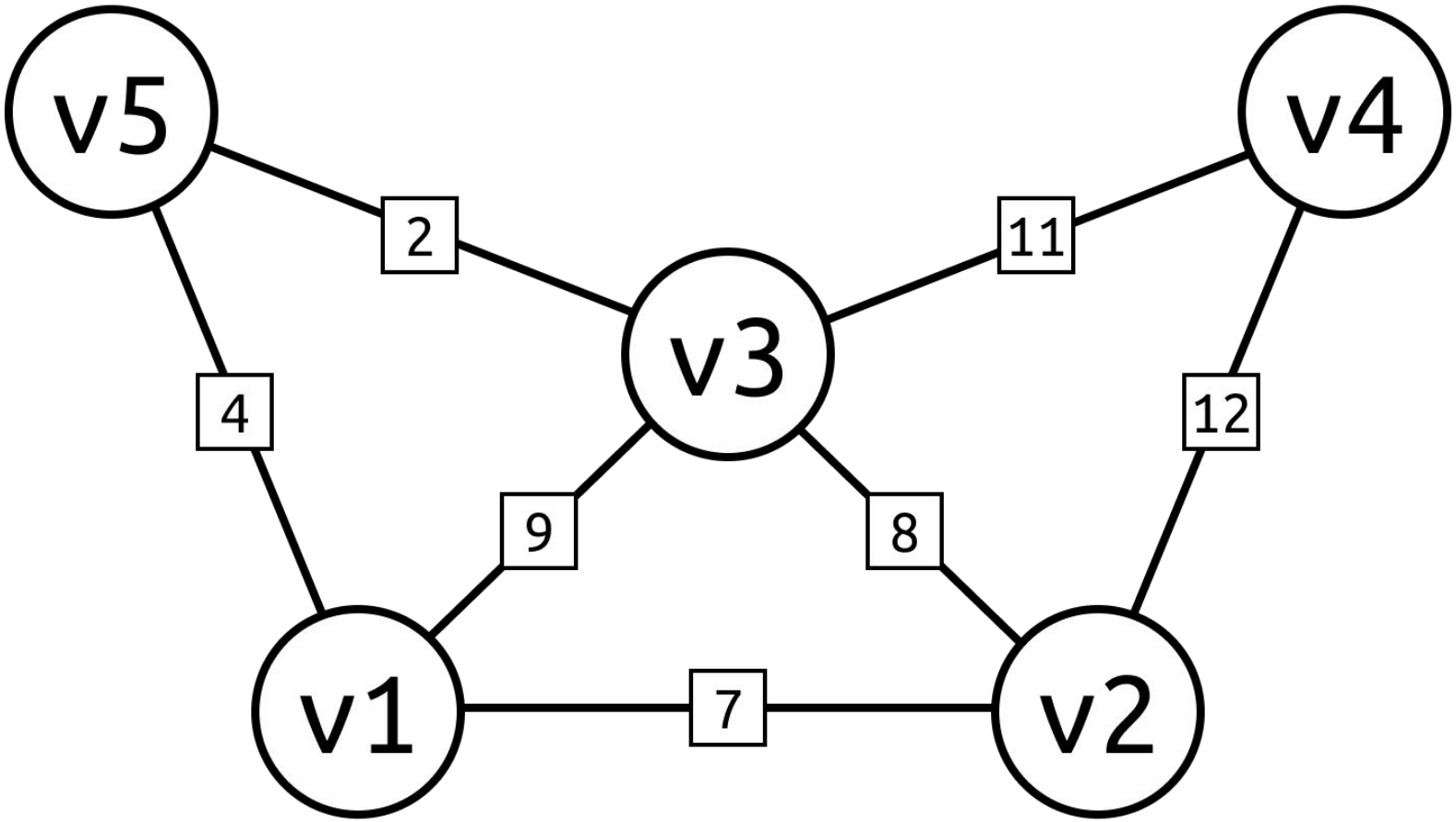
Алгоритм дейкстры

Алгоритм Дейкстры

- Алгоритм Дейкстры является алгоритмом поиска минимального пути от заданной вершины до всех остальных.
- Данный алгоритм будет продемонстрирован на примере взвешенного неориентированного графа.

