ыСанкт-Петербургский государственный университет телекоммуникаций

им. проф. М.А. Бонч-Бруевича

**Алгоритмы и структуры данных**

Отчёт по лабораторной работе №5  
«Алгоритмы поиска кратчайшего пути в графе**»**

Группа: ИКПИ-33

Студенты: Коньков М. Д.

Семенихин А. Р.

Санкт-Петербург

2025

# Цель работы

Целью работы является ознакомление с алгоритмами поиска кратчайших путей в графе и методикой оценки эффективности алгоритмов.

# Описание алгоритмов

Алгоритм Беллмана-Форда. Как и в других задачах динамического программирования, алгоритм вычисляет кратчайшие пути снизу вверх. Сначала он вычисляет самые короткие расстояния, то есть пути длиной не более, чем в одно ребро. Затем он вычисляет кратчайшие пути длиной не более двух ребер и так далее. После i-й итерации внешнего цикла вычисляются кратчайшие пути длиной не более i ребер. В любом простом пути может быть максимум |V|-1 ребер, поэтому внешний цикл выполняется именно |V|-1 раз. Идея заключается в том, что если мы вычислили кратчайший путь с не более чем i ребрами, то итерация по всем ребрам гарантирует получение кратчайшего пути с не более чем i + 1 ребрами

Сложность:

Алгоритм Флойда-Уоршалла.

Случай 1. Элемент k не входит в кратчайший путь pij, то есть от добавления дополнительной вершины мы ничего не выиграли и ничего не изменили, а значит стоимость кратчайшего пути dkij не изменился, соответственно

dkij = dk-1ij — просто перенимаем значение до увеличения k.

Случай 2. Элемент k входит в кратчайший путь pij, то есть после добавления новой вершины в можество разрешенных, кратчайший путь изменился и проходит теперь через вершину vk. Какую стоимость получит новый путь?  
  
Новый кратчайший путь разбит вершиной vk на pik и pkj, используем первое свойство, согласно ему, pik и pkj также кратчайшие пути от vi до vk и от vk до vj соответственно. Значит  
  
dkij = dkik + dkkj  
  
А так как в этих путях k либо конечный, либо начальный узел, то он не входит в множество промежуточных, соответственно его из него можно удалить:

dkij = dk-1ik + dk-1kj

Посмотрим на значение стоимости пути dkij в обоих случаях — оно в обоих случаях складывается из значений d для k-1, а значит имея начальные (k=0) значения для d, мы сможем расчитать d для всех последующих значений k. А значения d для k=0 мы знаем, это вес/стоимость рёбер графа, то есть соединений без промужуточных узлов.

При k=n (n — количество вершин) мы получим оптимальные значения d для всех пар вершин.

При увеличении с k-1 до k, какое значение мы сохраним для dkik? Минимумом значений случая 1 и 2, то есть смотрим дешевле ли старый путь или путь с добавлением дополнительной вершины.

Сложность:

# Пошаговое сравнение

**Алгоритм Беллмана-Форда:**

Пошаговое сравнение алгоритмов производится на графе, заданном матрицей смежности.

В таблице 1 отображена пошаговая работа алгоритма Беллмана-Форда.

*Таблица 1. Пошаговая работа алгоритма Беллмана-Форда.*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 |
| 0 | 0 | 5 | ∞ | 10 |
| 1 | ∞ | 0 | 3 | ∞ |
| 2 | ∞ | ∞ | 0 | 1 |
| 3 | ∞ | ∞ | ∞ | 0 |

Шаг 1: Исходная матрица кратчайших путей для вершины 0.

dist = [∞, ∞, ∞, ∞]

dist[0] = 0

Шаг 2: Анализируем первую строку матрицы смежности; добавляем в матрицу путей веса из ненулевых ячеек первой строки.

dist = [0, 5, ∞, 10]

Шаг 3: Путей длины 1 больше нет; т. к. ячейка 0-2 нулевая, спускаемся на строку ниже, отмечаем, что ячейка 1-2 ненулевая, следовательно существует путь 0-1-2; складываем веса рёбер на протяжении пути и добавляем в матрицу.

dist = [0, 5, 8, 10]

Шаг 4: Аналогично находим пути длиной более 2. Заметим, что ячейка 1-3 пустая, спускаемся на строку ниже, ячейка 2-3 ненулевая, следовательно существует путь 0-1-2-3, сумма весов рёбер – 9, что меньше значения 10 в матрице путей; заменяем значение.

dist = [0, 5, 8, 9]

**Алгоритм Флойда-Уоршалла:**

В ходе алгоритма последовательно проходим по строкам матрицы, заменяя значения длин путей, равные ∞, на значения, равные сумме значений в ячейках, соответствующих вершинам в вычисленном пути. Если при прохождении строки найдена ячейка со значением ∞, рассматриваем, есть ли в ячейках с номерами, не равными исходной и конечной точке пути, значение в ячейке, соответствующей конечной точке маршрута. Если такое значение существует, то составляется путь вида “исходная точка маршрута + новая точка + конечная точка маршрута”; в ячейку, значение которой равнялось ∞, записываем сумму значений ячеек всех входящих в маршрут точек.

