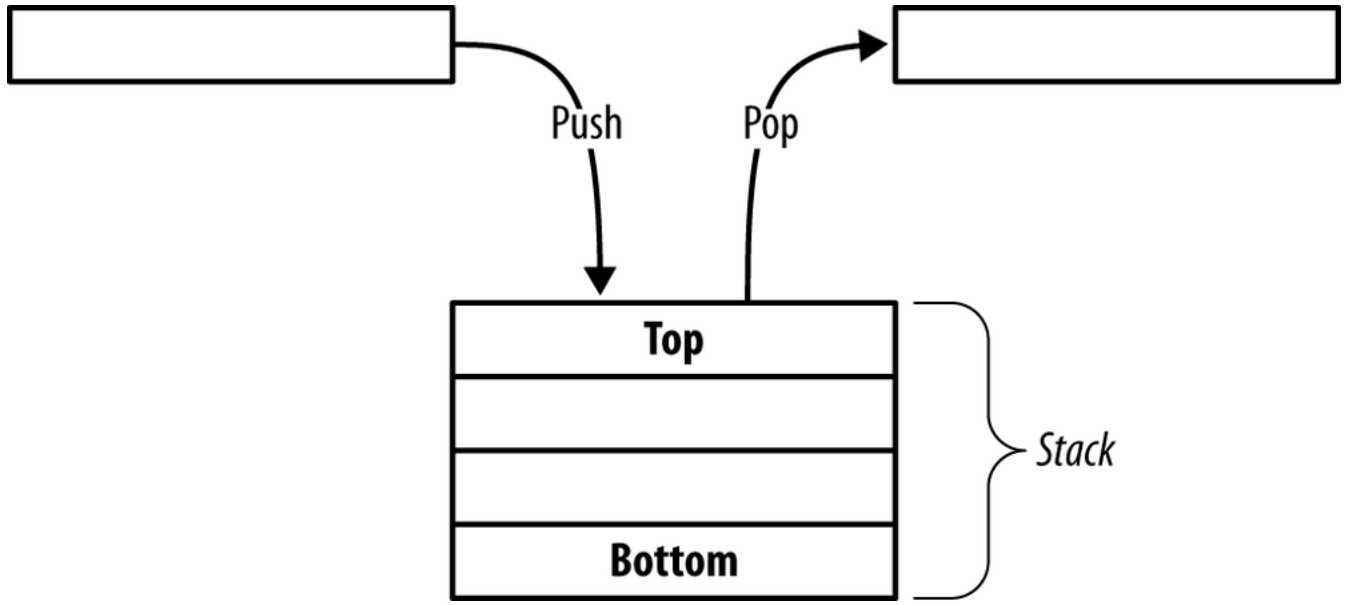
Простые структуры данных. Stack, Queue, Deque

М.К. Горденко mgordenko@hse.ru

Stack

- **Стек** — это структура данных, которая работает по принципу **FILO** (first in — last out; первый пришел — последний ушел) *или (Last-In-First-Out или **LIFO**)*.



Операции в стеке

- создание стека;
- печать (просмотр) стека;
- добавление элемента в *вершину стека (push)*;
- извлечение элемента из *вершины стека (pop)*;
- проверка пустоты стека;
- очистка стека.

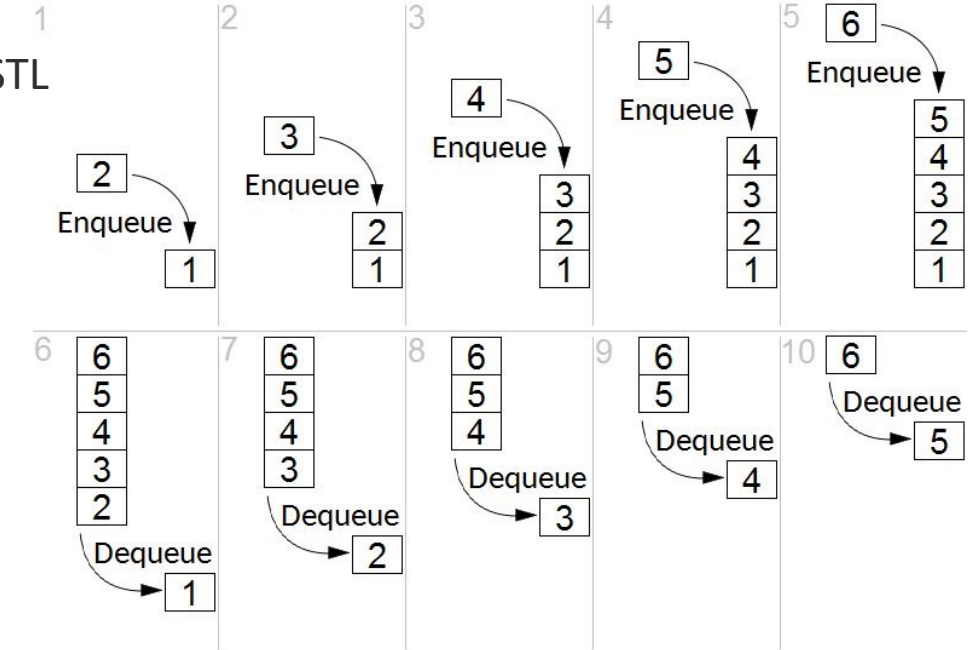
В C++ уже есть готовый шаблон — stack

```
Stack: empty
Push 1
Stack: 1
Push 2
Stack: 1 2
Push 3
Stack: 1 2 3
Push 4
Stack: 1 2 3 4
Pop
Stack: 1 2 3
Pop
Stack: 1 2
Pop
Stack: 1
```

Queue

- **Очередью** (англ. – queue) называется структура данных, из которой удаляется первым тот элемент, который был первым в очередь добавлен. То есть очередь в программировании соответствует «бытовому» понятию очереди. Очередь также называют структурой типа FIFO (first in, first out — первым пришел, первым ушел).

В C++ уже есть готовый STL контейнер — queue.

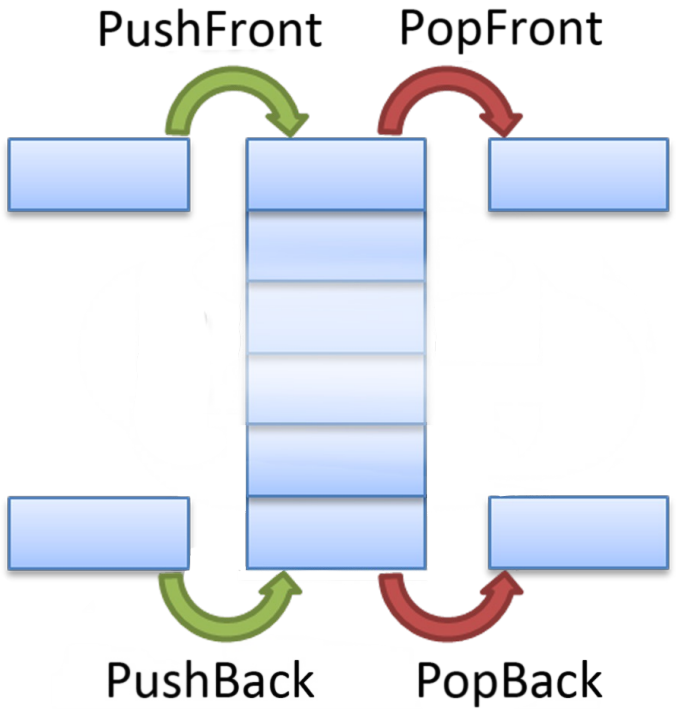


<queue>

- Создание queue <тип данных> <имя>;
- Добавить элемент push(); O(1)
- Удалить первый элемент pop(); O(1)
- Обратиться к первому элементу очереди front(); O(1)
- Обратиться к последнему элементу в очереди back(); O(1)
- Пуста ли очередь empty(); O(1)

Deque

- **Двусвязная очередь** — абстрактный тип данных, в котором элементы можно добавлять и удалять как в начало, так и в конец.



<deque>

- push_front Добавить (положить) в начало дека новый элемент
- push_back Добавить (положить) в конец дека новый элемент
- pop_front Извлечь из дека первый элемент
- pop_back Извлечь из дека последний элемент
- front Узнать значение первого элемента (не удаляя его)
- back Узнать значение последнего элемента (не удаляя его)
- size Узнать количество элементов в деке
- clear Очистить дек (удалить из него все элементы)

set

set — это контейнер, который автоматически сортирует добавляемые элементы в порядке возрастания. Но при добавлении одинаковых значений, set будет хранить только один его экземпляр. По-другому его еще называют множеством.

multiset

multiset — это контейнер, который также будет содержать элементы в отсортированном порядке при добавлении, но он хранит повторяющееся элементы, по сравнению с множеством set. Часто его называют мультимножество.

Теория графов

Алгоритм Дейкстры для нахождения кратчайших путей.

М.К. Горденко, mgordenko@hse.ru

Алгоритм Дейкстры

- Находит кратчайший путь от одной из вершин графа до всех остальных. Алгоритм работает только для графов без ребер отрицательного веса.