Пример графа

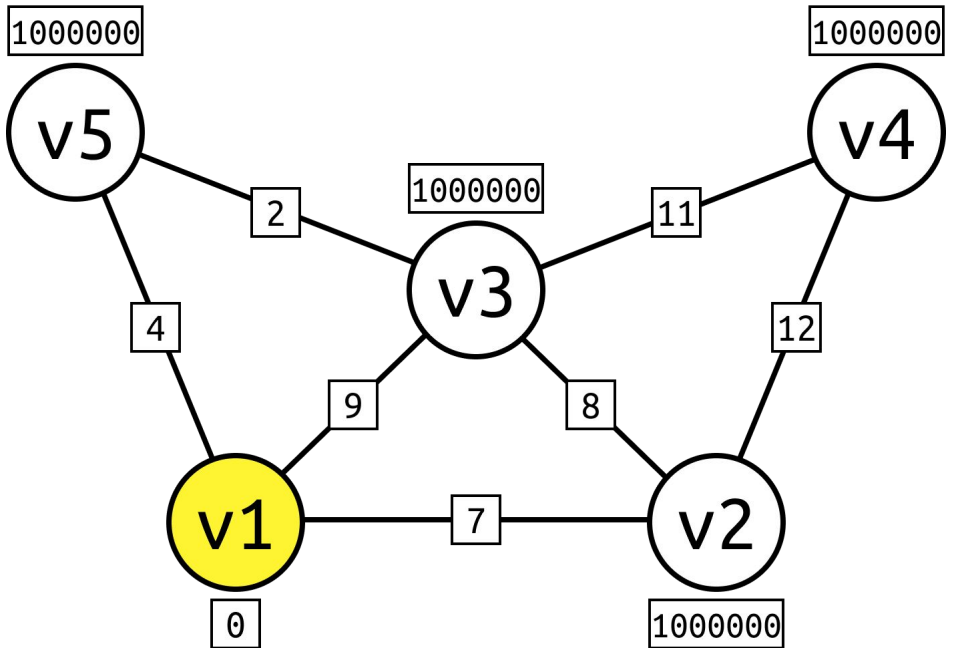


Анализ алгоритма

Изначально все вершины графа получают некоторую метку, которая характеризует расстояние до исходной вершины. Это расстояние не известно, поэтому будем считать что расстояние пока очень большое число.

Вершина, от которой начинается путь получает метку нуль.

Также будем отмечать обработанные вершины.



Анализ алгоритма.

- Алгоритм обходит соседей обрабатываемой вершины.
- Вычисляется расстояние до каждого из соседей как сумма метки обрабатываемой и веса ребра до соседней. Полученное значение сравнивается с меткой соседней вершины и, если полученное значение меньше, то это значение присваивается соседней вершине.
- После прохода по соседям, обрабатываемая вершина отмечается как обработанная.
- Ищется новая обрабатываемая вершина, как ближайшая к уже обработанной. При этом, новая обрабатываемая вершина не должна быть обработанной.

