МИНОБРНАУКИ РОССИИ
ФГБОУ ВО «СГУ ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»

# АЛГОРИТМ ФЛОЙДА-УОРШЕЛЛА

ЛАБОРАТОРНАЯ РАБОТА

студента 3 курса 331 группы
направления 100501 — Компьютерная безопасность
факультета КНиИТ
Токарева Никиты Сергеевича

Проверил
доцент ______________ А. Н. Гамова

Саратов 2022

# СОДЕРЖАНИЕ

## 1 Описание алгоритма

Этот алгоритм был одновременно опубликован в статьях Роберта Флойда (Robert Floyd) и Стивена Уоршелла (Варшалла) (Stephen Warshall) в 1962 г., по имени которых этот алгоритм и называется в настоящее время. Впрочем, в 1959 г. Бернард Рой (Bernard Roy) опубликовал практически такой же алгоритм, но его публикация осталась незамеченной. Алгоритм Флойда (или Флойда-Уоршелла, Floyd–Warshall) позволяет найти кратчайшее расстояние между любыми двумя вершинами в графе, при этом веса ребер могут быть как положительными, так и отрицательными. Данный алгоритм также использует идею динамического программирования.

Будем считать, что в графе $n$ вершин, пронумерованных числами от $0$ до $n-1$. Задана матрица $D$ весов ребер графа. Если в матрице в $i$-й строке в $j$-м столбце стоит 0, то это означает, что дуги из вершины $i$ в вершину $j$ нет. Однако при вводе данной матрицы $D$ все нулевые значения значения будут равны бесконечности $\infty$. Также будем считать, что $d_{ii} = 0$.

Ключевая идея алгоритма – разбиение процесса поиска кратчайших путей на

Перед $k$-ой ($0 \leq k \leq n-1$) фазой величина $d_{ij}$ равна длине кратчайшего пути из вершины $i$ в вершину $j$, если этому пути разрешается заходить только в вершины с номерами, меньшими $k$ (начало и конец пути не считаются).

Пусть теперь мы находимся на $k$-й фазе, и хотим пересчитать матрицу $D$ таким образом, чтобы она соответствовала требованиям уже для $k+1$-ой фазы. Зафиксируем какие-то вершины $i$ и $j$. У нас возникает два принципиально разных случая:

— Кратчайший путь из вершины $i$ в вершину $j$, которому разрешено дополнительно проходить через вершины $\{0, 1, \ldots, k\}$, совпадает с кратчайшим путём, которому разрешено проходить через вершины множества $\{0, 1, \ldots, k-1\}$.

  В этом случае величина $d_{ij}$ не изменится при переходе с $k$-й на $k+1$-ю фазу.

— "Новый"кратчайший путь стал лучше "старого"пути.

  Это означает, что "новый"кратчайший путь проходит через вершину $k$. Сразу отметим, что мы не потеряем общности, рассматривая далее только простые пути (т.е. пути, не проходящие по какой-то вершине дважды).

Тогда заметим, что если мы разобьём этот "новый"путь вершиной $k$ на две половинки (одна идущая $i \Rightarrow k$, а другая – $k \Rightarrow j$), то каждая из этих половинок уже не заходит в вершину $k$. Но тогда получается, что длина каждой из этих половинок была посчитана ещё на $k-1$-ой фазе или ещё раньше, и нам достаточно взять просто сумму $d_{ik} + d_{kj}$, она и даст длину "нового"кратчайшего пути.

Таким образом, вся работа, которую требуется произвести на $k$-ой фазе — это перебрать все пары вершин и пересчитать длину кратчайшего пути между ними. В результате после выполнения $n$-ой фазы в матрице расстояний $d_{ij}$ будет записана длина кратчайшего пути между $i$ и $j$, либо $\infty$, если пути между этими вершинами не существует.