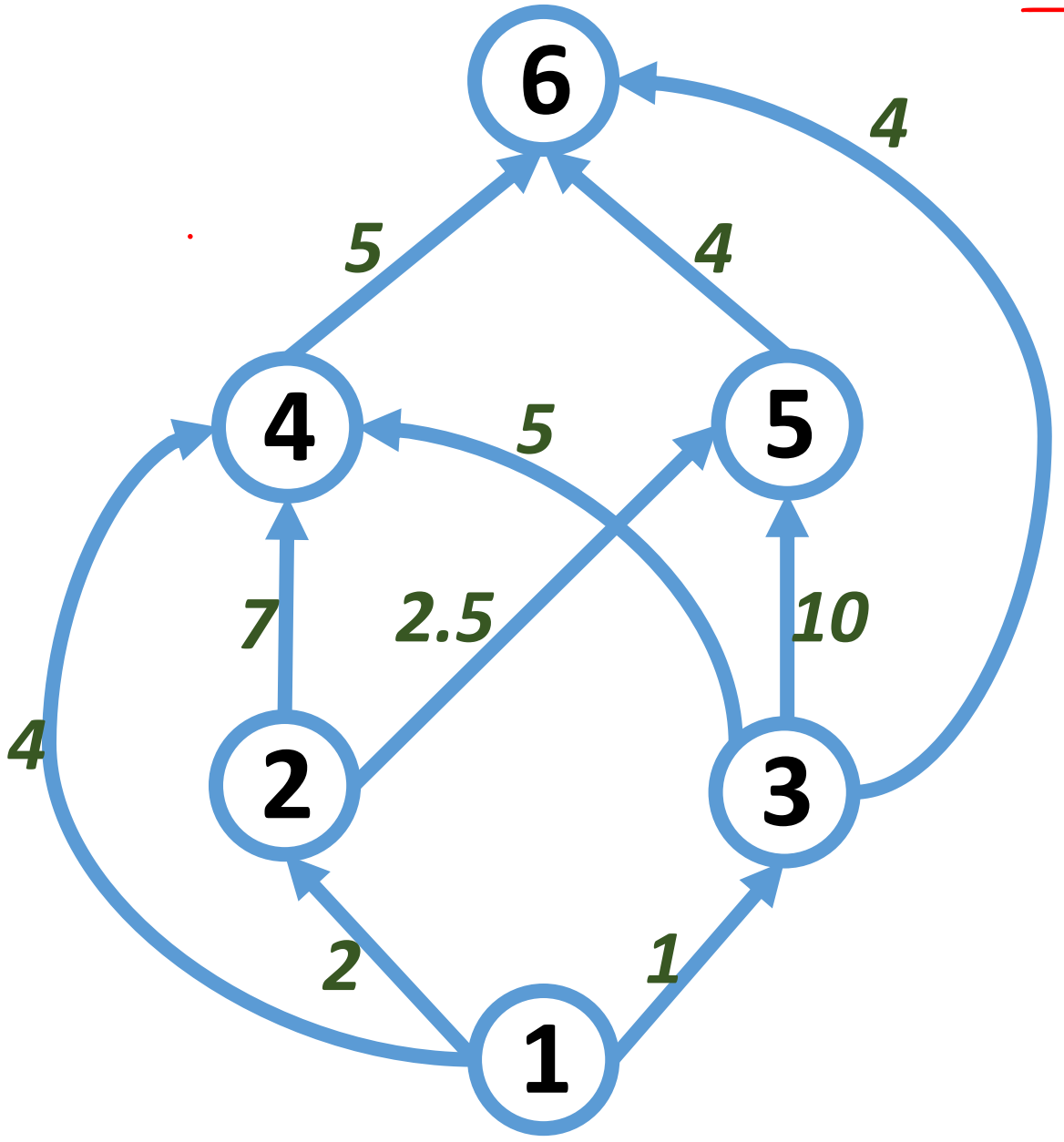
Алгоритм Дейкстры

- Шаг 1. Всем вершинам, за исключением первой, присваивается *вес* равный бесконечности, а первой вершине – 0.
- Шаг 2. Все вершины не выделены.
- Шаг 3. Первая *вершина* объявляется текущей.
- Шаг 4. *Вес* всех невыделенных вершин пересчитывается по формуле: *вес* невыделенной вершины есть минимальное число из старого веса данной вершины, суммы веса текущей вершины и веса *ребра*, соединяющего текущую вершину с невыделенной.
- Шаг 5. Среди невыделенных вершин ищется *вершина* с минимальным весом. Если таковая не найдена, то есть *вес* всех вершин равен бесконечности, то *маршрут* не существует. Следовательно, *выход*. Иначе, текущей становится найденная *вершина*. Она же выделяется.
- Шаг 6. Если текущей вершиной оказывается конечная, то *путь* найден, и его *вес* есть *вес* конечной вершины.
- Шаг 7. Переход на шаг 4.

М.К. Горденко, mgordenko@hse.ru

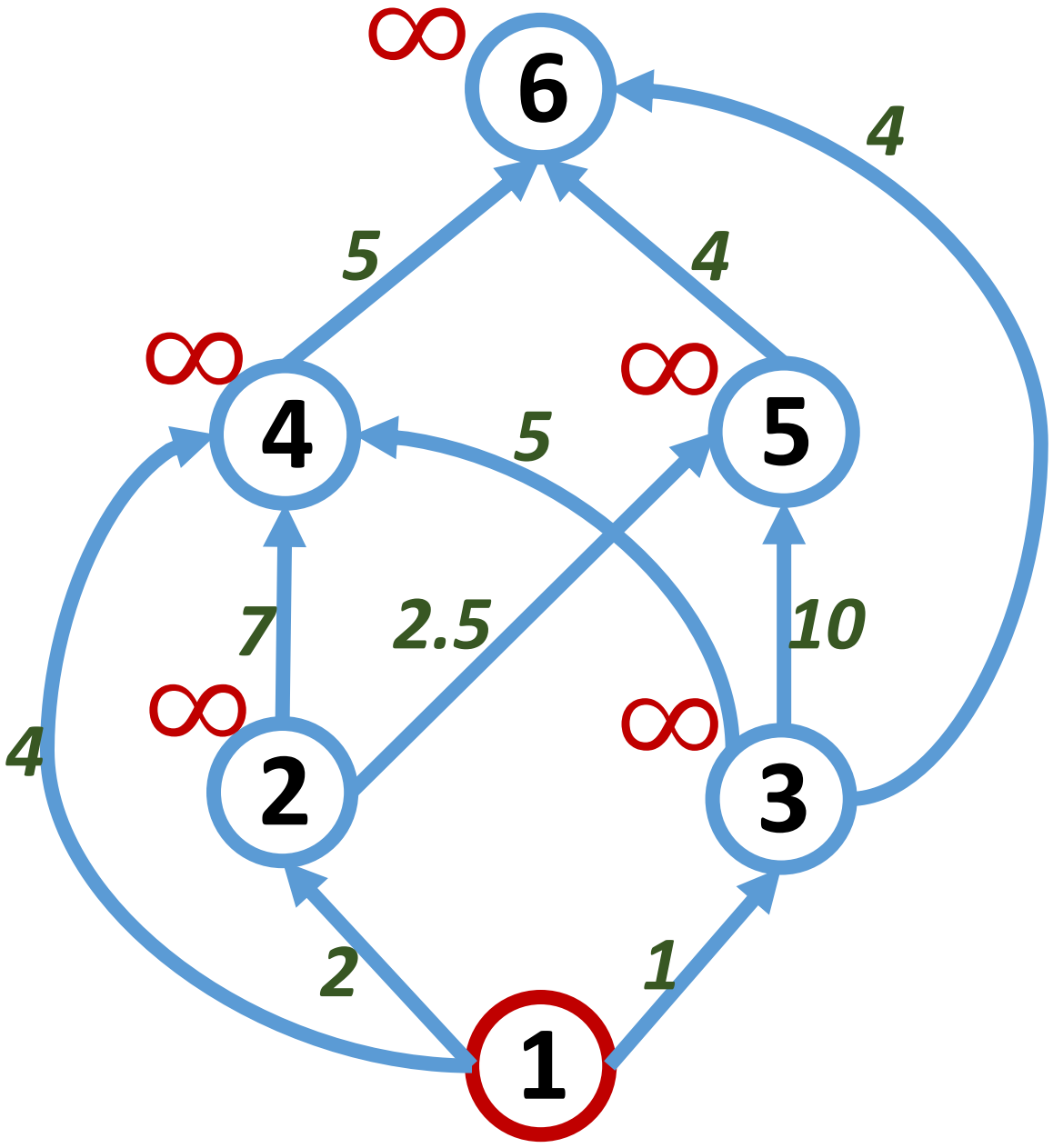
https://www.intuit.ru/studies/courses/648/504/lecture/11475

