Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе № 6

по дисциплине: "Логика и основы алгоритмизации в инженерных задачах"

на тему: "Поиск расстояний в графе"

Выполнили:

студенты группы 20ВВ2

Макарова А. Ю.

Тельнова А. Д.

Принял:

д.т.н., профессор

Митрохин М. А.

к.т.н., доцент

Юрова О. В.

Пенза, 2021

**Задание 1**

1. Сгенерируйте (используя генератор случайных чисел) матрицу смежности для неориентированного графа G. Выведите матрицу на экран.
2. Для сгенерированного графа осуществите процедуру поиска расстояний, реализованную в соответствии с приведенным выше описанием. При реализации алгоритма в качестве очереди используйте класс queue из стандартной библиотеки С++.
3. \* Реализуйте процедуру поиска расстояний для графа, представленного списками смежности.

**Описание работы программы**

**1 – 2.** Количество вершин графа N пользователь вводит с клавиатуры. Массив G[N][N] – матрица смежности графа заполняется случайными значениями. Результат выводится на экран с помощью собственной функции printmatrix (функция выводит квадратную матрицу G размерностью N).

srand(time(NULL));

for (int i = 0; i < N; i++)

for (int j = i; j < N; j++)

if (i == j)

G[i][j] = 0;

else

{

if (rand() % 10 < 4)

G[i][j] = 1;

else

G[i][j] = 0;

G[j][i] = G[i][j];

}

printf("\nМатрица смежности графа G:\n");

printmatrix(G, N);

После вывода матрицы смежности пользователь вводит с клавиатуры номер вершины num, расстояния до которой будут найдены.

Создается и инициализируется массив visited[num]. Все его элементы получают значение -1, по окончанию работы алгоритма поиска visited[i] – расстояние между вершиной старта и i-ой вершины. Значение num-1 передается функции DistenceBFS.

void DistanceBFS(int num)

{

queue <int> q;

q.push(num);

visited[num] = 0;

while (!q.empty())

{

num = q.front();

q.pop();

for (int i = 0; i < N; i++)

if (visited[i] == -1 && G[num][i] == 1)

{

q.push(i);

visited[i] = visited[num] + 1;

if (visited[num] + 1 > maxdepth)

maxdepth = visited[num] + 1;

}

}

}

Эта функция осуществляет поиск расстояний в графе, представленном матрицей смежности, на основе обхода в ширину. Номер вершины старта заносится в очередь. В массиве visited элемент с соответствующим номером получает значение 0, так как расстояние от вершины до самой себя равно 0.

Затем, пока очередь не пуста, в переменную num – номер текущей вершины - извлекается первый элемент очереди, который в самой очереди удаляется следующим шагом. Просматривается строка с номером num в исходной матрице смежности. Если находится не посещенная вершина, смежная с текущей, ее номер заносится в очередь, а элемент массива visited[i] получает значение visited[num]+1, то есть расстояние от вершины старта до найденной вершины равно расстоянию до текущей вершины, увеличенному на 1. Также это расстояние сравнивается с глобальной переменной maxdepth – максимальным найденным расстоянием, которое необходимо для вывода результата.

Результат выводится с помощью функции printdistance, которая на вход получает максимальное maxdepth.

void printdistance(int maxdepth)

{

for (int i = 0; i <= maxdepth; i++)

{

if (i == 0)

printf("\n\nВершина старта: ");

else

printf("\n\nВершины уровня %d: ", i);

for (int j = 0; j < N; j++)

{

if (visited[j] == i)

printf("%d ", j + 1);

}

}

}

**3.\*** Перед поиском расстояний в графе, представленном списками смежности, список смежности генерируется на основе матрицы G с помощью функции CreateList, массив visited вновь инициализируется -1, переменная maxdepth обнуляется.

Node\*\* listG = (Node\*\*)malloc(N \* sizeof(Node\*));

for (int i = 0; i < N; i++)

listG[i] = NULL;

CreateList(G, listG, N);

for (int i = 0; i < N; i++)

visited[i] = -1;

maxdepth = 0;

Вновь вводится номер вершины старта и передается функции DistanceBFSlist, уменьшенный на единицу.

void DistanceBFSlist(Node\*\* head, int num)

{

queue <int> q;

Node\*\* tmp\_node = head;

q.push(tmp\_node[num]->vertex);

visited[tmp\_node[num]->vertex] = 0;

while (!q.empty())

{

num = q.front();

q.pop();

while (tmp\_node[num])

{

if (visited[tmp\_node[num]->vertex] == -1)

{

q.push(tmp\_node[num]->vertex);

visited[tmp\_node[num]->vertex] = visited[num] + 1;

if (visited[num] + 1 > maxdepth)

maxdepth = visited[num] + 1;

}

tmp\_node[num] = tmp\_node[num]->next;

}

}

}

Эта функция осуществляет поиск расстояний в графе, представленном списками смежности, на основе обхода в ширину. Номер вершины старта заносится в очередь. В массиве visited элемент с соответствующим номером получает значение 0.

