Temp

```
#include <stdio.h> // ইনপুট-আউটপুট ফাংশন (printf, scanf) ব্যবহারের জন্য
#include <stdlib.h> // সাধারণ ইউটিলিটি ফাংশন (rand, srand) ব্যবহারের জন্য
#include <time.h> // সময়-সম্পর্কিত ফাংশন (clock, time) ব্যবহারের জন্য
#include <string.h> // এই প্রোগ্রামের জন্য অপরিহার্য নয়, তবে সাধারণত স্ট্রিং অপারেশনের জন্য ব্যবহৃত হয়
#include <math.h> // এই প্রোগ্রামের জন্য অপরিহার্য নয়, তবে গাণিতিক ফাংশন ব্যবহারের জন্য


#define MAXN 8000 // গ্রাফে সর্বোচ্চ 8000টি শীর্ষবিন্দু (Vertices) থাকতে পারে

// 'matrix' অ্যারেটি আমাদের অ্যাডজাসেন্সি ম্যাট্রিক্স। এটি n x n আকারের হবে।
// matrix[i][j] = 1 মানে i থেকে j পর্যন্ত একটি ধার আছে। 0 মানে নেই।
int matrix[MAXN][MAXN];
int row, column, n;  // 'n' হলো শীর্ষবিন্দুর সংখ্যা; 'row' ও 'column' লুপ ঘোরানোর জন্য

// ফাংশন প্রোটোটাইপ (Functions Prototypes)
void fill(void); // অ্যাডজাসেন্সি ম্যাট্রিক্সকে র‍্যান্ডম মান দিয়ে ভরার জন্য
void result(void); // ইন-ডিগ্রী ও আউট-ডিগ্রী হিসাব করার জন্য

int main()
{
clock_t start, end; // সময় পরিমাপের জন্য ভেরিয়েবল
double elapsed; // মোট সময় সেকেন্ডে রাখার জন্য
```

```
printf("Enter the amount of Vertices: ");
scanf("%d", &n); // ব্যবহারকারীর কাছ থেকে 'n' (শীর্ষবিন্দুর সংখ্যা) ইনপুট নেওয়া

// --- অ্যালগরিদম টাইমিং শুরু ---
start = clock();

// ১. ম্যাট্রিক্স র‍্যান্ডম ডেটা দিয়ে পূরণ করো
fill();

// ২. ইন-ডিগ্রী ও আউট-ডিগ্রী হিসাব করো
result();
```

```
    // --- অ্যালগরিদম টাইমিং শেষ ---
    end = clock();

    // মোট সময় হিসাব (সেকেন্ডে)
    elapsed = (double)(end - start) / CLOCKS_PER_SEC;
    printf("Time taken: %.2f sec.\\n", elapsed);

    return 0; // প্রোগ্রাম সফলভাবে শেষ হলো
}


// ফাংশন: fill
// কাজ: n x n অ্যাডজাসেন্সি ম্যাট্রিক্সকে 0 বা 1 র‍্যান্ডম মান দিয়ে পূরণ করা।
void fill()
{
// srand() ফাংশন র‍্যান্ডম সংখ্যা জেনারেশনকে প্রতিবার নতুন করে শুরু করে (সময় ব্যবহার করে)
srand((unsigned)time(NULL));
    // Outer Loop: প্রতিটি সারির জন্য (Source Vertex)
    for (row = 0; row < n; row++)
    {
        // Inner Loop: প্রতিটি কলামের জন্য (Destination Vertex)
        for (column = 0; column < n; column++)
        {
            // matrix[row][column]-এ 0 বা 1 রাখা হলো।
            // 1 মানে 'row' থেকে 'column' পর্যন্ত একটি ধার আছে।
            matrix[row][column] = rand() % 2;
        }
    }
}
// ফাংশন: result
// কাজ: মোট ইন-ডিগ্রী ও আউট-ডিগ্রী গণনা করা এবং হ্যান্ডশেকিং উপপাদ্য পরীক্ষা করা।
void result()
{
```

```
int in_degree = 0, out_degree = 0; // মোট ইন-ডিগ্রী ও আউট-ডিগ্রী রাখার জন্য ভেরিয়েবল
```

```
// --- ইন-ডিগ্রী (In-Degree) গণনা ---
// একটি ডিরেক্টেড গ্রাফে ইন-ডিগ্রী মানে: একটি নির্দিষ্ট Vertex-এ যতগুলো ধার এসে শেষ হয়।
for (row = 0; row < n; row++)
{
   for (column = 0; column < n; column++)
   {
      // In-Degree-র জন্য, আমরা Matrix-এর কলাম এবং সারি Swap করি: matrix[column][row]
      // আমরা একটি নির্দিষ্ট 'row'-তে (Destination) কতোটি 'column' (Source) থেকে ধার এসেছে, তা যোগ করি।
      in_degree += matrix[column][row];
   }
}

// --- আউট-ডিগ্রী (Out-Degree) গণনা ---
// আউট-ডিগ্রী মানে: একটি নির্দিষ্ট Vertex থেকে যতগুলো ধার বাইরে বেরিয়ে যায়।
for (row = 0; row < n; row++)
{
   for (column = 0; column < n; column++)
   {
      // Out-Degree-র জন্য, আমরা স্বাভাবিকভাবে যোগ করি: matrix[row][column]
      // আমরা একটি নির্দিষ্ট 'row' (Source) থেকে কতোটি 'column' (Destination)-এ ধার                গেছে, তা যোগ করি।
      out_degree += matrix[row][column];
   }
}

printf("Total In-Degree: %d\\n", in_degree);
printf("Total Out-Degree: %d\\n", out_degree);

// --- হ্যান্ডশেকিং উপপাদ্য পরীক্ষা ---
if(in_degree != out_degree)
{
   // তাত্ত্বিকভাবে, এই প্রিন্ট হওয়া উচিত নয়।
```

```
    printf("Sum of In Degree and Out Degree are not Equal\\n");
  }
  else
  {
    // গ্রাফ থিওরি অনুযায়ী, এটিই সঠিক ফলাফল।
    printf("Sum of In Degree and Out Degree are Equal\\n");
  }
```

}