Найти кратчайший путь из вершины 1 в вершину 6

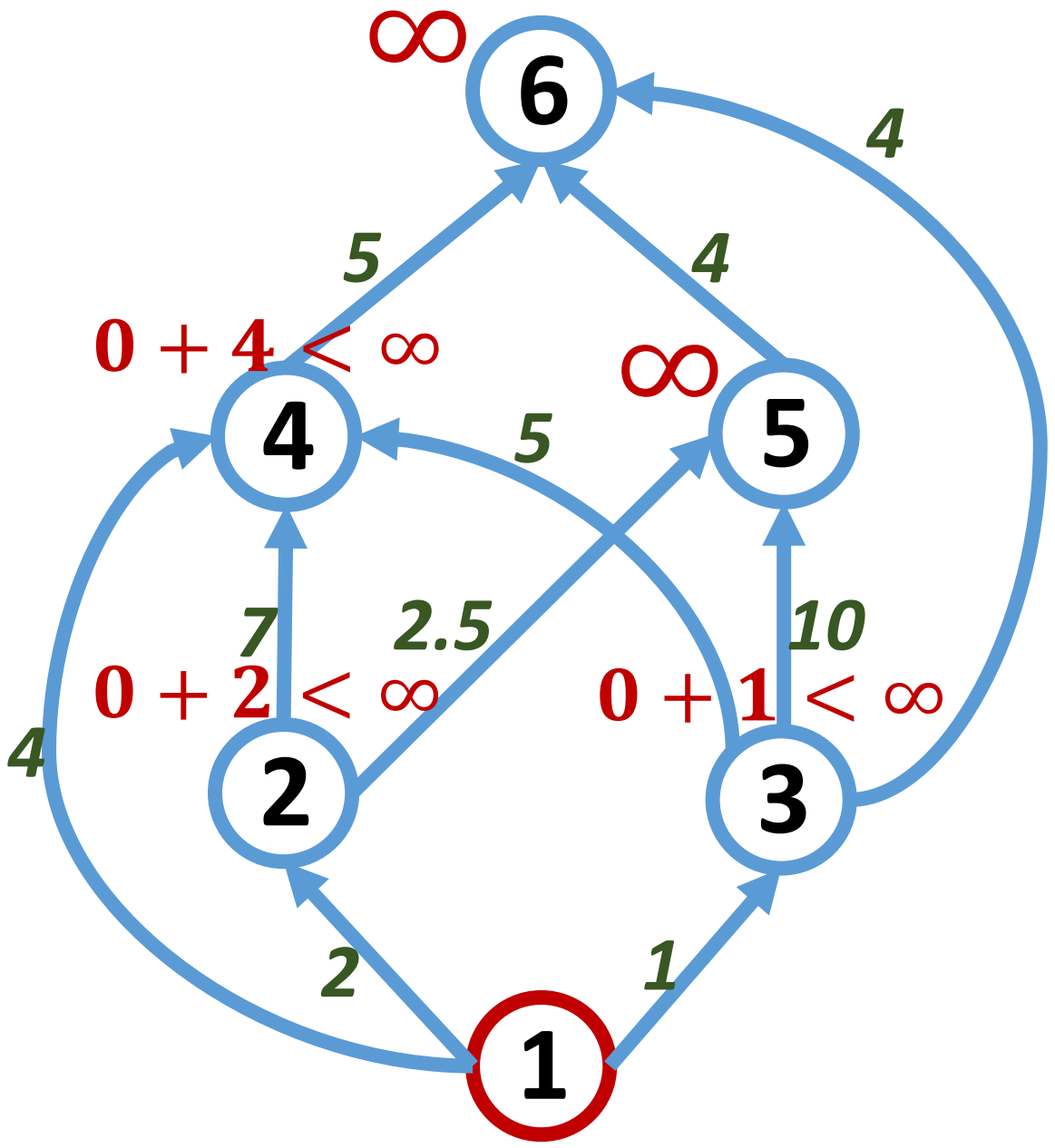


1	2	3	4	5	6
0	0	0	0	0	0

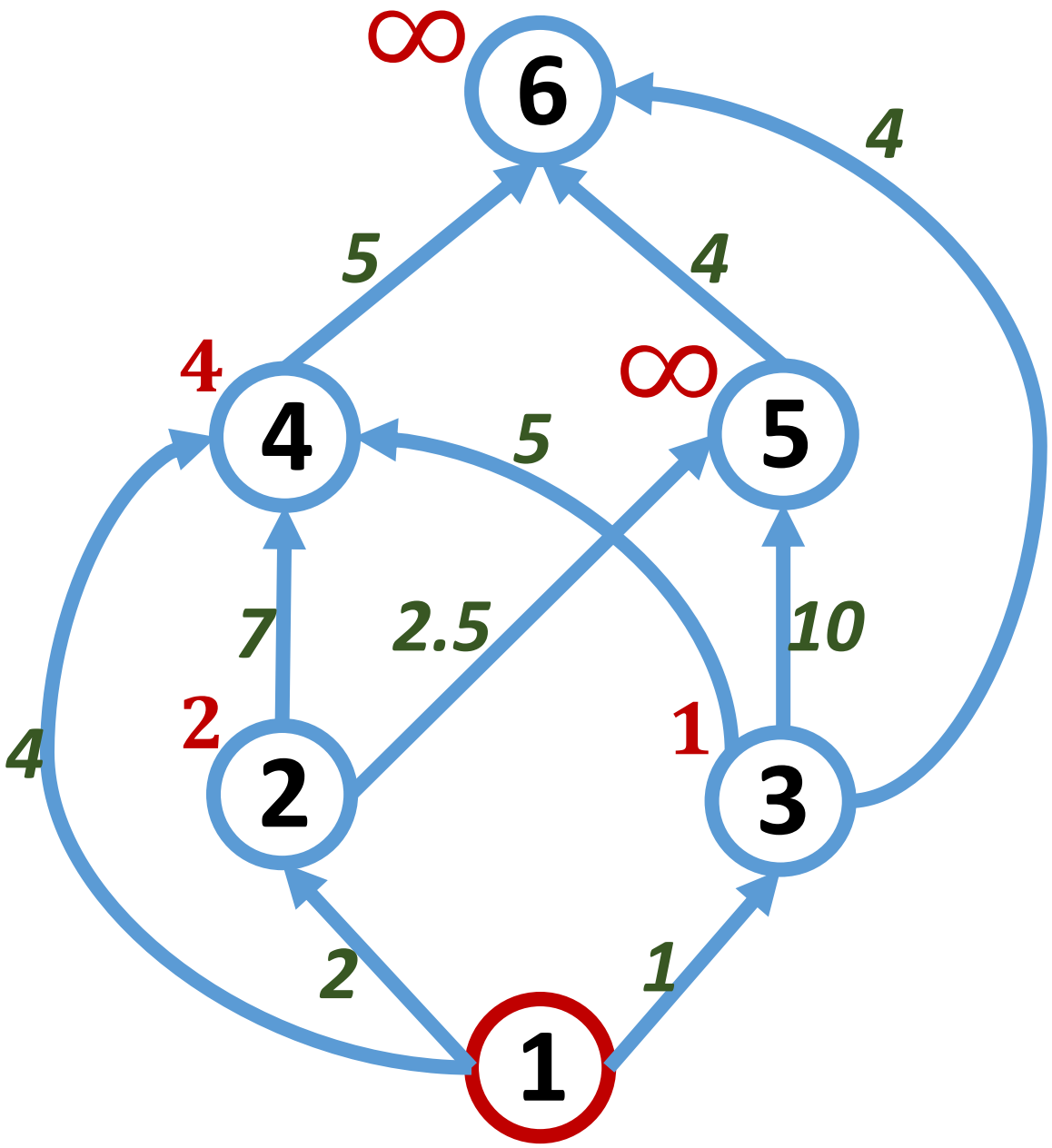
М.К. Горденко, mgordenko@hse.ru



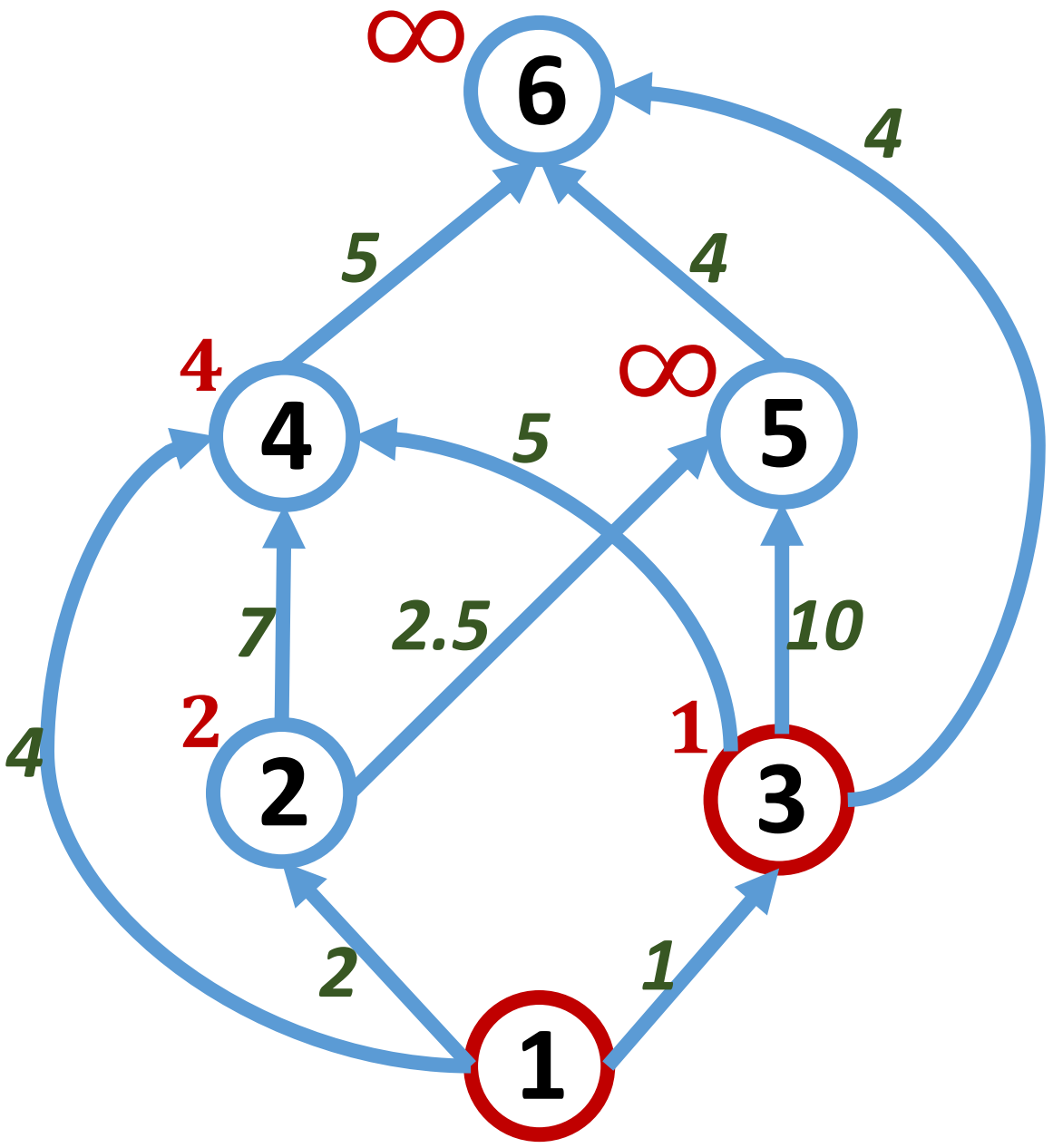
1	2	3	4	5	6
0	∞	∞	∞	∞	∞
0	0	0	0	0	0



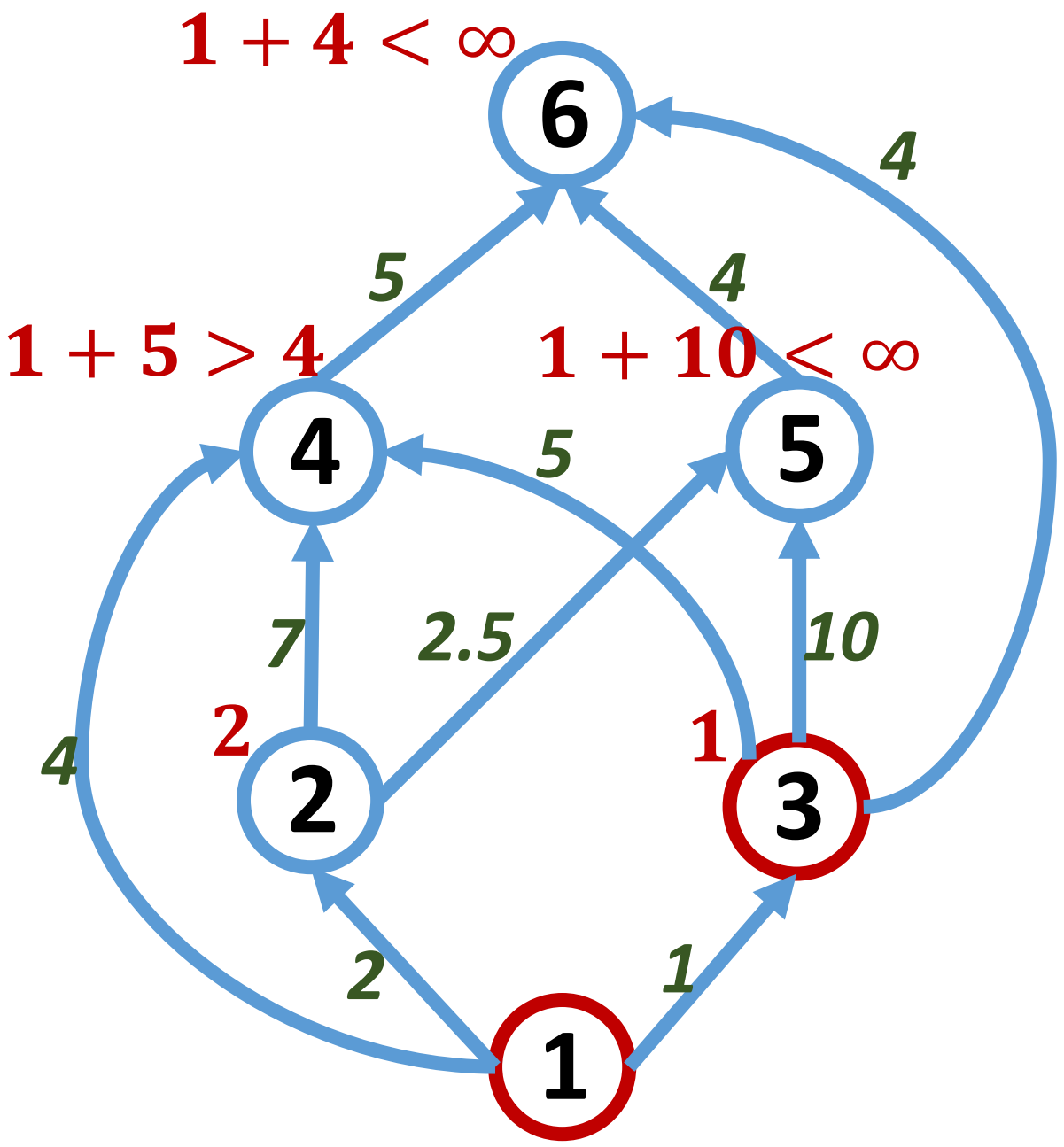
1	2	3	4	5	6
0	∞	∞	∞	∞	∞
	0+2<∞	0+1<∞	0+4<∞		
0	0	0	0	0	0