## 2 Код программы, реализующий алгоритм

Далее представлена реализация алгоритма Флойда, написанная на языке $C++$.

```
#include <iostream>
#include <vector>

using namespace std;

#define INF 10e5

vector <int> ans;

void floyd_algorithm (vector<vector<int>>& dst, int n) {
  for (int k = 0; k < n; k++) {
    for (int i = 0; i < n; i++) {
      for (int j = 0; j < n; j++) {
        if (dst[i][k] < INF && dst[k][j] < INF)
          dst[i][j] = min(dst[i][j], dst[i][k] + dst[k][j]);
      }
    }
  }
}

void print_matrix(vector<vector<int>>& dst) {

  for (int i = 0; i < dst.size(); i++) {
    cout << i + 1 << ": ";
    for (int j = 0; j < dst[i].size(); j++) {
      if (dst[i][j] == INF)
        dst[i][j] = 0;
      cout << dst[i][j] << ' ';
    }
    cout << endl;
  }
}

int main() {
  int n, x;
  cout << "Введите число элементов\n";
  cin >> n;
```

```
  vector <vector<int>> dst(n);
  cout << "Введите значения матрицы\n";
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
      cin >> x;
      if (x == 0 && i != j)
        dst[i].push_back(INF);
      else
        dst[i].push_back(x);
    }
  }

  floyd_algorithm(dst, n);

  cout << "Матрица кратчайших путей:\n";
  print_matrix(dst);
  return 0;
}
```

Далее приведем несколько примеров, подав некоторые случайные числа в качестве входных данных, чтобы проверить корректность, написанного кода.

На рисунке 1 представлен неориентированный взвешенный граф из 6 вершин. Из рисунка 2 видно, что поиск кратчайших путей сработал корректно.

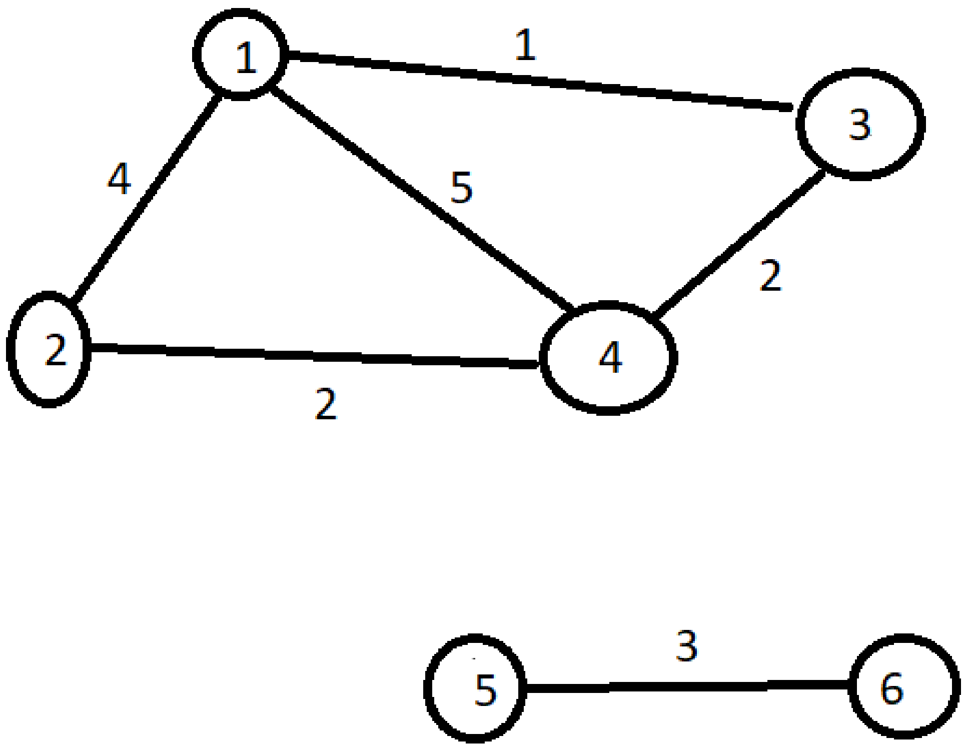


Рисунок 1 – Взвешенный неориентированный граф

```
Введите число элементов
6
Введите значения матрицы
0 4 1 5 0 0
4 0 0 2 0 0
1 0 0 2 0 0
5 2 2 0 0 0
0 0 0 0 0 3
0 0 0 0 3 0
Матрица кратчайших путей:
1: 0 4 1 3 0 0
2: 4 0 4 2 0 0
3: 1 4 0 2 0 0
4: 3 2 2 0 0 0
5: 0 0 0 0 0 3
6: 0 0 0 0 3 0
```

Рисунок 2 – Результат работы алгоритма

На рисунке 3 представлен ориентированный взвешенный граф из 6 вершин.

