

АЛГОРИТМ ФЛОЙДА—УОРШЕЛЛА

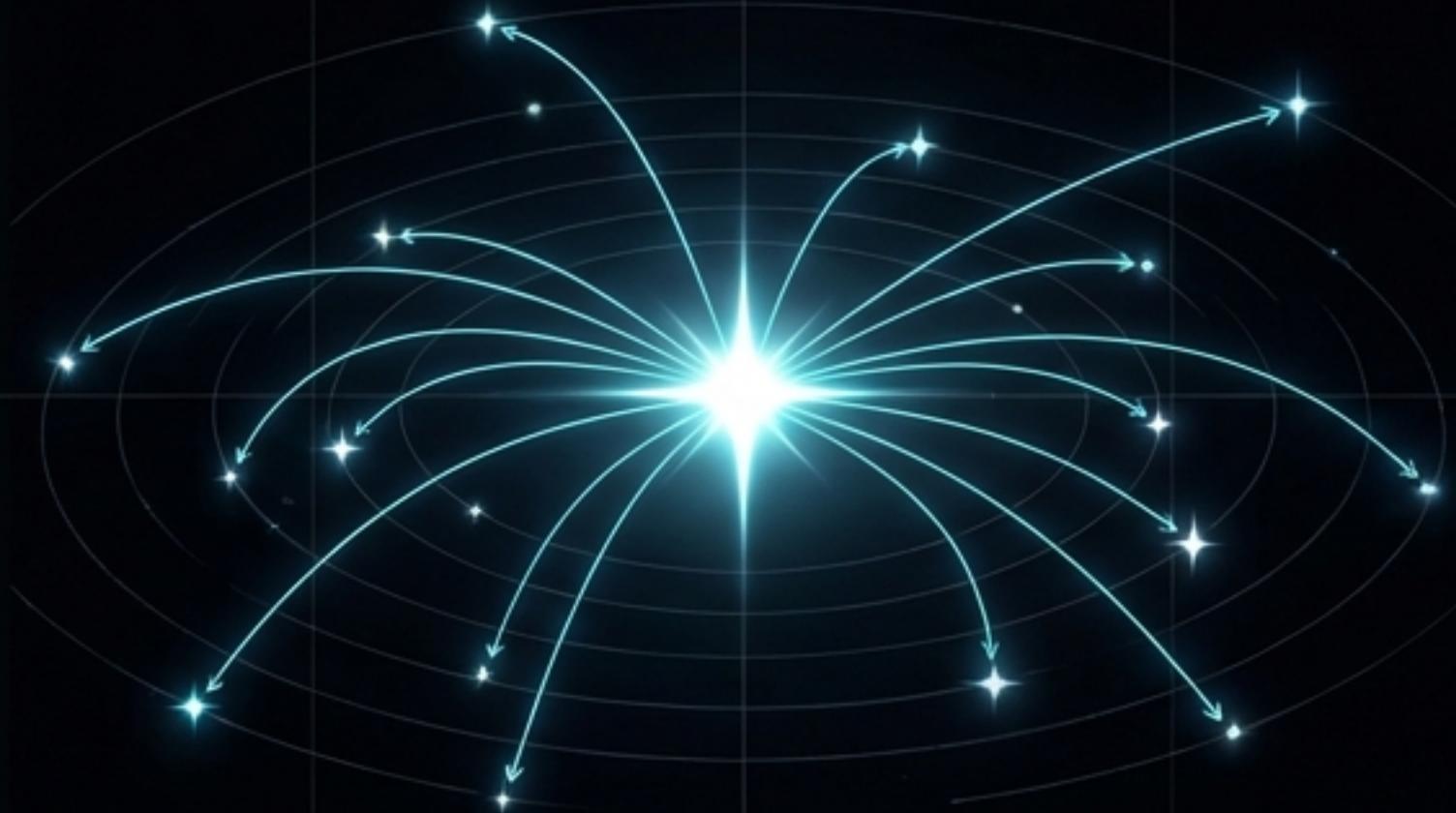
Навигация по вселенной данных: поиск кратчайших
путей между всеми парами вершин.