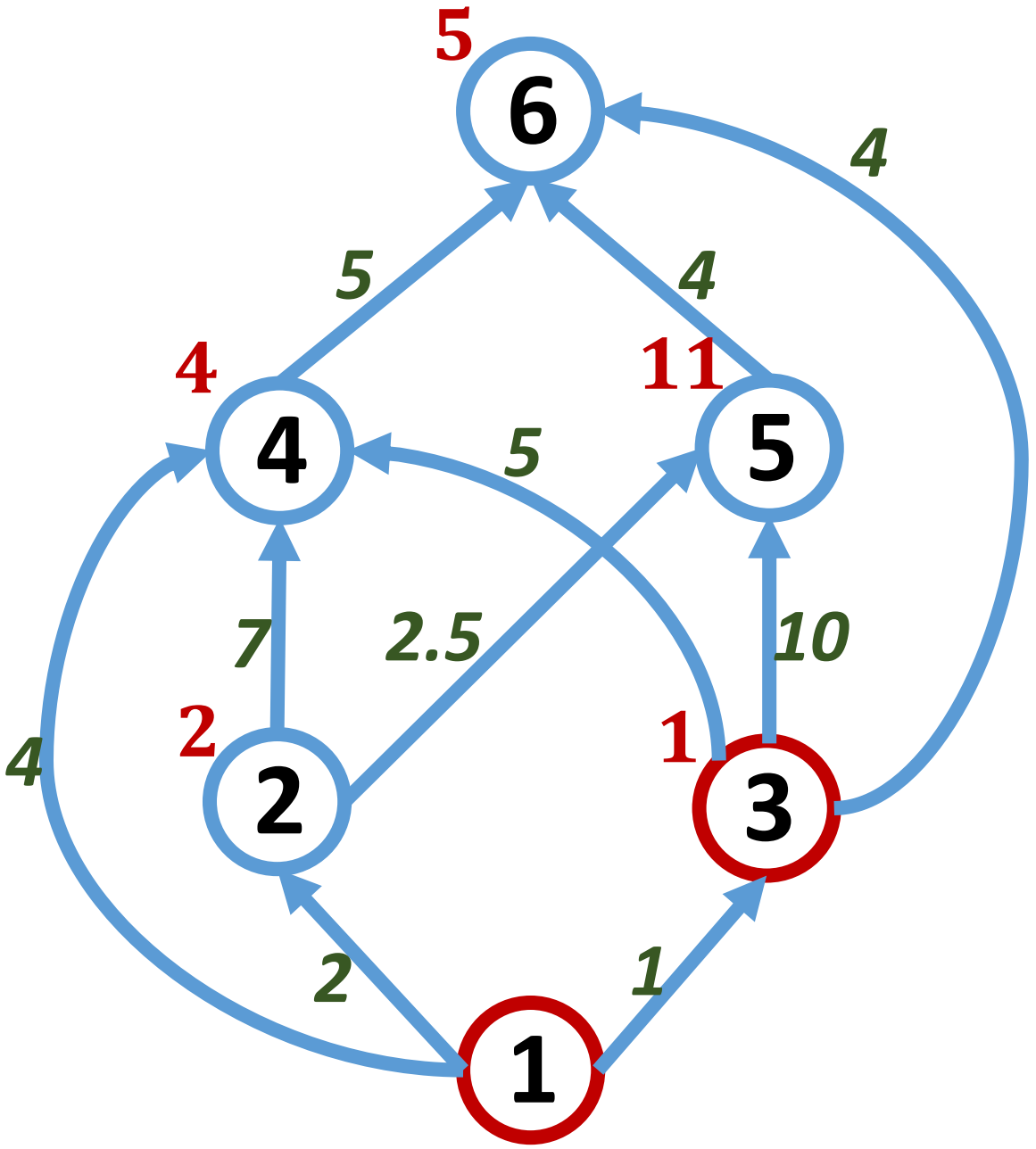
1	2	3	4	5	6
0	∞	∞	∞	∞	∞
	2	1	4	∞	∞
0	1	1	1	0	0



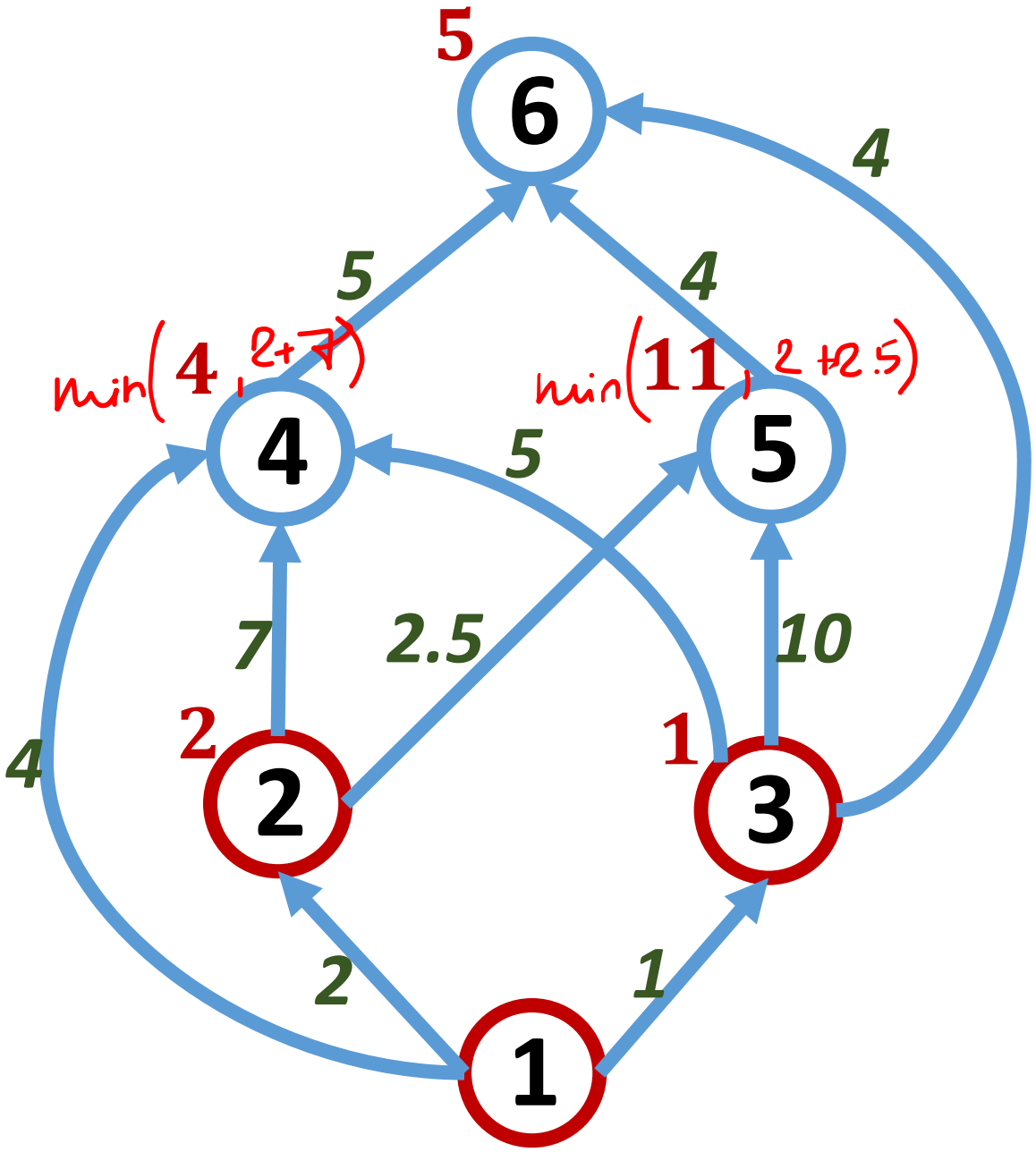
1	2	3	4	5	6
0	∞	∞	∞	∞	∞
	2	1	4	∞	∞
0	1	1	1	0	0



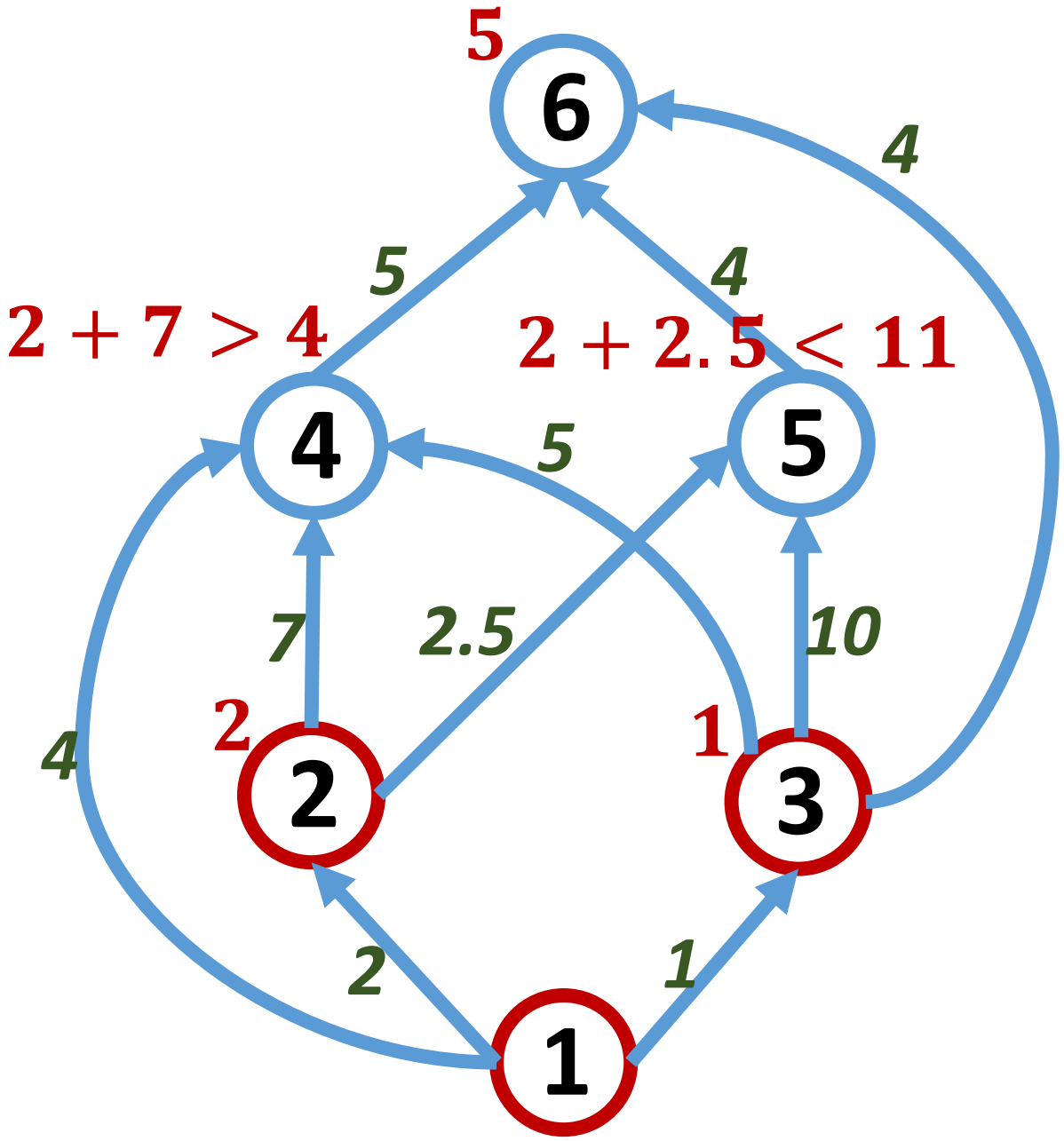
1	2	3	4	5	6
0	∞	∞	∞	∞	∞
	2	1	4	∞	∞
	2		1+5>4	1+10<∞	1+4<∞
0	1	1	1	0	0