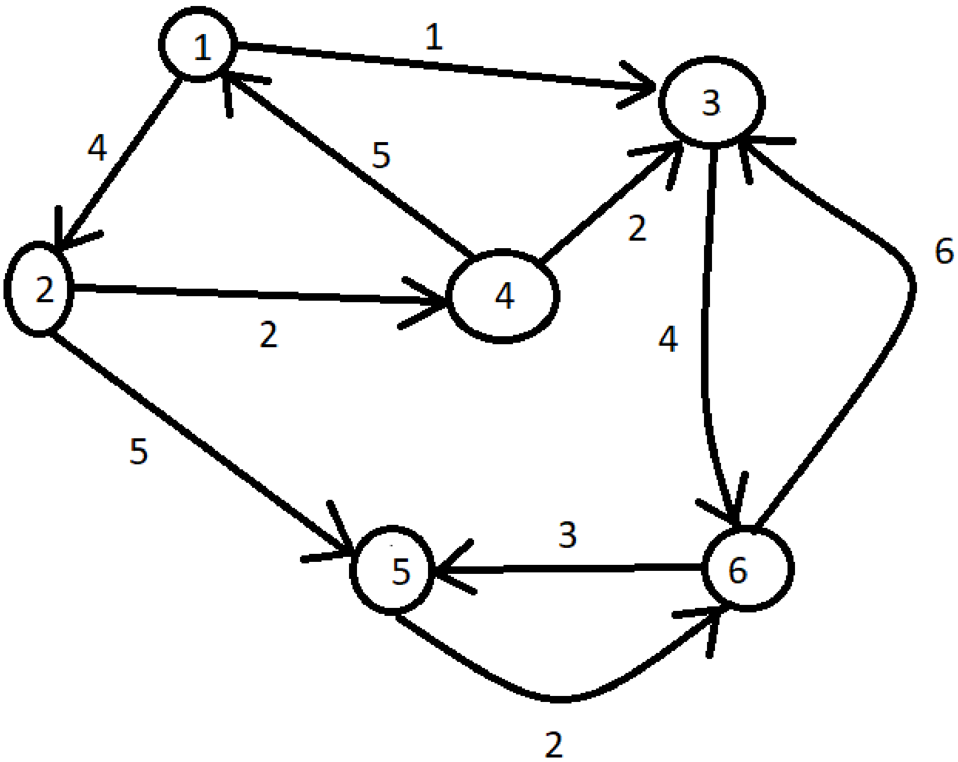


Рисунок 3 – Взвешенный ориентированный граф

А на рисунке 4 показан результат работы алгоритма Флойда.

```
Введите число элементов
6
Введите значения матрицы
0 4 1 0 0 0
0 0 0 2 5 0
0 0 0 0 0 4
5 0 2 0 0 0
0 0 0 0 0 2
0 0 6 0 3 0
Матрица кратчайших путей:
1: 0 4 1 6 8 5
2: 7 0 4 2 5 7
3: 0 0 0 0 7 4
4: 5 9 2 0 9 6
5: 0 0 8 0 0 2
6: 0 0 6 0 3 0
```

Рисунок 4 – Результат работы алгоритма

## 3 Анализ алгоритма

Очевидно, что сложность данного алгоритма равна $O(n^3)$, так как здесь используется три вложенных друг в друга цикла, в которых реализуется пересчитывание кратчайших путей.

Стоит отметить, что Алгоритм Флойда некорректно работает при наличии цикла отрицательного веса, но при этом если путь от $i$ до $j$ не содержит цикла отрицательного веса, то вес этого пути будет найден алгоритмом правильно. Также при помощи данного алгоритма можно определить наличие цикла отрицательного веса: если $i$ вершина лежит на цикле отрицательного веса, то значение $d_{ii}$ будет отрицательным после окончания алгоритма.

## ЗАКЛЮЧЕНИЕ

В данной лабораторной работе был рассмотрен алгоритм Флойда - Уоршелла и был проведен анализ оценки его сложности. Это послужило созданием её программной реализации, написанной на языке $C++$.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Статья "Алгоритм Флойда-Уоршелла"/ [Электронный ресурс] URL: https://ru.wikipedia.org/wiki/Алгоритм_Флойда_—_Уоршелла (дата обращения 26.02.2022), Яз. рус

2 Статья "Алгоритм Флойда-Уоршелла нахождения кратчайших путей между всеми парами вершин"/ [Электронный ресурс] URL: https://e-maxx.ru/algo/floyd_warshall_algorithm (дата обращения 26.02.2022), Яз. рус.

3 Стивен С. Скиен, "Алгоритмы. Руководство по разработке 2-е издание"/ Санкт-Петербург: БХВ-Петербург, 2021 г., Яз. рус.