Затем, пока очередь не пуста, в переменную num – номер текущей вершины - извлекается первый элемент очереди, который в самой очереди удаляется следующим шагом. Просматривается все элементы списка смежности текущей вершины. Если находится не посещенная вершина, ее номер заносится в очередь, а элемент массива visited[i] получает значение visited[num]+1, как и в реализации поиска расстояний для графа, представленного матрицей смежности. Аналогично находится максимальное найденной расстояние.

Результат выводится с помощью функции printdistance.

**Задание 2\***

1. Реализуйте процедуру поиска расстояний на основе обхода в глубину.
2. Реализуйте процедуру поиска расстояний на основе обхода в глубину для графа, представленного списками смежности.
3. Оцените время работы реализаций алгоритмов поиска расстояний на основе обхода в глубину и обхода в ширину для графов разных порядков.
4. Массив visited вновь инициализируется -1, а переменная maxdepth обнуляется. Номер вершины старта num вводится с клавиатуры.

Функция DistanceDFS получает на вход вершину старта num – 1 и начальное расстояние depth, равное при вызове функции 0.

void DistanceDFS(int num, int depth)

{

visited[num] = depth;

for (int i = 0; i < N; i++)

{

if (G[num][i] == 1 && visited[i] == -1)

DistanceDFS(i, depth + 1);

if (G[num][i] == 1 && visited[i] > depth)

DistanceDFS(i, depth + 1);

}

}

Эта функция осуществляет поиск расстояний в графе, представленном матрицей смежности, на основе обхода в глубину.

Элемент visited с индексом текущей вершины num получает значение depth.

Затем просматривается строка исходной матрицы с номером num. Если находится не посещенная вершина, смежная с текущей, вызывается DistanceDFS, которая на вход получает номер найденной вершины и текущее расстояние, увеличенное на 1. После этого вызова рекурсивно будут просмотрены все вершины графа и сформируется массив расстояний visited, но расстояния на данный момент могут быть не минимальными.

После этого, если для первой вершины, смежной со стартовой, расстояние в массиве visited окажется больше, чем расстояние, найденное в первом вызове функции, вновь вызовется DistanceDFS, которая на вход получит номер этой вершины и текущее расстояние, увеличенное на 1. Аналогично будут проверены все элементы массива расстояний.

В итоге сформируется массив расстояний, где максимальное расстояние будет найдено вне функции поиска.

for (int i = 0; i < N; i++)

if (visited[i] > maxdepth)

maxdepth = visited[i];

Результат выводится снова с помощью функции printdistance.

1. Массив visited вновь инициализируется -1, а переменная maxdepth обнуляется, генерируются новые списки смежности listGRAPH на основе исходной матрицы G. Номер вершины старта num вводится с клавиатуры.

Функция DistanceDFS получает на вход массив первых элементов списков смежности, вершину старта num – 1 и начальное расстояние depth, равное при вызове функции 0.

void DistanceDFSlist(Node\*\* list, int num, int depth)

{

visited[num] = depth;

Node\* tmp\_node = list[num];

while (tmp\_node)

{

if (visited[tmp\_node->vertex] == -1)

DistanceDFSlist(list, tmp\_node->vertex, depth + 1);

if (visited[tmp\_node->vertex] > depth + 1)

DistanceDFSlist(list, tmp\_node->vertex, depth + 1);

tmp\_node = tmp\_node->next;

}

}

Эта функция осуществляет поиск расстояний в графе, представленном списками смежности, на основе обхода в глубину.

Элемент visited с индексом текущей вершины num получает значение depth. Затем просматривается список смежности вершины num. Если находится не посещенная вершина, вызывается DistanceDFSlist, которая на вход получает номер найденной вершины и текущее расстояние, увеличенное на 1. После этого вызова рекурсивно будут просмотрены все вершины графа и сформируется массив расстояний visited, однако эти расстояния не будут минимальными.

После этого, если для первой вершины, смежной со стартовой, расстояние в массиве visited окажется больше, чем расстояние, найденное в первом вызове функции, вновь вызовется DistanceDFS, которая на вход получит номер этой вершины и текущее расстояние, увеличенное на 1. Аналогично будут проверены все элементы массива расстояний.

