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: Д. С. Пивницкий
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б-19

Дата: 01.01.2021
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 $1n$. 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 $Negativecycle$. 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. Элементы матрицы в одной строке разделяются пробелом.

1 Описание

Для написания алгоритма Джонсона нам понадобится написать еще два алгоритма, алгоритм Беллмана-Форда и Дейкстры. Сам алгоритм таков, если алгоритм Беллмана-Форда возвращает ложь, то у нас отрицательный цикл, иначе используем алгоритм Дейкстры, если нет отрицательных дуг, иначе формируем новый орграф с такими же кратчайшими путями, но без отрицательных дуг и снова запускаем алгоритм Дейкстры.

2 Исходный код

Код: main.cpp

```
#include "johnson.hpp"
#include "structures.hpp"

int main()
{
size_t n = 0;
size_t m = 0;
std::cin >> n >> m;
TGraph gr(n, m);
TMatrix dist(n, std::vector<int64_t>(n, INT64_MAX));
for(size_t i = 0; i < n; ++i)
dist[i][i] = 0;
size_t from = 0;
size_t to = 0;
int64_t weight = 0;
for(size_t k = 0; k < m; ++k)
{
std::cin >> from >> to >> weight;
gr.edges.push_back(TEdge{from - 1, to - 1, weight});
}
if(Johnson(gr, dist))
{
for(size_t i = 0; i < n; ++i)
{
for(size_t j = 0; j < n; ++j)
{
if(dist[i][j] == INT64_MAX)
std::cout << "inf";
else
std::cout << dist[i][j];
if(j != n - 1)
std::cout << ' ';
}
std::cout << "\n";
}
}
return 0;
}
```

Код: johnson.cpp

```cpp
#include "johnson.hpp"
#include "structures.hpp"

bool operator<(TEdge const& p1, TEdge const& p2) { return p1.weigth > p2.weigth; }

void Deikstra(TMatrix const& gr, size_t const& node, TMatrix& dist, size_t const& n)
{
dist[node][node] = 0;
std::priority_queue<TEdge> pq;
TEdge t = {node, 0, 0};
pq.push(t);
while(!pq.empty())
{
TEdge s = pq.top();
pq.pop();
for(size_t i = 0; i < n; ++i)
{
if(dist[node][i] - dist[node][s.from] > gr[s.from][i])
{
dist[node][i] = dist[node][s.from] + gr[s.from][i];
TEdge p = { i, 0, dist[node][i] };
pq.push(p);
}
}
}
}

bool BellmanFord(TGraph const& gr, size_t const& node, TMatrix& dist)
{
dist[node][node] = 0;
for(size_t j = 0; j < gr.v - 1; ++j)
{
for(auto& i: gr.edges)
if(dist[node][i.from] != INT64_MAX && dist[node][i.to] > dist[node][i.from] + i.weigth
    )
dist[node][i.to] = (dist[node][i.from] + i.weigth);
}
for(auto& i: gr.edges)
if(dist[node][i.from] != INT64_MAX && dist[node][i.to] > dist[node][i.from] + i.weigth
    )
return false;
return true;
}

bool Johnson(TGraph const& gr, TMatrix& dist)
{
TGraph new_gr;
new_gr.v = gr.v + 1;
```

```
47 new_gr.e = gr.e + gr.v;
48 new_gr.edges = gr.edges;
49 for(size_t i = 0; i < gr.v; ++i)
50 new_gr.edges.push_back(TEdge{gr.v, i, 0});
51 TMatrix new_dist(1, std::vector<int64_t>(new_gr.v, 0));
52 if(!BellmanFord(new_gr, 0, new_dist))
53 {
54 std::cout << "Negative cycle\n";
55 return false;
56 }
57 TMatrix graph(gr.v, std::vector<int64_t>(gr.v, INT64_MAX));
58 for(size_t i = 0; i < gr.v; ++i)
59 graph[i][i] = 0;
60 for(auto& i: gr.edges)
61 graph[i.from][i.to] = i.weigth + new_dist[0][i.from] - new_dist[0][i.to];
62 for(size_t i = 0; i < gr.v; ++i)
63 Deikstra(graph, i, dist, gr.v);
64 for(size_t i = 0; i < gr.v; ++i)
65 for(size_t j = 0; j < gr.v; ++j)
66 if(dist[i][j] != INT64_MAX)
67 dist[i][j] = dist[i][j] + new_dist[0][j] - new_dist[0][i];
68 return true;
69 }
```

3 Консоль

```
(py37) ~ /DA_labs/lab8$ make
g++ -g -O2 -pedantic -std=c++17 -Wall -Wextra -Werror main.cpp -o solution
(py37) ~ /DA_labs/lab8$ cat test.txt
5 4
1 2 -1
2 3 2
1 4 -5
3 1 1
(py37) ~ /DA_labs/lab8$ ./solution <test.txt
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
```

4 Тест производительности

Тест состоит из ввода графа и запуска алгоритма Джонсона 50000, 100000 и 200000 раз.

Моя реализация:

```
(py37) ~ /DA_labs/lab7$ make
g++ -g -O2 -pedantic -std=c++17 -Wall -Wextra -Werror main.cpp -o solution
(py37) ~ /DA_labs/lab8$ make bench
g++ -g -O2 -pedantic -std=c++17 -Wall -Wextra -Werror benchmark.cpp -o benchmark
(py37) ~ /DA_labs/lab8$ ./benchmark
Enter graph:
5 4
1 2 -1
2 3 2
1 4 -5
3 1 1
Enter test count:
50000
Time for Johnson on 50000 tests: 3 seconds
```

Алгоритм работает за $O(n * m + n^2 * lnn)$, в нашем случае $n = m -> O(n^2 * lnn)$, что лучше чем алгоритм Флойда-Уоршела, имеющего сложность $O(n^3)$ и явно лучше наивного перебора за $O(n^4)$.

5 Выводы

Выполнив девятую лабораторную работу по курсу «Дискретный анализ», я познакомился с алгоритмами на графах, такими как алгоритм Дейкстры, Беллмана-Форда и Джонсона, узнал в каких случаях нужно использовать алгоритм Джонсона и почему нельзя использовать Дейкстру(при наличии отрицательных ребер), для чего в алгоритме Джонсона нужен алгоритм Беллмана-Форда(для нахождения отрицательного цикла).

Список литературы

[1] Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн. *Алгоритмы: построение и анализ, 2-е издание.* — Издательский дом «Вильямс», 2007. Перевод с английского: И. В. Красиков, Н. А. Орехова, В. Н. Романов. — 1296 с. (ISBN 5-8459-0857-4 (рус.))