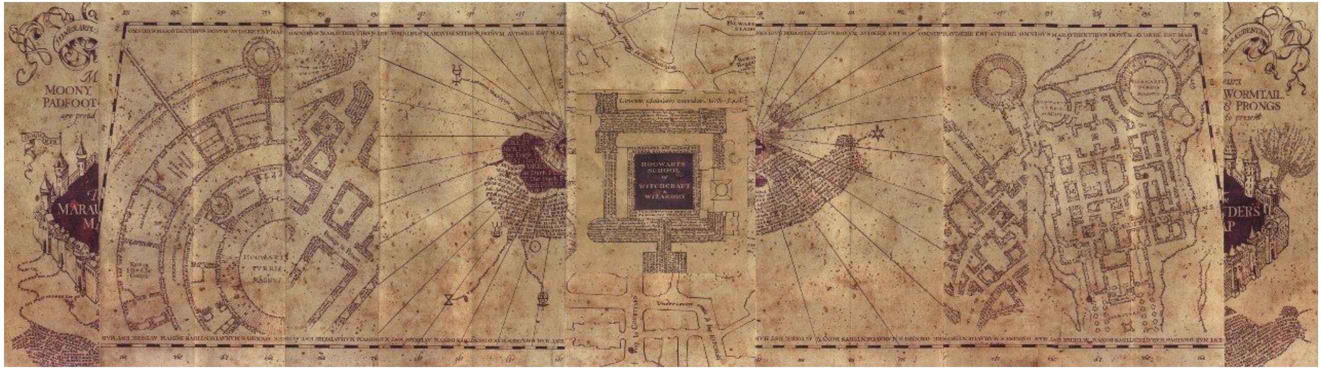
Мини-задача #41 (1 балл)

Найти город с наименьшим числом достижимых из него городов, которые расположены ближе, чем заданное расстояние.

https://leetcode.com/problems/find-the-city-with-the-smallest-number-of-neighbors-at-a-threshold-distance/description/

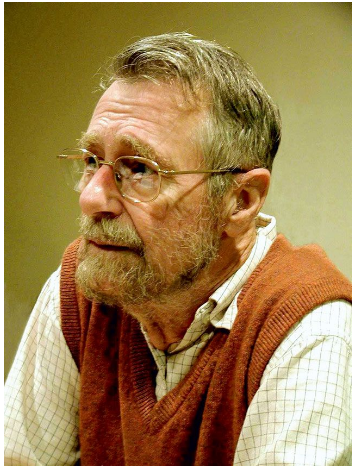


Алгоритмы и структуры данных

Алгоритм Беллмана-Форда, алгоритм Флойда-Уоршелла

Алгоритм Дейкстры поиска кратчайших путей

Задача: пусть дан взвешенный граф $G = \langle V, E \rangle$ без рёбер отрицательного веса. Найти кратчайшие пути от заданной вершины до всех остальных.



Алгоритм Дейкстры поиска кратчайших путей

Задача: пусть дан взвешенный граф $G = \langle V, E \rangle$ без рёбер отрицательного веса. Найти кратчайшие пути от заданной вершины до всех остальных.

У алгоритма Дейкстры две проблемы:

1. Рёбра отрицательного веса - вполне нормальная конфигурация графа (например, когда ребра - это не транзакции).

Алгоритм Дейкстры поиска кратчайших путей

Задача: пусть дан взвешенный граф $G = \langle V, E \rangle$ без рёбер отрицательного веса. Найти кратчайшие пути от заданной вершины до всех остальных.

У алгоритма Дейкстры две проблемы:

1. Рёбра отрицательного веса - вполне нормальная конфигурация графа (например, когда ребра - это не транзакции).

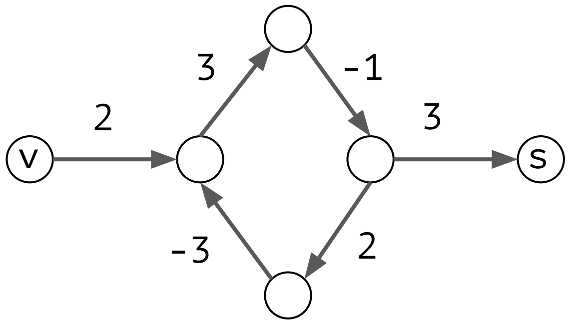
2. Для слишком больших графов становится невозможно загрузить их в память и запустить Дейкстру 😔

Циклы отрицательного веса

Наблюдение: рёбра отрицательного веса - это не проблема. А вот циклы отрицательного веса (для задач поиска кратчайшего пути) - это что-то странное.

Циклы отрицательного веса

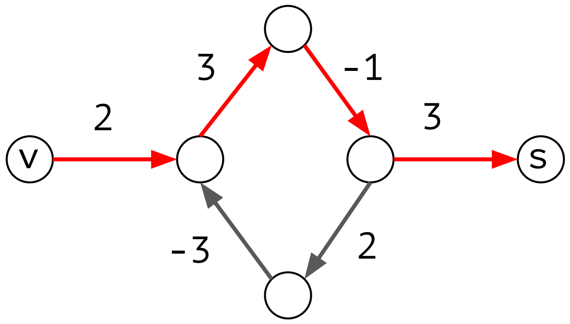
Наблюдение: рёбра отрицательного веса - это не проблема. А вот циклы отрицательного веса (для задач поиска кратчайшего пути) - это что-то странное.



Какой будет кратчайший путь из v в s?

Циклы отрицательного веса

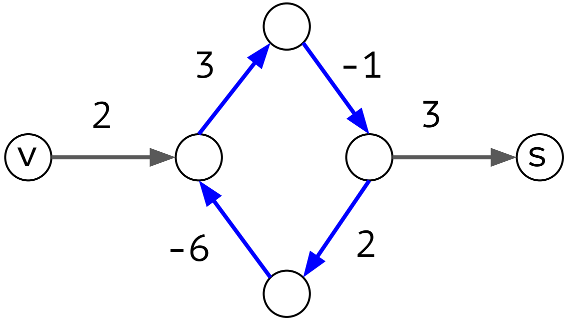
Наблюдение: рёбра отрицательного веса - это не проблема. А вот циклы отрицательного веса (для задач поиска кратчайшего пути) - это что-то странное.



Какой будет кратчайший путь из v в s?

Циклы отрицательного веса

Наблюдение: рёбра отрицательного веса - это не проблема. А вот циклы отрицательного веса (для задач поиска кратчайшего пути) - это что-то странное.

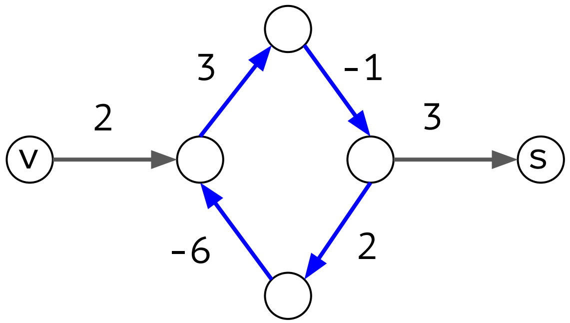


Какой будет кратчайший путь из v в s?

А если бы цикл был отрицательного веса?

Циклы отрицательного веса

Наблюдение: рёбра отрицательного веса - это не проблема. А вот циклы отрицательного веса (для задач поиска кратчайшего пути) - это что-то странное.



Какой будет кратчайший путь из v в s?

А если бы цикл был отрицательного веса?

На пути из v в s было бы выгодно накручивать петли по этому циклу, чтобы получать все меньший счет. Можно считать, что кратчайший путь $-\infty$.

Циклы отрицательного веса

Наблюдение: рёбра отрицательного веса - это не проблема. А вот циклы отрицательного веса (для задач поиска кратчайшего пути) - это что-то странное.

Варианты учитывания их в задачах:

1. Честно считать пути с учетом этих циклов, получать $-\infty$ (не очень практично и нужны правки в алгоритмы)

Циклы отрицательного веса

Наблюдение: рёбра отрицательного веса - это не проблема.
А вот циклы отрицательного веса (для задач поиска кратчайшего пути) - это что-то странное.

Варианты учитывания их в задачах:

1. Честно считать пути с учетом этих циклов, получать $-\infty$ (не очень практично и нужны правки в алгоритмы)

2. Находить кратчайшие пути, в которых нет циклов отрицательного веса (NP-трудная задача)

Циклы отрицательного веса

Наблюдение: рёбра отрицательного веса - это не проблема.
А вот циклы отрицательного веса (для задач поиска кратчайшего пути) - это что-то странное.

Варианты учитывания их в задачах:

1. Честно считать пути с учетом этих циклов, получать $-\infty$ (не очень практично и нужны правки в алгоритмы)

2. Находить кратчайшие пути, в которых нет циклов отрицательного веса (NP-трудная задача)

3. Решать задачи, где циклов нет (и обнаруживать факт их наличия и сами циклы)

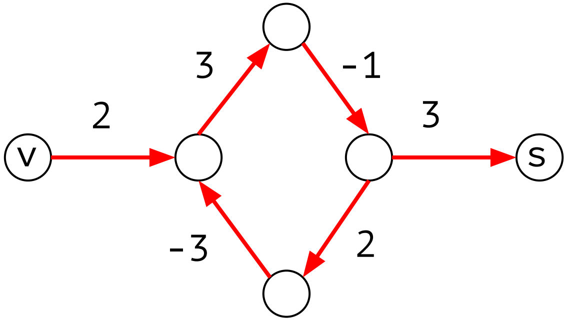
Циклы отрицательного веса

Пусть в графе нет циклов отрицательного веса. Тогда для любых двух (достижимых) вершин s и v, сколько может быть ребер в кратчайшем пути между ними?

Циклы отрицательного веса

Пусть в графе нет циклов отрицательного веса. Тогда для любых двух (достижимых) вершин s и v, сколько может быть ребер в кратчайшем пути между ними?

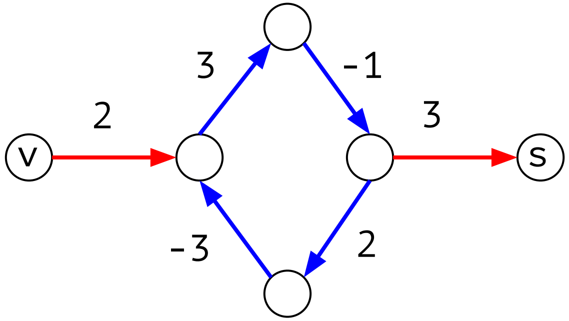
Пусть такой путь состоит хотя бы из |V| ребер =>



Циклы отрицательного веса

Пусть в графе нет циклов отрицательного веса. Тогда для любых двух (достижимых) вершин s и v, сколько может быть ребер в кратчайшем пути между ними?

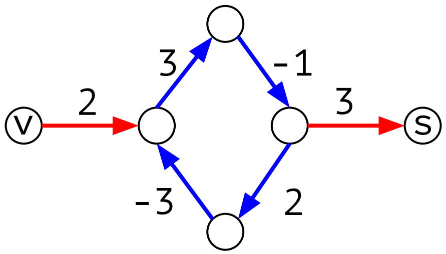
Пусть такой путь состоит хотя бы из |V| ребер => значит он соединяет |V| + 1 вершину, а значит какая-то из вершин повторяется, т.е. в пути есть цикл =>



Циклы отрицательного веса

Пусть в графе нет циклов отрицательного веса. Тогда для любых двух (достижимых) вершин s и v, сколько может быть ребер в кратчайшем пути между ними?

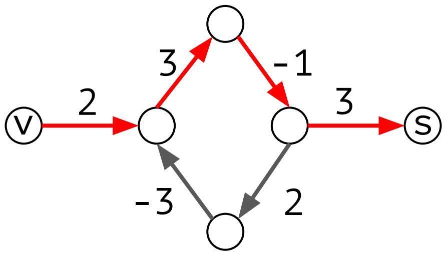
Пусть такой путь состоит хотя бы из |V| ребер => значит он соединяет |V| + 1 вершину, а значит какая-то из вершин повторяется, т.е. в пути есть цикл => но все циклы положительного веса, а значит, что этот цикл только удлиняет суммарный путь =>



Циклы отрицательного веса

Пусть в графе нет циклов отрицательного веса. Тогда для любых двух (достижимых) вершин s и v, сколько может быть ребер в кратчайшем пути между ними?

Пусть такой путь состоит хотя бы из |V| ребер => значит он соединяет |V| + 1 вершину, а значит какая-то из вершин повторяется, т.е. в пути есть цикл => но все циклы положительного веса, а значит, что этот цикл только удлиняет суммарный путь => в кратчайшем пути ⩽ |V|-1 ребер.



Задача поиска кратчайшего пути

Задача: пусть дан взвешенный граф $G = \langle V, E \rangle$.

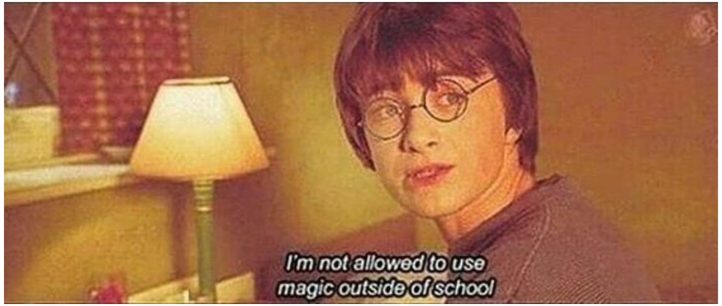
Если в графе нет циклов отрицательного веса, найти кратчайшие пути от заданной вершины s до всех остальных.

Задача поиска кратчайшего пути

Задача: пусть дан взвешенный граф $G = \langle V, E \rangle$.

Если в графе нет циклов отрицательного веса, найти кратчайшие пути от заданной вершины s до всех остальных.

Иначе, найти цикл отрицательного веса (это станет объяснением, почему не можем найти кратчайшие пути).



Оптимальная подструктура

Какую выбрать подструктуру в графовой задаче? В отличие от прошлых задач это не так очевидно.

Оптимальная подструктура

Какую выбрать подструктуру в графовой задаче? В отличие от прошлых задач это не так очевидно.

Например, выкидывать последнее из ребер из оптимального решения нельзя - полученное решение приведет вас уже в другую вершину!

Оптимальная подструктура

Какую выбрать подструктуру в графовой задаче? В отличие от прошлых задач это не так очевидно.

Например, выкидывать последнее из ребер из оптимального решения нельзя - полученное решение приведет вас уже в другую вершину!

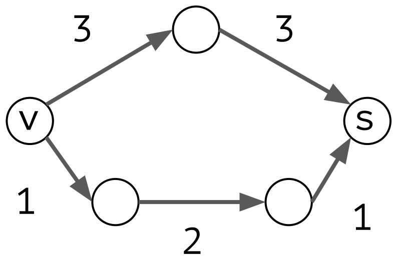
Вместо этого будем ограничивать количество ребер, которые можно использовать в пути.

Оптимальная подструктура

Какую выбрать подструктуру в графовой задаче? В отличие от прошлых задач это не так очевидно.

Например, выкидывать последнее из ребер из оптимального решения нельзя - полученное решение приведет вас уже в другую вершину!

Вместо этого будем ограничивать количество ребер, которые можно использовать в пути.



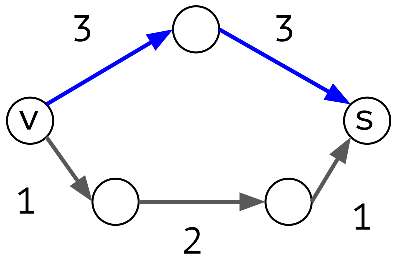
Оптимальная подструктура

Какую выбрать подструктуру в графовой задаче? В отличие от прошлых задач это не так очевидно.

Например, выкидывать последнее из ребер из оптимального решения нельзя - полученное решение приведет вас уже в другую вершину!

Вместо этого будем ограничивать количество ребер, которые можно использовать в пути.

1) Если можно брать максимум 2 ребра, то ответ - 6



Оптимальная подструктура

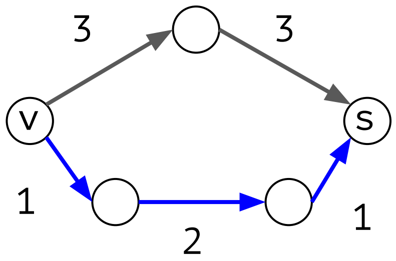
Какую выбрать подструктуру в графовой задаче? В отличие от прошлых задач это не так очевидно.

Например, выкидывать последнее из ребер из оптимального решения нельзя - полученное решение приведет вас уже в другую вершину!

Вместо этого будем ограничивать количество ребер, которые можно использовать в пути.

1) Если можно брать максимум 2 ребра, то ответ - 6

2) Если можно брать 3 ребра, то 4



Оптимальная подструктура

Пусть дан взвешенный граф $G = \langle V, E \rangle$ и вершина **s**, из которой будем искать кратчайшие пути в остальные вершины.

Оптимальная подструктура

Пусть дан взвешенный граф $G = \langle V, E \rangle$ и вершина **s**, из которой будем искать кратчайшие пути в остальные вершины.

Для каждой вершины $v \in V$ и $i \in \{1, 2, 3, \ldots\}$ введем $P_{v,i}$ - кратчайший путь от s до v, который использует не больше, чем i ребер.

Оптимальная подструктура

Пусть дан взвешенный граф $G = \langle V, E \rangle$ и вершина **s**, из которой будем искать кратчайшие пути в остальные вершины.

Для каждой вершины $v \in V$ и $i \in \{1, 2, 3, \ldots\}$ введем $P_{v,i}$ - кратчайший путь от s до v, который использует не больше, чем i ребер.

1-ый случай: в $P_{v,i}$ оказалось $\leq (i-1)$ ребер. Тогда: $P_{v,i} = P_{v,i-1}$ 🎉

Оптимальная подструктура

Пусть дан взвешенный граф $G = \langle V, E \rangle$ и вершина **s**, из которой будем искать кратчайшие пути в остальные вершины.