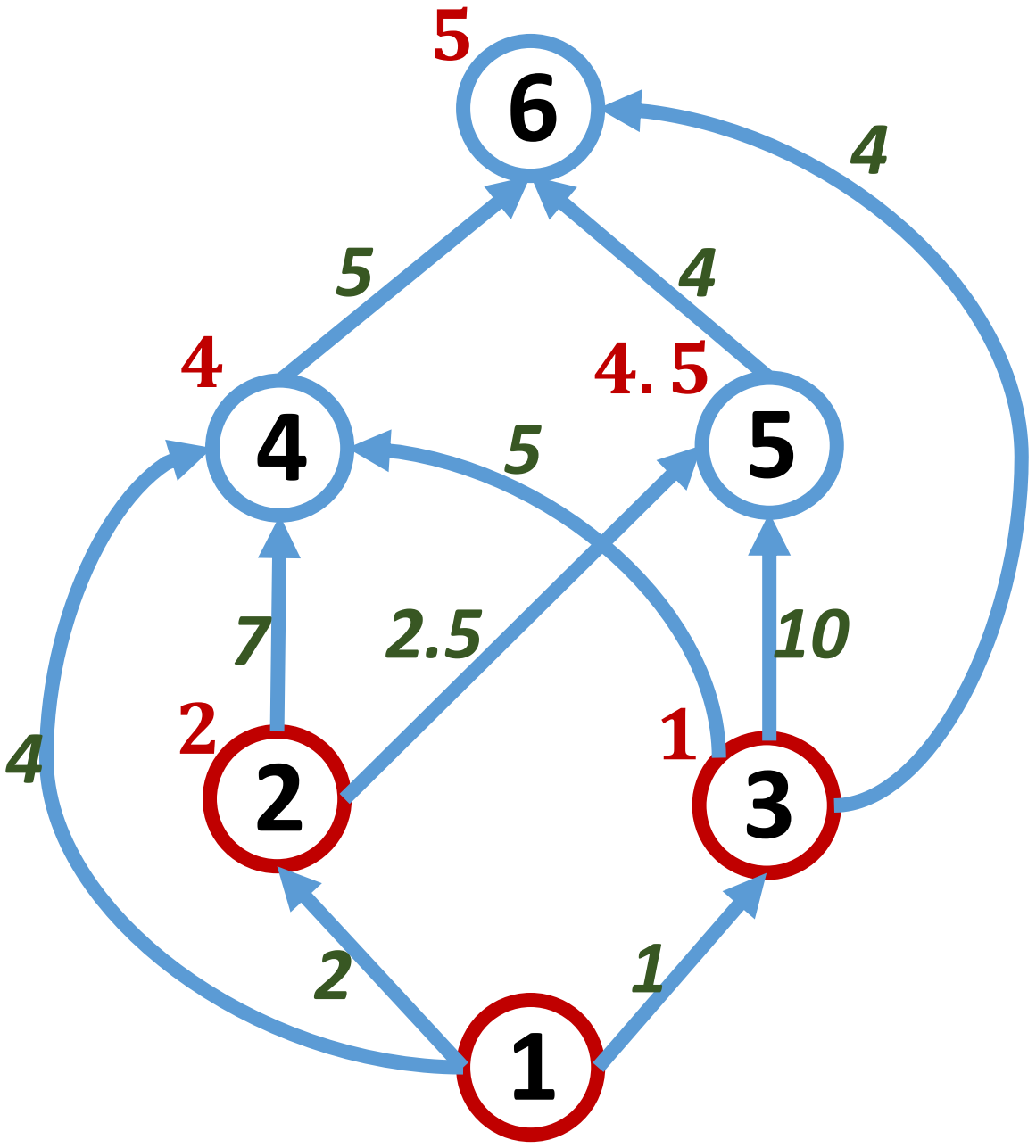
1	2	3	4	5	6
0	∞	∞	∞	∞	∞
	2	1	4	∞	∞
	2		4	11	5
0	1	1	1	3	3



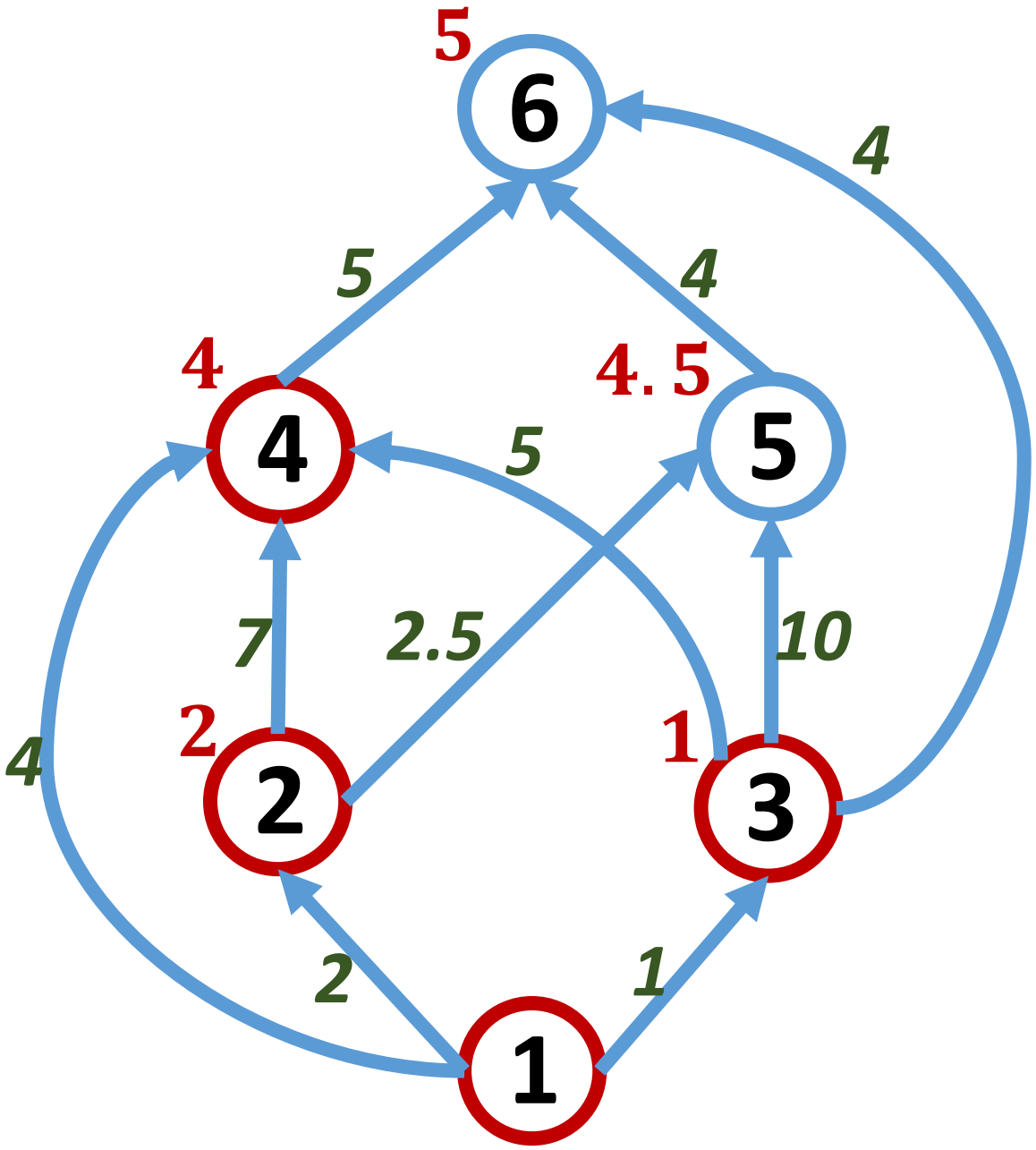
1	2	3	4	5	6
0	∞	∞	∞	∞	∞
	2	1	4	∞	∞
	2		4	11	5
0	1	1	1	3	3



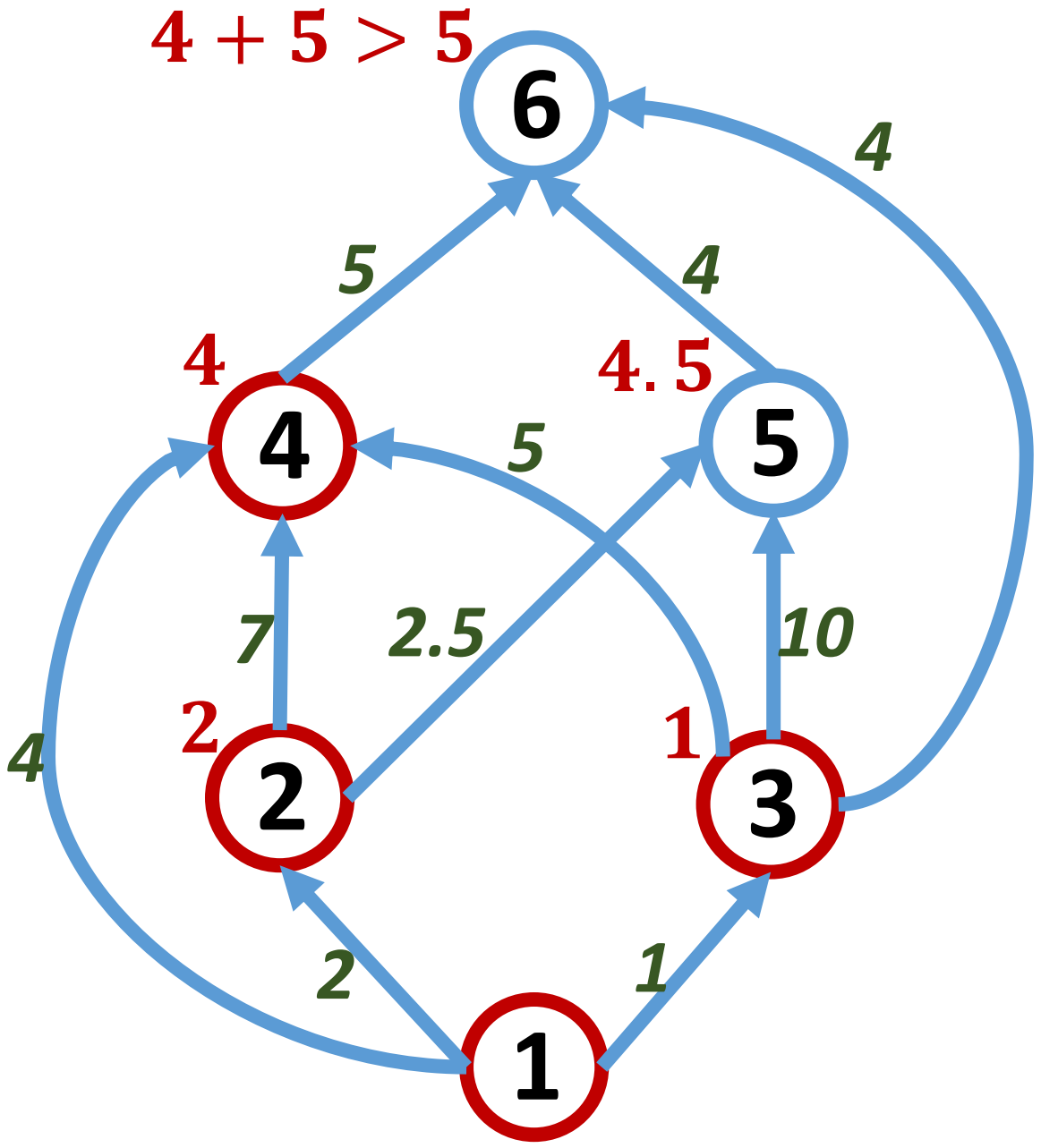
1	2	3	4	5	6
0	∞	∞	∞	∞	∞
	2	1	4	∞	∞
	2		4	11	5
			2+7>4	2+2.5< 11	5
0	1	1	1	3	3



