的邻接矩阵 -----
0 1 1 0 1
0 0 1 0 0
1 0 0 1 1
1 0 1 0 1
1 0 1 0 0

----- 统计结果 -----
Average outdegree: 4.00
```

6. 2测试2--非法输入和边界条件

```
请输入图的顶点数（n）：1
请输入图的边数（e）：55
错误：顶点数必须大于0，边数必须在0到顶点数*(顶点数-1)之间。
```

6.2 测试3--利用输出语句与手动画图计算结果校对

```
请输入图的顶点数（n）：5
请输入图的边数（e）：6

----- 生成图的参数 -----
顶点数：5
边数：6
------------------------

----- 随机生成的图的邻接矩阵 -----
0 0 0 0 1
1 0 1 0 0
0 1 0 0 1
0 0 0 0 0
1 0 0 0 0

----- 统计结果 -----
Average outdegree: 1.60
```

画一下图，发现完全符合。

手动计算传递闭包，也是符合的。