Для каждой вершины $v \in V$ и $i \in \{1, 2, 3, \dots\}$ введем $P_{v,i}$ - кратчайший путь от s до v, который использует не больше, чем i ребер.

1-ый случай: в $P_{v,i}$ оказалось $\leq (i-1)$ ребер. Тогда: $P_{v,i} = P_{v,i-1}$ 🎉

2-ой случай: в $P_{v,i}$ используется ровно **i** ребер.

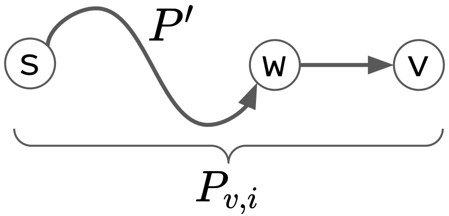
Оптимальная подструктура

Пусть дан взвешенный граф $G = \langle V, E \rangle$ и вершина **s**, из которой будем искать кратчайшие пути в остальные вершины.

Для каждой вершины $v \in V$ и $i \in \{1, 2, 3, \dots\}$ введем $P_{v,i}$ - кратчайший путь от s до v, который использует не больше, чем i ребер.

1-ый случай: в $P_{v,i}$ оказалось $\leq (i-1)$ ребер. Тогда: $P_{v,i} = P_{v,i-1}$ 🎉

2-ой случай: в $P_{v,i}$ используется ровно **i** ребер.



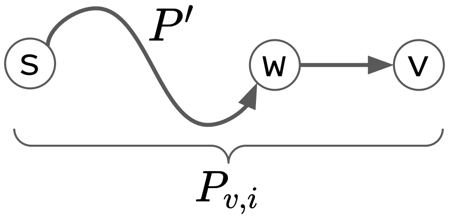
Оптимальная подструктура

Пусть дан взвешенный граф $G = \langle V, E \rangle$ и вершина **s**, из которой будем искать кратчайшие пути в остальные вершины.

Для каждой вершины $v \in V$ и $i \in \{1, 2, 3, \dots\}$ введем $P_{v,i}$ - кратчайший путь от s до v, который использует не больше, чем i ребер.

1-ый случай: в $P_{v,i}$ оказалось $\leq (i-1)$ ребер. Тогда: $P_{v,i} = P_{v,i-1}$ 🎉

2-ой случай: в $P_{v,i}$ используется ровно **i** ребер. Тогда утверждается, что $P' = P_{w,i-1}$.



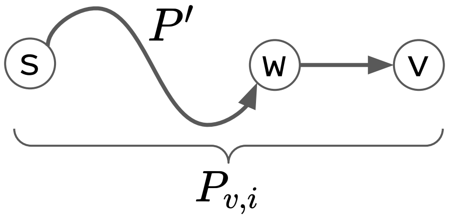
Оптимальная подструктура

Пусть дан взвешенный граф $G = \langle V, E \rangle$ и вершина **s**, из которой будем искать кратчайшие пути в остальные вершины.

Для каждой вершины $v \in V$ и $i \in \{1, 2, 3, \dots\}$ введем $P_{v,i}$ - кратчайший путь от s до v, который использует не больше, чем i ребер.

1-ый случай: в $P_{v,i}$ оказалось $\leq (i-1)$ ребер. Тогда: $P_{v,i} = P_{v,i-1}$ 🎉

2-ой случай: в $P_{v,i}$ используется ровно **i** ребер. Тогда утверждается, что $P' = P_{w,i-1}$. Действительно, покажем минимальность P'. Если нет, то есть P'' - кратчайший путь от s до w.



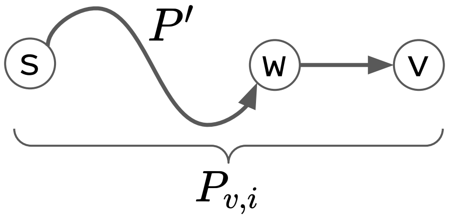
Оптимальная подструктура

Пусть дан взвешенный граф $G = \langle V, E \rangle$ и вершина **s**, из которой будем искать кратчайшие пути в остальные вершины.

Для каждой вершины $v \in V$ и $i \in \{1, 2, 3, \dots\}$ введем $P_{v,i}$ - кратчайший путь от s до v, который использует не больше, чем i ребер.

1-ый случай: в $P_{v,i}$ оказалось $\leq (i-1)$ ребер. Тогда: $P_{v,i} = P_{v,i-1}$ 🎉

2-ой случай: в $P_{v,i}$ используется ровно **i** ребер. Тогда утверждается, что $P' = P_{w,i-1}$. Действительно, покажем минимальность P'. Если нет, то есть P'' - кратчайший путь от s до w. Тогда $P'' + (w, v)$ короче $P_{v,i}$ => противоречие.



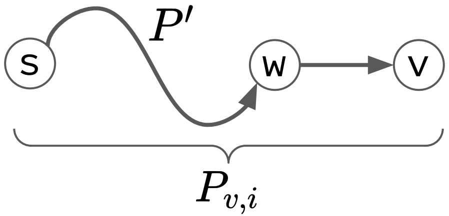
Оптимальная подструктура

Пусть дан взвешенный граф $G = \langle V, E \rangle$ и вершина **s**, из которой будем искать кратчайшие пути в остальные вершины.

Для каждой вершины $v \in V$ и $i \in \{1, 2, 3, \ldots\}$ введем $P_{v,i}$ - кратчайший путь от s до v, который использует не больше, чем i ребер.

1-ый случай: в $P_{v,i}$ оказалось $\leq (i-1)$ ребер. Тогда: $P_{v,i} = P_{v,i-1}$ 🎉

2-ой случай: в $P_{v,i}$ используется ровно **i** ребер. Тогда $P' = P_{w,i-1}$.



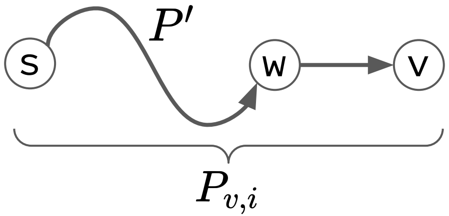
Оптимальная подструктура

Пусть дан взвешенный граф $G = \langle V, E \rangle$ и вершина **s**, из которой будем искать кратчайшие пути в остальные вершины.

Для каждой вершины $v \in V$ и $i \in \{1, 2, 3, \ldots\}$ введем $P_{v,i}$ - кратчайший путь от s до v, который использует не больше, чем i ребер.

1-ый случай: в $P_{v,i}$ оказалось $\le (i-1)$ ребер. Тогда: $P_{v,i} = P_{v,i-1}$ 🎉

2-ой случай: в $P_{v,i}$ используется ровно **i** ребер. Тогда $P' = P_{w,i-1}$.



А как через $P_{w,i-1}$ выражается $P_{v,i}$?

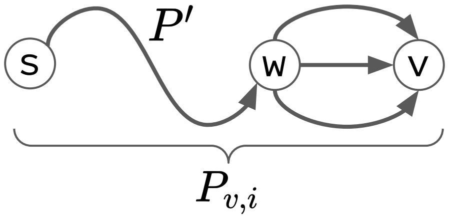
Оптимальная подструктура

Пусть дан взвешенный граф $G = \langle V, E \rangle$ и вершина **s**, из которой будем искать кратчайшие пути в остальные вершины.

Для каждой вершины $v \in V$ и $i \in \{1, 2, 3, \dots\}$ введем $P_{v,i}$ - кратчайший путь от s до v, который использует не больше, чем i ребер.

1-ый случай: в $P_{v,i}$ оказалось $\leq (i-1)$ ребер. Тогда: $P_{v,i} = P_{v,i-1}$ 🎉

2-ой случай: в $P_{v,i}$ используется ровно **i** ребер. Тогда $P' = P_{w,i-1}$.



А как через $P_{w,i-1}$ выражается $P_{v,i}$?

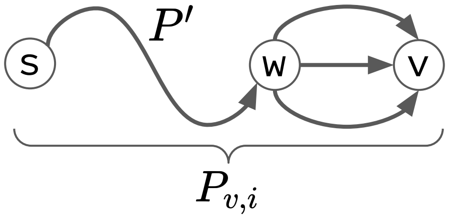
Оптимальная подструктура

Пусть дан взвешенный граф $G = \langle V, E \rangle$ и вершина **s**, из которой будем искать кратчайшие пути в остальные вершины.

Для каждой вершины $v \in V$ и $i \in \{1, 2, 3, \ldots\}$ введем $P_{v,i}$ - кратчайший путь от s до v, который использует не больше, чем i ребер.

1-ый случай: в $P_{v,i}$ оказалось $\leq (i-1)$ ребер. Тогда: $P_{v,i} = P_{v,i-1}$ 🎉

2-ой случай: в $P_{v,i}$ используется ровно **i** ребер. Тогда $P' = P_{w,i-1}$.



А как через $P_{w,i-1}$ выражается $P_{v,i}$?
$P_{v,i} = P_{w,i-1} + (w, v)$, где (w, v) - самое короткое ребро между w и v 🎉

Рекуррентное соотношение

Введем $L_{v,i}$- длину кратчайшего пути из **s** в **v**, использующего не больше, чем **i** ребер ($+\infty$, если такого пути нет).

Рекуррентное соотношение

Введем $L_{v,i}$- длину кратчайшего пути из **s** в **v**, использующего не больше, чем **i** ребер ($+\infty$, если такого пути нет).

Тогда для всех $v \in V$ и $i \in \{1, 2, 3, \dots\}$ верно:

$$L_{v,i} = \min \begin{cases} L_{v,i-1} \\ \min\limits_{(w,v) \in E} L_{w,i-1} + c_{w,v} \end{cases}$$

Здесь $c_{w,v}$ - вес ребра (w, v)

Рекуррентное соотношение

Введем $L_{v,i}$- длину кратчайшего пути из **s** в **v**, использующего не больше, чем **i** ребер ($+\infty$, если такого пути нет).

Тогда для всех $v \in V$ и $i \in \{1, 2, 3, \dots\}$ верно:

$$L_{v,i} = \min \begin{cases} L_{v,i-1} \\ \min\limits_{(w,v) \in E} L_{w,i-1} + c_{w,v} \end{cases}$$

Сколько нужно вычислить подзадач, чтобы посчитать $L_{v,i}$?

Здесь $c_{w,v}$ - вес ребра (w, v)

Рекуррентное соотношение

Введем $L_{v,i}$- длину кратчайшего пути из **s** в **v**, использующего не больше, чем **i** ребер ($+\infty$, если такого пути нет).

Тогда для всех $v \in V$ и $i \in \{1, 2, 3, \dots\}$ верно:

$$L_{v,i} = \min \begin{cases} L_{v,i-1} \\ \min\limits_{(w,v) \in E} L_{w,i-1} + c_{w,v} \end{cases}$$

Сколько нужно вычислить подзадач, чтобы посчитать $L_{v,i}$?

$$1 + in\_degree(v)$$

Здесь $c_{w,v}$ - вес ребра (w, v)

Рекуррентное соотношение

Введем $L_{v,i}$ - длину кратчайшего пути из **s** в **v**, использующего не больше, чем **i** ребер ($+\infty$, если такого пути нет).

Тогда для всех $v \in V$ и $i \in \{1, 2, 3, \dots\}$ верно:

$$L_{v,i} = \min \begin{cases} L_{v,i-1} \\ \min\limits_{(w,v) \in E} L_{w,i-1} + c_{w,v} \end{cases}$$

Сколько нужно вычислить подзадач, чтобы посчитать $L_{v,i}$?

$1 + in\_degree(v)$

от первого случая

Здесь $c_{w,v}$ - вес ребра (w, v)

Рекуррентное соотношение

Введем $L_{v,i}$ - длину кратчайшего пути из **s** в **v**, использующего не больше, чем **i** ребер ($+\infty$, если такого пути нет).

Тогда для всех $v \in V$ и $i \in \{1, 2, 3, \dots\}$ верно:

$$L_{v,i} = \min \begin{cases} L_{v,i-1} \\ \min\limits_{(w,v)\in E} L_{w,i-1} + c_{w,v} \end{cases}$$

Здесь $c_{w,v}$ - вес ребра (w, v)

Сколько нужно вычислить подзадач, чтобы посчитать $L_{v,i}$?

$$1 + \boxed{in\_degree(v)}$$

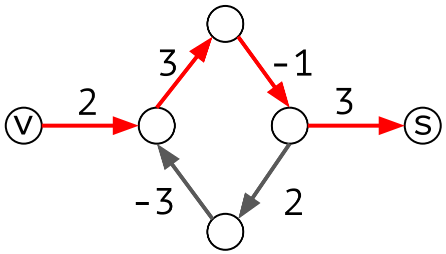
от второго случая

здесь in_degree(v) - количество входных дуг в v

Циклы отрицательного веса

Пусть в графе нет циклов отрицательного веса. Тогда для любых двух (достижимых) вершин s и v, сколько может быть ребер в кратчайшем пути между ними?

Пусть такой путь состоит хотя бы из |V| ребер => значит он соединяет |V| + 1 вершину, а значит какая-то из вершин повторяется, т.е. в пути есть цикл => но все циклы положительного веса, а значит, что этот цикл только удлиняет суммарный путь => в кратчайшем пути ⩽ |V|-1 ребер.



Рекуррентное соотношение

Если в графе нет циклов отрицательного веса, то $L_{v,i}$ нужно посчитать для $i \in \{1, 2, 3, \dots, |V| - 1\}$ и для всех $v \in V$.

$$L_{v,i} = \min \begin{cases} L_{v,i-1} \\ \min\limits_{(w,v)\in E} L_{w,i-1} + c_{w,v} \end{cases}$$

Алгоритм Беллмана-Форда

Заводим двумерный массив A. Первый индекс - вершина, куда ищем кратчайший путь, второй - доступное количество ребер.

$$L_{v,i} = \min \begin{cases} L_{v,i-1} \\ \min\limits_{(w,v)\in E} L_{w,i-1} + c_{w,v} \end{cases}$$

Алгоритм Беллмана-Форда

Заводим двумерный массив A. Первый индекс - вершина, куда ищем кратчайший путь, второй - доступное количество ребер.

Инициализация: ?

$$L_{v,i} = \min \begin{cases} L_{v,i-1} \\ \min\limits_{(w,v) \in E} L_{w,i-1} + c_{w,v} \end{cases}$$

Алгоритм Беллмана-Форда

Заводим двумерный массив A. Первый индекс - вершина, куда ищем кратчайший путь, второй - доступное количество ребер.

Инициализация: A[s][0] = 0, A[v][0] = $+\infty$, для остальных.

$$L_{v,i} = \min \begin{cases} L_{v,i-1} \\ \min\limits_{(w,v)\in E} L_{w,i-1} + c_{w,v} \end{cases}$$

Алгоритм Беллмана-Форда

Заводим двумерный массив A. Первый индекс - вершина, куда ищем кратчайший путь, второй - доступное количество ребер.

Инициализация: A[s][0] = 0, A[v][0] = $+\infty$, для остальных.

Заполняем массив:

```
for i in [1, |V|-1]:
  for v in V:
    A[v][i] = min(A[v][i-1], min(A[w][i-1] + C(w,v)))
```

$(w, v) \in E$

$$L_{v,i} = \min \begin{cases} L_{v,i-1} \\ \min\limits_{(w,v)\in E} L_{w,i-1} + c_{w,v} \end{cases}$$

Алгоритм Беллмана-Форда

Заводим двумерный массив A. Первый индекс - вершина, куда ищем кратчайший путь, второй - доступное количество ребер.

Инициализация: A[s][0] = 0, A[v][0] = $+\infty$, для остальных.

Заполняем массив:

```
for i in [1, |V|-1]:
  for v in V:
    A[v][i] = min(A[v][i-1], min(A[w][i-1] + C(w,v)))
```

$(w, v) \in E$

Сложность?

$$L_{v,i} = \min \begin{cases} L_{v,i-1} \\ \min\limits_{(w,v)\in E} L_{w,i-1} + c_{w,v} \end{cases}$$

Алгоритм Беллмана-Форда

Заводим двумерный массив A. Первый индекс - вершина, куда ищем кратчайший путь, второй - доступное количество ребер.

Инициализация: A[s][0] = 0, A[v][0] = $+\infty$, для остальных.

Заполняем массив:

$O(|V|$

```
for i in [1, |V|-1]:
  for v in V:
    A[v][i] = min(A[v][i-1], min(A[w][i-1] + C(w,v)))
```

$(w, v) \in E$

Сложность?

$$L_{v,i} = \min \begin{cases} L_{v,i-1} \\ \min\limits_{(w,v)\in E} L_{w,i-1} + c_{w,v} \end{cases}$$

Алгоритм Беллмана-Форда

Заводим двумерный массив A. Первый индекс - вершина, куда ищем кратчайший путь, второй - доступное количество ребер.

Инициализация: A[s][0] = 0, A[v][0] = $+\infty$, для остальных.

Заполняем массив:

$$O(|V| * |E|)$$

суммарно - проход по всем ребрам

```
for i in [1, |V|-1]:
  for v in V:
    A[v][i] = min(A[v][i-1], min(A[w][i-1] + C(w,v)))
```

$(w,v) \in E$

Сложность? $O(|V| * |E|)$

$$L_{v,i} = \min \begin{cases} L_{v,i-1} \\ \min\limits_{(w,v)\in E} L_{w,i-1} + c_{w,v} \end{cases}$$

Алгоритм Беллмана-Форда

Заводим двумерный массив A. Первый индекс - вершина, куда ищем кратчайший путь, второй - доступное количество ребер.

Инициализация: A[s][0] = 0, A[v][0] = $+\infty$, для остальных.