В итоге сформируется массив расстояний, где максимальное расстояние, необходимое для вывода результата, будет найдено вне функции поиска, как в предыдущем задании. Результат выводится с помощью функции printdistance.

**3.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Порядок графа** | **Время работы**  **DistanceBFS, сек** | **Время работы**  **DistanceBFSlist, сек** | **Время работы**  **DistanceDFS, сек** | **Время работы**  **DistanceDFSlist, сек** |
| 100 | 0,001 | 0,001 | 0,144 | 0,001 |
| 200 | 0,001 | 0,001 | 2,245 | 0,015 |
| 400 | 0,001 | 0,003 | 39,752 | 0,133 |
| 800 | 0,005 | 0,008 | 639,763 | 2,794 |
| 1000 | 0,005 | 0,015 | 1562,053 | 5,538 |

Быстрее всего поиск расстояний происходит при использовании алгоритма на основе обхода в ширину для графа, представленного матрицей смежности. Если граф представлен списками смежности, алгоритм на основе обхода в ширину так же эффективнее, чем алгоритм на основе поиска в глубину. Самый неэффективный случай – алгоритм поиска расстояний в графе, представленном матрицей смежности, на основе обхода в глубину. Он работает медленнее остальных на графах любого порядка.

**Результаты работы программы**

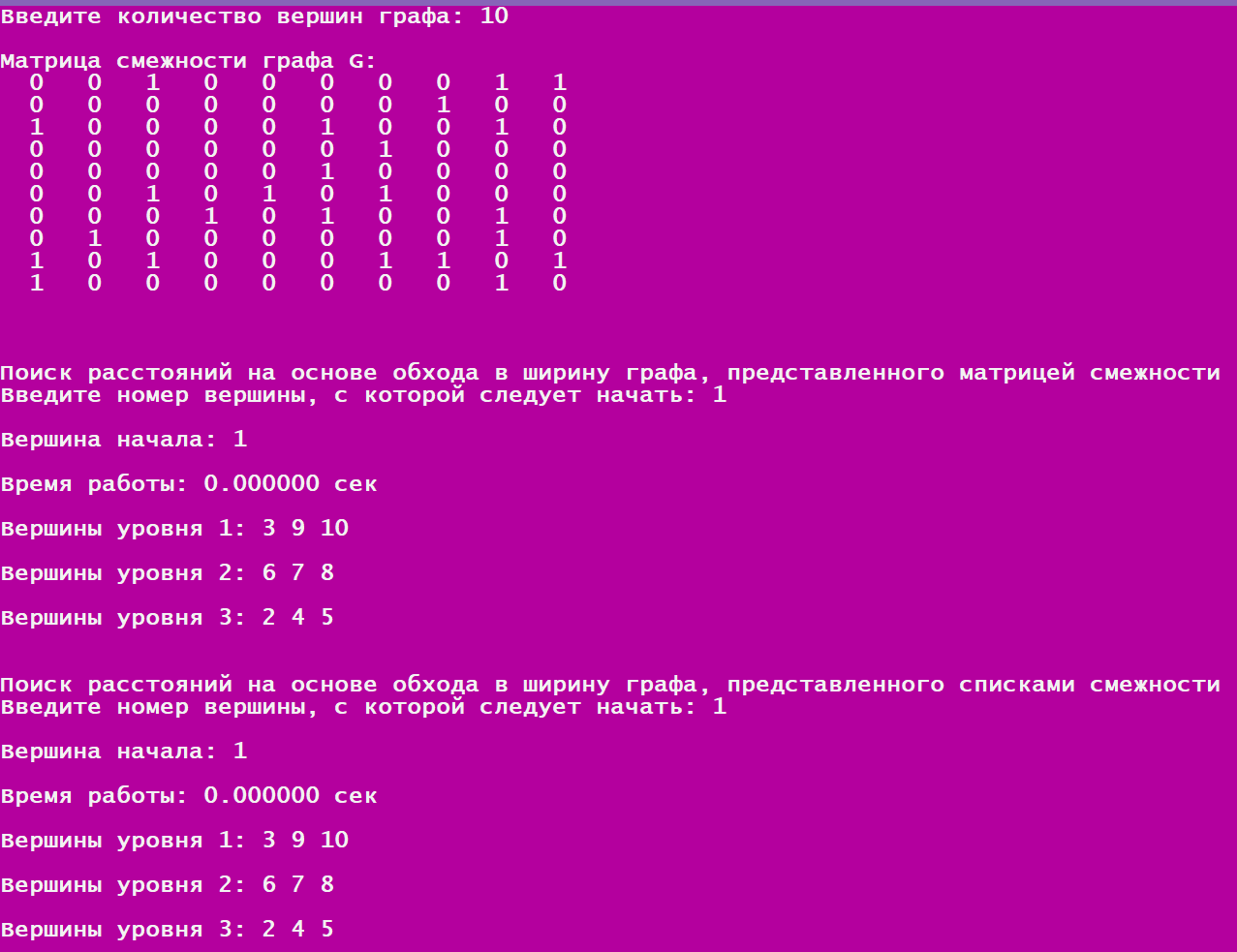


Рисунок 1 - Результат работы программы (начало)