Для Факультета Астрономии | Задача: Поиск максимального кратчайшего пути

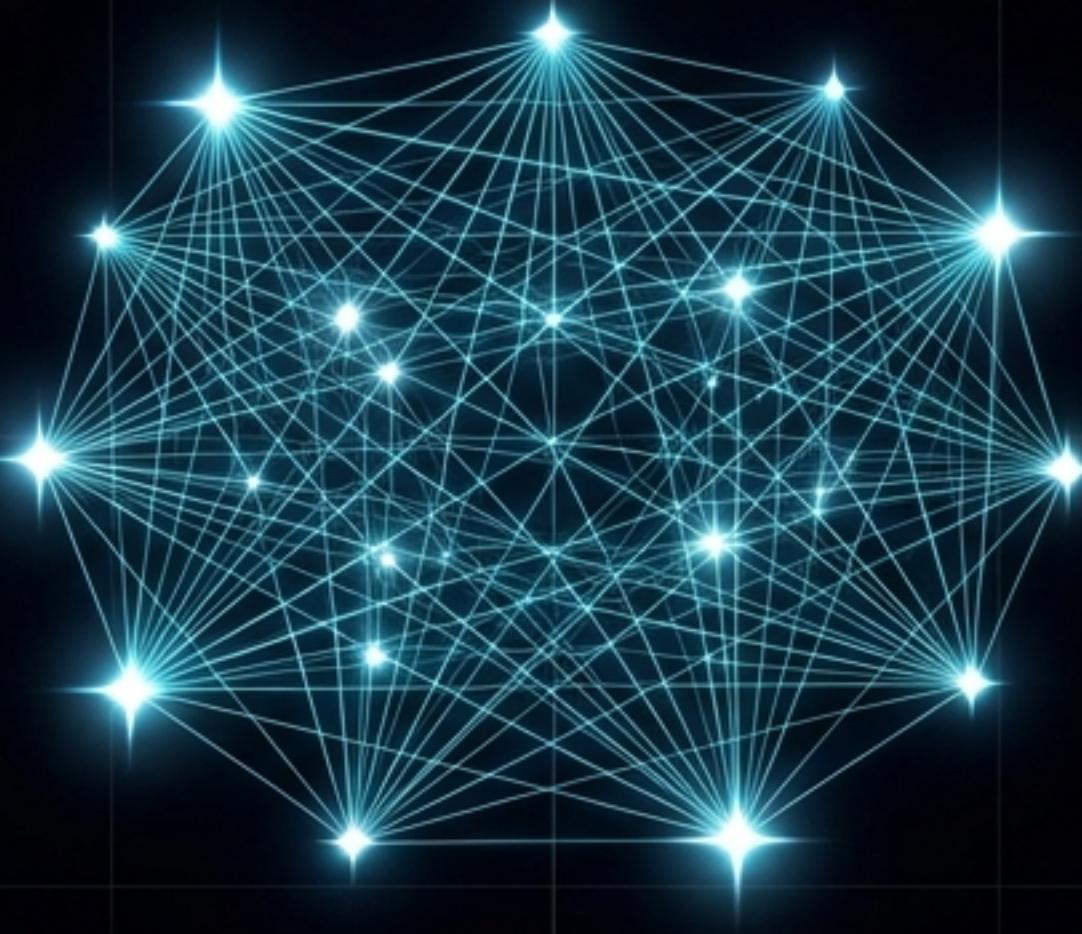
ЗАЧЕМ НАМ НОВАЯ КАРТА?

One-to-All / Dijkstra



Проблема (The Problem): Нам нужно найти кратчайшие пути не от одной точки, а между каждой возможной парой вершин (i, j).

All-to-All / Floyd-Warshall



Почему не Дейкстра?: Запуск Дейкстры N раз работает за $O(N * E \log N)$. На плотных графах (где много ребер) это медленнее.

Сложность Флойда—Уоршелла фиксирована: $\Theta(N^3)$.
Идеально для плотных матриц смежности ($N \leq 400$).