Заполняем массив:

$$O(|V| * |E|)$$

суммарно - проход по всем ребрам

```
for i in [1, |V|-1]:
  for v in V:
    A[v][i] = min(A[v][i-1], min(A[w][i-1] + C(w,v)))
```

$(w,v) \in E$

Сложность? $O(|V| * |E|)$. Дейкстра (через обычные хипы) при этом работает за $O((|V| + |E|)log(|V|))$

$$L_{v,i} = \min \begin{cases} L_{v,i-1} \\ \min\limits_{(w,v)\in E} L_{w,i-1} + c_{w,v} \end{cases}$$

Алгоритм Беллмана-Форда

Заводим двумерный массив A. Первый индекс - вершина, куда ищем кратчайший путь, второй - доступное количество ребер.

Инициализация: A[s][0] = 0, A[v][0] = $+\infty$, для остальных.

Заполняем массив:

$O(|V| * |E|)$

суммарно - проход по всем ребрам

```
for i in [1, |V|-1]:
  for v in V:
    A[v][i] = min(A[v][i-1], min(A[w][i-1] + C(w,v)))
```

$(w,v) \in E$

Сложность? $O(|V| * |E|)$. Дейкстра (через обычные хипы) при этом работает за $O((|V| + |E|)log(|V|))$. Т.е. для разреженных графов сложность сравнимая.

Алгоритм Беллмана-Форда: циклы отр. веса

Беллман-Форд работает с отрицательными весами (в отличие от Дейкстры), но что с циклами отрицательного веса?

Алгоритм Беллмана-Форда: циклы отр. веса

Беллман-Форд работает с отрицательными весами (в отличие от Дейкстры), но что с циклами отрицательного веса?

Утверждение: в графе **нет** циклов отрицательного веса ⇔ после дополнительной итерации алгоритма Беллмана-Форда будет справедливо A[v][|V|] = A[v][|V|-1] для всех v.

$$L_{v,i} = \min \begin{cases} L_{v,i-1} \\ \min\limits_{(w,v)\in E} L_{w,i-1} + c_{w,v} \end{cases}$$

Алгоритм Беллмана-Форда

Заводим двумерный массив A. Первый индекс - вершина, куда ищем кратчайший путь, второй - доступное количество ребер.

Инициализация: A[s][0] = 0, A[v][0] = $+\infty$, для остальных.

Заполняем массив:

добавим еще одну итерацию до |V|

```
for i in [1, |V|-1]:
  for v in V:
    A[v][i] = min(A[v][i-1], min(A[w][i-1] + C(w,v)))
```

$(w, v) \in E$

Алгоритм Беллмана-Форда: циклы отр. веса

Беллман-Форд работает с отрицательными весами (в отличие от Дейкстры), но что с циклами отрицательного веса?

Утверждение: в графе **нет** циклов отрицательного веса ⇔ после дополнительной итерации алгоритма Беллмана-Форда будет справедливо A[v][|V|] = A[v][|V|-1] для всех v.

Док-во: ⇒ следует из максимального количества ребер в кратчайшем пути (|V|-1) при отсутствии циклов отр. веса.

Алгоритм Беллмана-Форда: циклы отр. веса

Беллман-Форд работает с отрицательными весами (в отличие от Дейкстры), но что с циклами отрицательного веса?

Утверждение: в графе **нет** циклов отрицательного веса ⇔ после дополнительной итерации алгоритма Беллмана-Форда будет справедливо A[v][|V|] = A[v][|V|-1] для всех v.

Док-во: ⇐ пусть после доп. итерации выяснилось, что A[v][|V|] = A[v][|V|-1] для всех v.

Алгоритм Беллмана-Форда: циклы отр. веса

Беллман-Форд работает с отрицательными весами (в отличие от Дейкстры), но что с циклами отрицательного веса?

Утверждение: в графе **нет** циклов отрицательного веса ⇔ после дополнительной итерации алгоритма Беллмана-Форда будет справедливо A[v][|V|] = A[v][|V|-1] для всех v.

Док-во: ⇐ пусть после доп. итерации выяснилось, что A[v][|V|] = A[v][|V|-1] для всех v.

```
for i in [1, |V|-1]:
  for v in V:
    A[v][i] = min(A[v][i-1], min(A[w][i-1] + C(w,v)))
```

$$\min_{(w,v) \in E}$$

Алгоритм Беллмана-Форда: циклы отр. веса

Беллман-Форд работает с отрицательными весами (в отличие от Дейкстры), но что с циклами отрицательного веса?

Утверждение: в графе **нет** циклов отрицательного веса ⇔ после дополнительной итерации алгоритма Беллмана-Форда будет справедливо A[v][|V|] = A[v][|V|-1] для всех v.

Док-во: ⇐ пусть после доп. итерации выяснилось, что A[v][|V|] = A[v][|V|-1] для всех v.

т.е. мы всегда выбираем A[v][|V|-1]

```
for i in [1, |V|-1]:
  for v in V:
    A[v][i] = min(A[v][i-1], min(A[w][i-1] + C(w,v)))
                                 (w,v) ∈ E
```

Алгоритм Беллмана-Форда: циклы отр. веса

Док-во: ⇐ пусть после доп. итерации выяснилось, что A[v][|V|] = A[v][|V|-1] для всех v.

т.е. мы всегда выбираем A[v][|V|-1]

```
for i in [1, |V|-1]:
  for v in V:
    A[v][i] = min(A[v][i-1], min(A[w][i-1] + C(w,v)))
```

$(w, v) \in E$

Алгоритм Беллмана-Форда: циклы отр. веса

Док-во: ⇐ пусть после доп. итерации выяснилось, что A[v][|V|] = A[v][|V|-1] для всех v.

т.е. мы всегда выбираем A[v][|V|-1]

```
for i in [1, |V|-1]:
  for v in V:
    A[v][i] = min(A[v][i-1], min(A[w][i-1] + C(w,v)))
```

$(w, v) \in E$

Введем: $d(v) = A[v][|V|] = A[v][|V| - 1]$

Алгоритм Беллмана-Форда: циклы отр. веса

Док-во: $\Leftarrow$ пусть после доп. итерации выяснилось, что A[v][|V|] = A[v][|V|-1] для всех v.

т.е. мы всегда выбираем A[v][|V|-1]

```
for i in [1, |V|-1]:
  for v in V:
    A[v][i] = min(A[v][i-1], min(A[w][i-1] + C(w,v)))
```

$(w,v) \in E$

Введем: $d(v) = A[v][|V|] = A[v][|V|-1]$, тогда $\forall v, w \in V : d(v) \leq d(w) + c_{w,v}$

Алгоритм Беллмана-Форда: циклы отр. веса

Док-во: $\Leftarrow$ пусть после доп. итерации выяснилось, что A[v][|V|] = A[v][|V|-1] для всех v.

т.е. мы всегда выбираем A[v][|V|-1]

```
for i in [1, |V|-1]:
  for v in V:
    A[v][i] = min(A[v][i-1], min(A[w][i-1] + C(w,v)))
```

$(w, v) \in E$

Введем: $d(v) = A[v][|V|] = A[v][|V| - 1]$, тогда $\forall v, w \in V : d(v) \leq d(w) + c_{w,v}$

Выберем произвольный цикл C.

Алгоритм Беллмана-Форда: циклы отр. веса

Док-во: ⇐ пусть после доп. итерации выяснилось, что A[v][|V|] = A[v][|V|-1] для всех v.

т.е. мы всегда выбираем A[v][|V|-1]

```
for i in [1, |V|-1]:
  for v in V:
    A[v][i] = min(A[v][i-1], min(A[w][i-1] + C(w,v)))
```

$(w,v) \in E$

Введем: $d(v) = A[v][|V|] = A[v][|V|-1]$, тогда $\forall v, w \in V : d(v) \le d(w) + c_{w,v}$

Выберем произвольный цикл C. Тогда $\sum_{(v,w) \in C} c_{v,w} \ge$

Алгоритм Беллмана-Форда: циклы отр. веса

Док-во: $\Leftarrow$ пусть после доп. итерации выяснилось, что A[v][|V|] = A[v][|V|-1] для всех v.

т.е. мы всегда выбираем A[v][|V|-1]

```
for i in [1, |V|-1]:
  for v in V:
    A[v][i] = min(A[v][i-1], min(A[w][i-1] + C(w,v)))
```

$(w,v) \in E$

Введем: $d(v) = A[v][|V|] = A[v][|V|-1]$, тогда $\forall v, w \in V : d(v) \leq d(w) + c_{w,v}$

Выберем произвольный цикл C. Тогда $\sum\limits_{(v,w)\in C} c_{v,w} \geq \sum\limits_{(v,w)\in C} d(v) - d(w)$

Алгоритм Беллмана-Форда: циклы отр. веса

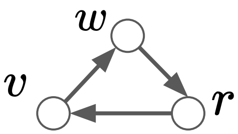
Док-во: $\Leftarrow$ пусть после доп. итерации выяснилось, что A[v][|V|] = A[v][|V|-1] для всех v.

т.е. мы всегда выбираем A[v][|V|-1]

```
for i in [1, |V|-1]:
  for v in V:
    A[v][i] = min(A[v][i-1], min(A[w][i-1] + C(w,v)))
                             (w,v) ∈ E
```

Введем: $d(v) = A[v][|V|] = A[v][|V|-1]$, тогда $\forall v, w \in V : d(v) \leq d(w) + c_{w,v}$

Выберем произвольный цикл C. Тогда $\sum\limits_{(v,w) \in C} c_{v,w} \geq \sum\limits_{(v,w) \in C} d(v) - d(w)$



Алгоритм Беллмана-Форда: циклы отр. веса

Док-во: ⇐ пусть после доп. итерации выяснилось, что A[v][|V|] = A[v][|V|-1] для всех v.

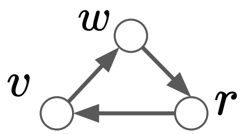
т.е. мы всегда выбираем A[v][|V|-1]

```
for i in [1, |V|-1]:
  for v in V:
    A[v][i] = min(A[v][i-1], min(A[w][i-1] + C(w,v)))
```

$(w,v) \in E$

Введем: $d(v) = A[v][|V|] = A[v][|V|-1]$, тогда $\forall v, w \in V : d(v) \leq d(w) + c_{w,v}$

Выберем произвольный цикл C. Тогда $\sum\limits_{(v,w)\in C} c_{v,w} \geq \sum\limits_{(v,w)\in C} d(v) - d(w)$



$(d(v) - d(w)) + (d(w) - d(r)) + (d(r) - d(v)) = 0$

Алгоритм Беллмана-Форда: циклы отр. веса

Док-во: ⇐ пусть после доп. итерации выяснилось, что A[v][|V|] = A[v][|V|-1] для всех v.

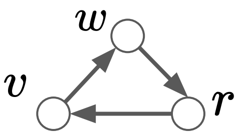
т.е. мы всегда выбираем A[v][|V|-1]

```
for i in [1, |V|-1]:
  for v in V:
    A[v][i] = min(A[v][i-1], min(A[w][i-1] + C(w,v)))
```

$(w,v) \in E$

Введем: $d(v) = A[v][|V|] = A[v][|V|-1]$, тогда $\forall v, w \in V : d(v) \leq d(w) + c_{w,v}$

Выберем произвольный цикл C. Тогда $\sum\limits_{(v,w)\in C} c_{v,w} \geq \sum\limits_{(v,w)\in C} d(v) - d(w) = 0$ □



$$(d(v) - d(w)) + (d(w) - d(r)) + (d(r) - d(v)) = 0$$

Алгоритм Беллмана-Форда: циклы отр. веса

Беллман-Форд работает с отрицательными весами (в отличие от Дейкстры), но что с циклами отрицательного веса?

Утверждение: в графе **нет** циклов отрицательного веса ⇔ после дополнительной итерации алгоритма Беллмана-Форда будет справедливо A[v][|V|] = A[v][|V|-1] для всех v. □

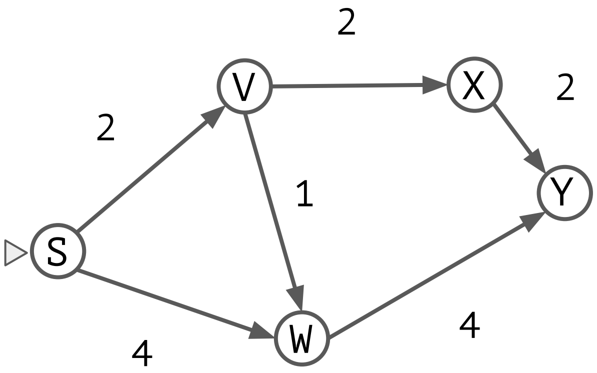
Алгоритм Беллмана-Форда: циклы отр. веса

Беллман-Форд работает с отрицательными весами (в отличие от Дейкстры), но что с циклами отрицательного веса?

Утверждение: в графе **нет** циклов отрицательного веса ⇔ после дополнительной итерации алгоритма Беллмана-Форда будет справедливо A[v][|V|] = A[v][|V|-1] для всех v. □

Тогда, чтобы понять, есть ли циклы отрицательного веса, нужно просто сделать одну дополнительную итерацию алгоритма. Если что-то поменялось => цикл есть.

Алгоритм Беллмана-Форда: пример



$$L_{v,i} = \min \begin{cases} L_{v,i-1} \\ \min\limits_{(w,v)\in E} L_{w,i-1} + c_{w,v} \end{cases}$$

Алгоритм Беллмана-Форда

Заводим двумерный массив A. Первый индекс - вершина, куда ищем кратчайший путь, второй - доступное количество ребер.

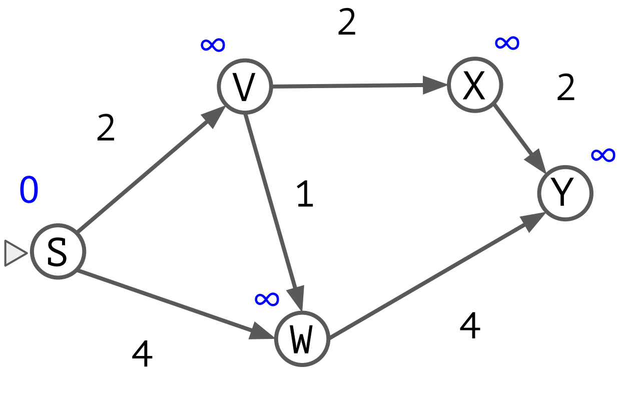
Инициализация: A[s][0] = 0, A[v][0] = $+\infty$, для остальных.

Заполняем массив:

```
for i in [1, |V|-1]:
  for v in V:
    A[v][i] = min(A[v][i-1], min(A[w][i-1] + C(w,v)))
```

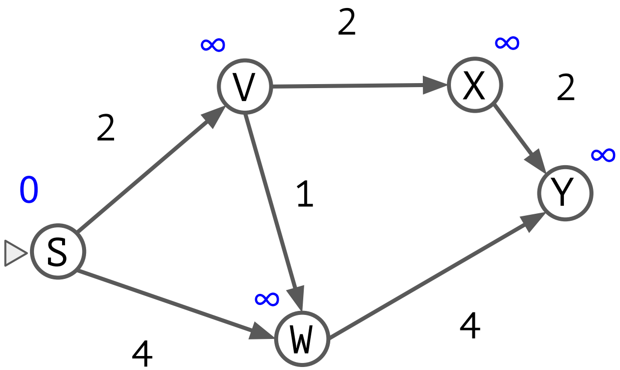
$(w, v) \in E$

Алгоритм Беллмана-Форда: пример



S	0					
V	∞					
W	∞					
X	∞					
Y	∞					

Алгоритм Беллмана-Форда: пример

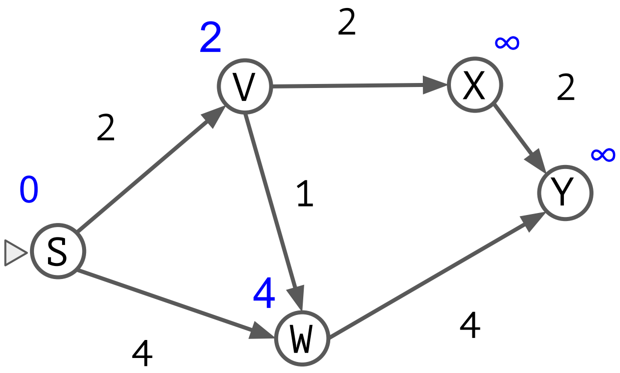


S	0					
V	∞					
W	∞					
X	∞					
Y	∞					

```
for i in [1, |V|-1]:
  for v in V:
    A[v][i] = min(A[v][i-1], min(A[w][i-1] + C(w,v)))
```

$(w, v) \in E$

Алгоритм Беллмана-Форда: пример

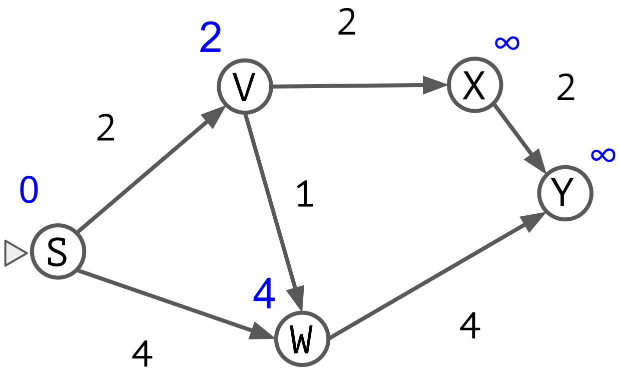


S	0	0				
V	∞	2				
W	∞	4				
X	∞					
Y	∞					

```
for i in [1, |V|-1]:
  for v in V:
    A[v][i] = min(A[v][i-1], min(A[w][i-1] + C(w,v)))
```

$$\min_{(w,v) \in E}$$

Алгоритм Беллмана-Форда: пример

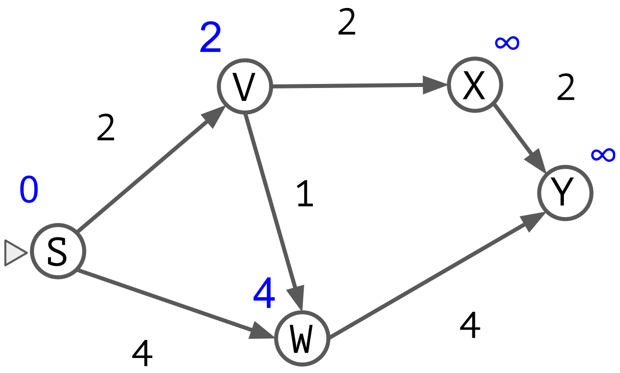


S	0	0				
V	∞	2				
W	∞	4				
X	∞	∞				
Y	∞	∞				

```
for i in [1, |V|-1]:
  for v in V:
    A[v][i] = min(A[v][i-1], min(A[w][i-1] + C(w,v)))
```

