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: А. О. Знай
Преподаватель: С. А. Михайлова
Группа: М8О-301Б-21

Дата: 02.12.2023
Оценка:
Подпись:

Москва, 2023

Лабораторная работа №9

Задача: 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 и . В следующих m строках записаны ребра. Каждая строка содержит три числа – номера вершин, соединенных ребром, и вес данного ребра. Вес ребра – целое число от -109 до 109.

Формат вывода: Если граф содержит цикл отрицательного веса, следует вывести строку "Negative cycle"(без кавычек). В противном случае следует вывести 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

Алгоритм Джонсона представляет собой комбинацию алгоритмов Белмана-Форда и Дейкстра. Такая комбинация позволяет нам решать задачу по поиску кратчайщих путей в графах с отрицательными ребрами, а при наличии отрицательных циклов - находить их и останавливаться без ошибок.

Суть заключается в том, что мы вначале добавляем вершину к нашему графу и связываем ее с каждой вершиной ребром с весом 0. Проходимся по новому графу алгоритмом Б-Ф для новой вершины. Результат такого прохода - кратчайшее растояние между добавленной вершиной и всеми остальными, он нам нужен чтобы избавиться от отрицательных ребер с помощью формулы: newD = D + BFResult(u) - BFResult(v), - где D - вес ребра из вершины u в вершину v. Теперь мы уверены, что в графе нет отрицательных ребер и мы просто запускаем алгоритм Дейкстры для каждой вершины, находя кратчайшие расстояния до остальных. Затем нам надо пересчитать пути, используем ту же формулу: orig = DeykstraRes[i][j] - BFResult[i] + BFResult[j] Сложность данного алгоритма складывается из двух остальных: Б-Ф - O(VE), Дейсктра (при использовании двоичной кучи) - O(ElogV). Джонсон - O(VElogV)

[1]