Шаг 1: k=0, исходная матрица кратчайших путей. Начинаем движение с левого верхнего угла по первой строке слева направо. В таблице 2 отображена пошаговая работа алгоритма Флойда-Уоршалла.

*Таблица 2. Пошаговая работа алгоритма Флойда-Уоршалла.*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 |
| 0 | 0 | 5 | ∞ | 10 |
| 1 | ∞ | 0 | 3 | ∞ |
| 2 | ∞ | ∞ | 0 | 1 |
| 3 | ∞ | ∞ | ∞ | 0 |

Шаг 2: k=1, ячейка 0-2 нулевая, следовательно рассматриваем строки 1 и 3; для строки 1 есть значение в ячейке 1-2, следовательно сумма значений 0-1, 1-2 d = 5 + 3 = 8; добавляем значение 8 в ячейку 0-2.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 |
| 0 | 0 | 5 | 8 | 10 |
| 1 | ∞ | 0 | 3 | ∞ |
| 2 | ∞ | ∞ | 0 | 1 |
| 3 | ∞ | ∞ | ∞ | 0 |

Шаг 3: k=2, ячейка 1-3 нулевая, следовательно рассмотрим единственную подходящую строку 2, в которой есть значение в столбце 3, сумма значений ячеек 1-2, 2-3: d = 3 + 1 = 4; добавляем значение 4 для ячейки 1-3.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 |
| 0 | 0 | 5 | 8 | 10 |
| 1 | ∞ | 0 | 3 | 4 |
| 2 | ∞ | ∞ | 0 | 1 |
| 3 | ∞ | ∞ | ∞ | 0 |

Шаг 4: k=3. В строках 0, 1, 2, 3 выше главной диагонали нет ячеек со значением ∞. Останов.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 |
| 0 | 0 | 5 | 8 | 10 |
| 1 | ∞ | 0 | 3 | 4 |
| 2 | ∞ | ∞ | 0 | 1 |
| 3 | ∞ | ∞ | ∞ | 0 |

По результатам пошагового сравнения, алгоритм Флойда-Уоршалла и алгоритм Беллмана-Форда на графе из 4 вершин выполняются за одинаковое число шагов.

# Описание программы

Программа для выполнения настоящей работы обеспечивает выполнение процедур поиска кратчайшего пути в фоновом режиме и подсчёт временных затрат на работу каждого из алгоритмов. Имеется возможность указать число итераций, в течение которых будут сняты показания времени.

При запуске программы следует ввести количество итераций, количество вершин и ребёр в графе согласно инструкциям. По завершении работы процедур построения кратчайших путей на экран будет выведено среднее время работы обоих алгоритмов. Для завершения работы программы следует ввести символ q, для продолжения работы — любой другой символ.

Программа создана с использованием языка программирования C++ в среде программирования Visual Studio Code для работы в системе под управлением ОС Windows.

# Полученные результаты

Результаты оценки временной сложности алгоритмов (в микросекундах) представлены в таблице 3 и отображены на рисунке 1. Оценки произведены на разреженных графах.

*Таблица 3. Зависимость времени работы алгоритмов одномерной оптимизации от заданной точности вычислений (в миллисекундах).*

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Количество вершин / Алгоритм | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 |
| Алгоритм Беллмана-Форда | 0,01 | 0,06 | 0,21 | 0,46 | 0,97 | 1,28 | 2,02 | 2,95 | 5,36 | 6,99 |
| Алгоритм Флойда-Уоршалла | 0,03 | 0,18 | 0,51 | 1,17 | 2,38 | 3,26 | 5,11 | 7,56 | 13,76 | 24,44 |

*Рис. 1. Зависимость времени работы алгоритмов нахождения кратчайшего пути от количества вершин в графе.*

Данные, отображенные в таблице и на рисунке, показывают, что на разреженных графах алгоритм алгоритм Беллмана-Форда работает в среднем быстрее, чем алгоритм Флойда-Уоршалла.

# Вывод

В ходе проведённой лабораторной работы получены следующие результаты:

1. Изучены алгоритмы поиска кратчайшего пути Беллмана-Форда и Флойда-Уоршалла.
2. Создана программа, позволяющая выполнить оценку временной сложности алгоритмов.
3. Проведена оценка временной сложности алгоритмов Беллмана-Форда и Флойда-Уоршалла на разреженных графах.
4. По результатам оценки, на разреженных графах алгоритм Беллмана-Форда работает в среднем быстрее, чем алгоритм Флойда-Уоршалла.