СИНХРОНИЧНОСТЬ ОТКРЫТИЯ



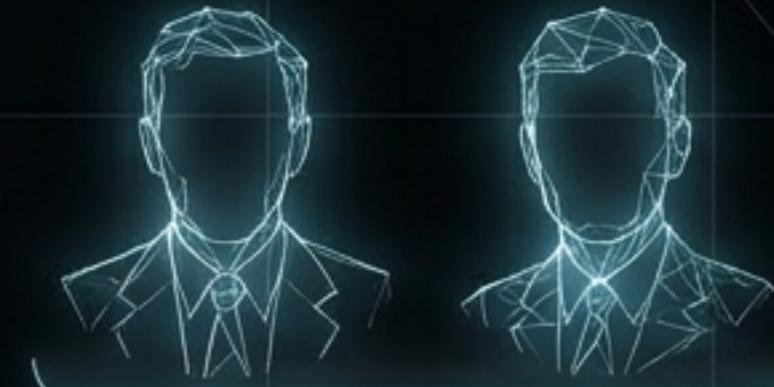
1956

Стивен Клини
(Kleene's Algorithm)



1959

Бернар Руа
(Bernard Roy)



1962

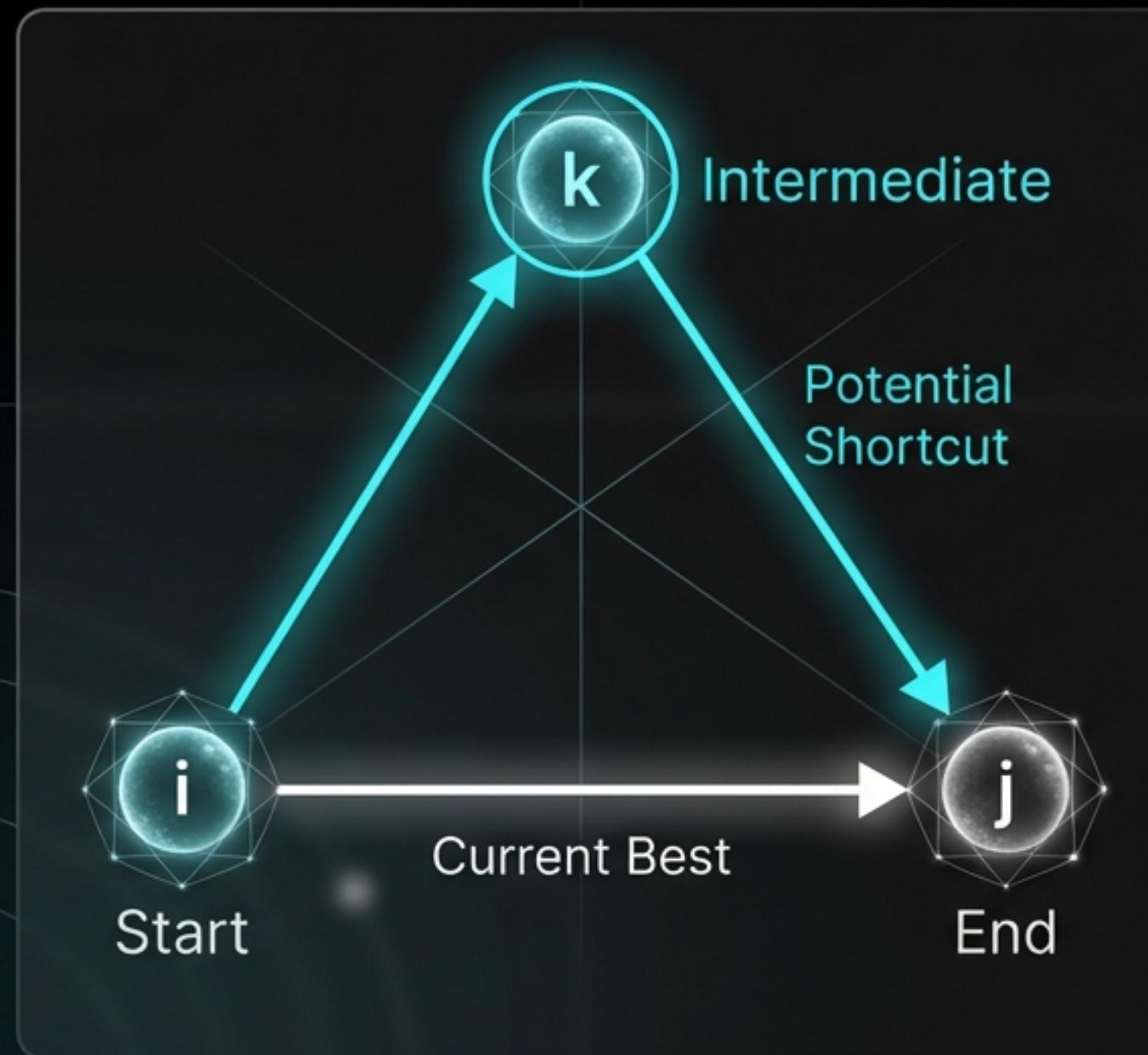
Роберт Флойд (Robert Floyd)
Стивен Уоршелл (Stephen Warshall)

Алгоритм был опубликован в 1962 году Робертом Флойдом. Однако идея витала в воздухе:

- 1959: **Бернар Руа** публикует схожую идею.
- 1962: **Стивен Уоршелл** описывает метод для транзитивного замыкания.
- **Связь с Клини:** Алгоритм структурно идентичен методу Клини для превращения конечных автоматов в регулярные выражения.