$(w,v) \in E$

Алгоритм Беллмана-Форда: пример

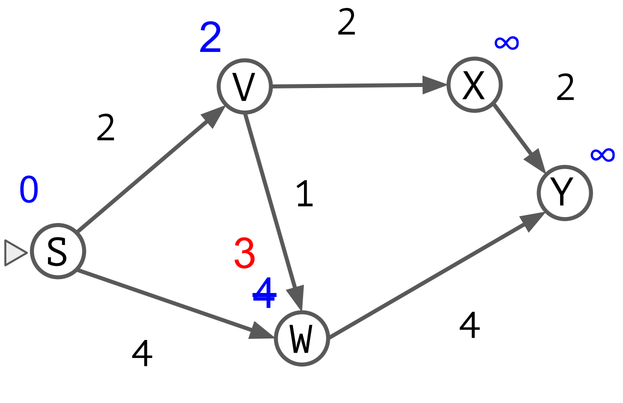


S	0	0	0			
V	∞	2	2			
W	∞	4				
X	∞	∞				
Y	∞	∞				

```
for i in [1, |V|-1]:
  for v in V:
    A[v][i] = min(A[v][i-1], min(A[w][i-1] + C(w,v)))
```

$$\min_{(w,v) \in E}$$

Алгоритм Беллмана-Форда: пример

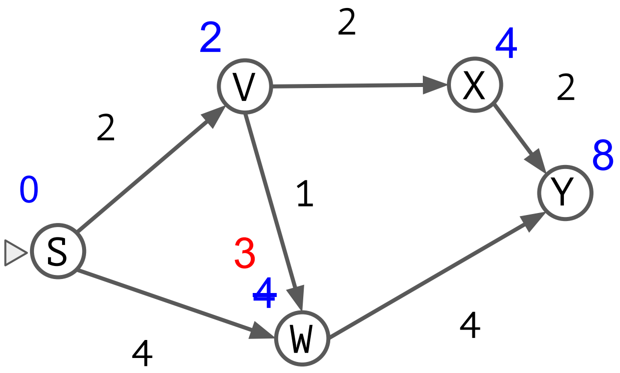


S	0	0	0			
V	∞	2	2			
W	∞	4	3			
X	∞	∞				
Y	∞	∞				

```
for i in [1, |V|-1]:
  for v in V:
    A[v][i] = min(A[v][i-1], min(A[w][i-1] + C(w,v)))
```

$(w, v) \in E$

Алгоритм Беллмана-Форда: пример

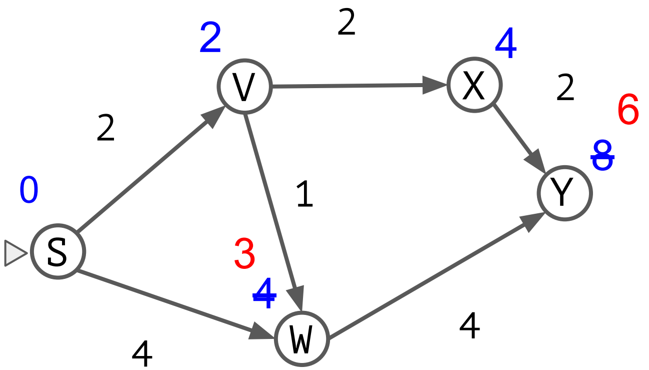


S	0	0	0			
V	∞	2	2			
W	∞	4	3			
X	∞	∞	4			
Y	∞	∞	8			

```
for i in [1, |V|-1]:
  for v in V:
    A[v][i] = min(A[v][i-1], min(A[w][i-1] + C(w,v)))
```

$(w,v) \in E$

Алгоритм Беллмана-Форда: пример



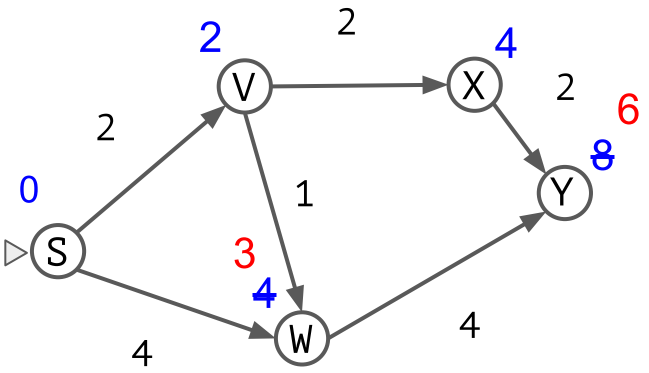
S	0	0	0	0		
V	∞	2	2	2		
W	∞	4	3	3		
X	∞	∞	4	4		
Y	∞	∞	8	6		

```
for i in [1, |V|-1]:
  for v in V:
    A[v][i] = min(A[v][i-1], min(A[w][i-1] + C(w,v)))
```

$(w, v) \in E$

Алгоритм Беллмана-Форда: пример

Еще одна итерация в холостую, изменений не будет



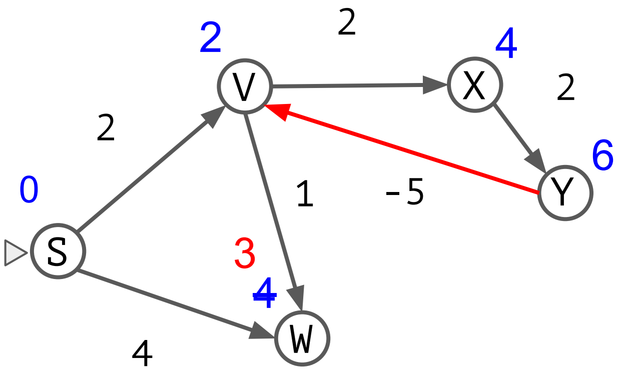
S	0	0	0	0	0	
V	∞	2	2	2	2	
W	∞	4	3	3	3	
X	∞	∞	4	4	4	
Y	∞	∞	8	6	6	

Ответы в этой колонке

```
for i in [1, |V|-1]:
  for v in V:
    A[v][i] = min(A[v][i-1], min(A[w][i-1] + C(w,v)))
```

$$\min_{(w,v)\in E}$$

Алгоритм Беллмана-Форда: пример



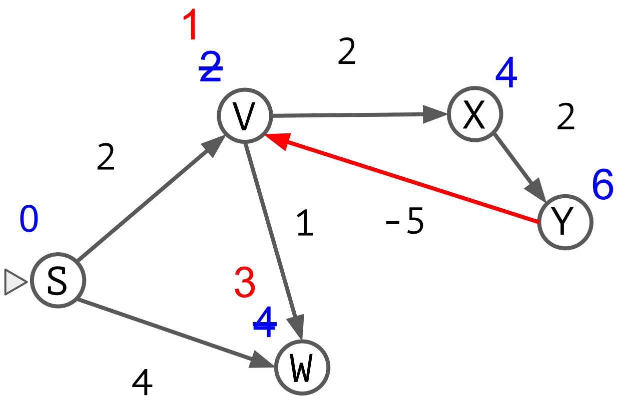
S	0	0	0	0		
V	∞	2	2	2		
W	∞	4	3	3		
X	∞	∞	4	4		
Y	∞	∞	∞	6		

```
for i in [1, |V|-1]:
  for v in V:
    A[v][i] = min(A[v][i-1], min(A[w][i-1] + C(w,v)))
```

$(w, v) \in E$

Алгоритм Беллмана-Форда: пример

Пока штатная ситуация, может это и не цикл отр. веса



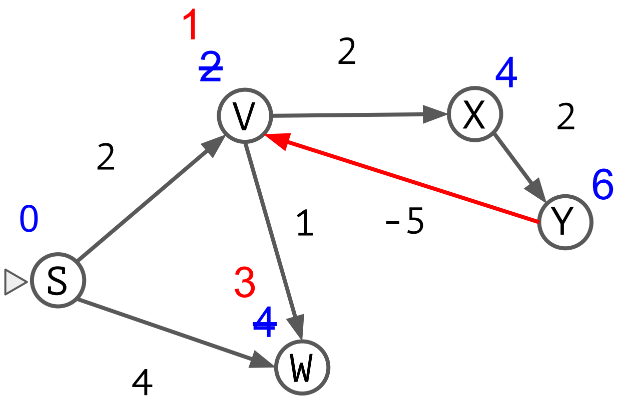
S	0	0	0	0	0	
V	∞	2	2	2	1	
W	∞	4	3	3	3	
X	∞	∞	4	4	4	
Y	∞	∞	∞	6	6	

```
for i in [1, |V|-1]:
  for v in V:
    A[v][i] = min(A[v][i-1], min(A[w][i-1] + C(w,v)))
```

$(w,v) \in E$

Алгоритм Беллмана-Форда: пример

А вот теперь мы уверены, что цикл отр. веса есть!



S	0	0	0	0	0	0
V	∞	2	2	2	1	1
W	∞	4	3	3	3	**2**
X	∞	∞	4	4	4	**3**
Y	∞	∞	∞	6	6	6

```
for i in [1, |V|-1]:
  for v in V:
    A[v][i] = min(A[v][i-1], min(A[w][i-1] + C(w,v)))
```

$(w,v) \in E$

Алгоритм Беллмана-Форда: детали

1. Как находить кратчайшие пути (и циклы отрицательного веса)?

Алгоритм Беллмана-Форда: детали

1. Как находить кратчайшие пути (и циклы отрицательного веса)?

   Как всегда: идем по таблице в обратную сторону. Для циклов отрицательного веса смотрим от мест, где на дополнительной итерации что-то поменялось.

Алгоритм Беллмана-Форда: детали

1. Как находить кратчайшие пути (и циклы отрицательного веса)?

   Как всегда: идем по таблице в обратную сторону. Для циклов отрицательного веса смотрим от мест, где на дополнительной итерации что-то поменялось.

2. Можно ли остановить алгоритм раньше?

Алгоритм Беллмана-Форда: детали

1. Как находить кратчайшие пути (и циклы отрицательного веса)?

   Как всегда: идем по таблице в обратную сторону. Для циклов отрицательного веса смотрим от мест, где на дополнительной итерации что-то поменялось.

2. Можно ли остановить алгоритм раньше?

   Да, если ничего не поменялось на очередной итерации, то уже и не поменяется, можно останавливаться.

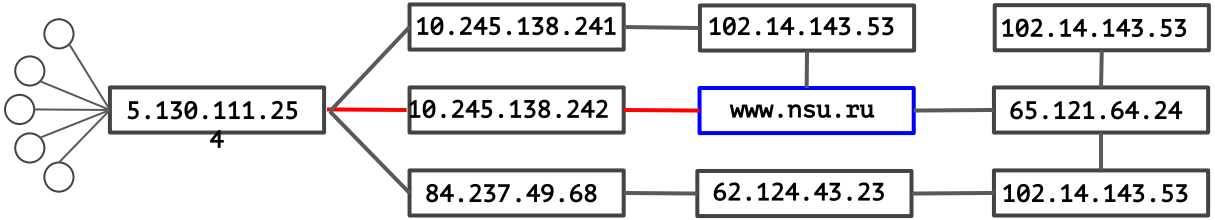
Сетевой протокол RIP

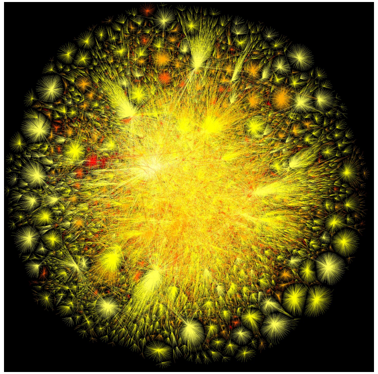
RIP (**R**outing **I**nformation **P**rotocol) - протокол маршрутизации сети, основанный на алгоритме Беллмана-Форда.

Сетевой протокол RIP

RIP (**R**outing **I**nformation **P**rotocol) - протокол маршрутизации сети, основанный на алгоритме Беллмана-Форда.

Решает (решал) проблему поиска кратчайшего пути в слишком больших графах.





Алгоритм Беллмана-Форда

```
Инициализация: A[s][0] = 0, A[v][0] = +∞ , для остальных.

for i in [1, |V|-1]:
  for v in V:
    A[v][i] = min(A[v][i-1], min(A[w][i-1] + C(w,v)))
```

$$A[v][i] = \min\left(A[v][i-1],\ \min_{(w,v)\in E}\left(A[w][i-1] + C(w,v)\right)\right)$$

Алгоритм Беллмана-Форда

Инициализация: A[s][0] = 0, A[v][0] = $+\infty$, для остальных.

```
for i in [1, |V|-1]:
  for v in V:
    A[v][i] = min(A[v][i-1], min(A[w][i-1] + C(w,v)))
```

$$\min_{(w,v) \in E}$$

Интуитивно понятно, что алгоритм можно превратить в распределенный. Действительно, для каждого A[v][i] нам нужна информация только о результатах с прошлой итерации.

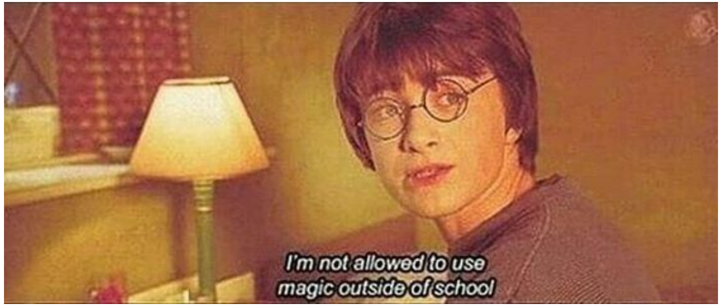
Весь граф не нужен, в память его грузить необязательно. Что нужно поменять в алгоритме?

Задача поиска кратчайшего пути

Задача: пусть дан взвешенный граф $G = \langle V, E \rangle$.

Если в графе нет циклов отрицательного веса, найти кратчайшие пути от заданной вершины s до всех остальных.

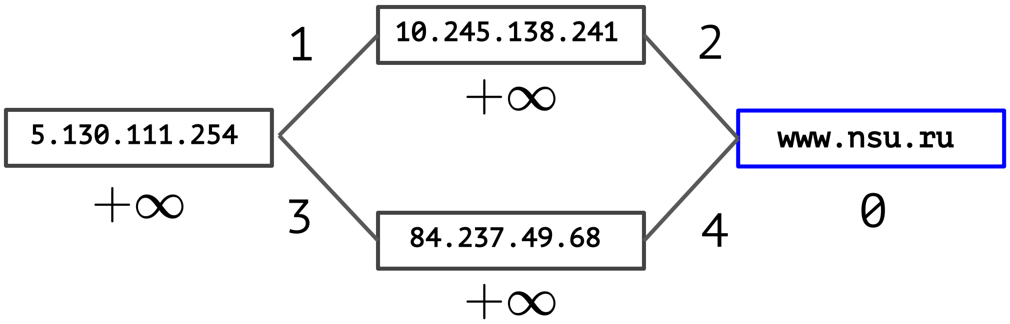
Иначе, найти цикл отрицательного веса (это станет объяснением, почему не можем найти кратчайшие пути).



Сетевой протокол RIP

RIP (**R**outing **I**nformation **P**rotocol) - протокол маршрутизации сети, основанный на алгоритме Беллмана-Форда.

1) В отличие от классического Беллмана-Форда алгоритм становится не source-oriented, а destination-oriented.

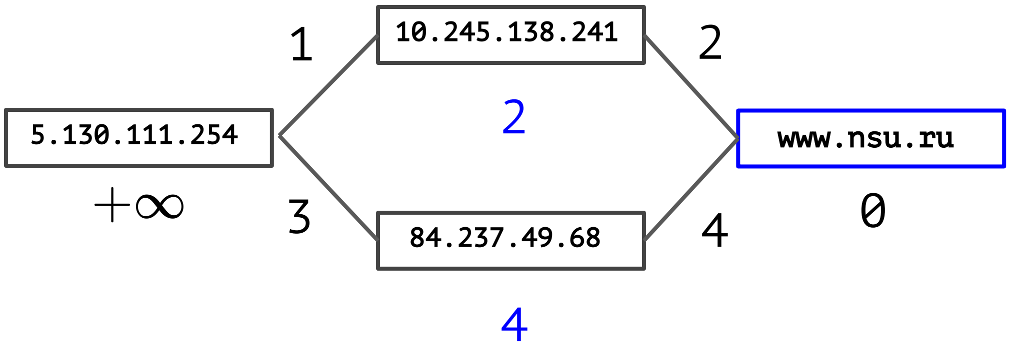


Будем искать кратчайшие пути **в** www.nsu.ru

Сетевой протокол RIP

RIP (**R**outing **I**nformation **P**rotocol) - протокол маршрутизации сети, основанный на алгоритме Беллмана-Форда.

1) В отличие от классического Беллмана-Форда алгоритм становится не source-oriented, а destination-oriented.

