Московский авиационный институт

(национальный исследовательский институт)

Институт «Компьютерные науки и прикладная математика»

Курсовая проект

«Эвристический поиск на графах»

по курсу

«Дискретный анализ»

V семестр

Студент: Меркулов Ф. А.
Группа: М8О-307Б-21

Руководитель: Сорокин С. А.

Оценка:

Дата:

Москва, 2023

Условие

Ваша программа должна читать входные данные из стандартного потока ввода и выводить ответ на стандартный поток вывода. Реализуйте алгоритм A* для неориентированного графа. Расстояние между соседями вычисляется как простое евклидово расстояние на плоскости.

Формат ввода

В первой строке вам даны два числа n и m $(1 \le n \le 10^4, 1 \le m \le 10^5)$ — количество вершин и рёбер в графе. В следующих n строках вам даны пары чисел xy $(-10^9 \le x, y \le 10^9)$, описывающие положение вершин графа в двумерном пространстве. В следующих m строках даны пары чисел в отрезке от 1 до n, описывающие рёбра графа. Далее дано число q $(1 \le q \le 300)$ и в следующих q строках даны запросы в виде пар чисел ab $(1 \le a, b \le n)$ на поиск кратчайшего пути между двумя вершинами.

Формат вывода

В ответ на каждый запрос выведите единственное число — длину кратчайшего пути между заданными вершинами с абсолютной либо относительной точностью 10^{-6}, если пути между вершинами не существует выведите -1.

Метод решения

Требуется реализовать алгоритм поиска кратчайшего пути во взвешенном графе. С этой задачей отлично справляется алгоритм Дейкстры, но в силу особенности задачи так же применим алгоритм поиска A*.

Алгоритм Дейкстры хранит для каждой вершины длину кратчайшего пути до неё и на каждом шаге ищет вершину с минимальной такой величиной. Алгоритм обходит все смежные вершины и запоминает, что текущая вершина с минимальным кратчайшим путем была посещена.
Изначально пути до всех вершин равны бесконечности, а в старте значение равно нулю. Алгоритм имеет сложность $O(n^2 + m)$ в простой реализации и $O(n\,logn + m)$ в реализации с использованием двоичной кучи.

Алгоритм поиска A* работает похожим образом, но для выбора вершины на каждом шаге использует функцию $f(v) = g(v) + h(v)$, где $g(v)$ — кратчайший путь от стартовой вершины до v, $h(v)$ — эвристическая функция, которая вычисляет приближённое значение кратчайшего пути до финиша. В моём случае функции $h(v)$ — евклидово расстояние на плоскости между координатами вершин v и финиша. Алгоритм имеет временную сложность $O(|E|)$ и пространственную $O(|V|)$, где V — множество вершин в графе, а
E — множество дуг графа.

Описание программы

В функции **main** программа считывает входные данные, включая количество вершин и рёбер, координаты точек и связи между ними, формируя граф. Затем она считывает количество запросов на поиск пути, для каждого запроса сбрасывает информацию о стоимости пути и закрытых вершинах, и запускает функцию **aStar** для нахождения и вывода кратчайшего пути между парой точек.

Функция **euclideanDistance** вычисляет Евклидово расстояние между двумя точками, которое используется в алгоритме как эвристическая функция $h(v)$.

Функция **aStar** реализует сам алгоритм A*. Она инициализирует открытое множество вершин с помощью очереди с приоритетами. Вершины извлекаются из очереди по одной, и для каждой из них:

- Если вершина является целевой, функция возвращает стоимость пути до неё.
- Для каждого соседа рассматриваемой вершины проверяется, не был ли он уже посещён (закрыт).
- Если новый путь к соседу короче, чем известный ранее, обновляется стоимость пути до соседа, и сосед добавляется в открытое множество с новым приоритетом, который включает в себя стоимость пути и эвристическое расстояние до цели.

Дневник отладки

Очень много раз получал $\boldsymbol{TL4}$ в связи с тем, что в приоритетной очереди использовал 3 элемента вместо 2, причём третий элемент был фиктивным и никак не использовался в алгоритме.

Потом все остальные посылки до $\boldsymbol{OK}$ получал $\boldsymbol{TL5}$, что было вызвано использование типа long double (в эвристической функции, которая используется чуть ли не больше всех остальных операций в программе, используется функция sqrt, которой приходилось делать больше действий на вычисление значений с большей точностью и следовательно тратилось больше времени), замена на double решила эту проблему.

Тест производительности

В тесте сравниваются алгоритм Дейкстры и алгоритм поиска A*.
Каждый тест содержит 50 запросов на поиск.
В тестах представлены несвязные и плотные графы, чтобы сравнить алгоритмы в разных условиях.

Количество вершин (n)	Количество рёбер (m)	Алгоритм A*, мс	Алгоритм Дейкстры, мс
100	50	0.875	0.890
100	200	3.625	8.834
100	10^3	3.586	34.628
10^3	500	4.184	4.289
10^3	2 * 10^3	90.385	172.436
10^3	10^4	40.902	397.362
10^4	5 * 10^3	9.567	11.228
10^4	2 * 10^4	980.609	2034.514
10^4	10^5	663.586	5719.548

Видно, что A* не сильно побеждает алгоритм Дейкстры при несвязных графах, потому что алгоритм Дейкстры так же быстро как A* определяет, что пути нет.
Чем плотнее граф, тем быстрее оказывается A*, так как он сразу выбирает наиболее подходящую вершину с использованием эвристики.

Выводы

В ходе выполнения курсового проекта я изучил алгоритмы поиска кратчайших путей в графах и реализовал алгоритм Дейкстры и A* с эвристикой, применяемой к своему заданию.

Почти во всех реализациях задачах мы каким-либо образом можем оценить расстояние до цели (расстояние в пространстве или Манхэттенское расстояние), поэтому алгоритм A* становится применим.

Как было видно выше в сравнении скорости работы алгоритмов Дейкстры и A*, A* в несколько раз быстрее на больших и плотных графах. С учётом того, как быстро растут города, улицы и жилые кварталы, граф города становится всё плотнее и плотнее из-за чего актуальность алгоритма A*, только повышается и повышается.