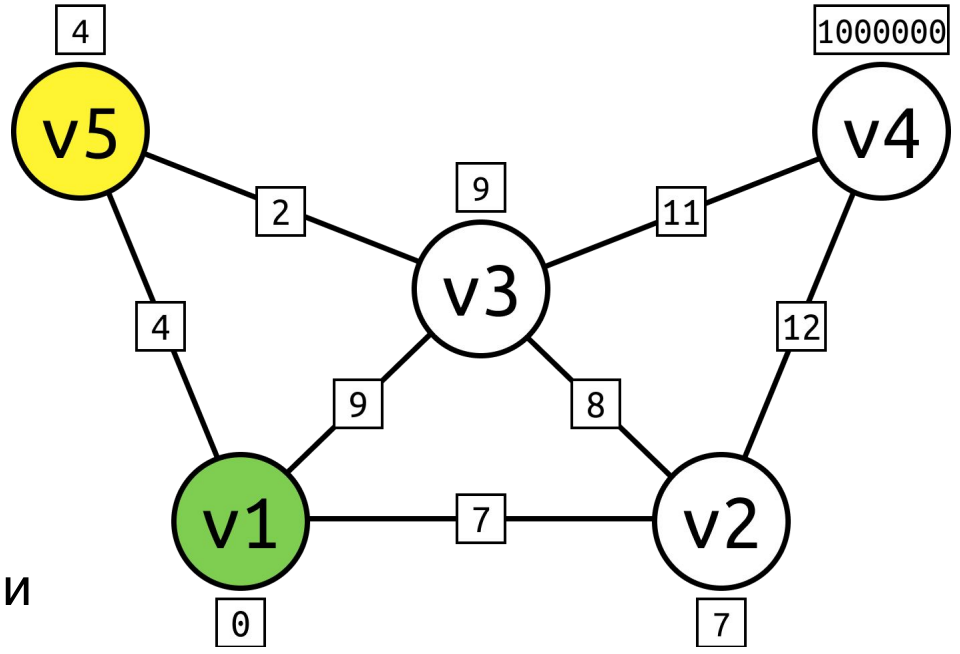
Работа алгоритма. Шаг 1

После обработки вершины *v1* значения меток вершин *v2*, *v3* и *v5* изменились:

0+7 = 7 < 1 000 000, поэтому метка v2 = 7
0+9 = 9 < 1 000 000, поэтому метка v3 = 9
0+4 = 4 < 1 000 000, поэтому метка v5 = 4

Вершина *v1* помечается обработанной.

Затем алгоритм находит ближайшую вершину *v5* и начинает обрабатывать её.



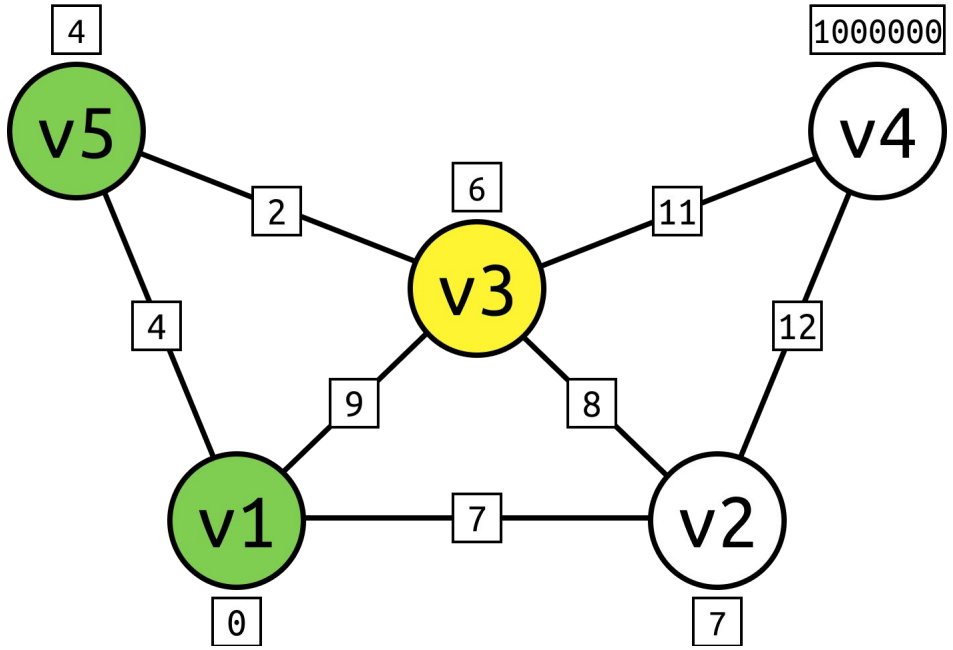
Работа алгоритма. Шаг 2

После обработки вершины *v5*, значение метки вершины *v3* изменилось:

4+2 = 6 < 9, поэтому метка v3 = 6

Вершина *v5* помечается обработанной.

Алгоритм вновь находит ближайшую необработанную вершину *v3* и начинает обрабатывать её.