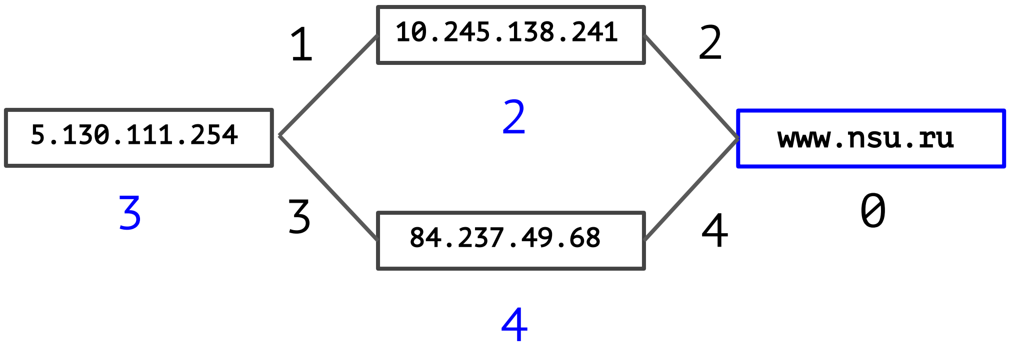


Будем искать кратчайшие пути **в** www.nsu.ru

Сетевой протокол RIP

RIP (**R**outing **I**nformation **P**rotocol) - протокол маршрутизации сети, основанный на алгоритме Беллмана-Форда.

1) В отличие от классического Беллмана-Форда алгоритм становится не source-oriented, а destination-oriented.



Будем искать кратчайшие пути **в** www.nsu.ru

Сетевой протокол RIP

RIP (**R**outing **I**nformation **P**rotocol) - протокол маршрутизации сети, основанный на алгоритме Беллмана-Форда.

1) В отличие от классического Беллмана-Форда алгоритм становится не source-oriented, а destination-oriented.

   Ищем для конкретной вершины **t** кратчайшие пути ИЗ остальных вершин в нее.

   Каждая вершина хранит кратчайший путь в **t** и ближайший узел на этом кратчайшем пути.

Сетевой протокол RIP

RIP (**R**outing **I**nformation **P**rotocol) - протокол маршрутизации сети, основанный на алгоритме Беллмана-Форда.

1) В отличие от классического Беллмана-Форда алгоритм становится не source-oriented, а destination-oriented.

   Ищем для конкретной вершины **t** кратчайшие пути ИЗ остальных вершин в нее.

   Каждая вершина хранит кратчайший путь в **t** и ближайший узел на этом кратчайшем пути.

   (обычно t - это не любая вершина в интернете, а gateway подсети. Во "внешнем" интернете узлов не так много: сотни тысяч)

Сетевой протокол RIP

RIP (**R**outing **I**nformation **P**rotocol) - протокол маршрутизации сети, основанный на алгоритме Беллмана-Форда.

1) source-oriented -> destination-oriented

2) асинхронность. Раньше для получения A[i][v] мы ждали, пока посчитаются **все** A[i-1][w]

Алгоритм Беллмана-Форда

Инициализация: A[s][0] = 0, A[v][0] = $+\infty$, для остальных.

```
for i in [1, |V|-1]:
  for v in V:
    A[v][i] = min(A[v][i-1], min(A[w][i-1] + C(w,v)))
```

$\min_{(w,v) \in E}$

Сетевой протокол RIP

RIP (**R**outing **I**nformation **P**rotocol) - протокол маршрутизации сети, основанный на алгоритме Беллмана-Форда.

1) source-oriented -> destination-oriented

2) асинхронность. Раньше для получения A[i][v] мы ждали, пока посчитаются **все** A[i-1][w].

   Но так ли это нужно?

Алгоритм Беллмана-Форда

```
Инициализация: A[s][0] = 0, A[v][0] = +∞ , для остальных.

for i in [1, |V|-1]:
  for v in V:
    A[v][i] = min(A[v][i-1], min(A[w][i-1] + C(w,v)))
                                (w,v) ∈ E
```

Сетевой протокол RIP

RIP (**R**outing **I**nformation **P**rotocol) - протокол маршрутизации сети, основанный на алгоритме Беллмана-Форда.

1) source-oriented -> destination-oriented

2) асинхронность. Раньше для получения A[i][v] мы ждали, пока посчитаются **все** A[i-1][w].

   Но так ли это нужно? Нет! Нужны только A[i-1][v] и нужны A[i-1][w], где w - соседи v.

Сетевой протокол RIP

RIP (**R**outing **I**nformation **P**rotocol) - протокол маршрутизации сети, основанный на алгоритме Беллмана-Форда.

1) source-oriented -> destination-oriented

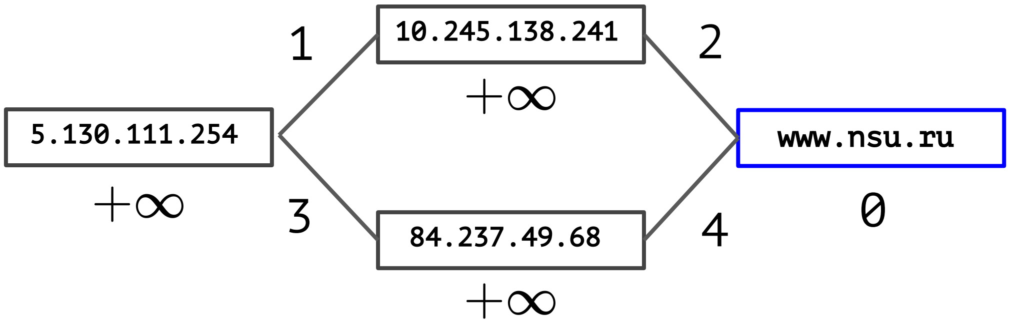
2) асинхронность. Раньше для получения A[i][v] мы ждали, пока посчитаются **все** A[i-1][w].

   Но так ли это нужно? Нет! Нужны только A[i-1][v] и нужны A[i-1][w], где w - соседи v.

   Для RIP - это меняется. Вместо того, чтобы ждать всех предыдущих, наоборот: при обновлении расстояния посылаем сообщение всем своим соседям, что что-то поменялось.

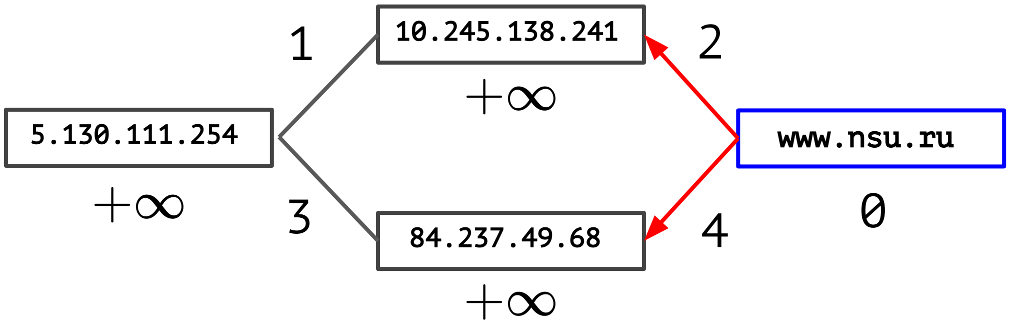
Сетевой протокол RIP

RIP (**R**outing **I**nformation **P**rotocol) - протокол маршрутизации сети, основанный на алгоритме Беллмана-Форда.



Сетевой протокол RIP

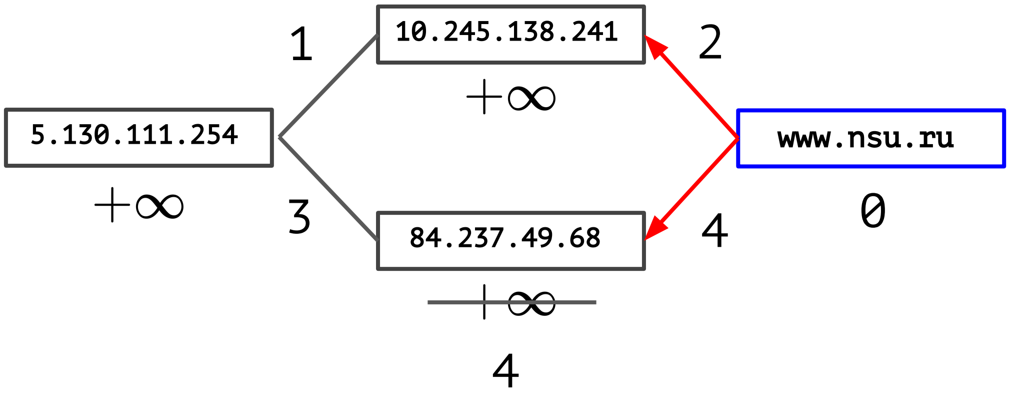
RIP (**R**outing **I**nformation **P**rotocol) - протокол маршрутизации сети, основанный на алгоритме Беллмана-Форда.



Наша цель готова (у нее обновилось расстояние), она посылает сообщения соседям

Сетевой протокол RIP

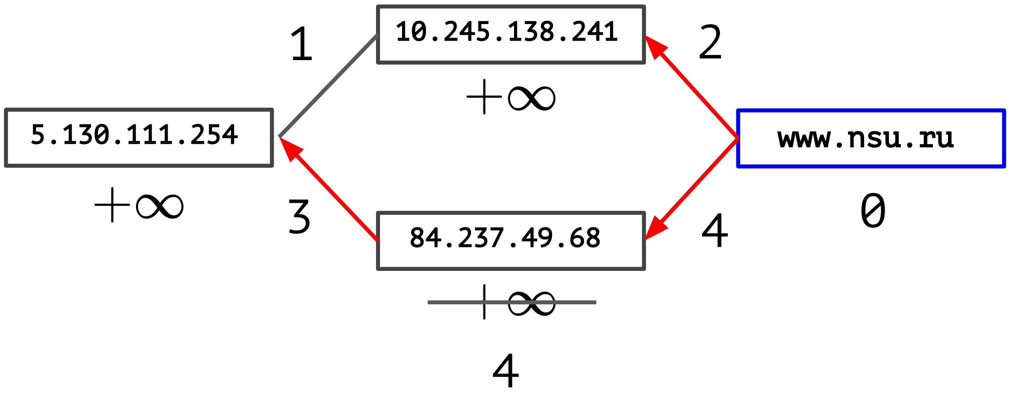
RIP (**R**outing **I**nformation **P**rotocol) - протокол маршрутизации сети, основанный на алгоритме Беллмана-Форда.



Наша цель готова (у нее обновилось расстояние), она посылает сообщения соседям

Сетевой протокол RIP

RIP (**R**outing **I**nformation **P**rotocol) - протокол маршрутизации сети, основанный на алгоритме Беллмана-Форда.

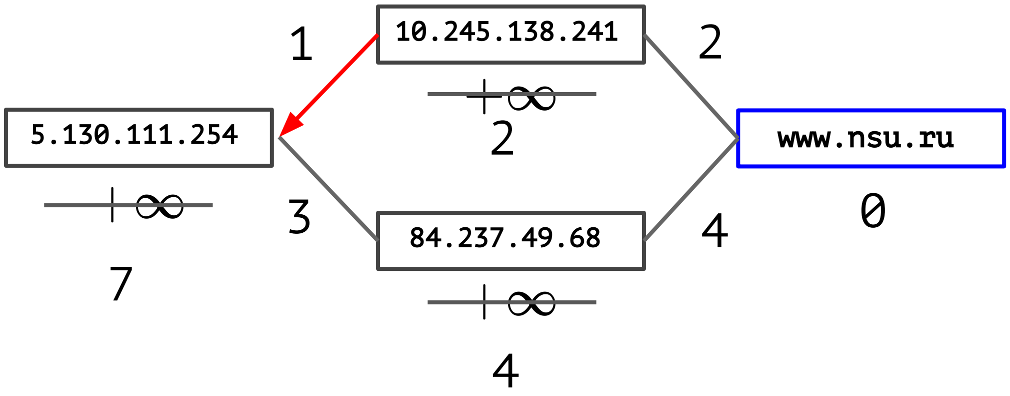


Наша цель готова (у нее обновилось расстояние), она посылает сообщения соседям

Теперь 84.234.49.68 готова, она тоже посылает сигнал соседям

Сетевой протокол RIP

RIP (**R**outing **I**nformation **P**rotocol) - протокол маршрутизации сети, основанный на алгоритме Беллмана-Форда.



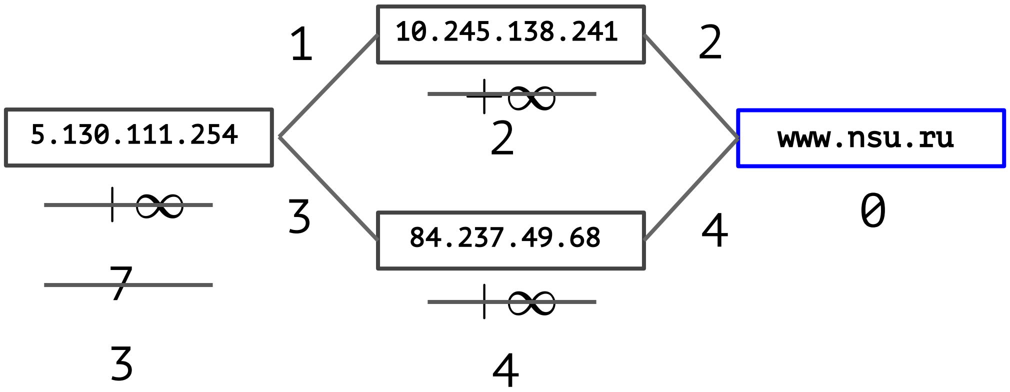
Наша цель готова (у нее обновилось расстояние), она посылает сообщения соседям

Потом сообщение дошло и до 10.245.138.241

До 5.130.111.254 в это время уже дошло первое сообщение.

Сетевой протокол RIP

RIP (**R**outing **I**nformation **P**rotocol) - протокол маршрутизации сети, основанный на алгоритме Беллмана-Форда.



Наша цель готова (у нее обновилось расстояние), она посылает сообщения соседям

Потом сообщение дошло и до 10.245.138.241

В конце еще раз обнолвляем 5.130.111.254

Сетевой протокол RIP

RIP (**R**outing **I**nformation **P**rotocol) - протокол маршрутизации сети, основанный на алгоритме Беллмана-Форда.

1) source-oriented -> destination-oriented

2) асинхронность. Вместо ожидания, пока все подзадачи будут готовы, посылаем сообщение соседям, когда у нас обновилось расстояние.

Сетевой протокол RIP

RIP (**R**outing **I**nformation **P**rotocol) - протокол маршрутизации сети, основанный на алгоритме Беллмана-Форда.

1) source-oriented -> destination-oriented

2) асинхронность. Вместо ожидания, пока все подзадачи будут готовы, посылаем сообщение соседям, когда у нас обновилось расстояние.

3) обработка ошибок: что делать, если узел стал недоступен?

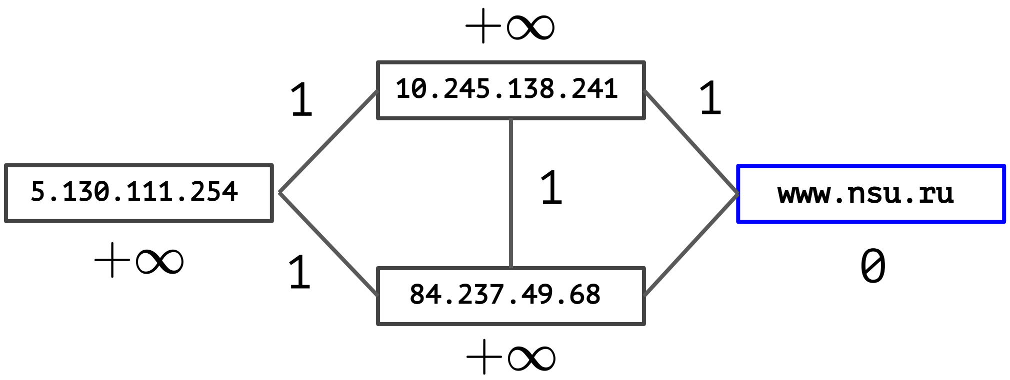
Сетевой протокол RIP

RIP (**R**outing **I**nformation **P**rotocol) - протокол маршрутизации сети, основанный на алгоритме Беллмана-Форда.

1) source-oriented -> destination-oriented

2) асинхронность

3) обработка ошибок: что делать, если узел стал недоступен?



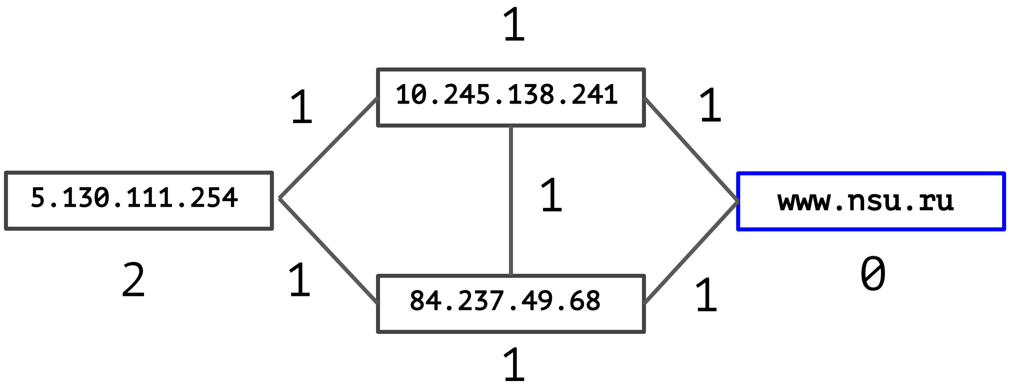
Сетевой протокол RIP

RIP (**R**outing **I**nformation **P**rotocol) - протокол маршрутизации сети, основанный на алгоритме Беллмана-Форда.

1) source-oriented -> destination-oriented

2) асинхронность

3) обработка ошибок: что делать, если узел стал недоступен?