Это не просто поиск пути, это фундаментальная алгебраическая операция над транзитивными отношениями.

ПРИНЦИП ПРОМЕЖУТОЧНОЙ ВЕРШИНЫ



Мы используем **Динамическое Программирование**.

Вопрос: Можно ли сократить путь из **i** в **j**, если пройти через вершину **k**?

$$D_{ij} = \min(D_{ij}, D_{ik} + D_{kj})$$

На шаге **k** мы "открываем" вершину **k** и проверяем, улучшит ли она наши маршруты между всеми **i** и **j**.

ИНИЦИАЛИЗАЦИЯ МАТРИЦЫ D(0)

0	5	12	3	∞
∞	0	7	2	∞
∞	∞	0	9	∞
∞	1	8	0	∞
∞	∞	4	6	0

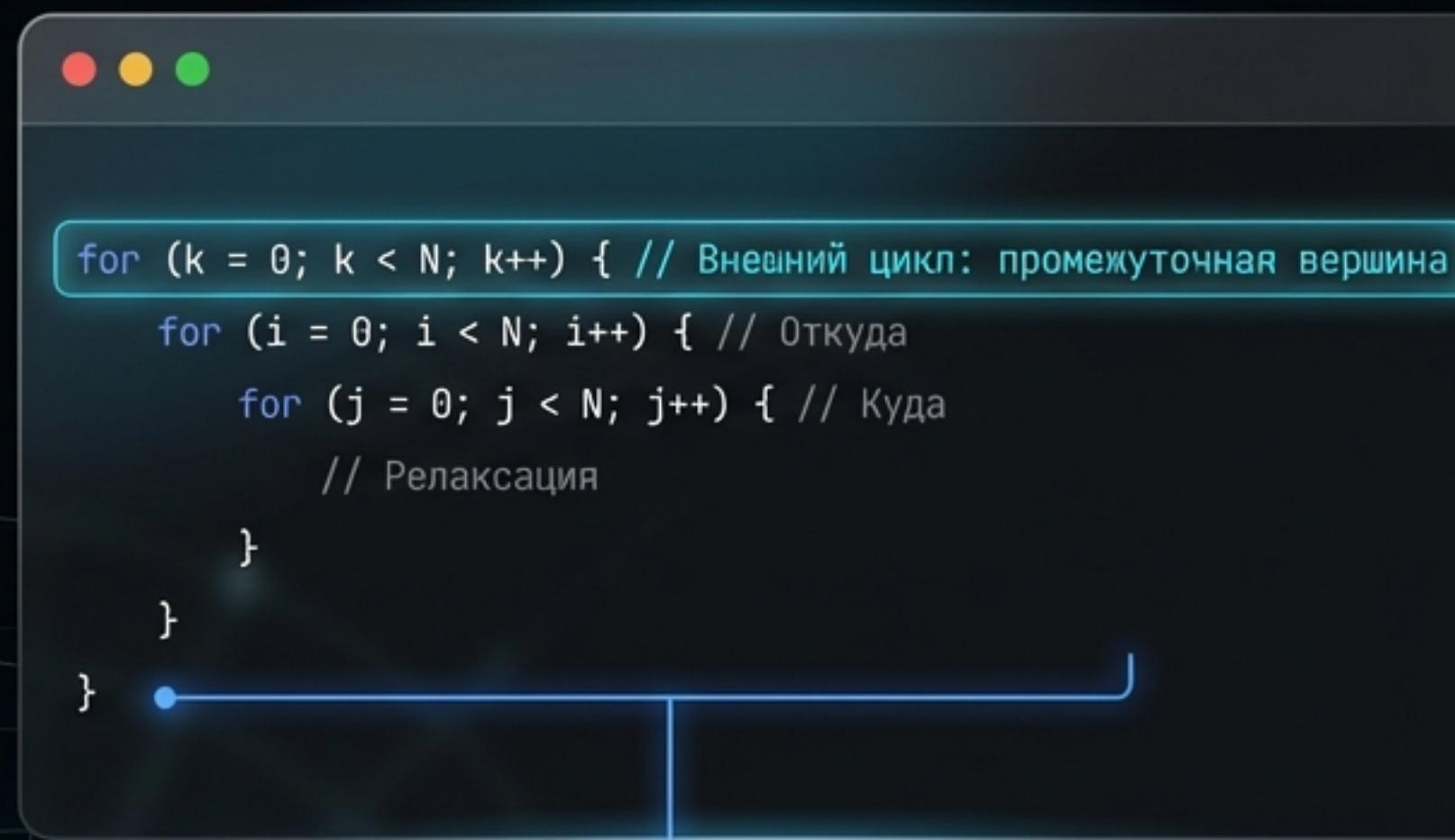
$D[i][i] = 0$
(Расстояние до себя)