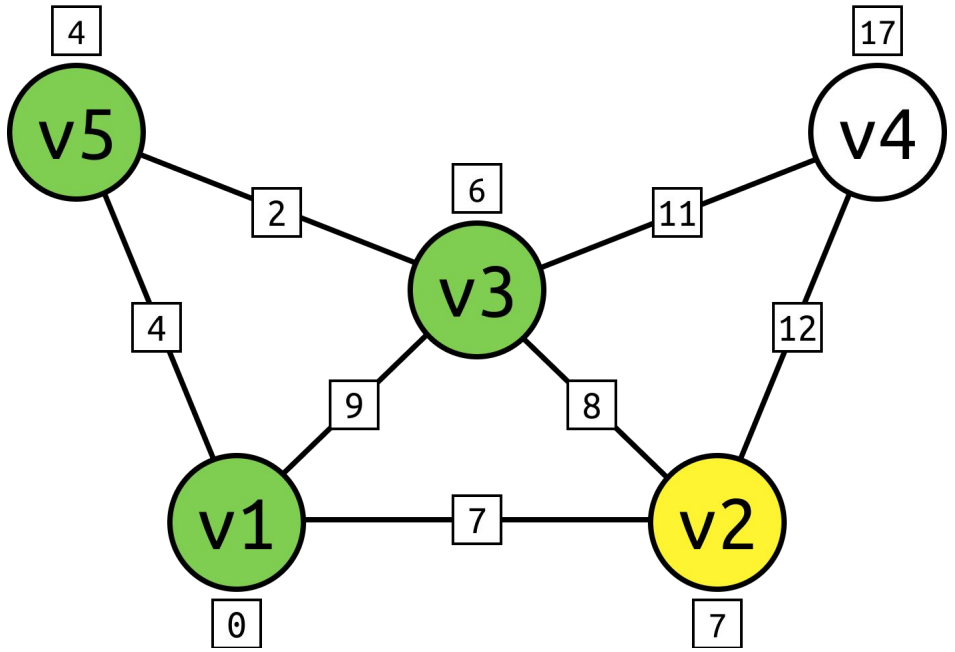
Работа алгоритма. Шаг 3

После обработки вершины *v3* значения меток вершин *v2* и *v4* такие:

6+8 = 14 > 7, поэтому метка *v2* не изменяется
6+11 = 17 < 1 000 000, поэтому метка *v4* = 17

Вершина *v3* помечается обработанной.

Алгоритм вновь находит ближайшую необработанную вершину *v2* и начинает обрабатывать её.



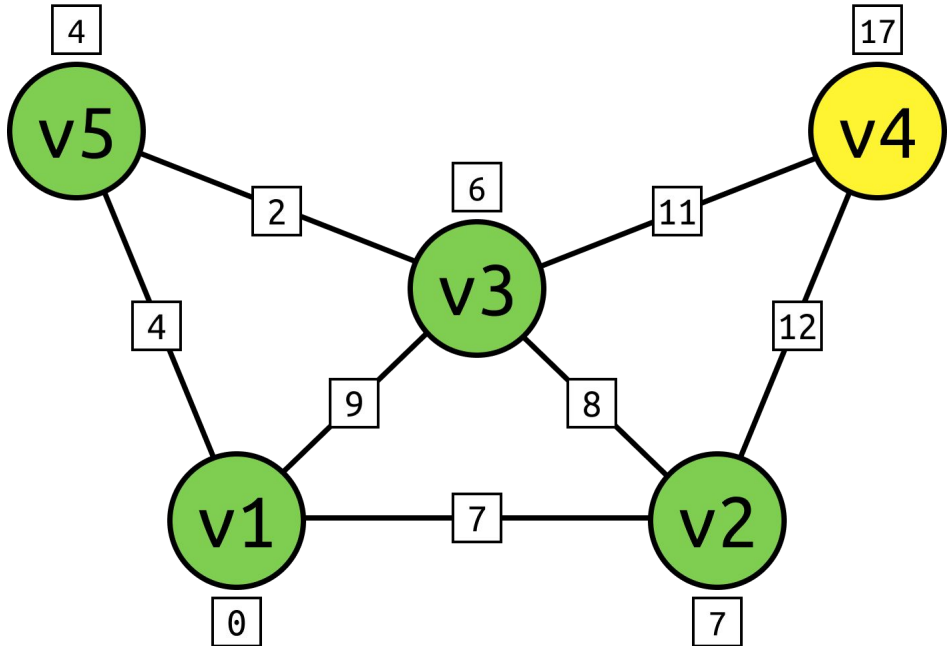
Работа алгоритма. Шаг 4

После обработки вершины *v2* значение метки вершины v4:

7+12 = 19 > 17, поэтому метка *v4* не изменяется

Вершина v2 помечается обработанной.

Алгоритм вновь находит ближайшую необработанную вершину *v4* и начинает обрабатывать её.

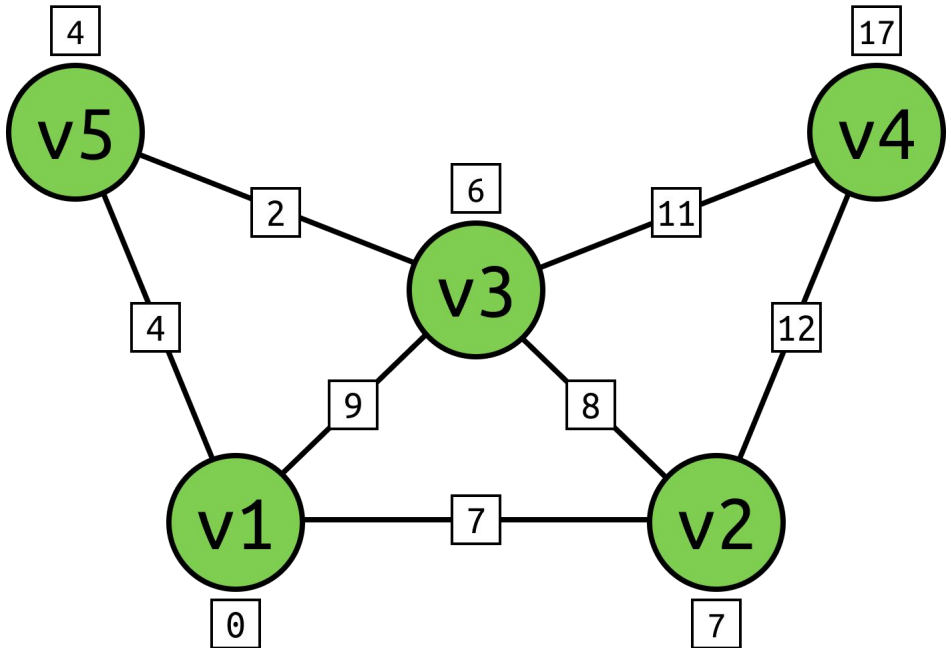


Работа алгоритма. Шаг 4

У вершины *v4* нет необработанных соседних вершин, поэтому она помечается обработанной.

Обход графа завершён, алгоритм закончил работу.

Теперь метки характеризуют минимальное расстояние от *v1* до *v2-v4*



Реализация на C++

```
template <class T>
void Graph<T>::FillLabels(T& startVertex)
{
    //Заполнение меток расстояния
    for (int i = 0, size = vertexList.size(); i < size; ++i)
        labelList[i] = 1000000;
    int pos = GetVertPos(startVertex);
    labelList[pos] = 0;
};
```

Реализация на C++

```
template <class T>
bool Graph<T>::AllVisited(vector<bool>& visitedVerts)
{
    //Проверяет, все ли ноды были посещены
    bool flag = true;
    for (int i = 0; i < this->vertexList.size(); i++)
        if (visitedVerts[i] != true)
            flag = false;
    return flag;
};
```

Реализация на C++

```
template <class T>
void Graph<T>::Dijkstra(T& startVertex)
{
    for (int i = 0; i < vertexList.size(); i++)
        for (int j = 0; j < vertexList.size(); j++)
            if (adjMatrix[i][j] < 0)
                return;
    if (GetVertPos(startVertex) == -1)
        return;

    vector<bool> visitedVerts(vertexList.size());
    fill(visitedVerts.begin(), visitedVerts.end(), false);
```