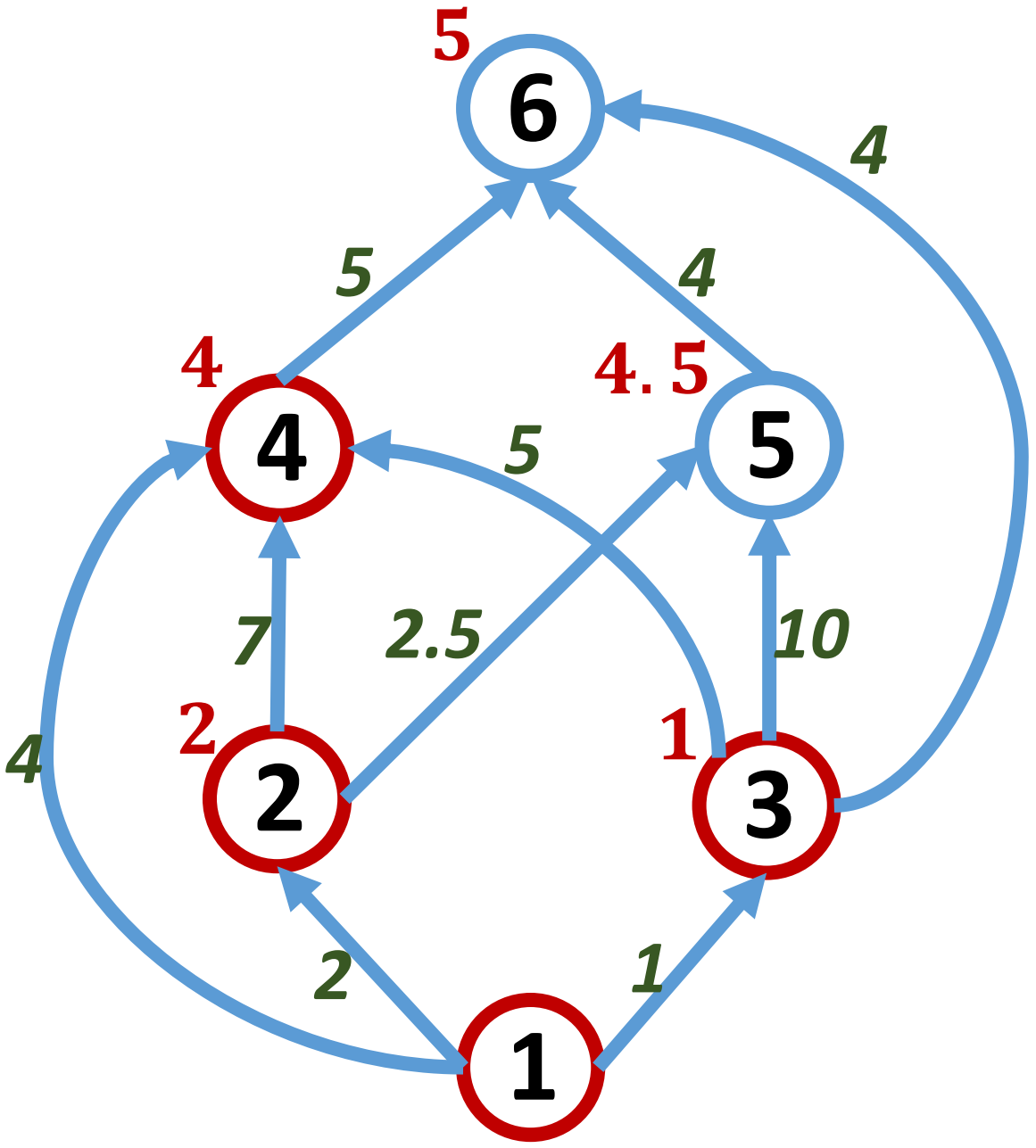
1	2	3	4	5	6
0	∞	∞	∞	∞	∞
	2	1	4	∞	∞
	2		4	11	5
			4	4.5	5
0	1	1	1	2	3



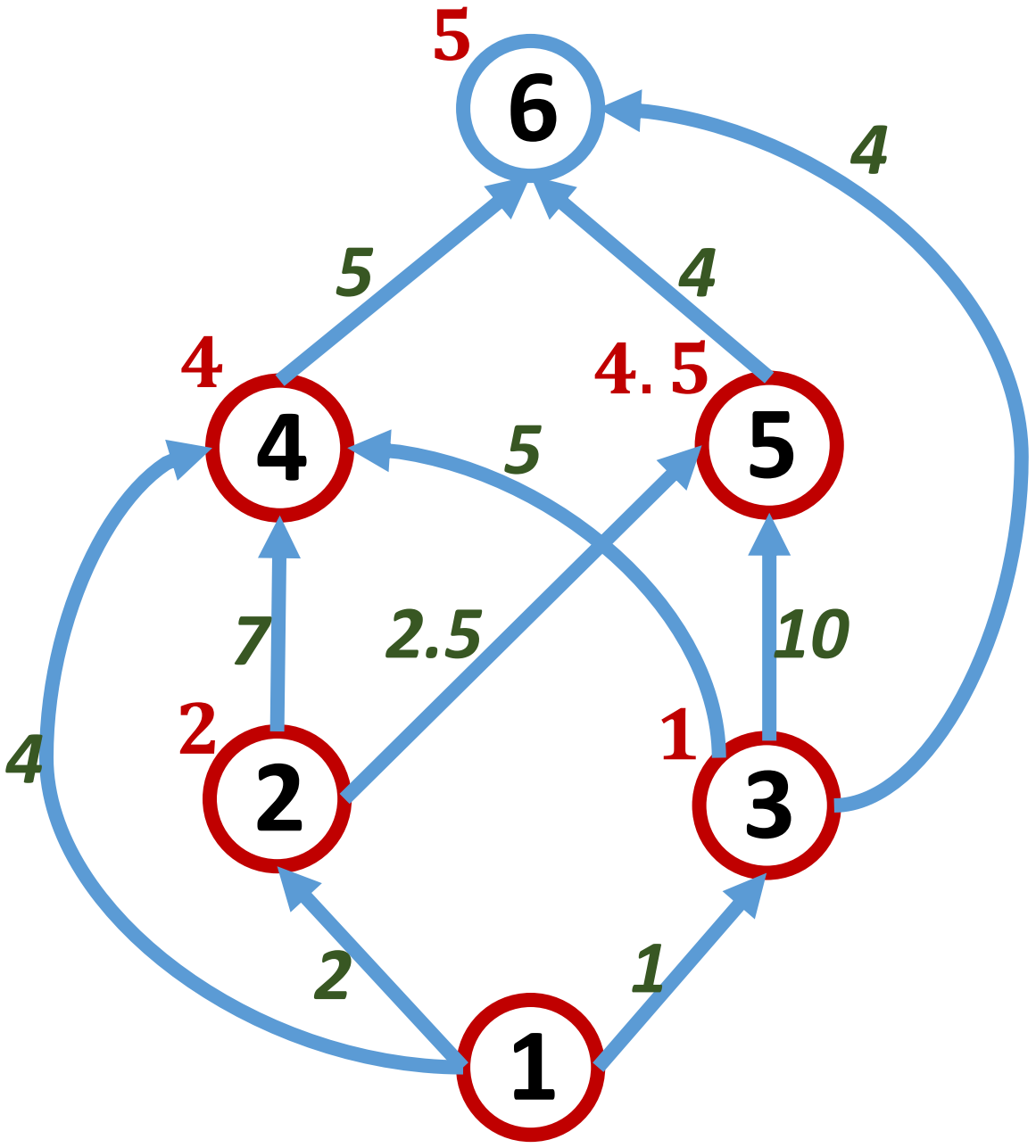
1	2	3	4	5	6
0	∞	∞	∞	∞	∞
	2	**1**	4	∞	∞
	2		4	11	5
			4	4.5	5
0	1	1	1	2	3



1	2	3	4	5	6
0	∞	∞	∞	∞	∞
	2	1	4	∞	∞
	2		4	11	5
			4	4.5	5
					4+5>5
0	1	1	1	2	3

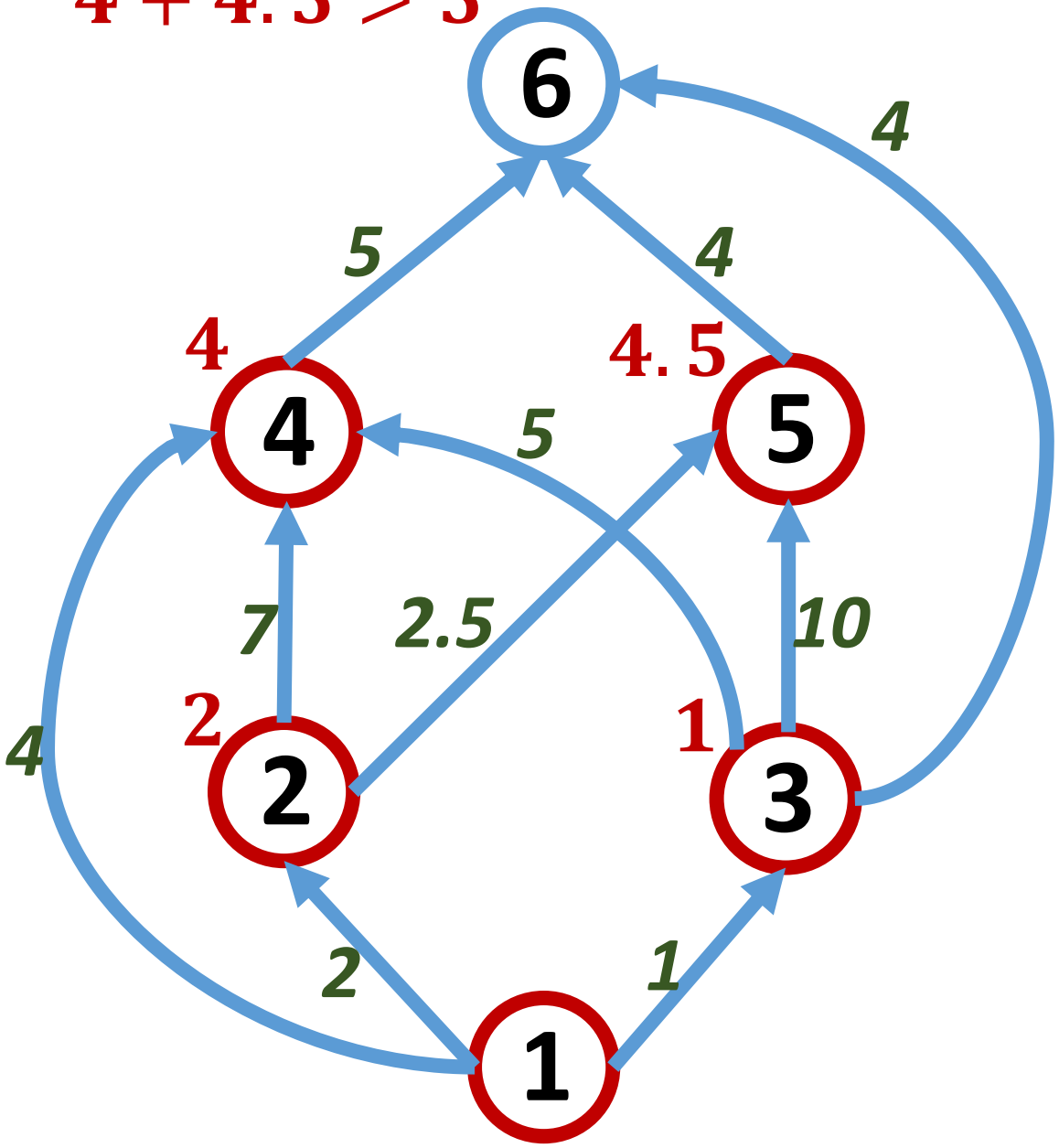


1	2	3	4	5	6
0	∞	∞	∞	∞	∞
	2	**1**	4	∞	∞
	2		4	11	5
			4	4.5	5
				4.5	5
0	1	1	1	2	3