Код

```
#include <string>
#include <iostream>
#include <vector>
#include <stack>
#include <memory>
#include <chrono>
#include <map>
#include <cmath>
#include <algorithm>
#include <queue>

using namespace std;

long long inf = INT64_MAX;

class Compare {
public:
    bool operator()(pair<int, long long> below, pair<int, long long> above)
    {
        if (below.second > above.second) {
            return true;
        }
        return false;
    }
};

bool BelmanFord(const int N, vector<long long > & belmanFordResult, const vector<
    vector<pair<int, long long> > > adjacency ) {

    vector<vector<long long> > distance(N + 2, vector<long long>(N + 1, inf));
    distance[0][0] = 0;

    int i;
    bool isSame;
    for (i = 1; i <= N; i++) {
        isSame = true;
        for (int v = 0; v <= N; v++) {
            if (distance[i - 1][v] == inf) {
                continue;
            }
            for (int j = 0; j < adjacency[v].size(); j++) {
                int u = adjacency[v][j].first;
                if (distance[i - 1][v] + adjacency[v][j].second < distance[i][u]) {
                    distance[i][u] = distance[i - 1][v] + adjacency[v][j].second;

                }
            }
        }
```

```
48          for (int u = 0; u <= N; u++) {
49              isSame = isSame && distance[i][u] == distance[i - 1][u];
50          }
51          if (isSame) {
52              break;
53          }
54      }
55      if (!isSame) {
56          for (int v = 0; v <= N; v++) {
57              if (distance[N][v] == inf) {
58                  continue;
59              }
60              for (int j = 0; j < adjacency[v].size(); j++) {
61                  int u = adjacency[v][j].first;
62                  distance[N + 1][u] = distance[N][u];
63                  if (distance[N][v] + adjacency[v][j].second < distance[N + 1][u]) {
64                      return true;
65                  }
66              }
67          }
68      }
69
70      for (int j = 0; j <= N; j++) {
71          belmanFordResult[j] = distance[i][j];
72      }
73      return false;
74  }
75
76  vector<long long> Deikstra(const int S, const int N, const vector<vector<pair<int,
        long long> > > adjacency) {
77      priority_queue<pair<int, long long>, vector<pair<int, long long> >, Compare> minq;
78      minq.push(make_pair(S, 0));
79
80      vector<bool > visited(N + 1, false);
81      vector<long long > distance(N + 1, inf);
82      distance[S] = 0;
83
84      while (!minq.empty()) {
85          int curV = minq.top().first;
86          long long curDist = minq.top().second;
87
88          if (visited[curV]) {
89              minq.pop();
90              continue;
91          }
92
93          for (int i = 0; i < adjacency[curV].size(); i++) {
94              if (distance[curV] + adjacency[curV][i].second < distance[adjacency[curV][i
                    ].first]) {
```

```
95                distance[adjacency[curV][i].first] = distance[curV] + adjacency[curV][i
                      ].second;
96                minq.push(make_pair(adjacency[curV][i].first, distance[curV] + adjacency
                      [curV][i].second));
97            }
98        }
99
100       visited[curV] = true;
101       minq.pop();
102   }
103   return distance;
104}
105
106int main() {
107   ios_base::sync_with_stdio(false);
108   cin.tie(nullptr);
109   cout.tie(nullptr);
110
111   int N, M;
112   cin >> N >> M;
113
114   vector<vector<pair<int, long long> > > adjacency(N + 1);
115
116   for (int i = 0; i <= N; i++) {
117       adjacency[0].push_back(make_pair(i, 0));
118   }
119
120   bool hasNegativeNums = false;
121   for (int i = 1; i <= M; i++) {
122       int u, v, w;
123       cin >> u >> v >> w;
124       if (w < 0) {
125           hasNegativeNums = true;
126       }
127       adjacency[u].push_back(make_pair(v, w));
128   }
129
130   vector<long long > belmanFordResult(N + 1, 0);
131
132   if (hasNegativeNums) {
133       bool hasNegativeCycle = BelmanFord(N, belmanFordResult, adjacency);
134       if (hasNegativeCycle) {
135           cout << "Negative cycle" << '\n';
136           return 0;
137       }
138       else {
139           for (int i = 1; i <= N; i++) {
140               for (int j = 0; j < adjacency[i].size(); j++) {
141                   adjacency[i][j].second = adjacency[i][j].second + belmanFordResult[i
```

```
                        ] - belmanFordResult[adjacency[i][j].first];
                }
            }
        }
    }

    for (int i = 1; i <= N; i++) {
        vector<long long> deikstraRes = Deikstra(i, N, adjacency);
        for (int j = 1; j <= N; j++) {
            if ( deikstraRes[j] == inf) {
                if (j == N) {
                    cout << "inf";
                }
                else {
                    cout << "inf" << ' ';
                }
            }
            else {
                if (j == N) {
                    cout << deikstraRes[j] - belmanFordResult[i] + belmanFordResult[j];
                }
                else {
                    cout << deikstraRes[j] - belmanFordResult[i] + belmanFordResult[j]
                        << ' ';
                }
            }
        }
        cout << '\n';
    }

}
```

2 Консоль

```
artemznaj@MacBook-Air-Artem lab9 % ./a.out
5 4
1 2 -1
2 3 2
1 4 -5
3 1 1
0 -1 1 -5 inf
3 0 2 -2 inf
1 0 0 -4 inf
inf inf inf 0 inf
inf inf inf inf 0
artemznaj@MacBook-Air-Artem lab9 % ./a.out
3 3
1 2 -1
2 3 1
3 1 -2
Negative cycle
```

3 Тест производительности

Для изучения производительности сравним время работы программы на тестах разных размеров с m = $10^3, 10^5$

```
1. $10^3$
artemznaj@MacBook-Air-Artem lab9 % ./a* <tests/1.in
10215ms
Process finished with exit code 0
2.$10^5$
artemznaj@MacBook-Air-Artem lab9 % ./a* <tests/2.in
328485ms
Process finished with exit code 0
```

4 Выводы

Выполнив девятую лабораторную работу по курсу «Дискретный анализ», я научился решать задачу по поиску кратчайщих путей между всеми парами вершин с помощью алгоритма Джонсона. Мне понравился данный алгоритм, потому что он использует два других базовых алгоритма и идею о пересчете веса, это в совокупности дает нам возможность решать ту жу задачу, но теперь и на графах с отрицательными весами, при этом быстрее чем просто запуском Б-Ф для всех вершин.

Список литературы

[1] Жадный алгоритм https://ru.wikipedia.org/

[2] Лекции Н.К.Макарова, Московский авиационный институт.