Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №9 по курсу «Дискретный анализ»

Студент: К. М. Воронов
Преподаватель: А. А. Кухтичев
Группа: М8О-207Б-19

Дата:
Оценка:
Подпись:

Москва, 2021

Лабораторная работа №9

Задача: Разработать программу на языке C или C++, реализующую указанный алгоритм согласно заданию:

Задан взвешенный ориентированный граф, состоящий из n вершин и m ребер. Вершины пронумерованы целыми числами от 1 до n. Необходимо найти длины кратчайших путей между всеми парами вершин при помощи алгоритма Джонсона. Длина пути равна сумме весов ребер на этом пути. Обратите внимание, что в данном варианте веса ребер могут быть отрицательными, поскольку алгоритм умеет с ними работать. Граф не содержит петель и кратных ребер.

Формат входных данных
В первой строке заданы $1 \leq n \leq 2000$, $1 \leq m \leq 4000$. В следующих m строках записаны ребра. Каждая строка содержит три числа – номера вершин, соединенных ребром, и вес данного ребра. Вес ребра – целое число от -10^9 до 10^9.

Формат результата
Если граф содержит цикл отрицательного веса, следует вывести строку "Negative cycle"(без кавычек). В противном случае следует выести матрицу из n строк и n столбцов, где j-е число в i-й строке равно длине кратчайшего пути из вершины i в вершину j. Если такого пути не существует, на соответствующей позиции должно стоять слово "inf"(без кавычек). Элементы матрицы в одной строке разделяются пробелом.

1 Описание

Идея этого алгоритма сформулирована в [1]: "На основе данного орграфа сформировать новый орграф таким образом, чтобы в нём не было отрицательных дуг, а все кратчайшие пути были такими же. Пропустить полученный орграф через алгоритм Дейкстры и на выходе получить верный ответ"

Для того, чтобы убрать отрицательные дуги в орграфе, надо к весу каждого ребра прибавить некую функцию H(x) и, как говорилось выше, пропустить граф через алгоритм Дейкстры. После нахождения кратчайших путей, нужно вычесть эту функцию для каждого ребра, чтобы получить пути исходного графа. Для вычисления этой функции нужно воспользоваться алгоритмом Форда-Беллмана. Создадим вершину с ребрами к каждой другой вершине и весами на них 0. Посчитав из неё кратчайшие пути до других вершин, мы найдем H(x) и посчитаем веса по формуле: $w(x, y) = w(x, y) + H(x) - H(y)$.

Для графа с n вершинами алгоритм Форда-Беллмана работает следующим образом: мы $m \leq n$ раз проходимся по всем вершинами графа, смотрим на её соседей и пытаемся улучшить ответ, то есть найти наименьший путь. В начале путь до вершины, от которой мы ищем, равен 0, а до остальных из неё - бесконечность. Если при $n + 1$ попытке что-то меняется - значит граф имеет отрицательный цикл.

В алгоритме Дейкстры, мы будем смотреть только ребра той вершины, до которой сейчас найденный вес минимальный, и больше её трогать не будем. Неотрицательные веса рёбер гарантируют, что минимальный взятый сейчас вес для вершины является уже ответом для неё, потому что иначе существовала другая вершина с меньшим весом, из которой мы могли бы попасть в эту.