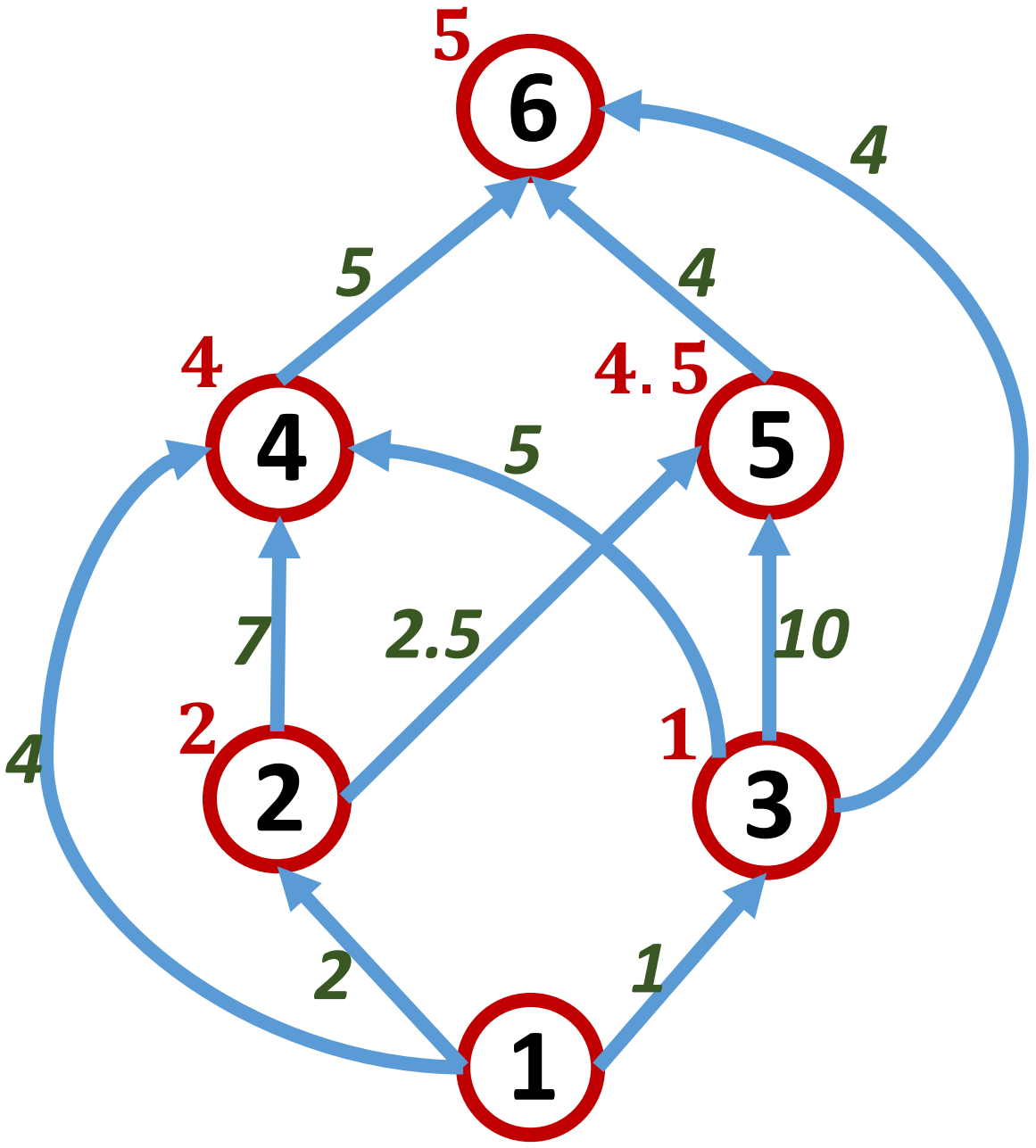


1	2	3	4	5	6
0	∞	∞	∞	∞	∞
	2	**1**	4	∞	∞
	2		4	11	5
			4	4.5	5
				4.5	5
0	1	1	1	2	3

$4 + 4.5 > 5$



1	2	3	4	5	6
0	∞	∞	∞	∞	∞
	2	1	4	∞	∞
	2		4	11	5
			4	4.5	5
				4.5	5
					4+4.5 >5
0	1	1	1	2	3



1	2	3	4	5	6
0	∞	∞	∞	∞	∞
	2	**1**	4	∞	∞
	2		4	11	5
			4	4.5	5
				4.5	5
					5
0	1	1	1	2	3

Алгоритм Флойда-Уоршелла

Алгоритм Флойда-Уоршелла

Алгоритм Флойда – Уоршелла – динамический алгоритм вычисления значений кратчайших путей для каждой из вершин графа. Метод работает на взвешенных графах, с положительными и отрицательными весами ребер, но без отрицательных циклов, являясь, таким образом, более общим в сравнении с алгоритмом Дейкстры, т. к. последний не работает с отрицательными весами ребер.

https://ru.wikipedia.org/wiki/Алгоритм_Флойда_—_Уоршелла

Алгоритм Флойда-Уоршелла

Пусть вершины графа $G = (V,\ E),\ |V| = n$ пронумерованы от 1 до n и введено обозначение d_{ij}^k для длины кратчайшего пути от i до j, который кроме самих вершин $i,\ j$ проходит только через вершины $1 \ldots k$. Очевидно, что d_{ij}^0 — длина (вес) ребра $(i,\ j)$, если таковое существует (в противном случае его длина может быть обозначена как ∞).

Существует два варианта значения $d_{ij}^k,\ k \in (1,\ \ldots,\ n)$:

1. Кратчайший путь между $i,\ j$ не проходит через вершину k, тогда $d_{ij}^k = d_{ij}^{k-1}$
2. Существует более короткий путь между $i,\ j$, проходящий через k, тогда он сначала идёт от i до k, а потом от k до j. В этом случае, очевидно, $d_{ij}^k = d_{ik}^{k-1} + d_{kj}^{k-1}$

Таким образом, для нахождения значения функции достаточно выбрать минимум из двух обозначенных значений.

Тогда рекуррентная формула для d_{ij}^k имеет вид:

d_{ij}^0 — длина ребра $(i,\ j)$;

$d_{ij}^k = \min(d_{ij}^{k-1},\ d_{ik}^{k-1} + d_{kj}^{k-1})$.

https://e-maxx.ru/algo/floyd_warshall_algorithm

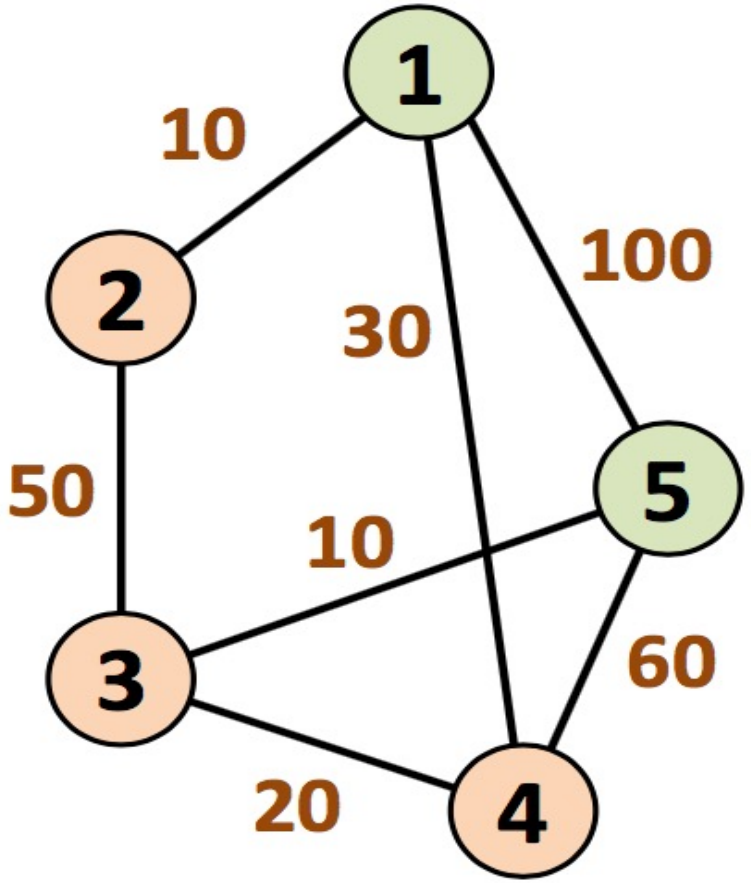
На вход программе подаётся граф, заданный в виде матрицы смежности — двумерного массива $d[][]$ размера $n \times n$, в котором каждый элемент задаёт длину ребра между соответствующими вершинами.

Требуется, чтобы выполнялось $d[i][i] = 0$ для любых i.

```
for (int k=0; k<n; ++k)
        for (int i=0; i<n; ++i)
                for (int j=0; j<n; ++j)
                        d[i][j] = min (d[i][j], d[i][k] + d[k][j]);
```

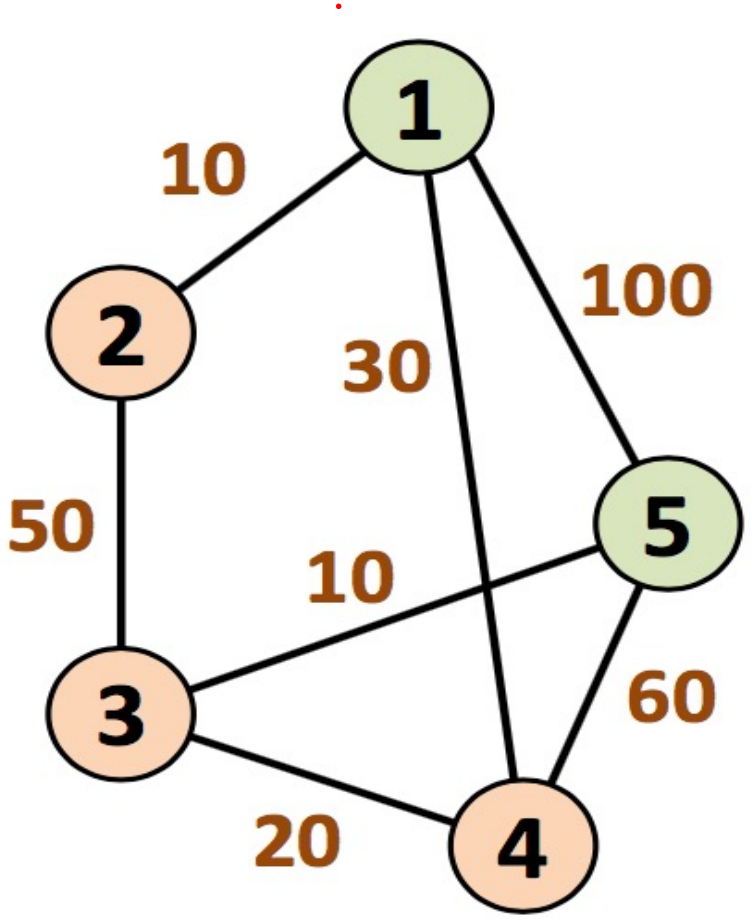
Предполагается, что если между двумя какими-то вершинами **нет ребра**, то в матрице смежности было записано какое-то большое число (достаточно большое, чтобы оно было больше длины любого пути в этом графе); тогда это ребро всегда будет невыгодно брать, и алгоритм сработает правильно.