$D[i][j] = w_{ij}$
(Вес ребра)

$D[i][j] = \infty$
(Нет связи)

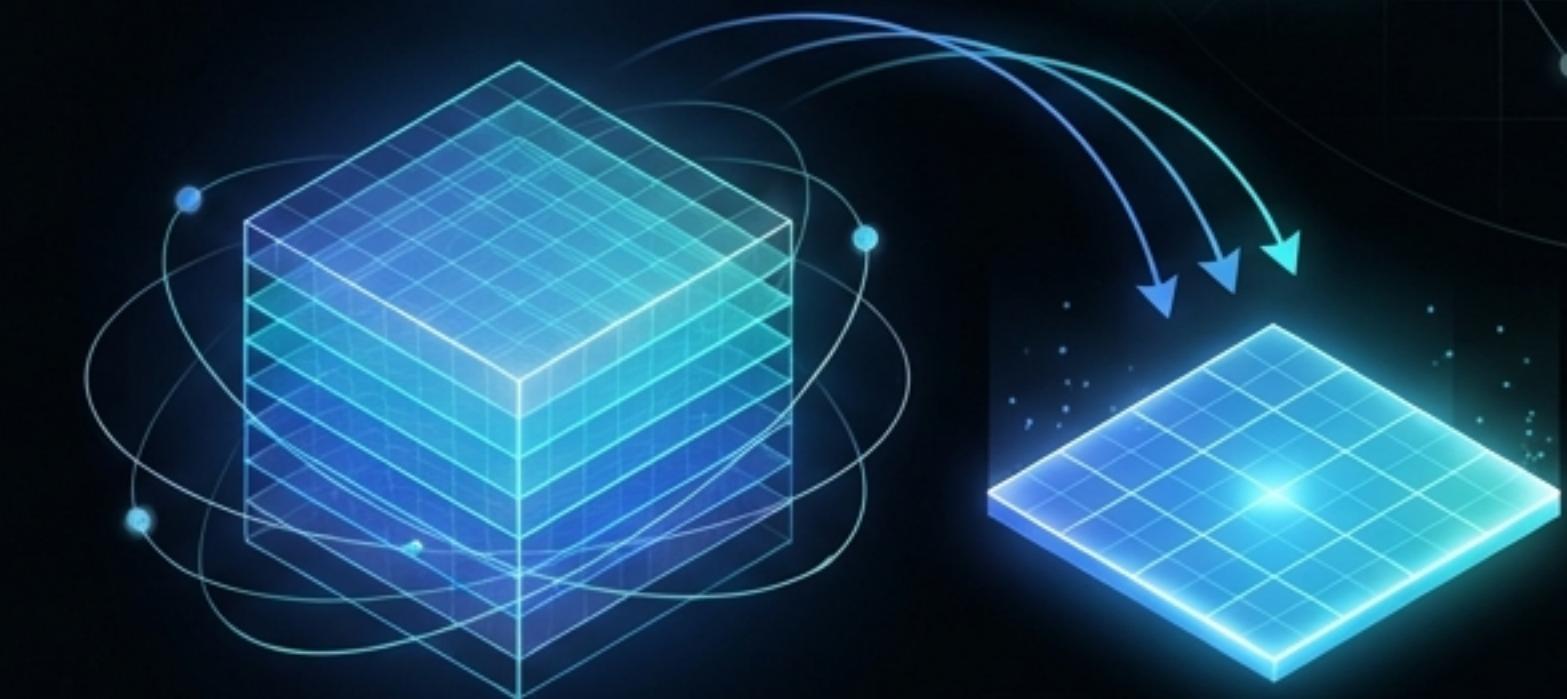
ВАЖНО:
В исходных данных -1
обозначает отсутствие ребра.
При чтении -1 необходимо
заменить на константу INF
(например, 10^9), иначе
логика `min()` сломается.

