Введение в теорию графов.

Хайрулин Сергей Сергеевич
s.khayrulin@gmail.com

Overview

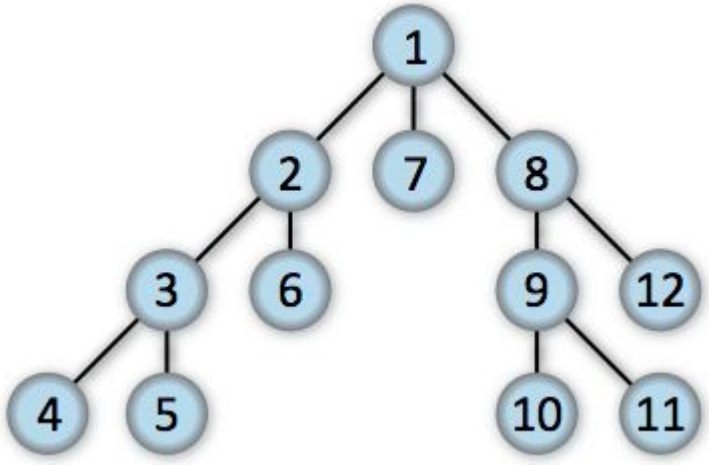
1. Алгоритмы обхода графа
   - В глубину DFS
   - В ширину BFS
2. Сильно связные компоненты графа
   - Алгоритм поиска сильно связных компонент
3. Остовные деревья минимальной стоимости
   - Ациклические графы
   - Свойства
   - Алгоритм Прима
   - Алгоритм Крускала

Литература и др. источники

1. Дональд Эрвин Кнут. Искусство программирования (Том 1, 2, 3) // Вильямс 2015.
2. Альфред В. Ахо, Джон Э. Хопкрофт, Джеффри Д. Ульман. Структуры данных и алгоритмы // Вильямс 2000.
3. Емеличев В. А., Мельников О. И., Сарванов В. И., Тышкевич Р. И. Лекции по теории графов // М.: Наука, 1990.
4. Харари Ф. Теория графов // М.: Мир, 1973.

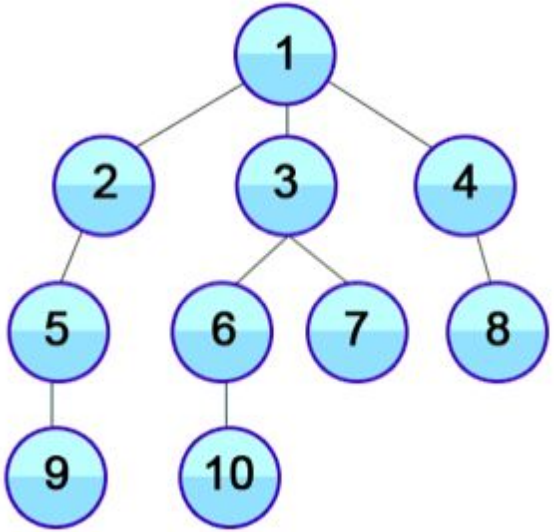
Обход графа в глубину DFS

Поиск в глубину (англ. Depth-first search, **DFS**) — один из методов обхода графа. Стратегия поиска в глубину, как и следует из названия, состоит в том, чтобы идти «вглубь» графа, насколько это возможно.



Обход графа в ширину BFS

Поиск в ширину (англ. breadth-first search, **BFS**) — метод обхода графа и поиска пути в графе.



Подумайте, как модифицировать алгоритмы обхода, для работы со всеми графами. Ориентированными/неориентированными.

Сильная связность

Сильно связной компонентой - ориентированного графа называется максимальное множество вершин, в котором существуют пути из любой вершины в любую другую вершину.

Алгоритм поиска сильно связных компонент

1. Сначала выполняется поиск в глубину на графе G. Вершины нумеруются в порядке завершения рекурсивно вызванной процедуры *dfs*.
2. Конструируется новый ориентированный граф G^r, путем обращения направления всех дуг графа G.

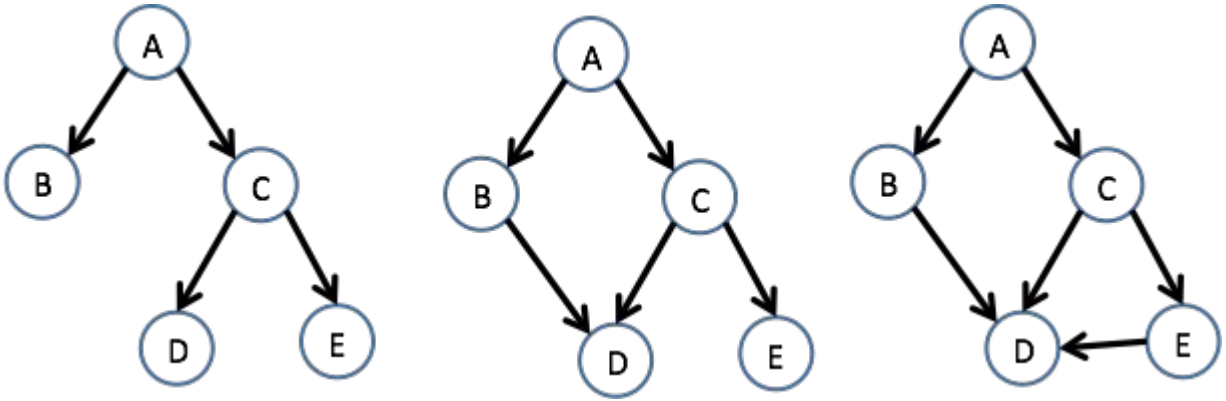
Алгоритм поиска сильно связных компонент

3. Выполняется поиск в глубину на графе G^r, начиная с вершины с наибольшим номером, присвоенным на шаге 1. Если проведенный поиск не охватывает всех вершин, то начинается новый поиск с вершины, имеющей наибольший номер среди оставшихся вершин.
4. Каждое дерево в полученном остовном лесу является сильно связной компонентой графа *G*.