Необходимо найти кратчайшие пути между каждой парой вершин в графе, представленном на рисунке



	1	2	3	4	5
1					
2					
3					
4					
5					

Необходимо найти кратчайшие пути между каждой парой вершин в графе, представленном на рисунке



	1	2	3	4	5
1	0	10	∞	30	100
2	10	0	50	∞	∞
3	∞	50	0	20	10
4	30	∞	20	0	60
5	100	∞	10	60	0

d[i][j] = min(d[i][j], d[i][k] + d[k][j])
Например, k = 1

d[1][1] = min(d[1][1], d[1][1] + d[1][1])
…
d[1][5] = min(d[1][5], d[1][1] + d[5][1])

d[2][2] = min(d[2][2], d[2][1] + d[1][2])
d[2][3] = min(d[2][3], d[2][1] + d[1][3])
d[2][4] = min(d[2][4], d[2][1] + d[1][4])
d[2][5] = min(d[2][5], d[2][1] + d[1][5])

d[3][3] = min(d[3][3], d[3][1] + d[1][3])
d[3][4] = min(d[3][4], d[3][1] + d[1][4])
d[3][5] = min(d[3][5], d[3][1] + d[1][5])

d[4][4] = min(d[4][4], d[4][1] + d[1][4])
d[4][5] = min(d[4][5], d[4][1] + d[1][5])

d[5][5] = min(d[5][5], d[5][1] + d[1][5])

	1	**2**	**3**	**4**	**5**
1	0	10	∞	30	100
2	10	0	50	∞	∞
3	∞	50	0	20	10
4	30	∞	20	0	60
5	100	∞	10	60	0

d[i][j] = min(d[i][j], d[i][k] + d[k][j])
Например, k = 1

d[1][1] = min(d[1][1], d[1][1] + d[1][1])
…
d[1][5] = min(d[1][5], d[1][1] + d[5][1])

d[2][2] = min(d[2][2], d[2][1] + d[1][2])
d[2][3] = min(d[2][3], d[2][1] + d[1][3])
d[2][4] = min(d[2][4], d[2][1] + d[1][4])
d[2][5] = min(d[2][5], d[2][1] + d[1][5])

d[3][3] = min(d[3][3], d[3][1] + d[1][3])
d[3][4] = min(d[3][4], d[3][1] + d[1][4])
d[3][5] = min(d[3][5], d[3][1] + d[1][5])

d[4][4] = min(d[4][4], d[4][1] + d[1][4])
d[4][5] = min(d[4][5], d[4][1] + d[1][5])

d[5][5] = min(d[5][5], d[5][1] + d[1][5])

	1	2	3	4	5
1	0	10	∞	30	100
2	10	0	50	**40**	**110**
3	∞	50	0	20	10
4	30	**40**	20	0	60
5	100	**110**	10	60	0

d[i][j] = min(d[i][j], d[i][k] + d[k][j])

А теперь, k = 2

d[1][1] = min(d[1][1], d[1][2] + d[2][1])
d[1][2] = min(d[1][2], d[1][2] + d[2][2])
d[1][3] = min(d[1][3], d[1][2] + d[2][3])
d[1][4] = min(d[1][4], d[1][2] + d[2][4])
d[1][5] = min(d[1][5], d[1][2] + d[2][5])

d[2][2] = min(d[2][2], d[2][2] + d[2][2])
…
d[2][5] = min(d[2][5], d[2][2] + d[2][5])

d[3][3] = min(d[3][3], d[3][2] + d[2][3])
d[3][4] = min(d[3][4], d[3][2] + d[2][4])
d[3][5] = min(d[3][5], d[3][2] + d[2][5])

d[4][4] = min(d[4][4], d[4][2] + d[2][4])
d[4][5] = min(d[4][5], d[4][2] + d[2][5])

d[5][5] = min(d[5][5], d[5][2] + d[2][5])

	1	**2**	**3**	**4**	**5**
1	0	10	∞	30	100
2	10	0	50	40	110
3	∞	50	0	20	10
4	30	40	20	0	60
5	100	110	10	60	0

d[i][j] = min(d[i][j], d[i][k] + d[k][j])

А теперь, k = 2

d[1][1] = min(d[1][1], d[1][2] + d[2][1])
d[1][2] = min(d[1][2], d[1][2] + d[2][2])
d[1][3] = min(d[1][3], d[1][2] + d[2][3])
d[1][4] = min(d[1][4], d[1][2] + d[2][4])
d[1][5] = min(d[1][5], d[1][2] + d[2][5])

d[2][2] = min(d[2][2], d[2][2] + d[2][2])
…
d[2][5] = min(d[2][5], d[2][2] + d[2][5])

d[3][3] = min(d[3][3], d[3][2] + d[2][3])
d[3][4] = min(d[3][4], d[3][2] + d[2][4])
d[3][5] = min(d[3][5], d[3][2] + d[2][5])

d[4][4] = min(d[4][4], d[4][2] + d[2][4])
d[4][5] = min(d[4][5], d[4][2] + d[2][5])

d[5][5] = min(d[5][5], d[5][2] + d[2][5])

	1	2	3	4	5
1	0	10	**60**	30	100
2	10	0	50	40	110
3	**60**	50	0	20	10
4	30	40	20	0	60
5	100	110	10	60	0

d[i][j] = min(d[i][j], d[i][k] + d[k][j])
А если k = 3?

d[1][2] = min(d[1][2], d[1][3] + d[3][2])
d[1][3] = min(d[1][3], d[1][3] + d[3][3])
d[1][4] = min(d[1][4], d[1][3] + d[3][4])
d[1][5] = min(d[1][5], d[1][3] + d[3][5])

d[2][3] = min(d[2][3], d[2][3] + d[3][3])
d[2][4] = min(d[2][4], d[2][3] + d[3][4])
d[2][5] = min(d[2][5], d[2][3] + d[3][5])

d[3][4] = min(d[3][4], d[3][3] + d[3][4])
d[3][5] = min(d[3][5], d[3][3] + d[3][5])

d[4][5] = min(d[4][5], d[4][3] + d[3][5])

	1	2	3	4	5
1	0	10	60	30	100
2	10	0	50	40	110
3	60	50	0	20	10
4	30	40	20	0	60
5	100	110	10	60	0

d[i][j] = min(d[i][j], d[i][k] + d[k][j])
А если k = 3?

d[1][2] = min(d[1][2], d[1][3] + d[3][2])
d[1][3] = min(d[1][3], d[1][3] + d[3][3])
d[1][4] = min(d[1][4], d[1][3] + d[3][4])
d[1][5] = min(d[1][5], d[1][3] + d[3][5])

d[2][3] = min(d[2][3], d[2][3] + d[3][3])
d[2][4] = min(d[2][4], d[2][3] + d[3][4])
d[2][5] = min(d[2][5], d[2][3] + d[3][5])

d[3][4] = min(d[3][4], d[3][3] + d[3][4])
d[3][5] = min(d[3][5], d[3][3] + d[3][5])

d[4][5] = min(d[4][5], d[4][3] + d[3][5])

	1	**2**	**3**	**4**	**5**
1	0	10	60	30	**70**
2	10	0	50	40	**60**
3	60	50	0	20	10
4	30	40	20	0	**30**
5	**70**	**60**	10	**30**	0

d[i][j] = min(d[i][j], d[i][k] + d[k][j])
Что насчет k = 4?

d[1][2] = min(d[1][2], d[1][4] + d[4][2])
d[1][3] = min(d[1][3], d[1][4] + d[4][3])
d[1][4] = min(d[1][4], d[1][4] + d[4][4])
d[1][5] = min(d[1][5], d[1][4] + d[4][5])

d[2][3] = min(d[2][3], d[2][4] + d[4][3])
d[2][4] = min(d[2][4], d[2][4] + d[4][4])
d[2][5] = min(d[2][5], d[2][4] + d[4][5])

d[3][4] = min(d[3][4], d[3][4] + d[4][4])
d[3][5] = min(d[3][5], d[3][4] + d[4][5])

d[4][5] = min(d[4][5], d[4][4] + d[4][5])

	1	**2**	**3**	**4**	**5**
1	0	10	60	30	70
2	10	0	50	40	60
3	60	50	0	20	10
4	30	40	20	0	30
5	70	60	10	30	0

d[i][j] = min(d[i][j], d[i][k] + d[k][j])
Что насчет k = 4?

d[1][2] = min(d[1][2], d[1][4] + d[4][2])
d[1][3] = min(d[1][3], d[1][4] + d[4][3])
d[1][4] = min(d[1][4], d[1][4] + d[4][4])
d[1][5] = min(d[1][5], d[1][4] + d[4][5])

d[2][3] = min(d[2][3], d[2][4] + d[4][3])
d[2][4] = min(d[2][4], d[2][4] + d[4][4])
d[2][5] = min(d[2][5], d[2][4] + d[4][5])

d[3][4] = min(d[3][4], d[3][4] + d[4][4])
d[3][5] = min(d[3][5], d[3][4] + d[4][5])

d[4][5] = min(d[4][5], d[4][4] + d[4][5])

	1	2	3	4	5
1	0	10	50	30	60
2	10	0	50	40	60
3	50	50	0	20	10
4	30	40	20	0	30
5	60	60	10	30	0

d[i][j] = min(d[i][j], d[i][k] + d[k][j])
И напоследок, k = 5?

d[1][2] = min(d[1][2], d[1][5] + d[5][2])
d[1][3] = min(d[1][3], d[1][5] + d[5][3])
d[1][4] = min(d[1][4], d[1][5] + d[5][4])
d[1][5] = min(d[1][5], d[1][5] + d[5][5])

d[2][3] = min(d[2][3], d[2][5] + d[5][3])
d[2][4] = min(d[2][4], d[2][5] + d[5][4])
d[2][5] = min(d[2][5], d[2][5] + d[5][5])

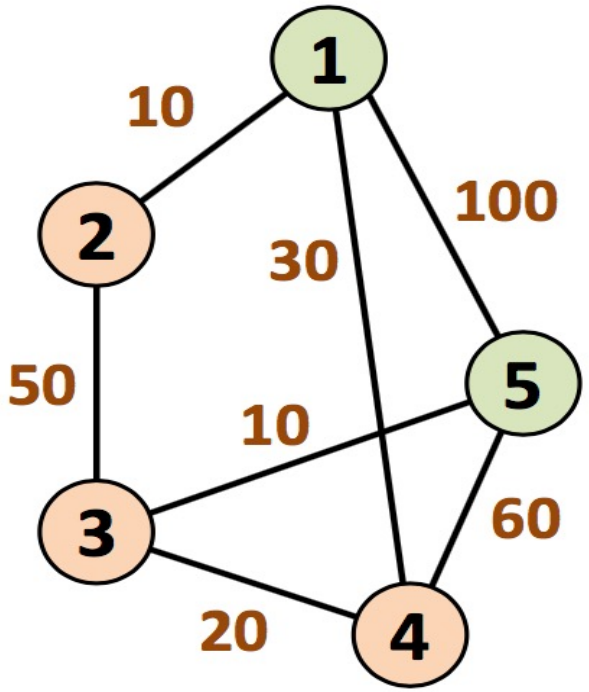
d[3][4] = min(d[3][4], d[3][5] + d[5][4])
d[3][5] = min(d[3][5], d[3][5] + d[5][5])

d[4][5] = min(d[4][5], d[4][5] + d[5][5])

	1	**2**	**3**	**4**	**5**
1	0	10	50	30	60
2	10	0	50	40	60
3	50	50	0	20	10
4	30	40	20	0	30
5	60	60	10	30	0

d[i][j] = min(d[i][j], d[i][k] + d[k][j])
И напоследок, k = 5?

Менять нечего!



	1	2	3	4	5
1	0	10	50	30	60
2	10	0	50	40	60
3	50	50	0	20	10
4	30	40	20	0	30
5	60	60	10	30	0