АРХИТЕКТУРА ТРЕХ ЦИКЛОВ



```
for (k = 0; k < N; k++) { // Внешний цикл: промежуточная вершина
    for (i = 0; i < N; i++) { // Откуда
        for (j = 0; j < N; j++) { // Куда
            // Релаксация
        }
    }
}
```

$O(N^3) \rightarrow O(N^2)$



- **Порядок важен!** Цикл по k (промежуточной вершине) обязан быть **внешним**.
- **Память:** Нам не нужны матрицы $D(0)...D(N)$. Обновления делаем *in-place*.
- **Итог:** $O(N^2)$ памяти вместо $O(N^3)$.

МОМЕНТ РЕЛАКСАЦИИ

```
if (dist[i][k] < INF && dist[k][j] < INF) {  
    if (dist[i][j] > dist[i][k] + dist[k][j]) {  
        dist[i][j] = dist[i][k] + dist[k][j];  
    }  
}
```

Critical Check: Защита от переполнения (Overflow).

В C/C++ сложение (INF + число) может дать отрицательный результат, если INF слишком велик.

Безопасный INF = 10^9 (меньше INT_MAX).

ОТРИЦАТЕЛЬНЫЕ ВЕСА И ЦИКЛЫ

-5

Работает с отрицательными ребрами.

- **Преимущество:** Алгоритм работает с отрицательными весами (в отличие от Дейкстры).

-5



-2

Отрицательный цикл (Negative Cycle).

-4

- **Ограничение:** Граф не должен содержать отрицательных циклов (сумма < 0).

Как обнаружить? Если в конце $D[i][i] < 0$, значит вершина i – часть отрицательного цикла.

ПОСТАНОВКА ЗАДАЧИ

Input Format:

- Число N ($1 \leq N \leq 100$)
- Матрица смежности $N \times N$
- '-1' означает отсутствие ребра

Input:

```
6
0 6 8 -1 -1 -1
5 0 5 -1 -1 -1
1 7 0 -1 -1 -1
-1 -1 -1 0 6 -1
-1 -1 -1 -1 0 3
-1 -1 -1 2 -1 0
```

Output:

```
9
```



Goal: Найти пару вершин, кратчайшее расстояние между которыми МАКСИМАЛЬНО среди всех пар.

Mathematical Goal: $\max(D[i][j])$, где $D[i][j] \neq \text{INF}$ и $i \neq j$.

Задача: Найти диаметр обитаемой части вселенной.

ЧТЕНИЕ И ПОДГОТОВКА ДАННЫХ



scanf('%d', &val)

val == -1?

Yes

dist[i][j] = INF

No

dist[i][j] = val

```
scanf_s("%d", &val);
if (val == -1)
    dist[i][j] = INF; // -1 превращаем в бесконечность
else
    dist[i][j] = val; // Сохраняем вес
```

```
setlocale(LC_ALL, "Russian");
// Для корректного
отображения кириллицы
```

ВЫПОЛНЕНИЕ АЛГОРИТМА

0	∞						
∞	∞	0	3	5	12	7	∞
∞	∞	∞	5	12	∞	7	2
∞	∞	∞	0	5	2	9	∞
∞	∞	∞	∞	0	5	3	0
∞	∞	∞	9	∞	7	3	∞
∞	∞	∞	2	5	7	2	5
∞	∞	∞	3	∞	12	7	0

Запускаем классический
Флойд-Уоршелл.

Сложность: $N \leq 100 \rightarrow$
 $\sim 1,000,000$ операций.