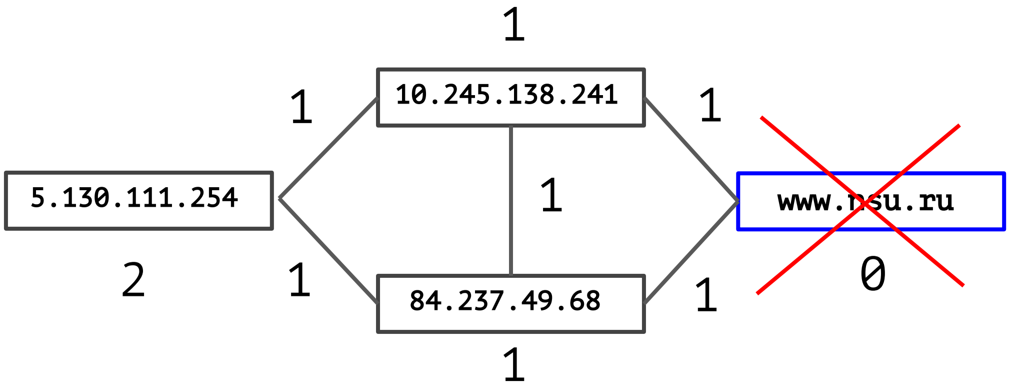
Сетевой протокол RIP

RIP (**R**outing **I**nformation **P**rotocol) - протокол маршрутизации сети, основанный на алгоритме Беллмана-Форда.

1) source-oriented -> destination-oriented

2) асинхронность

3) обработка ошибок: что делать, если узел стал недоступен?



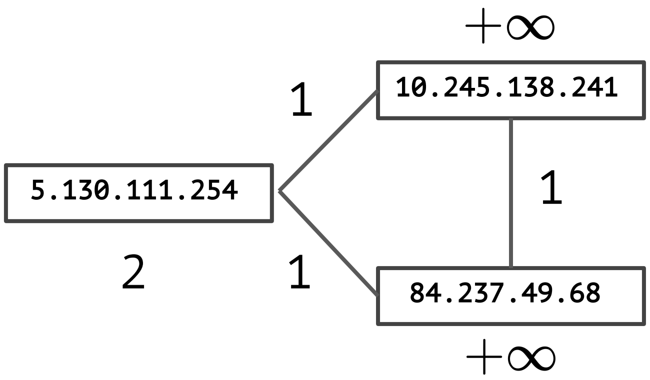
Сетевой протокол RIP

RIP (**R**outing **I**nformation **P**rotocol) - протокол маршрутизации сети, основанный на алгоритме Беллмана-Форда.

1) source-oriented -> destination-oriented

2) асинхронность

3) обработка ошибок: что делать, если узел стал недоступен?



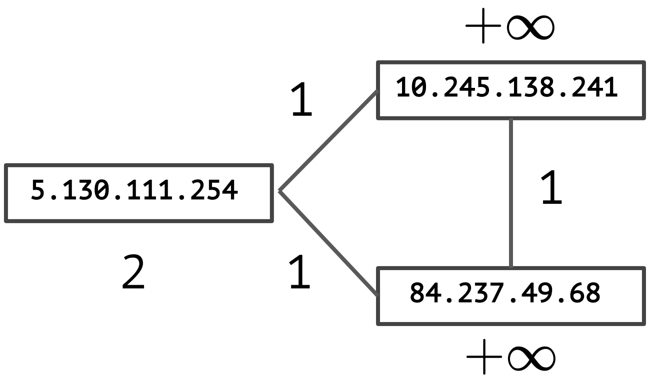
Сетевой протокол RIP

RIP (**R**outing **I**nformation **P**rotocol) - протокол маршрутизации сети, основанный на алгоритме Беллмана-Форда.

1) source-oriented -> destination-oriented

2) асинхронность

3) обработка ошибок: что делать, если узел стал недоступен?



Наивный алгоритм попытается обновить расстояния от 10.245.138.241 и 84.237.49.68…

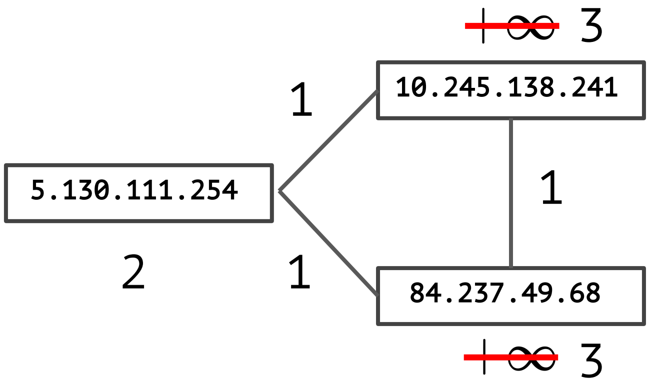
Сетевой протокол RIP

RIP (**R**outing **I**nformation **P**rotocol) - протокол маршрутизации сети, основанный на алгоритме Беллмана-Форда.

1) source-oriented -> destination-oriented

2) асинхронность

3) обработка ошибок: что делать, если узел стал недоступен?



Наивный алгоритм попытается обновить расстояния от 10.245.138.241 и 84.237.49.68…

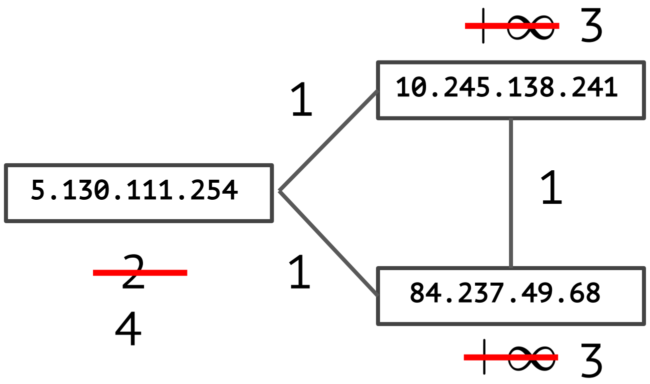
Сетевой протокол RIP

RIP (**R**outing **I**nformation **P**rotocol) - протокол маршрутизации сети, основанный на алгоритме Беллмана-Форда.

1) source-oriented -> destination-oriented

2) асинхронность

3) обработка ошибок: что делать, если узел стал недоступен?



Наивный алгоритм попытается обновить расстояния от 10.245.138.241 и 84.237.49.68…

и попадет в неприятную ситуацию.

Сетевой протокол RIP

RIP (**R**outing **I**nformation **P**rotocol) - протокол маршрутизации сети, основанный на алгоритме Беллмана-Форда.

1) source-oriented -> destination-oriented

2) асинхронность

3) обработка ошибок: что делать, если узел стал недоступен?

   Необходимо обновлять все кратчайшие пути до пропавшей вершины, а не только соседей пропавших.

Сетевой протокол RIP

RIP (**R**outing **I**nformation **P**rotocol) - протокол маршрутизации сети, основанный на алгоритме Беллмана-Форда.

1) source-oriented -> destination-oriented

2) асинхронность

3) обработка ошибок

На практике RIP работал не больше, чем с 15-ю транзитными участками и сильно грузил сеть. Есть улучшенные варианты RIP2 и RIPng, но более распространены альтернативы (см. IGRP)

Кратчайшие пути между всеми парами вершин

Кратчайшие пути между всеми парами вершин

Пусть теперь мы хотим найти не кратчайший путь от вершины s до остальных, а кратчайшие пути между всеми парами вершин.

Кратчайшие пути между всеми парами вершин

Пусть теперь мы хотим найти не кратчайший путь от вершины s до остальных, а кратчайшие пути между всеми парами вершин.

Как решать?

Кратчайшие пути между всеми парами вершин

Пусть теперь мы хотим найти не кратчайший путь от вершины s до остальных, а кратчайшие пути между всеми парами вершин.

Как решать?

1) Если нет рёбер отрицательного веса: то |V| раз запустим Дейкстру (каждый запуск построит кратчайшие пути из одной вершины до всех остальных).

   Сложность?

Кратчайшие пути между всеми парами вершин

Пусть теперь мы хотим найти не кратчайший путь от вершины s до остальных, а кратчайшие пути между всеми парами вершин.

Как решать?

1) Если нет рёбер отрицательного веса: то |V| раз запустим Дейкстру (каждый запуск построит кратчайшие пути из одной вершины до всех остальных).

   Сложность? O(|V|*(|V|+|E|)*log|V|)

Кратчайшие пути между всеми парами вершин

Пусть теперь мы хотим найти не кратчайший путь от вершины s до остальных, а кратчайшие пути между всеми парами вершин.

Как решать?

1) Если нет рёбер отрицательного веса: то |V| раз запустим Дейкстру (каждый запуск построит кратчайшие пути из одной вершины до всех остальных).

Сложность? $O(|V| * \underbrace{\mathbf{(|V|+|E|)*log|V|}}_{\text{Сложность Дейкстры на бинарных хипах}})$

Кратчайшие пути между всеми парами вершин

Пусть теперь мы хотим найти не кратчайший путь от вершины s до остальных, а кратчайшие пути между всеми парами вершин.

Как решать?

1) Если нет рёбер отрицательного веса: то |V| раз запустим Дейкстру (каждый запуск построит кратчайшие пути из одной вершины до всех остальных).

   Сложность? $O(|V|^2 log(|V|) + |V||E|log(|V|))$

   Дальше зависит от |E|. Если |E| = O(|V|), то получаем сложность $O(|V|^2 log(|V|))$

Кратчайшие пути между всеми парами вершин

Пусть теперь мы хотим найти не кратчайший путь от вершины s до остальных, а кратчайшие пути между всеми парами вершин.

Как решать?

1) Если нет рёбер отрицательного веса: то |V| раз запустим Дейкстру (каждый запуск построит кратчайшие пути из одной вершины до всех остальных).

   Сложность? $O(|V|^2 log(|V|) + |V||E|log(|V|))$

   Дальше зависит от |E|. Если $|E| = O(|V|^2)$, то получаем сложность $O(|V|^3 log(|V|))$

Кратчайшие пути между всеми парами вершин

Пусть теперь мы хотим найти не кратчайший путь от вершины s до остальных, а кратчайшие пути между всеми парами вершин.

Как решать?

2) Если есть рёбра отрицательного веса?

Кратчайшие пути между всеми парами вершин

Пусть теперь мы хотим найти не кратчайший путь от вершины s до остальных, а кратчайшие пути между всеми парами вершин.

Как решать?

2) Если есть рёбра отрицательного веса? Тогда |V| раз запустим алгоритм Беллмана-Форда! Сложность?

$$L_{v,i} = \min \begin{cases} L_{v,i-1} \\ \min\limits_{(w,v)\in E} L_{w,i-1} + c_{w,v} \end{cases}$$

Алгоритм Беллмана-Форда

Заводим двумерный массив A. Первый индекс - вершина, куда ищем кратчайший путь, второй - доступное количество ребер.

Инициализация: A[s][0] = 0, A[v][0] = $+\infty$, для остальных.

Заполняем массив:

$O(|V| * |E|)$

суммарно - проход по всем ребрам

```
for i in [1, |V|-1]:
  for v in V:
    A[v][i] = min(A[v][i-1], min(A[w][i-1] + C(w,v)))
                             (w,v) ∈ E
```

Сложность? $O(|V| * |E|)$. Дейкстра (через обычные хипы) при этом работает за $O((|V| + |E|)log(|V|))$. Т.е. для разреженных графов сложность сравнимая.

Кратчайшие пути между всеми парами вершин

Пусть теперь мы хотим найти не кратчайший путь от вершины s до остальных, а кратчайшие пути между всеми парами вершин.

Как решать?

2) Если есть рёбра отрицательного веса? Тогда |V| раз запустим алгоритм Беллмана-Форда! Сложность?

$$O(|V| * \underbrace{|V| * |E|}_{\text{Сложность Беллмана-Форда}})$$

Кратчайшие пути между всеми парами вершин

Пусть теперь мы хотим найти не кратчайший путь от вершины s до остальных, а кратчайшие пути между всеми парами вершин.

Как решать?

2) Если есть рёбра отрицательного веса? Тогда |V| раз запустим алгоритм Беллмана-Форда! Сложность?

$O(|V| * |V| * |E|)$, т.е. если $|E| = O(|V|) \Rightarrow O(|V|^3)$

Кратчайшие пути между всеми парами вершин

Пусть теперь мы хотим найти не кратчайший путь от вершины s до остальных, а кратчайшие пути между всеми парами вершин.

Как решать?

2) Если есть рёбра отрицательного веса? Тогда |V| раз запустим алгоритм Беллмана-Форда! Сложность?

$O(|V| * |V| * |E|)$, т.е. если $|E| = O(|V|) \Rightarrow O(|V|^3)$

$$|E| = O(|V|^2) \Rightarrow O(|V|^4)$$

Кратчайшие пути между всеми парами вершин

Пусть теперь мы хотим найти не кратчайший путь от вершины s до остальных, а кратчайшие пути между всеми парами вершин.

Как решать?

2) Если есть рёбра отрицательного веса? Тогда |V| раз запустим алгоритм Беллмана-Форда! Сложность?

$O(|V| * |V| * |E|)$, т.е. если $|E| = O(|V|) \Rightarrow O(|V|^3)$ разреженный граф

$|E| = O(|V|^2) \Rightarrow O(|V|^4)$ плотный граф

Кратчайшие пути между всеми парами вершин

Пусть теперь мы хотим найти не кратчайший путь от вершины s до остальных, а кратчайшие пути между всеми парами вершин.

Как решать?

2) Если есть рёбра отрицательного веса? Тогда |V| раз запустим алгоритм Беллмана-Форда! Сложность?

$O(|V| * |V| * |E|)$, т.е. если $|E| = O(|V|) \Rightarrow O(|V|^3)$ разреженный граф

$|E| = O(|V|^2) \Rightarrow O(|V|^4)$ плотный граф

Можем ли мы лучше?

Особенно для плотных графов!

Алгоритм Флойда-Уоршелла

Это будет динамика, а значит нам снова нужна оптимальная подструктура.

Алгоритм Флойда-Уоршелла

Это будет динамика, а значит нам снова нужна оптимальная подструктура.

Пусть все наши вершины как-то занумерованы: $V = \{1, 2, \ldots, n\}$
Введем обозначение: $V^{(k)} = \{1, 2, \ldots, k\}$

Алгоритм Флойда-Уоршелла

Это будет динамика, а значит нам снова нужна оптимальная подструктура.

Пусть все наши вершины как-то занумерованы: $V = \{1, 2, \ldots, n\}$
Введем обозначение: $V^{(k)} = \{1, 2, \ldots, k\}$

Теперь фиксируем две вершины: i и j. Кроме того, фиксируем некое $k \in \{1, 2..., n\}$.

Алгоритм Флойда-Уоршелла

Это будет динамика, а значит нам снова нужна оптимальная подструктура.

Пусть все наши вершины как-то занумерованы: $V = \{1, 2, \ldots, n\}$
Введем обозначение: $V^{(k)} = \{1, 2, \ldots, k\}$

Теперь фиксируем две вершины: i и j. Кроме того, фиксируем некое $k \in \{1, 2..., n\}$. Рассмотрим все возможные пути из вершины i в вершину j, такие что все внутренние вершины на этом пути из $V^{(k)} = \{1, 2, \ldots, k\}$.

Алгоритм Флойда-Уоршелла

Это будет динамика, а значит нам снова нужна оптимальная подструктура.

Пусть все наши вершины как-то занумерованы: $V = \{1, 2, \ldots, n\}$
Введем обозначение: $V^{(k)} = \{1, 2, \ldots, k\}$

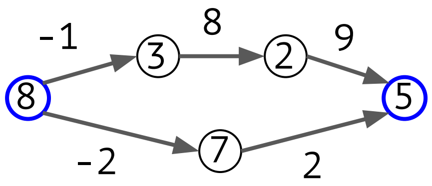
Теперь фиксируем две вершины: i и j. Кроме того, фиксируем некое $k \in \{1, 2..., n\}$. Рассмотрим все возможные пути из вершины i в вершину j, такие что все внутренние вершины на этом пути из $V^{(k)} = \{1, 2, \ldots, k\}$. И рассмотрим кратчайший из этих путей P.

Алгоритм Флойда-Уоршелла

Это будет динамика, а значит нам снова нужна оптимальная подструктура.

Пусть все наши вершины как-то занумерованы: $V = \{1, 2, \ldots, n\}$
Введем обозначение: $V^{(k)} = \{1, 2, \ldots, k\}$

Теперь фиксируем две вершины: i и j. Кроме того, фиксируем некое $k \in \{1, 2..., n\}$. Рассмотрим все возможные пути из вершины i в вершину j, такие что все внутренние вершины на этом пути из $V^{(k)} = \{1, 2, \ldots, k\}$. И рассмотрим кратчайший из этих путей P.



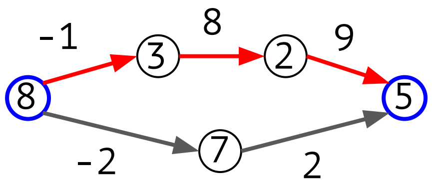
$i = 8, j = 5, k = 4$

Алгоритм Флойда-Уоршелла

Это будет динамика, а значит нам снова нужна оптимальная подструктура.

Пусть все наши вершины как-то занумерованы: $V = \{1, 2, \ldots, n\}$
Введем обозначение: $V^{(k)} = \{1, 2, \ldots, k\}$

Теперь фиксируем две вершины: i и j. Кроме того, фиксируем некое $k \in \{1, 2..., n\}$. Рассмотрим все возможные пути из вершины i в вершину j, такие что все внутренние вершины на этом пути из $V^{(k)} = \{1, 2, \ldots, k\}$. И рассмотрим кратчайший из этих путей P.



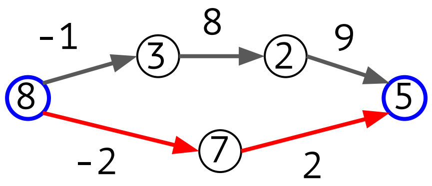
$$i = 8, j = 5, k = 4 \Rightarrow P = 8 \to 3 \to 2 \to 5$$

Алгоритм Флойда-Уоршелла

Это будет динамика, а значит нам снова нужна оптимальная подструктура.