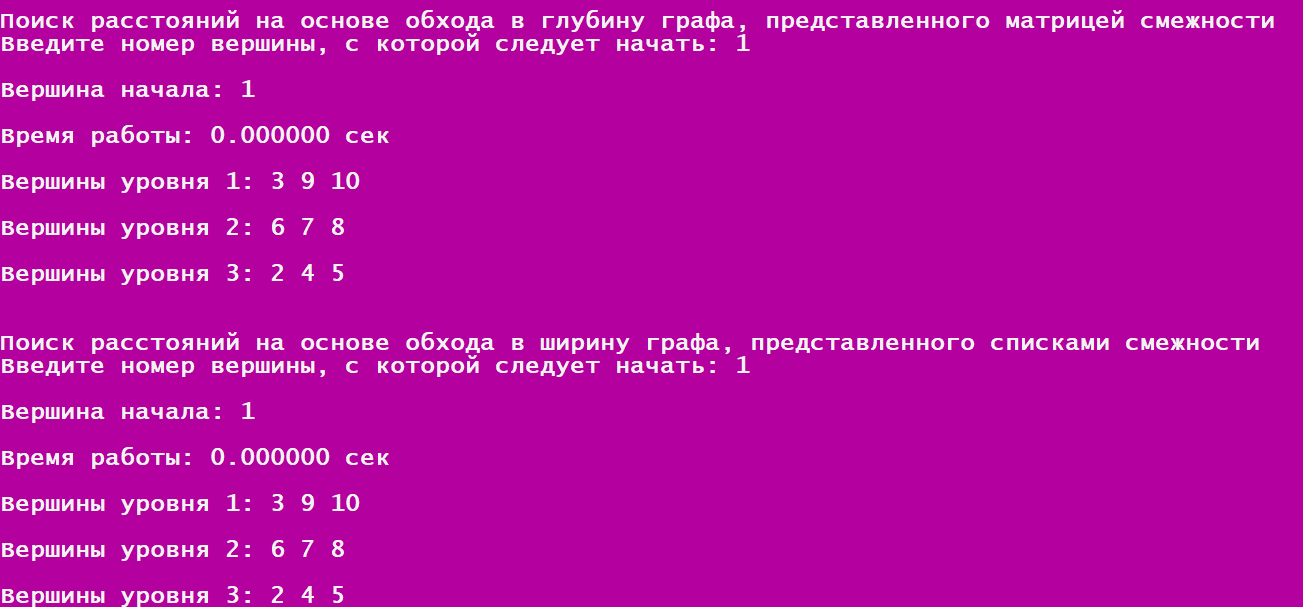


Рисунок 2 - Результат работы программы (продолжение)

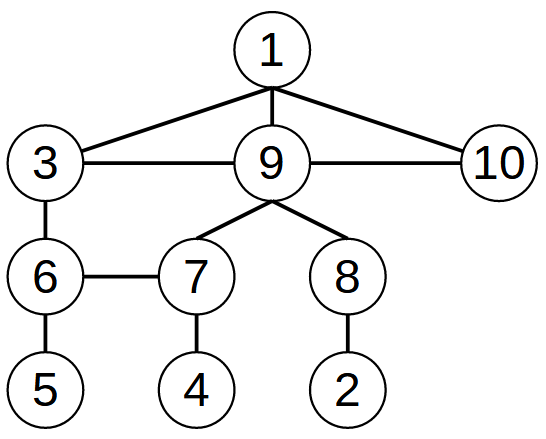


Рисунок 3 - Граф, соответствующий сгенерированной матрице смежности

При проверке вручную расстояния между вершиной 1 и остальными вершинами совпадают с расстояниями, полученными в программе.

**Вывод:** Реализовали алгоритмы поиска расстояний в графе на основе обхода в глубину и обхода в ширину для графов, представленных матрицей смежности и списками смежности. Оценили время работы всех алгоритмов на графах разных порядков.