Реализация на C++

```
template <class T>
void Graph<T>::Dijkstra(T& startVertex)
{
    for (int i = 0; i < vertexList.size(); i++)
        for (int j = 0; j < vertexList.size(); j++)
            if (adjMatrix[i][j] < 0)
                return;
    if (GetVertPos(startVertex) == -1)
        return;

    vector<bool> visitedVerts(vertexList.size());
    fill(visitedVerts.begin(), visitedVerts.end(), false);
```

```
this->FillLabels(startVertex);
T curSrc = startVertex;
vector<T> neighbors;
```

Реализация на C++

```
while (!this->AllVisited(visitedVerts))
{
    neighbors = this->GetNbrs(curSrc);
    int startLabel = labelList[GetVertPos(curSrc)];

    //Самый близкий сосед
    T* minNeighbor_ptr = nullptr;
    int minW = 1000000;
```

Реализация на C++

```
for (int i = 0; i < neighbors.size(); ++i)
{
    int weight = this->GetWeight(curSrc, neighbors[i]);
    int nIndex = this->GetVertPos(neighbors[i]);
    int nextLabel = labelList[nIndex];
    if (startLabel + weight < nextLabel)
    {
        labelList[nIndex] = startLabel + weight;
    };
    if (!visitedVerts[nIndex] && minW > labelList[nIndex])
    {
        minW = labelList[nIndex];
        minNeighbor_ptr = &neighbors[i];
    };
};
```

Реализация на C++

```
for (int i = 0; i < neighbors.size(); ++i)
{
    int weight = this->GetWeight(curSrc, neighbors[i]);
    int nIndex = this->GetVertPos(neighbors[i]);
    int nextLabel = labelList[nIndex];
    if (startLabel + weight < nextLabel)
    {
        labelList[nIndex] = startLabel + weight;
    };
    if (!visitedVerts[nIndex] && minW > labelList[nIndex])
    {
        minW = labelList[nIndex];
        minNeighbor_ptr = &neighbors[i];
    };
};
```

Реализация на C++

```
        visitedVerts[GetVertPos(curSrc)] = true;
        if (minNeighbor_ptr != nullptr)
        {
            curSrc = *minNeighbor_ptr;
        };
};
```

Реализация на C++

```
/* Вывод результата работы алгоритма на экран */
for (int i = 0; i < GetVertPos(startVertex); ++i)
{
    cout << "Кратчайшее расстояние от вершины " << startVertex
        << " до вершины " << vertexList[i] << " равно "
        << labelList[GetVertPos(vertexList[i])] << endl;
};

for (int i = GetVertPos(startVertex) + 1; i < vertexList.size(); ++i)
{
    cout << "Кратчайшее расстояние от вершины " << startVertex
        << " до вершины " << vertexList[i] << " равно "
        << labelList[GetVertPos(vertexList[i])] << endl;
};
```

Результаты работы

```
Введите количество вершин графа: 5

Введите количество ребер графа: 7

Вершина: 5

Вершина: 1

Вершина: 4

Вершина: 2

Вершина: 3

Исходная вершина: 1

Конечная вершина: 5

Вес ребра: 4

Исходная вершина: 1

Конечная вершина: 2

Вес ребра: 7

Исходная вершина: 1

Конечная вершина: 3

Вес ребра: 9

Исходная вершина: 5

Конечная вершина: 3

Вес ребра: 2

Исходная вершина: 3

Конечная вершина: 2
```

```
Матрица смежности графа:
-  1  5  4  3  2
1  0  4  0  9  7
5  4  0  0  2  0
4  0  0  0  11  12
3  9  2  11  0  8
2  7  0  12  8  0
Кратчайшее расстояние от вершины 1 до вершины 5 равно 4
Кратчайшее расстояние от вершины 1 до вершины 4 равно 17
Кратчайшее расстояние от вершины 1 до вершины 3 равно 6
Кратчайшее расстояние от вершины 1 до вершины 2 равно 7
```