|  |  |  |
| --- | --- | --- |
|  | | |
| Федеральное государственное бюджетное  образовательное учреждение высшего образования «Новосибирский государственный технический университет» | | |
|  | | |
| Кафедра прикладной математики | | |
| по дисциплине «Дискретная математика» | | |
| **Алгоритмы поиска в ширину и глубину** | | |
|  | | |
|  | Бригада 4 | Лезнёв Артём |
| Группа ПМ-15 | Дроздов Даниил |
| Вариант 7 | Мамаев Никита |
|  | Палий Никита |
|  |  |
|  |  |
| Преподаватель | Рояк Михаил эмМануилович |
|  |  |
| Новосибирск, 2022 | | |

1. **Задание**

Определить, существует ли маршрут между двумя заданными вершинами в неориентированном графе. Граф в памяти представлять в виде матрицы.

1. **Распределение задач в бригаде**

Создание тестов для поисков ошибок в программе – Лезнёв Артём

Создание тестов для оценки фактической сложности алгоритма – Мамаев Никита

Программа с использованием поиска в глубину – Дроздов Даниил

Программа с использованием поиска в ширину – Палий Никита

1. **Анализ задачи**

*Входные данные*: в первой строчке v1, v2 - вершины графа, между которыми требуется определить наличие маршрута, во второй n и m – количество вершин и ребер в графе, далее в m строках ai и bi – номера вершин, между которыми существует ребро.

*Выходные данные*: результат проверки: “true”, если между введёнными вершинами графа есть маршрут, или ”false” в обратном случае.

*Метод решения*: решение строится из определения маршрута: «чередующаяся последовательность вершин и рёбер в которой любые два соседних элемента инцидентны». Записываем граф в виде матрицы, в которой на пересечениях строк n со столбцами m находятся значения true, указывающие на наличие ребра между вершинами графа n и m, либо false, указывающие на её отсутствие. Начнем из первой указанной вершины поиск второй указанной вершины. Если её удалось найти, то между вершинами есть маршрут. Чтобы избежать зацикливания, будем отмечать каждую проверенную вершину в коллекции сет.

*Поиск в глубину*: создаём стек для обхода графа. Начинаем обход с первой указанной вершины: проверяем, не является ли она искомой, отмечаем её и добавляем её в стек. Затем, пока стек не стал пуст или искомая вершина не найдена, обрабатываем текущую вершину. Сначала ищем первого неотмеченного соседа текущей вершины. Если он является искомой, то поиск завершен – маршрут найден, возвращаем «true». Если нет, отмечаем его, добавляем текущую вершину в стек и назначаем его новой текущей вершиной. В случае, если такового не найдено, выталкиваем из стека старую текущую вершину и назначаем её текущей. Если обход завершен, а искомая вершина не нашлась – то маршрута нет, возвращаем «false».

*Поиск в ширину*: для алгоритма потребуется очередь. На каждом шагу алгоритм берет из начала очереди вершину и добавляет все неотмеченные смежные с ней вершины в конец очереди, проверяя и отмечая их. Если среди них найдена искомая – маршрут найден, вернуть «true». Если очередь пуста, то алгоритм завершает работу и возвращает «false».

1. **Теоретическая сложность алгоритмов:**
   1. В ширину:

Как видно из фрагмента алгоритма, обход итерирует по строкам матрицы в поисках неотмеченного соседа вершин, соответствующей текущим вершин. В худшем случае он обработает n строк по n элементов в каждой, в результате – n2 элементов матрицы, где n – число вершин.

Сложность такого алгоритма – O(n2).

В худшем случае, при увеличении количества вершин в k раз, затраты времени на обход должны увеличиваться в k2 раз. Худший случай тот, в котором количество ребер между вершинами максимальное, и при этом все кроме последней (в порядке обхода) вершины не имеют ребра с искомой.

Фрагмент программы обхода в ширину:

while (!s.empty() && !pathFound)

{

. . .

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

{

if (neighbors(current, i) && !visited.count(i))

{

. . .

}

}

}

* 1. В глубину:

Аналогично обходу в ширину, обход итерирует по строкам матрицы в поисках неотмеченного соседа вершин, соответствующей текущим вершин.

Сложность такого алгоритма такая же – O(n2).

В худшем случае, при увеличении количества вершин в k раз, затраты времени на обход должны увеличиваться в k2 раз.

Фрагмент программы обхода в ширину:

while (!s.empty() && !pathFound)

{

for (int i = 0;

i < graphSize && !(neighbors(current, i) &&

!visited.count(i));

i++, unvisited++);

. . .

}

1. **Тесты для поиска ошибок**

|  |  |  |
| --- | --- | --- |
| **№** | **Входные данные** | **Комментарий** |
| **1** | 12 11  0 1  1 2  2 3  2 4  4 5  4 7  7 8  8 6  5 6  6 9  9 10 |  |
| **1.1** | 6 6 | Маршрут существует |
| **1.2** | 0 10 |
| **1.3** | 3 8 |
| **1.4** | 11 11 |
| **2** | 12 17  0 3  0 4  0 5  1 3  1 4  1 5  2 3  2 4  2 5  6 7  7 8  8 9  9 10  10 11  11 6  6 10  6 9 |  |
| **2.1** | 6 8 | Маршрут существует |
| **2.2** | 2 0 |
| **2.3** | 10 2 | Маршрут не существует |
| **2.4** | 0 9 |
| **3** | 13 17  0 0  6 11  11 3  2 2  7 7  10 10  9 9  2 7  7 10  10 9  12 1  1 5  5 8  8 4  4 12  1 4  1 8 |  |
| **3.1** | 6 10 | Маршрут не существует |
| **3.2** | 0 4 |
| **3.3** | 0 0 | Маршрут существует |
| **3.4** | 12 5 |
| **3.5** | 2 9 |
| **4** | 9 8  5 5  7 7  6 6  8 8  5 7  7 6  6 8  8 5 |  |
| **4.1** | 5 6 | Маршрут существует |
| **4.2** | 3 4 | Маршрут не существует |
| **4.3** | 0 1 |
| **4.4** | 2 0 |
| **4.5** | 2 7 |
| **4.6** | 3 5 |

1. **Тесты для оценки сложности**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **№** | **Входные данные** | **Поиск в глубину** | | **Поиск в ширину** | |
|  | **Число вершин(n)** | **Время выполнения ti (секунды)** | **ti/t-1** | **Время выполнения ti (секунды)** | **ti/t-1** |
| **1** | n = 512 | 0.001 | - | 0.001 | - |
| **2** | n = 1024 | 0.003 | 3.0 | 0.007 | 7.0 |
| **3** | n = 2048 | 0.016 | 5.3 | 0.034 | 4.8 |
| **4** | n = 4096 | 0.078 | 4.8 | 0.171 | 5.0 |
| **5** | n = 8192 | 0.426 | 5.4 | 0.945 | 5.5 |
| **6** | n = 16384 | 1.824 | 4.3 | 4.214 | 4.4 |
| **7** | n = 32768 | 6.547 | 3.6 | 12.773 | 3.0 |

Вывод: при увеличении количества вершин в 2 раза наблюдается увеличение времени работы алгоритмов в среднем примерно в 4 раза. Это говорит о том, что сложность алгоритмов – O(n2).