2 Исходный код

Для хранения графа я использую две структуры: TEdge(вершина 1(From), вершина 2(To), вес(Weight)), TList(вершина(To), вес(Weight)). Первую структуру удобно использовать в алгоритме Форда-Беллмана, а вторую - Дейкстры. Также для алгоритма Дейкстры удобно использовать очередь с приорететом, так как она хранит наверху самый большой элемент. Создав структуру TPriority(включающую в себя расстояние(Distance) и вершину(Vertex)), напишем для неё оператор $<$ как $>$, чтобы наверху был наименьший элемент.

```
1  #include<iostream>
2  #include<vector>
3  #include<queue>
4
5  using namespace std;
6  const long long INF = 92233720368547758;
7
8  struct TEdge {
9      int From;
10     int To;
11     long long Weight;
12 };
13 struct TList {
14     int To;
15     long long Weight;
16 };
17 struct TPriority{
18     long long Distance;
19     int Vertex;
20 };
21
22 bool operator<(const TPriority &l, const TPriority &r) {
23     return l.Distance > r.Distance;
24 }
25
26 bool FordBellman(vector<TEdge>& edges, vector<long long> &distances, int n) {
27     int i;
28     bool changed = false;
29     for (i = 0; i < n; ++i) {
30         changed = false;
31         for (long unsigned int j = 0; j < edges.size(); ++j) {
32             if ((distances[edges[j].From] != INF) && (distances[edges[j].To] >
                   distances[edges[j].From] + edges[j].Weight)) {
33                 changed = true;
34                 distances[edges[j].To] = distances[edges[j].From] + edges[j].Weight;
35             }
36         }
37         if (!changed) {
```

```
38              break;
39          }
40      }
41      if (i == n) {
42          changed = false;
43          for (long unsigned int j = 0; j < edges.size(); ++j) {
44              if ((distances[edges[j].From] != INF) && (distances[edges[j].To] >
                    distances[edges[j].From] + edges[j].Weight)) {
45                  changed = true;
46                  distances[edges[j].To] = distances[edges[j].From] + edges[j].Weight;
47              }
48          }
49      }
50      return !changed;
51  }
52
53  void H(vector<vector<TList>>& edges, vector<long long> &distances) {
54      for (int i = 0; i < edges.size(); ++i) {
55          for(int j = 0; j < edges[i].size(); ++j) {
56              edges[i][j].Weight = edges[i][j].Weight + distances[i] - distances[edges[i
                    ][j].To];
57          }
58      }
59  }
60
61  vector<long long> Dijkstra(vector<vector<TList>>& edges, vector<long long> &distances,
        int s) {
62      priority_queue<TPriority> pq;
63      vector<bool> relaxed(distances.size());
64      pq.push(TPriority{0, s});
65      while (!pq.empty()) {
66          TPriority p = pq.top();
67          pq.pop();
68          if (relaxed[p.Vertex - 1]) {
69              continue;
70          }
71          relaxed[p.Vertex - 1] = 1;
72          for (int i = 0; i < edges[p.Vertex].size(); ++i) {
73              if((distances[p.Vertex - 1] != INF) && (distances[edges[p.Vertex][i].To -
                    1] > distances[p.Vertex - 1] + edges[p.Vertex][i].Weight)) {
74                  distances[edges[p.Vertex][i].To - 1] = distances[p.Vertex - 1] + edges[p
                        .Vertex][i].Weight;
75                  pq.push(TPriority{distances[edges[p.Vertex][i].To - 1], edges[p.Vertex][
                        i].To});
76              }
77          }
78      }
79      return distances;
80  }
```

```
81
82  int main() {
83      int n, m;
84      cin >> n >> m;
85      vector<TEdge> edgesS;
86      vector<vector<TList>> edges(n + 1);
87      for (int i = 0; i < m; ++i) {
88          int from, to, weight;
89          cin >> from >> to >> weight;
90          edgesS.push_back({from, to, weight});
91          edges[from].push_back({to, weight});
92      }
93
94      for (int i = 1; i <= n; ++i) {
95          edgesS.push_back({0, i, 0});
96      }
97
98      vector<long long> distances(n + 1, INF);
99      distances[0] = 0;
100     bool flag = FordBellman(edgesS, distances, n + 1);
101
102     if (!flag) {
103         cout << "Negative cycle\n";
104         return 0;
105     }
106
107     H(edges, distances);
108
109     vector<vector<long long>> result(n);
110     vector<long long> distances2(distances.size() - 1);
111
112     for (int i = 0; i < n; ++i) {
113         distances2.assign(distances2.size(), INF);
114         distances2[i] = 0;
115         result[i] = Dijkstra(edges, distances2, i + 1);
116     }
117
118     for (int i = 0; i < result.size(); ++i) {
119         for (int j = 0; j < result[i].size(); ++j) {
120             if (result[i][j] == INF) {
121                 cout << "inf ";
122             } else {
123                 cout << result[i][j] + distances[j + 1] - distances[i + 1]<< " ";
124             }
125         }
126         cout << endl;
127     }
128 }
```

3 Консоль

```
kirill@kirill-G3-3779:~/DA/lab9$ cat test.txt
4
8
1 2 -2
2 4 6
1 4 5
4 1 -1
1 3 7
3 4 -4
2 3 8
3 2 3
kirill@kirill-G3-3779:~/DA/lab9$ ./s* <test.txt
0 -2 6 2
3 0 8 4
-5 -7 0 -4
-1 -3 5 0
```

4 Тест производительности

Сравним время работы алгоритма Джонсона с алгоритмом Форда-Беллмана на 10^4 вершин и рёбер и $5 * 10^4$ вершин и рёбер.

Умножение

```
kirill@kirill-G3-3779:~/DA/lab9$ ./s* <test.txt
Алгоритм Джонсона: 585.004ms
Алгоритм Форда-Беллмана: 4497.156ms


kirill@kirill-G3-3779:~/DA/lab9$ ./s* <test.txt
Алгоритм Джонсона: 11525.340ms
Алгоритм Форда-Беллмана: 159006.410ms
```

Из-за того, что алгоритм Форда-Беллмана работает за O(n*m), а Дейкстры O(m*logn), алгоритм Джонсона справляется быстрее

5 Выводы

Выполнив девятую лабораторную работу по курсу «Дискретный анализ», я вспомнил, как работать с графами, узнал новый алгоритм - алгоритм Джонсона, а также повторил уже знакомые - Форда-Беллмана и Дейкстры. Работа с графами очень актуальна, например, когда посчитать минимальное расстояние между городами, странами и т.д. Также, углубившись в алгоритмы поиска кратчайших путей в графах, я познакомился с алгоритмом Форда-Уоршелла, основанном на полном переборе через матрицу, и алгоритмом A*, который является улучшенной версией Дейкстры, так как использует эвристику для приближенного вычисления расстояния до конечной точки.

Список литературы

[1] *Алгоритм Джонсона на орграфе с отрицательными дугами.*
URL: `https://habr.com/ru/company/otus/blog/510942/` (дата обращения: 07.06.2021)

[2] *Алгоритм Джонсона.*
URL: `https://ru.wikipedia.org/wiki/Алгоритм_Джонсона` (дата обращения: 09.06.2021).