Пусть все наши вершины как-то занумерованы: $V = \{1, 2, \ldots, n\}$
Введем обозначение: $V^{(k)} = \{1, 2, \ldots, k\}$

Теперь фиксируем две вершины: i и j. Кроме того, фиксируем некое $k \in \{1, 2..., n\}$. Рассмотрим все возможные пути из вершины i в вершину j, такие что все внутренние вершины на этом пути из $V^{(k)} = \{1, 2, \ldots, k\}$. И рассмотрим кратчайший из этих путей P.



$i = 8, j = 5, k = 4 \Rightarrow P = 8 \to 3 \to 2 \to 5$

$i = 8, j = 5, k = 8 \Rightarrow P = 8 \to 7 \to 5$

Оптимальная подструктура

Зафиксировали две вершины: i и j и некое $k \in \{1, 2..., n\}$.
P - кратчайший путь из i в j, в котором все внутренние вершины из $V^{(k)} = \{1, 2, \dots, k\}$.

Оптимальная подструктура

Зафиксировали две вершины: i и j и некое $k \in \{1, 2..., n\}$.
P - кратчайший путь из i в j, в котором все внутренние вершины из $V^{(k)} = \{1, 2, \ldots, k\}$. Предположим сначала, что нет циклов отрицательного веса.

Оптимальная подструктура

Зафиксировали две вершины: i и j и некое $k \in \{1, 2..., n\}$.
P - кратчайший путь из i в j, в котором все внутренние вершины из $V^{(k)} = \{1, 2, \ldots, k\}$. Предположим сначала, что нет циклов отрицательного веса.

Тогда верно одно из двух:

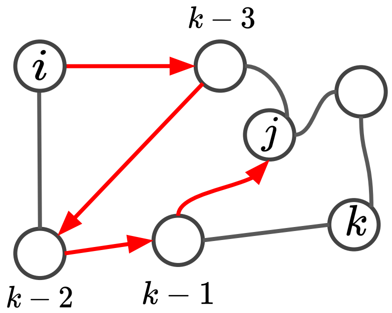
1) Вершина k не входит в этот кратчайший путь P.

Оптимальная подструктура

Зафиксировали две вершины: i и j и некое $k \in \{1, 2..., n\}$.
P - кратчайший путь из i в j, в котором все внутренние вершины из $V^{(k)} = \{1, 2, \ldots, k\}$. Предположим сначала, что нет циклов отрицательного веса.

Тогда верно одно из двух:

1) Вершина k не входит в этот кратчайший путь P.

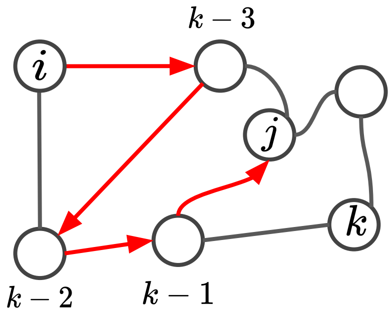


Оптимальная подструктура

Зафиксировали две вершины: i и j и некое $k \in \{1, 2..., n\}$.
P - кратчайший путь из i в j, в котором все внутренние вершины из $V^{(k)} = \{1, 2, \ldots, k\}$. Предположим сначала, что нет циклов отрицательного веса.

Тогда верно одно из двух:

1) Вершина k не входит в этот кратчайший путь P.



Тогда P - это кратчайший путь из i в j, внутренние вершины которого все из $V^{(k-1)}$!

Оптимальная подструктура

Зафиксировали две вершины: i и j и некое $k \in \{1, 2..., n\}$.
P - кратчайший путь из i в j, в котором все внутренние вершины из $V^{(k)} = \{1, 2, \ldots, k\}$. Предположим сначала, что нет циклов отрицательного веса.

Тогда верно одно из двух:

P кр. из i в j для $V^{(k-1)}$

1) Вершина k не входит в этот кратчайший путь P
2) Вершина k входит в этот кратчайший путь P.

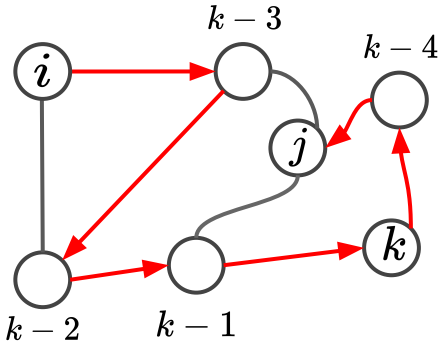
Оптимальная подструктура

Зафиксировали две вершины: i и j и некое $k \in \{1, 2 ..., n\}$.
P - кратчайший путь из i в j, в котором все внутренние вершины из $V^{(k)} = \{1, 2, \ldots, k\}$. Предположим сначала, что нет циклов отрицательного веса.

Тогда верно одно из двух:

P кр. из i в j для $V^{(k-1)}$

1) Вершина k не входит в этот кратчайший путь P.
2) Вершина k входит в этот кратчайший путь P.



Оптимальная подструктура

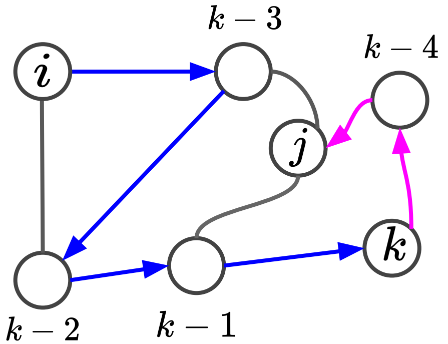
Зафиксировали две вершины: i и j и некое $k \in \{1, 2..., n\}$.
P - кратчайший путь из i в j, в котором все внутренние вершины из $V^{(k)} = \{1, 2, \ldots, k\}$. Предположим сначала, что нет циклов отрицательного веса.

Тогда верно одно из двух:

P кр. из i в j для $V^{(k-1)}$

1) Вершина k не входит в этот кратчайший путь P.
2) Вершина k входит в этот кратчайший путь P.

   Тогда путь разбивается на P_1 и P_2
   Что можем про них сказать?



Оптимальная подструктура

Зафиксировали две вершины: i и j и некое $k \in \{1, 2..., n\}$.
P - кратчайший путь из i в j, в котором все внутренние вершины из $V^{(k)} = \{1, 2, \ldots, k\}$. Предположим сначала, что нет циклов отрицательного веса.

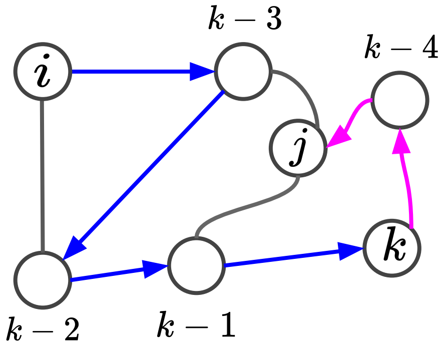
Тогда верно одно из двух:

P кр. из i в j для $V^{(k-1)}$

1) Вершина k не входит в этот кратчайший путь P.
2) Вершина k входит в этот кратчайший путь P.

P_1 - кр. из i в k для $V^{(k-1)}$
P_2 - кр. из k в j для $V^{(k-1)}$

Док-во аналогично Беллману-Форду.



Алгоритм Флойда-Уоршелла

Заведем трехмерный массив A, в котором индексы будут i,j и k.

Алгоритм Флойда-Уоршелла

Заведем трехмерный массив A, в котором индексы будут i,j и k.

Инициализация: что будет в элементах массива A[i, j, 0]?
(т.е. для пути из i в j в котором нет промежуточных вершин)

Алгоритм Флойда-Уоршелла

Заведем трехмерный массив A, в котором индексы будут i,j и k.

Инициализация: что будет в элементах массива A[i, j, 0]?
(т.е. для пути из i в j в котором нет промежуточных вершин)

1) i = j => A[i, j, 0] = 0

Алгоритм Флойда-Уоршелла

Заведем трехмерный массив A, в котором индексы будут i,j и k.

Инициализация: что будет в элементах массива A[i, j, 0]?
(т.е. для пути из i в j в котором нет промежуточных вершин)

1) i = j => A[i, j, 0] = 0
2) i != j, и $(i,j) \in E$ => A[i, j, 0] = $c_{i,j}$

Алгоритм Флойда-Уоршелла

Заведем трехмерный массив A, в котором индексы будут i,j и k.

Инициализация: что будет в элементах массива A[i, j, 0]?
(т.е. для пути из i в j в котором нет промежуточных вершин)

1) i = j => A[i, j, 0] = 0
2) i != j, и $(i,j) \in E$ => A[i, j, 0] = $c_{i,j}$
3) i != j, и $(i,j) \notin E$ => A[i, j, 0] = $+\infty$

Алгоритм Флойда-Уоршелла

Заведем трехмерный массив A, в котором индексы будут i,j и k.

Инициализация: A[i, j, 0] проинициализирован для всех i и j.

Заполняем массив:

```
for k in [1, n]:
  for i in [1, n]:
    for j in [1, n]:
```

Алгоритм Флойда-Уоршелла

Заведем трехмерный массив A, в котором индексы будут i,j и k.

Инициализация: A[i, j, 0] проинициализирован для всех i и j.

Заполняем массив:

```
for k in [1, n]:
  for i in [1, n]:
    for j in [1, n]:
```

$$A[i,j,k] = \min \begin{cases} A[i,j,k-1] \\ A[i,k,k-1] + A[k,j,k-1] \end{cases}$$

Алгоритм Флойда-Уоршелла

Заведем трехмерный массив A, в котором индексы будут i,j и k.

Инициализация: A[i, j, 0] проинициализирован для всех i и j.

Заполняем массив:

Сложность бросается в глаза: $O(|V|^3)$

```
for k in [1, n]:
  for i in [1, n]:
    for j in [1, n]:
```

$$A[i,j,k] = \min \begin{cases} A[i,j,k-1] \\ A[i,k,k-1] + A[k,j,k-1] \end{cases}$$

Кратчайшие пути между всеми парами вершин

Пусть теперь мы хотим найти не кратчайший путь от вершины s до остальных, а кратчайшие пути между всеми парами вершин.

Как решать?

2) Если есть рёбра отрицательного веса? Тогда |V| раз запустим алгоритм Беллмана-Форда! Сложность?

$O(|V| * |V| * |E|)$, т.е. если $|E| = O(|V|) \Rightarrow O(|V|^3)$ — разреженный граф

$|E| = O(|V|^2) \Rightarrow O(|V|^4)$ — плотный граф

Можем ли мы лучше?

Особенно для плотных графов!

Алгоритм Флойда-Уоршелла

Заведем трехмерный массив A, в котором индексы будут i,j и k.

Инициализация: A[i, j, 0] проинициализирован для всех i и j.

Заполняем массив:

```
for k in [1, n]:
  for i in [1, n]:
    for j in [1, n]:
```

$$A[i,j,k] = \min \begin{cases} A[i,j,k-1] \\ A[i,k,k-1] + A[k,j,k-1] \end{cases}$$

Сложность бросается в глаза: $O(|V|^3)$

Для плотных графов это лучше, чем |V| запусков Беллмана-Форда.

Алгоритм Флойда-Уоршелла

А что по поводу циклов отрицательной длины?

Алгоритм Флойда-Уоршелла

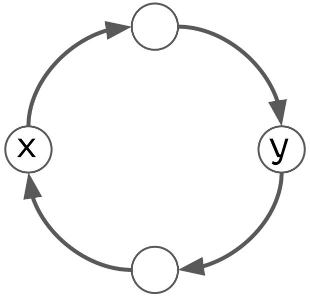
А что по поводу циклов отрицательной длины?

Утверждение: они есть ⇔ после алгоритма Флойда хотя бы для одного i будет справедливо: A[i, i, |V|] < 0

Алгоритм Флойда-Уоршелла

А что по поводу циклов отрицательной длины?

Утверждение: они есть ⇔ после алгоритма Флойда хотя бы для одного i будет справедливо: A[i, i, |V|] < 0

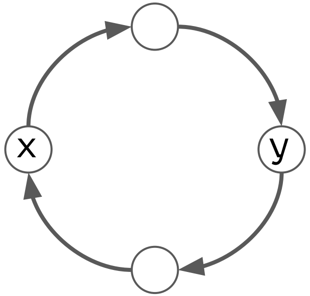


⇒ Пусть есть цикл отрицательного веса.

Алгоритм Флойда-Уоршелла

А что по поводу циклов отрицательной длины?

Утверждение: они есть ⇔ после алгоритма Флойда хотя бы для одного i будет справедливо: A[i, i, |V|] < 0



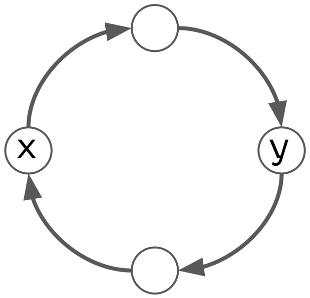
⇒ Пусть есть цикл отрицательного веса.

В какой-то момент алгоритм попытается построить путь из **x** в **x** такой, что в $V^{(k)}$ попадут все вершины из этого цикла.

Алгоритм Флойда-Уоршелла

А что по поводу циклов отрицательной длины?

Утверждение: они есть ⇔ после алгоритма Флойда хотя бы для одного i будет справедливо: A[i, i, |V|] < 0



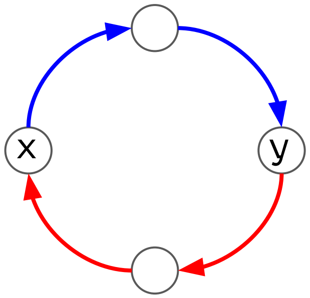
⇒ Пусть есть цикл отрицательного веса.

В какой-то момент алгоритм попытается построить путь из **x** в **x** такой, что в $V^{(k)}$ попадут все вершины из этого цикла (включая максимальную - **y**)

Алгоритм Флойда-Уоршелла

А что по поводу циклов отрицательной длины?

Утверждение: они есть ⇔ после алгоритма Флойда хотя бы для одного i будет справедливо: A[i, i, |V|] < 0



⇒ Пусть есть цикл отрицательного веса.

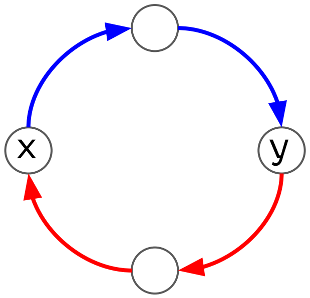
В какой-то момент алгоритм попытается построить путь из **x** в **x** такой, что в $V^{(k)}$ попадут все вершины из этого цикла (включая максимальную - **y**)

$$A[x,x,y] = \min \begin{cases} A[x,x,y-1] \\ A[x,y,y-1] + A[y,x,y-1] \end{cases}$$

Алгоритм Флойда-Уоршелла

А что по поводу циклов отрицательной длины?

Утверждение: они есть ⇔ после алгоритма Флойда хотя бы для одного i будет справедливо: A[i, i, |V|] < 0



⇒ Пусть есть цикл отрицательного веса.

В какой-то момент алгоритм попытается построить путь из **x** в **x** такой, что в $V^{(k)}$ попадут все вершины из этого цикла (включая максимальную - **y**)

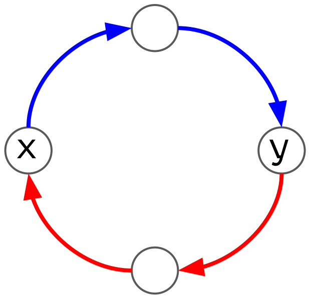
половинки цикла отрицательного веса

$$A[x,x,y] = \min \begin{cases} A[x,x,y-1] \\ \boxed{A[x,y,y-1]} + \boxed{A[y,x,y-1]} \end{cases}$$

Алгоритм Флойда-Уоршелла

А что по поводу циклов отрицательной длины?

Утверждение: они есть ⇔ после алгоритма Флойда хотя бы для одного i будет справедливо: A[i, i, |V|] < 0

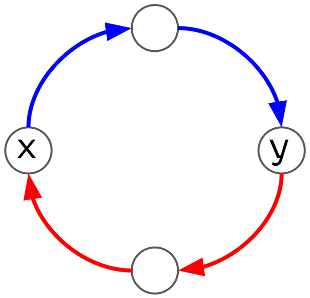


⇐ Изначально A[i, i, 0] = 0.

Алгоритм Флойда-Уоршелла

А что по поводу циклов отрицательной длины?

Утверждение: они есть ⇔ после алгоритма Флойда хотя бы для одного i будет справедливо: A[i, i, |V|] < 0



⇐ Изначально A[i, i, 0] = 0. Если на какой-то итерации мы выбрали для A[i, i, _] отрицательное значение, то это значит, что был путь из i в i проходящий через некий j, такой, что A[i, j, j - 1] + A[j, i, j - 1] < 0 => это и был цикл отрицательного веса.

Алгоритм Флойда-Уоршелла: оптимизация памяти

Сколько тратим памяти?

```
for k in [1, n]:
  for i in [1, n]:
    for j in [1, n]:
```

$$A[i,j,k] = \min \begin{cases} A[i,j,k-1] \\ A[i,k,k-1] + A[k,j,k-1] \end{cases}$$

Алгоритм Флойда-Уоршелла: оптимизация памяти

Сколько тратим памяти? O(|V|^3). А это точно необходимо?

```
for k in [1, n]:
  for i in [1, n]:
    for j in [1, n]:
```

$$A[i,j,k] = \min \begin{cases} A[i,j,k-1] \\ A[i,k,k-1] + A[k,j,k-1] \end{cases}$$

Алгоритм Флойда-Уоршелла: оптимизация памяти

Сколько тратим памяти? O(|V|^3). А это точно необходимо?

```
for k in [1, n]:
  for i in [1, n]:
    for j in [1, n]:
```

$$A[i, j, k] = \min \begin{cases} A[i, j, k-1] \\ A[i, k, k-1] + A[k, j, k-1] \end{cases}$$

Как минимум можно было бы просто взять две матрицы: одна хранит прошлую итерацию, вторая - текущую.