# Исходный код программы

#include <iostream>

#include <vector>

#include <time.h>

#include <algorithm>

#include <limits>

#include <queue>

#define MAX\_W 100

const int inf = std::numeric\_limits<int>::max();

std::vector<std::vector<int>> genGraph(int n, int v)

{

std::vector<std::vector<int>> matrixGraph(n, std::vector<int>(n, inf));

std::vector<bool> used(n, false);

std::queue<int> next;

next.push(0);

int unused\_n = n - 1;

int unused\_v = v;

while (unused\_n > 0)

{

int amountOfV = rand() % (v / n) + 1;

int cur\_src = next.front();

next.pop();

for (int i = 0; i < amountOfV && unused\_n > 0; i++)

{

int w = rand() % MAX\_W + 1;

matrixGraph[cur\_src][unused\_n] = w;

matrixGraph[unused\_n][cur\_src] = w;

next.push(unused\_n);

unused\_n--;

unused\_v--;

}

}

while (unused\_v > 0)

{

int src = rand() % n;

int dest = rand() % n;

if (src == dest || matrixGraph[src][dest] != inf)

{

continue;

}

int w = rand() % MAX\_W + 1;

matrixGraph[src][dest] = w;

matrixGraph[dest][src] = w;

unused\_v--;

}

return matrixGraph;

}

std::vector<int> leader;

int getLeader(int x)

{

if (x == leader[x])

{

return x;

}

return leader[x] = getLeader(leader[x]);

}

bool unite(int x, int y)

{

x = getLeader(x);

y = getLeader(y);

if (x == y)

{

return false;

}

if (rand() % 2 == 0)

{

std::swap(x, y);

}

leader[x] = y;

return true;

}

std::vector<int> bellman\_ford(const std::vector<std::vector<int>>& graph, int source) {

int n = graph.size();

std::vector<int> dist(n, inf);

dist[source] = 0;

for (int i = 0; i < n - 1; i++) {

for (int u = 0; u < n; u++) {

for (int v = 0; v < n; v++) {

if (graph[u][v] != inf && dist[u] != inf && dist[v] > dist[u] + graph[u][v]) {

dist[v] = dist[u] + graph[u][v];

}

}

}

}

return dist;

}

std::vector<std::vector<int>> floyd\_warshall(const std::vector<std::vector<int>>& graph) {

int n = graph.size();

std::vector<std::vector<int>> dist(n, std::vector<int>(n, inf));

for (int u = 0; u < n; u++) {

for (int v = 0; v < n; v++) {

if (graph[u][v] != inf) {

dist[u][v] = graph[u][v];

}

}

}

for (int k = 0; k < n; k++) {

for (int u = 0; u < n; u++) {

for (int v = 0; v < n; v++) {

if (dist[u][k] != inf && dist[k][v] != inf && dist[u][v] > dist[u][k] + dist[k][v]) {

dist[u][v] = dist[u][k] + dist[k][v];

}

}

}

}

return dist;

}

int main()

{

// {10,12},{20,25},{30,37},{40,50},{50,62},{60,75},{70,87},{80,100},{90,112},{100,125}

srand(static\_cast<unsigned int>(time(NULL)));

char c = ' ';

while (c != 'q')

{

int r = 0;

std::cout << "Iterations:" << std::endl;

std::cin >> r;

int n = 0;

std::cout << "Vertex count:" << std::endl;

std::cin >> n;

int m = 0;

std::cout << "Edges count:" << std::endl;

std::cin >> m;

double time\_f1\_sum = 0, time\_f2\_sum = 0;

for (int k = 0; k < r; k++)

{

std::vector<std::vector<int>> matGraph = genGraph(n, m);

clock\_t start = clock();

bellman\_ford(matGraph, 0);

clock\_t end = clock();

double seconds = (double)(end - start) / CLOCKS\_PER\_SEC;

time\_f1\_sum += seconds;

matGraph = genGraph(n, m);

start = clock();

floyd\_warshall(matGraph);

end = clock();

seconds = (double)(end - start) / CLOCKS\_PER\_SEC;

time\_f2\_sum += seconds;

}

double time\_1 = time\_f1\_sum / (double)r;

std::cout << "The time (bellman\_ford): " << time\_1 << " seconds" << std::endl;

double time\_2 = time\_f2\_sum / (double)r;

std::cout << "The time (floyd-warshall): " << time\_2 << " seconds" << std::endl;

std::cout << "Enter any char to continue; enter 'q' to exit." << std::endl;

std::cin >> c;

}

return 0;

}