Курсовая работа по курсу дискретного анализа: Эвристический поиск на графах

Выполнил студент группы М8О-308Б-21 МАИ *Белоносов Кирилл.*

Условие

1. Общая постановка задачи: Реализовать алгоритм эвристического поиска на графе А*

2. Вариант задания(2):

   Алгоритм: Ваша программа должна читать входные данные из стандартного потока ввода и выводить ответ на стандартный поток вывода. Реализуйте алгоритм А* 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 вершнами.

   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

Для выполнения данной работы предполагается использоваться алгоритм А*, который используется для поиска кратчайшего пути во взвешенном графе. Алгоритм А* предполагает наличие некой эвристики, с помощью которой мы будем оценивать расстояние до конечной точки. В поставленной задаче лучшей эвристикой будет эвклидова метрика. Тогда на каждом шаге алгоритма выбор новой вершины происходит с помощью следующей функции f(u) = g(u) + h(u): h(u) - наша эвристика (расстояние от нашей вершины до конечной точки), g(u) - кратчайшее расстояние от стартовой вершины до текущей. Для выбора новой вершины можно хранить пару (f(u), u) и выбирать каждый раз пару с наименьшей функцией f(u) - для этого можно использовать бинарную кучу. Тогда сложность алгоритма составит O(MlogM) - где M - число ребер в графе

Описание программы

Для хранения пары f и u я создал структуру fu, также я переопределил оператор сравнения, чтобы на верхушке priority_queue лежала минимальная пара.

```
struct Point {
    ll x, y;
};

struct fu {
    double f;
    int u;
};

bool operator < (const fu& lhs, const fu& rhs) {
     return lhs.f > rhs.f;
}
```

Функция h рассчитывает эвристику

```
const double INF = 1e18;

double h(const Point& p1, const Point& p2) {
    return sqrt(pow(p1.x - p2.x, 2) + pow(p1.y - p2.y, 2));
}
```

Сама работа алгоритма происходит в функции AStar, которая на вход принимает стартовую вершину, конечную вершину и сам граф. Внутри заведем массив d чтобы хранить расстояние от начальной вершины до текущей

```
double AStar(int start, int goal, const Graph& g, const
   vector<Point>& vertices) {
    vector<bool> visited(g.size(), false);
    vector<double> d(g.size(), INF);
    d[start] = 0;
    priority_queue<fu> pq;
    pq.push({d[start] + h(vertices[start], vertices[goal]), start});
    while(!pq.empty()) {
        fu cur = pq.top();
        if(cur.u == goal)
            return d[cur.u];
        pq.pop();
        visited[cur.u] = true;
        for(int v: g[cur.u]) {
            if(visited[v])
                continue;
            double tmp = d[cur.u] + h(vertices[cur.u], vertices[v]);
            if(tmp < d[v]) {
```

```
                d[v] = tmp;
                pq.push({d[v] + h(vertices[v], vertices[goal]), v});
            }
        }
    }
    return -1;
}
```

Дневник отладки

Попытка 1-11

Ошибки были связаны с расчетом эвристики, нужно было взять значение INF = 1e18, а не 1e9

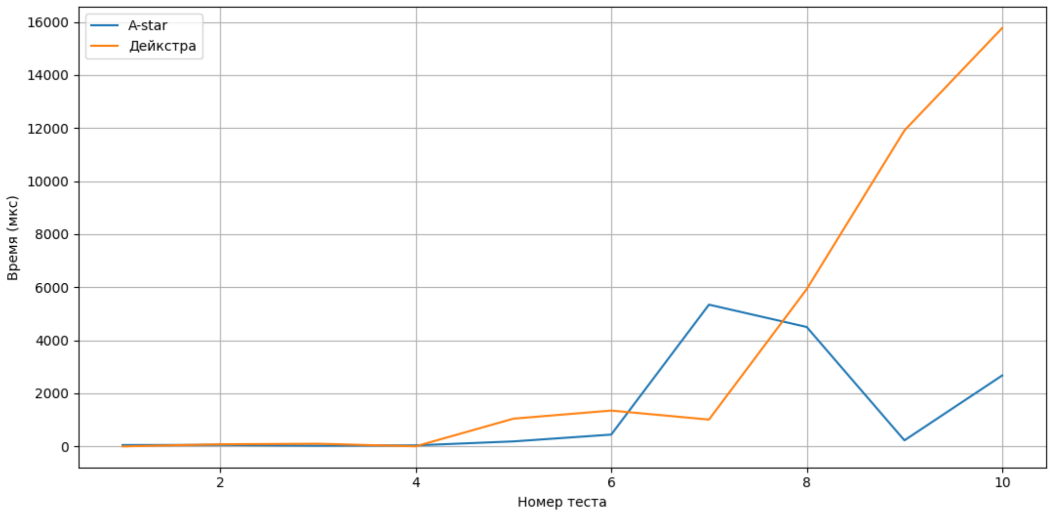
0.1 Результаты тестирования

Тест производительности

Тестирование программы производилось на 10 сгенерированных тестах на графах из условия (неориентированный граф, где каждой вершине соответствует точка):

1. n = 100 m = 100
2. n = 100 m = 500
3. n = 100 m = 1000
4. n = 1000 m = 1000
5. n = 1000 m = 5000
6. n = 1000 m = 10000
7. n = 10000 m = 10000
8. n = 10000 m = 50000
9. n = 10000 m = 75000
10. n = 10000 m = 100000

Вставка:



A-star показал себя чуть лучше дейкстры, кроме одного из тестовых примеров. Скорее всего путь в a-star находится быстрее благодаря эвристике

Выводы

В результате выполнения данной лабораторной работы я познакомился с эвристическим алгоритмом поиска в графе - A*. Реализовал его, а также реализовал алгоритм дейкстры для сравнения производительности. Алгоритм A-star применим в тех ситуациях, у нас достаточно много ребер и где мы можем ввести эвристику. Как можно заметить в тестировании программ, то чем больше ребер в графе, тем более эффективным становится A-star. A-star имеет больше применение в навигации, также он часто используется в играх, для передвижения персонажа