Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Курсовой проект по курсу «Дискретный анализ»

Студент: А. А. Почечура
Преподаватель: С. А. Сорокин
Группа: М8О-306Б

Дата:
Оценка:
Подпись:

Москва, 2022

Курсовой проект: базовый вариант

Задача: Реализуйте алгоритм A^* для графа на решетке.

1 Описание

Требуется реализовать алгоритм A^*, который позволяет найти кратчайший путь между двумя вершинами графа. В моём варианте граф задан в виде решётки, поэтому в данном случае будет осуществляться поиск кратчайшего пути между двумя заданными точками поля. Данный алгоритм представляет из себя подсчёт для каждой посещённой точки функции $f(u) = g(u) + h(u)$. Переход осуществляется в ту точку, у которой значение функции $f(u)$ наименьшее среди всех рассмотренных на данный момент точек. Функция $g(u)$ показывает наименьшую стоимость пути в вершину u из стартовой вершины, а функция $h(u)$ — эвристическое приближение стоимости пути от u до конечной цели. Для моей эвристики функция $h(u)$ представляет из себя манхэттенское расстояние: $h(v) = |v.xgoal.x| + |v.ygoal.y|$.

2 Исходный код

Все точки будут храниться с помощью структуры *point*, которая содержит для каждой точки координаты, функцию $f(u)$ и $g(u)$. В моём варианте вместо *priority queue* можно использовать *deque*. Изначально в дек добавляются все точки, в которые мы можем попасть из начальной. Для них просчитываются $f(u)$ и $g(u)$. В деке элементы всегда находятся в порядке неубывания по значению функции $f(u)$. Затем мы берём верхний элемент дека (назовём её текущей точкой) и смотрим, в какие точки из текущей мы можем попасть. Если точка ещё не использована (это проверяется с помощью массива *used*) и доступна, мы считаем для неё значения $f(u)$ и $g(u)$. В зависимости от значения $f(u)$ определяется, добавится данная точка в голову дека или в хвост. Алгоритм сразу же завершает работу в момент, когда мы попадаем в конечную точку из запроса. Ответом будем являться функция $g(u)$, посчитанная для конечной точки.

```
1  #include <iostream>
2  #include <vector>
3  #include <queue>
4
5  using namespace std;
6
7  struct point {
8    int x;
9    int y;
10   int func_g;
11   int func_f;
12 };
13
14 int func_f(point a, int x2, int y2) {
15   return abs(a.x - x2) + abs(a.y - y2) + a.func_g;
16 }
17
18 int c1[4] = { -1, 1, 0, 0 };
19 int c2[4] = { 0, 0, -1, 1 };
20
21 int find(int x1, int y1, int x2, int y2, vector<vector<char>>& v) {
22   deque<point> d;
23   vector<vector<bool>> used(v.size(), vector<bool>(v[0].size()));
24   point a;
25   a.func_g = 1;
26   for (int i = 0;i < 4;i++) {
27     a.x = x1 + c1[i];
28     a.y = y1 + c2[i];
29     if (a.x >= 0 && a.x <= v.size() - 1 && a.y >= 0 && a.y <= v[0].size() - 1) {
30       if (d.empty() || func_f(a, x2, y2) <= (d.front()).func_f) {
31         d.push_front({ a.x, a.y, 1, func_f(a, x2, y2) });
32       }
```

```
33      else {
34        d.push_back({ a.x, a.y, 1, func_f(a, x2, y2) });
35      }
36    }
37  }
38  int ans = -1;
39  int x, y;
40  while (!d.empty()) {
41    a = d.front();
42    d.pop_front();
43    if (v[a.x][a.y] == '#' || used[a.x][a.y]) {
44      continue;
45    }
46    if (a.x == x2 && a.y == y2) {
47      ans = a.func_g;
48      break;
49    }
50    used[a.x][a.y] = true;
51    x = a.x;
52    y = a.y;
53    for (int i = 0;i < 4;i++) {
54      a.x = x + c1[i];
55      a.y = y + c2[i];
56      if (a.x >= 0 && a.x <= v.size() - 1 && a.y >= 0 && a.y <= v[0].size() - 1 && !
            used[a.x][a.y]) {
57        if (d.empty() || func_f(a, x2, y2) <= (d.front()).func_f) {
58          d.push_front({ a.x, a.y, a.func_g + 1, func_f(a, x2, y2) });
59        }
60        else {
61          d.push_back({ a.x, a.y, a.func_g + 1, func_f(a, x2, y2) });
62        }
63      }
64    }
65  }
66  return ans;
67 }
68
69 int main() {
70  ios::sync_with_stdio(false);
71  cin.tie(nullptr); cout.tie(nullptr);
72  int n, m;
73  cin >> n >> m;
74  vector<vector<char>> v(n, vector<char>(m));
75  for (int i = 0;i < n;i++) {
76    for (int j = 0;j < m;j++) {
77      cin >> v[i][j];
78    }
79  }
80  int q;
```

```
81    cin >> q;
82    int x1, y1, x2, y2;
83    while (q--) {
84      cin >> x1 >> y1 >> x2 >> y2;
85      if (x1 == x2 && y1 == y2) {
86        cout << 0 << '\n';
87      }
88      else {
89        x1--;
90        x2--;
91        y1--;
92        y2--;
93        cout << find(x1, y1, x2, y2, v) << '\n';
94      }
95    }
96  }
```

3 Консоль

```
root@DESKTOP-5HM2HTK:~# cat <test
5 5
#....
.#.#.
.#.##
.#...
...##
5
2 1 1 2
1 2 2 1
3 1 2 5
4 5 2 1
4 5 1 4
root@DESKTOP-5HM2HTK:~# g++ kp.cpp
root@DESKTOP-5HM2HTK:~# ./a.out <test
10
10
11
8
6
root@DESKTOP-5HM2HTK:~#
```

4 Тест производительности

```
root@DESKTOP-5HM2HTK:~# g++ generator5.cpp
root@DESKTOP-5HM2HTK:~# ./a.out tests
root@DESKTOP-5HM2HTK:~# g++ benchmark.cpp
root@DESKTOP-5HM2HTK:~# ./a.out <tests/1.t
Size of field is: 1000 X 1000
Number of requests is: 5
A star time: 1370ms
BFS time: 3458ms
root@DESKTOP-5HM2HTK:~#
```

Как видно, алгоритм A^* работает почти в три раза быстрее, чем стандратный обход в ширину на поле размером 1000×1000 . И это всего лишь на 5-ти тестах. Такое сильное ускорение получается из-за того, что в алгоритме A^* используется функция $f(u)$, которая позволяет сразу найти направление, в которую следует двигаться. Если на пути не встретиться значительных преград, то данный алгоритм даёт огромное преимущество перед обходом в ширину.

5 Выводы

Выполнив курсовой проект по курсу «Дискретный анализ», я ознакомился с новым алгоритмом поиска кратчайшего пути в графе. Также в ходе работы я познакомился с новым алгоритмом обхода в ширину, который называется 0-1BFS. Данные алгоритмы было интересно и полезно изучить для решения будущих задач.

Список литературы

[1] *Алгоритм A* — Итмо-вики.*
URL: `https://neerc.ifmo.ru/wiki/index.php?title=Алгоритм_A*` (дата обращения: 8.12.2022).

[2] *Обход в ширину — Итмо-вики.*
URL: `https://neerc.ifmo.ru/wiki/index.php?title=Обход_в_ширину` (дата обращения: 8.12.2022).