Алгоритм Флойда-Уоршелла: оптимизация памяти

Сколько тратим памяти? O(|V|^3). А это точно необходимо?

```
for k in [1, n]:
  for i in [1, n]:
    for j in [1, n]:
```

$$A[i, j, k] = \min \begin{cases} A[i, j, k-1] \\ A[i, k, k-1] + A[k, j, k-1] \end{cases}$$

Как минимум можно было бы просто взять две матрицы: одна хранит прошлую итерацию, вторая - текущую. Но на самом деле все еще лучше: ведь мы никогда не увеличиваем A[i,j,_]!

Алгоритм Флойда-Уоршелла: оптимизация памяти

Сколько тратим памяти? O(|V|^2).

```
for k in [1, n]:
  for i in [1, n]:
    for j in [1, n]:
      B[i, j] = min(B[i,j], B[i,k] + B[k,j])
```

Классическая запись алгоритма Флойда-Уоршелла.

Алгоритм Флойда-Уоршелла: оптимизация памяти

Сколько тратим памяти? O(|V|^2).

```
for k in [1, n]:
  for i in [1, n]:
    for j in [1, n]:
      B[i, j] = min(B[i,j], B[i,k] + B[k,j])
```

Классическая запись алгоритма Флойда-Уоршелла.

Вопрос: а это точно корректно? Ведь B[i, k] и B[k, j] могли быть уже обновлены на текущей итерации, точно ли корректно использовать именно их?

Алгоритм Флойда-Уоршелла: оптимизация памяти

Сколько тратим памяти? O(|V|^2).

```
for k in [1, n]:
  for i in [1, n]:
    for j in [1, n]:
      B[i, j] = min(B[i,j], B[i,k] + B[k,j])
```

Классическая запись алгоритма Флойда-Уоршелла.

Утверждение: пусть B[i,j] - значение 2D массива B сразу после k итераций, а d(i,j) - кратчайшее расстояние из i в j.

Алгоритм Флойда-Уоршелла: оптимизация памяти

Сколько тратим памяти? O(|V|^2).

```
for k in [1, n]:
  for i in [1, n]:
    for j in [1, n]:
      B[i, j] = min(B[i,j], B[i,k] + B[k,j])
```

Классическая запись алгоритма Флойда-Уоршелла.

Утверждение: пусть B[i,j] - значение 2D массива B сразу после k итераций, а d(i,j) - кратчайшее расстояние из i в j.

Тогда d(i,j) <= B[i,j] <= A[i,j,k]

Алгоритм Флойда-Уоршелла: оптимизация памяти

Сколько тратим памяти? O(|V|^2).

```
for k in [1, n]:
  for i in [1, n]:
    for j in [1, n]:
      B[i, j] = min(B[i,j], B[i,k] + B[k,j])
```

Классическая запись алгоритма Флойда-Уоршелла.

Утверждение: пусть B[i,j] - значение 2D массива B сразу после k итераций, а d(i,j) - кратчайшее расстояние из i в j.

Тогда d(i,j) <= B[i,j] <= A[i,j,k]

Если это правда, то d(i, j) = B[i,j] после всех итераций.

Алгоритм Флойда-Уоршелла: оптимизация памяти

Утверждение: пусть B[i,j] - значение 2D массива B сразу после k итераций, а d(i,j) - кратчайшее расстояние из i в j.

Тогда d(i,j) <= $\boxed{\texttt{B[i,j] <= A[i,j,k]}}$

Алгоритм Флойда-Уоршелла: оптимизация памяти

Утверждение: пусть B[i,j] - значение 2D массива B сразу после k итераций, а d(i,j) - кратчайшее расстояние из i в j.

Тогда d(i,j) <= B[i,j] <= A[i,j,k]

Доказательство: по индукции, база верна по заполнению массивов, пусть верно для k-1 итерации.

Алгоритм Флойда-Уоршелла: оптимизация памяти

Утверждение: пусть B[i,j] - значение 2D массива B сразу после k итераций, а d(i,j) - кратчайшее расстояние из i в j.

Тогда d(i,j) <= $\boxed{\text{B[i,j] <= A[i,j,k]}}$

Доказательство: по индукции, база верна по заполнению массивов, пусть верно для k-1 итерации. Тогда верно одно из двух:

Алгоритм Флойда-Уоршелла: оптимизация памяти

Сколько тратим памяти?

```
for k in [1, n]:
  for i in [1, n]:
    for j in [1, n]:
```

$$A[i,j,k] = \min \begin{cases} A[i,j,k-1] \\ A[i,k,k-1] + A[k,j,k-1] \end{cases}$$

Алгоритм Флойда-Уоршелла: оптимизация памяти

Утверждение: пусть B[i,j] - значение 2D массива B сразу после k итераций, а d(i,j) - кратчайшее расстояние из i в j.

Тогда $d(i,j) \le \boxed{B[i,j] \le A[i,j,k]}$

Доказательство: по индукции, база верна по заполнению массивов, пусть верно для k-1 итерации. Тогда верно одно из двух:

1) A[i,j,k] = A[i,k,k-1] + A[k,j,k-1]

Алгоритм Флойда-Уоршелла: оптимизация памяти

Утверждение: пусть B[i,j] - значение 2D массива B сразу после k итераций, а d(i,j) - кратчайшее расстояние из i в j.

Тогда d(i,j) <= $\boxed{\text{B[i,j] <= A[i,j,k]}}$

Доказательство: по индукции, база верна по заполнению массивов, пусть верно для k-1 итерации. Тогда верно одно из двух:

1) A[i,j,k] = A[i,k,k-1] + A[k,j,k-1] >= B[i,k] + B[k,j]

имеются в виду значения
B после итерации k-1

Алгоритм Флойда-Уоршелла: оптимизация памяти

Сколько тратим памяти? O(|V|^2).

```
for k in [1, n]:
  for i in [1, n]:
    for j in [1, n]:
      B[i, j] = min(B[i,j], B[i,k] + B[k,j])
```

Классическая запись алгоритма Флойда-Уоршелла.

Алгоритм Флойда-Уоршелла: оптимизация памяти

Утверждение: пусть B[i,j] - значение 2D массива B сразу после k итераций, а d(i,j) - кратчайшее расстояние из i в j.

Тогда $d(i,j) \le \boxed{B[i,j] \le A[i,j,k]}$

Доказательство: по индукции, база верна по заполнению массивов, пусть верно для k-1 итерации. Тогда верно одно из двух:

1) A[i,j,k] = A[i,k,k-1] + A[k,j,k-1] >= B[i,k] + B[k,j]

имеются в виду значения
B после итерации k-1

Алгоритм Флойда-Уоршелла: оптимизация памяти

Утверждение: пусть B[i,j] - значение 2D массива B сразу после k итераций, а d(i,j) - кратчайшее расстояние из i в j.

Тогда $d(i,j) \le \boxed{B[i,j] \le A[i,j,k]}$

Доказательство: по индукции, база верна по заполнению массивов, пусть верно для k-1 итерации. Тогда верно одно из двух:

1) A[i,j,k] = A[i,k,k-1] + A[k,j,k-1] >= B[i,k] + B[k,j] >= >= B[i,j]

уже на k-ой итерации, именно так этот элемент и задается

имеются в виду значения B после итерации k-1

Алгоритм Флойда-Уоршелла: оптимизация памяти

Утверждение: пусть B[i,j] - значение 2D массива B сразу после k итераций, а d(i,j) - кратчайшее расстояние из i в j.

Тогда $d(i,j) <= \boxed{B[i,j] <= A[i,j,k]}$

Доказательство: по индукции, база верна по заполнению массивов, пусть верно для k-1 итерации. Тогда верно одно из двух:

1) A[i,j,k] = A[i,k,k-1] + A[k,j,k-1] >= … >= B[i,j]

Алгоритм Флойда-Уоршелла: оптимизация памяти

Утверждение: пусть B[i,j] - значение 2D массива B сразу после k итераций, а d(i,j) - кратчайшее расстояние из i в j.

Тогда d(i,j) <= $\boxed{\text{B[i,j] <= A[i,j,k]}}$

Доказательство: по индукции, база верна по заполнению массивов, пусть верно для k-1 итерации. Тогда верно одно из двух:

1) A[i,j,k] = A[i,k,k-1] + A[k,j,k-1] >= … >= B[i,j]
2) A[i,j,k] = A[i,j,k-1]

Алгоритм Флойда-Уоршелла: оптимизация памяти

Утверждение: пусть B[i,j] - значение 2D массива B сразу после k итераций, а d(i,j) - кратчайшее расстояние из i в j.

Тогда d(i,j) <= $\boxed{\text{B[i,j] <= A[i,j,k]}}$

Доказательство: по индукции, база верна по заполнению массивов, пусть верно для k-1 итерации. Тогда верно одно из двух:

1) A[i,j,k] = A[i,k,k-1] + A[k,j,k-1] >= … >= B[i,j]
2) A[i,j,k] = A[i,j,k-1] >= B[i,j]

значение B после итерации k-1, т.е. по индукции

Алгоритм Флойда-Уоршелла: оптимизация памяти

Утверждение: пусть B[i,j] - значение 2D массива B сразу после k итераций, а d(i,j) - кратчайшее расстояние из i в j.

Тогда d(i,j) <= $\boxed{\text{B[i,j] <= A[i,j,k]}}$

Доказательство: по индукции, база верна по заполнению массивов, пусть верно для k-1 итерации. Тогда верно одно из двух:

1) A[i,j,k] = A[i,k,k-1] + A[k,j,k-1] >= … >= B[i,j]
2) A[i,j,k] = A[i,j,k-1] >= B[i,j]

значение B после итерации k-1, т.е. по индукции

но т.к. элементы B со временем только уменьшаются, то верно и для итерации k □

Алгоритм Флойда-Уоршелла: оптимизация памяти

Утверждение: пусть B[i,j] - значение 2D массива B сразу после k итераций, а d(i,j) - кратчайшее расстояние из i в j.

Тогда d(i,j) <= B[i,j] <= A[i,j,k]

Доказательство: пусть после некоторой итерации k впервые случилось: d(i,j) > B[i,j].

Алгоритм Флойда-Уоршелла: оптимизация памяти

Утверждение: пусть B[i,j] - значение 2D массива B сразу после k итераций, а d(i,j) - кратчайшее расстояние из i в j.

Тогда $\boxed{d(i,j) \le B[i,j]} \le A[i,j,k]$

Доказательство: пусть после некоторой итерации k впервые случилось: d(i,j) > B[i,j]. Это означает, что B[i,j] изменился на этой итерации, т.е. B[i,j] = B[i,k] + B[k,j]

Алгоритм Флойда-Уоршелла: оптимизация памяти

Утверждение: пусть B[i,j] - значение 2D массива B сразу после k итераций, а d(i,j) - кратчайшее расстояние из i в j.

Тогда $\boxed{d(i,j) \leq B[i,j]} \leq A[i,j,k]$

Доказательство: пусть после некоторой итерации k впервые случилось: d(i,j) > B[i,j]. Это означает, что B[i,j] изменился на этой итерации, т.е. B[i,j] = B[i,k] + B[k,j]

Но раз это случилось впервые, то d(i,j) > B[i,j] = B[i,k] + B[k,j] >= d(i,k) + d(k,j)

Алгоритм Флойда-Уоршелла: оптимизация памяти

Утверждение: пусть B[i,j] - значение 2D массива B сразу после k итераций, а d(i,j) - кратчайшее расстояние из i в j.

Тогда $\boxed{\texttt{d(i,j) <= B[i,j]}}$ <= A[i,j,k]

Доказательство: пусть после некоторой итерации k впервые случилось: d(i,j) > B[i,j]. Это означает, что B[i,j] изменился на этой итерации, т.е. B[i,j] = B[i,k] + B[k,j]

Но раз это случилось впервые, то d(i,j) > B[i,j] = B[i,k] + B[k,j] >= d(i,k) + d(k,j) >= d(i,j) по неравенству треугольника.

Алгоритм Флойда-Уоршелла: оптимизация памяти

Утверждение: пусть B[i,j] - значение 2D массива B сразу после k итераций, a d(i,j) - кратчайшее расстояние из i в j.

Тогда d(i,j) <= B[i,j] <= A[i,j,k]

Доказательство: пусть после некоторой итерации k впервые случилось: d(i,j) > B[i,j]. Это означает, что B[i,j] изменился на этой итерации, т.е. B[i,j] = B[i,k] + B[k,j]

Но раз это случилось впервые, то d(i,j) > B[i,j] = B[i,k] + B[k,j] >= d(i,k) + d(k,j) >= d(i,j) по неравенству треугольника. Но тогда B[i,j] >= d(i,j) - противоречие. □

Алгоритм Флойда-Уоршелла: оптимизация памяти

Сколько тратим памяти? O(|V|^2).

```
for k in [1, n]:
  for i in [1, n]:
    for j in [1, n]:
      B[i, j] = min(B[i,j], B[i,k] + B[k,j])
```

Классическая запись алгоритма Флойда-Уоршелла.

Утверждение: пусть B[i,j] - значение 2D массива B сразу после k итераций, а d(i,j) - кратчайшее расстояние из i в j.

Тогда d(i,j) <= B[i,j] <= A[i,j,k]

Если это правда, то d(i, j) = B[i,j] после всех итераций.

Алгоритм Флойда-Уоршелла: оптимизация памяти

Сколько тратим памяти? O(|V|^2).

```
for k in [1, n]:
  for i in [1, n]:
    for j in [1, n]:
      B[i, j] = min(B[i,j], B[i,k] + B[k,j])
```

Классическая запись алгоритма Флойда-Уоршелла.

Алгоритм Флойда-Уоршелла: восстановление пути

```
for k in [1, n]:
  for i in [1, n]:
    for j in [1, n]:
      B[i, j] = min(B[i,j], B[i,k] + B[k,j])
```

Классическая запись алгоритма Флойда-Уоршелла.

Вопрос: а как восстановить сами кратчайшие пути?

Алгоритм Флойда-Уоршелла: восстановление пути

```
for k in [1, n]:
  for i in [1, n]:
    for j in [1, n]:
      B[i, j] = min(B[i,j], B[i,k] + B[k,j])
```

Классическая запись алгоритма Флойда-Уоршелла.

Ответ: чтобы восстановить кратчайшие пути заводят еще двумерный массив next[i, j], в котором запоминают выбор, который мы делали при релаксации.

Алгоритм Флойда-Уоршелла: оптимизация памяти

В инициализации: next[i, j] = j, если есть ребро (i,j).

```
for k in [1, n]:
  for i in [1, n]:
    for j in [1, n]:
      if B[i,j] > B[i,k] + B[k,j]):
        B[i,j] = B[i,k] + B[k,j]
        next[i, j] = next[i, k]
```

Ответ: чтобы восстановить кратчайшие пути заводят еще двумерный массив next[i, j], в котором запоминают выбор, который мы делали при релаксации.

Алгоритм Флойда-Уоршелла: оптимизация памяти

В инициализации: next[i, j] = j, если есть ребро (i,j).

```
for k in [1, n]:
  for i in [1, n]:
    for j in [1, n]:
      if B[i,j] > B[i,k] + B[k,j]):
        B[i,j] = B[i,k] + B[k,j]
        next[i, j] = next[i, k]
```

После окончания алгоритма, чтобы найти путь из u в v, достаточно проитерироваться линейно:

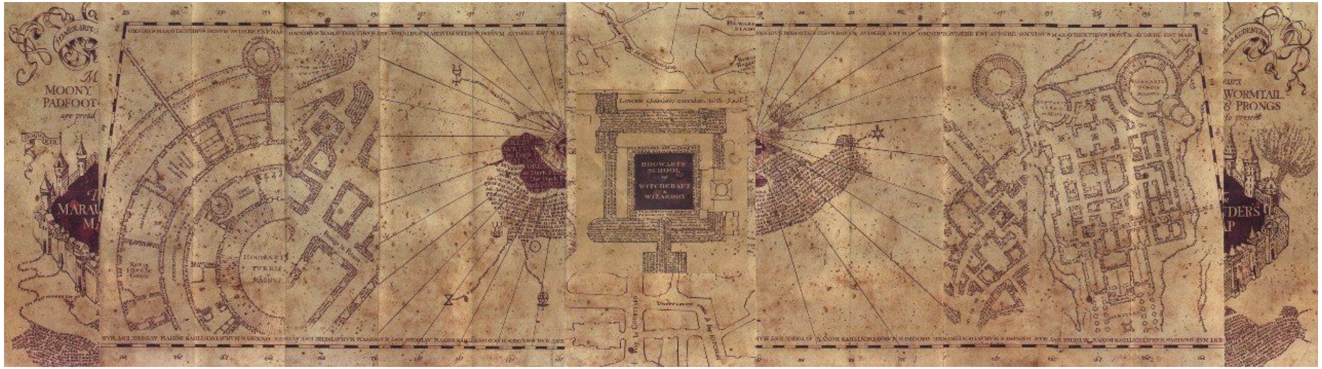
```
u <- next[i, j]
while u != j:
   u <- next[u, j]
```

Ответ: чтобы восстановить кратчайшие пути заводят еще двумерный массив next[i, j], в котором запоминают выбор, который мы делали при релаксации.

Мини-задача #41 (1 балл)

Найти город с наименьшим числом достижимых из него городов, которые расположены ближе, чем заданное расстояние.

https://leetcode.com/problems/find-the-city-with-the-smallest-number-of-neighbors-at-a-threshold-distance/description/



Takeaways

- Применение динамики к графам требует изобретательности в поиске оптимальной подзадачи.

- Беллман-Форд снова ищет кратчайшие пути из одной вершины, но теперь и с учетом отрицательных ребер (и даже находит циклы отр. веса)

- Флойд-Уоршелл ищет все кратчайшие пути лучше, чем запуск |V| Дейкстр для плотных графов.