Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Курсовой проект по курсу «Дискретный анализ»

Студент: Д. С. Пивницкий
Преподаватель: С. А. Сорокин
Группа: М8О-206Б

Дата: 01.01.2021
Оценка:
Подпись:

Москва, 2021

Курсовой проект

Задача: Эвристический поиск в графе

Реализуйте алгоритм A* для неориентированного графа.

Формат входных данных: В первой строке вам даны два числа n и m - количество вершин и рёбер в графе. В следующих n строках вам даны пары чисел, 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 и в следующих q строках даны запросы в виде пар чисел на поиск кратчайшего пути между двумя вершинами.

Формат результата: 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". Расстояние между соседями вычисляется как простое евклидово расстояние на плоскости.

1 Эвристический поиск в графе

Алгоритм поиска по первому наилучшему совпадению на графе, то есть A* просматривает все пути пошагово, ведущие от начальной вершины в конечную, пока не найдёт минимальный. Он просматривает сначала те маршруты, которые «кажутся» ведущими к цели. От жадного алгоритма, который тоже является алгоритмом поиска по первому лучшему совпадению, его отличает то, что при выборе вершины он учитывает, помимо прочего, весь пройденный до неё путь. Составляющая $g(x)$ - это стоимость пути от начальной вершины, а не от предыдущей, как в жадном алгоритме. В начале работы просматриваются узлы, смежные с начальным; выбирается тот из них, который имеет минимальное значение $f(x)$, после чего этот узел раскрывается. На каждом этапе алгоритм оперирует с множеством путей из начальной точки до всех ещё не раскрытых (листовых) вершин графа — множеством частных решений, которое размещается в очереди с приоритетом. Приоритет пути определяется по значению $f(x) = g(x) + h(x)$. Алгоритм продолжает свою работу до тех пор, пока значение $f(x)$ целевой вершины не окажется меньшим, чем любое значение в очереди, либо пока всё дерево не будет просмотрено. Из множества решений выбирается решение с наименьшей стоимостью.

Алгоритм A* и допустим, и обходит при этом минимальное количество вершин, благодаря тому, что он работает с «оптимистичной» оценкой пути через вершину. Оптимистичной в том смысле, что, если он пойдёт через эту вершину, у алгоритма «есть шанс», что реальная стоимость результата будет равна этой оценке, но никак не меньше. Но, поскольку A* является информированным алгоритмом, то есть таким алгоритмом, который использует знания, относящиеся к конкретной задаче, такое равенство может быть вполне возможным.

Когда A* завершает поиск, он нашёл путь, истинная стоимость которого меньше, чем оценка стоимости любого пути через любой открытый узел. Но поскольку эти оценки являются оптимистичными, соответствующие узлы можно без сомнений отбросить. Таким образом, A* никогда не упустит возможности минимизировать длину пути, и потому является допустимым.

Предположим теперь, что некий алгоритм B вернул в качестве результата путь, длина которого больше оценки стоимости пути через некоторую вершину. На основании эвристической информации, для алгоритма B нельзя исключить возможность, что этот путь имел и меньшую реальную длину, чем результат. Соответственно, пока алгоритм B просмотрел меньше вершин, чем A*, он не будет допустимым.

2 Код

```
#include <iostream>
#include <vector>
#include <cmath>
#include <queue>
#include <iomanip>

using namespace std;

struct coords {
double x;
double y;
};

double dist(const coords &f1, const coords &f2) {
return sqrt((f1.x - f2.x) * (f1.x - f2.x) + (f1.y - f2.y) * (f1.y - f2.y));
}

class approx {
public:
int index;
double path;
double distance;
friend bool operator< (const approx &f1, const approx &f2) {
if ((f1.path + f1.distance) != (f2.path + f2.distance))
return (f1.path + f1.distance) > (f2.path + f2.distance);
return f1.index < f2.index;
}
approx(const int &new_index, const coords &u, const coords &v, const double &new_path)
    {
index = new_index;
path = new_path;
distance = dist(u, v);
}
};

double res(const int u, const int v, int n, const vector<coords> &vert, const vector<
    vector<int>> &g) {
vector<double> d(n, -10000);
priority_queue<approx> pq;
d[u] = 0;
pq.push(approx(u, vert[u], vert[v], 0));

while (!pq.empty()) {
approx tmp = pq.top();
pq.pop();
if (tmp.index == v)
break;
```

```
46 if (tmp.path > d[tmp.index])
47 continue;
48
49 for (int i = 0; i < g[tmp.index].size(); ++i) {
50 int next_vert = g[tmp.index][i];
51 if (d[next_vert] < 0 || (d[tmp.index] + dist(vert[tmp.index], vert[next_vert])) < d[
       next_vert]){
52 d[next_vert] = d[tmp.index] + dist(vert[tmp.index], vert[next_vert]);
53 pq.push(approx(next_vert, vert[next_vert], vert[v], d[next_vert]));
54 }
55 }
56 }
57 return d[v];
58 }
59
60 int main()
61 {
62 int n, m;
63 cin >> n >> m;
64 vector<coords> vert(n);
65 for (auto &i : vert)
66 cin >> i.x >> i.y;
67 vector<vector<int>> g(n);
68
69 for (int i = 0; i < m; ++i) {
70 int u, v;
71 cin >> u >> v;
72 g[u - 1].push_back(v - 1);
73 g[v - 1].push_back(u - 1);
74 }
75 int q;
76 cin >> q;
77
78 while (q) {
79 int u, v;
80 cin >> u >> v;
81 double solve = res(u - 1, v - 1, n, vert, g);
82 if (solve > 0)
83 cout << fixed << setprecision(6) << solve << '\n';
84 else
85 cout << "-1\n";
86 --q;
87 }
88 return 0;
89 }
```

3 Выводы

Выполнив курсовой проект я изучил эвристический поиск в графе. Порядок обхода вершин определяется эвристической функцией. Эта функция - сумма двух других: функции стоимости достижения рассматриваемой вершины (x) из начальной (обычно обозначается как $g(x)$ и может быть как эвристической, так и нет), и функции эвристической оценки расстояния от рассматриваемой вершины к конечной (обозначается как $h(x)$). Функция $h(x)$ должна быть допустимой эвристической оценкой, то есть не должна переоценивать расстояния к целевой вершине. Например, для задачи маршрутизации $h(x)$ может представлять собой расстояние до цели по прямой линии, так как это физически наименьшее возможное расстояние между двумя точками. Итак, A* проходит наименьшее количество вершин графа среди допустимых алгоритмов, использующих такую же точную (или менее точную) эвристику.