Performance: Мгновенно
для современного CPU.

```
// Динамическое выделение памяти
int** dist = (int**)malloc(N * sizeof(int*));
```

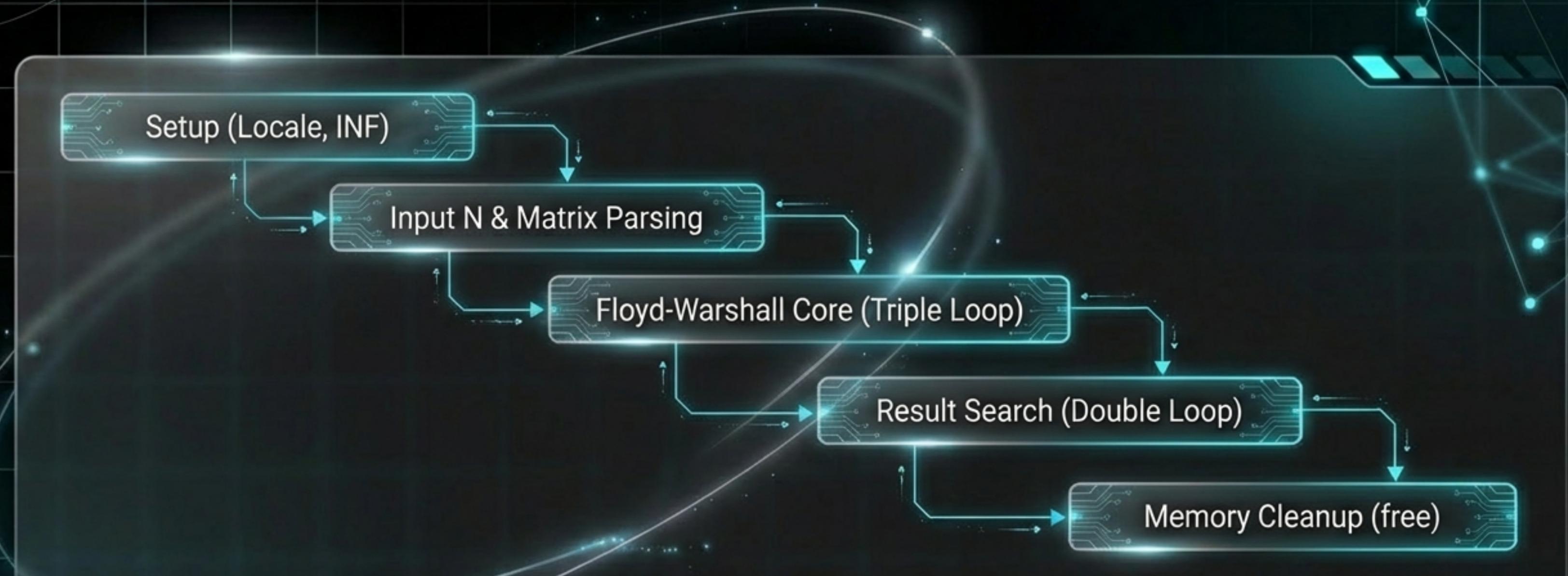
ПОИСК МАКСИМУМА

INF	X	3	5	12	7	INF	INF
INF	X	12	0	5	2	INF	INF
INF	X	0	7	3	9	INF	INF
INF	X	9	7	3	0	INF	INF
INF	X	2	5	7	5	0	INF
INF	X	3	12	X	7	12	7
INF	X	12	7	X	X	X	X
INF	X	X	X	X	X	X	X

```
if (i != j && dist[i][j] < INF) {  
    if (dist[i][j] > max_dist) {  
        max_dist = dist[i][j];  
        start = i + 1;  
        end = j + 1;  
    }  
}
```

Игнорируем:
1. Диагональ ($i == j$)
2. Недостижимые вершины (INF)

ПОЛНЫЙ КОД РЕШЕНИЯ



Структура программы из Source.txt
Безопасная работа с памятью и буфером ввода.

ГРАНИЧНЫЕ СЛУЧАИ

Сценарий:

Несвязный граф. Если `max_dist == -1`, значит, нет ни одного пути между разными вершинами.



```
if (max_dist == -1)
    printf("В графе нет достижимых пар
вершин!");
else
    printf("Максимальное кратчайшее
расстояние: %d", max_dist);
```

ИТОГИ: ПОРЯДОК ИЗ ХАОСА

Алгоритм:

Флойда—Уоршелла

***Время:** $O(N^3)$

***Память:** $O(N^2)$

***Применение:** Плотные графы,
матрицы смежности,
транзитивное замыкание.



Мы превратили набор разрозненных наблюдений
в полное знание о связности вселенной.