Ациклические орграф

Ориентированный ациклический граф — это орграф, не имеющий циклов. Можно сказать, что ациклический орграф более общая структура, чем дерево, но менее общая по сравнению с обычным ориентированным графом.

Ациклические орграф



Ациклические орграф

Как проверить орграф на ацикличность?

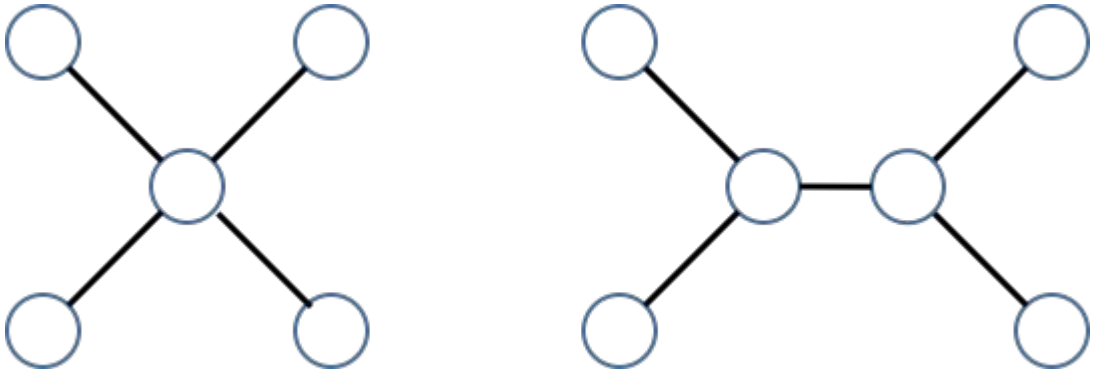
Остовные деревья минимальной стоимости

Пусть G = *(V, E)* — связный граф, в котором каждое ребро <*v*, *w*> помечено числом *c(v, w),* которое называется *стоимостью ребра.*

Остовные деревья минимальной стоимости

Связный ациклический граф, представляющий собой "дерево без корня", называют ***свободным деревом***.

Остовные деревья минимальной стоимости



Остовные деревья минимальной стоимости

Остовным деревом **графа** *G* называется свободное дерево, содержащее все вершины *V* графа G. ***Стоимость*** остовного дерева вычисляется как сумма стоимостей всех ребер, входящих в это дерево.

Остовные деревья минимальной стоимости

1. Каждое свободное дерево с числом вершин n, n > 1, имеет в точности n - 1 ребер
2. Если в свободное дерево добавить новое ребро, то обязательно получится цикл

Остовные деревья минимальной стоимости

Пусть $G = \langle V, E \rangle$ - связный граф с заданной функцией стоимости, определенной на множестве ребер. Обозначим через U - подмножество вершин множества V. Если $\langle u, v \rangle$ - такое ребро наименьшей стоимости, что $u \in U$ и $v \in V \setminus U$, тогда для графа G существует остовное дерево минимальной стоимости, содержащее ребро $\langle u, v \rangle$.

Остовные деревья минимальной стоимости

Как это доказать?

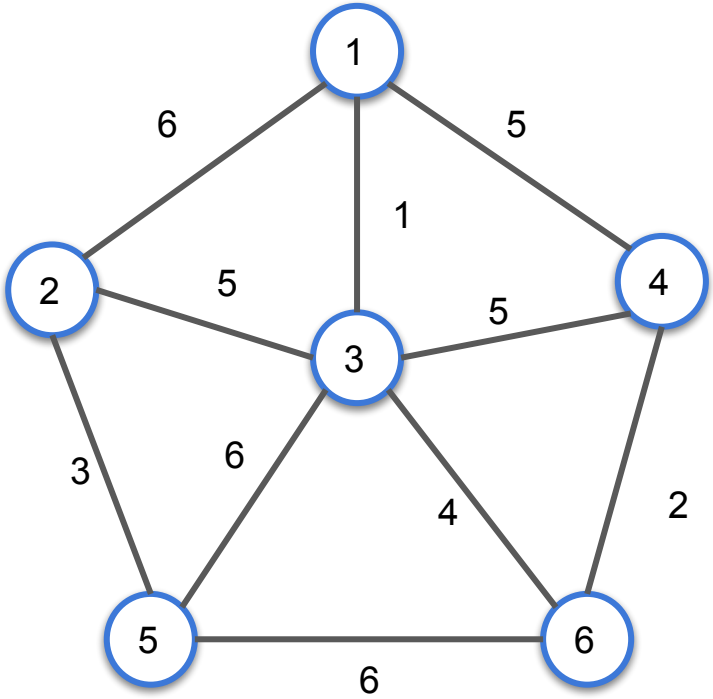
Алгоритм Прима (Prim)

Алгоритм Прима (Prim) – в этом алгоритме строится множество вершин **U**, из которого "вырастает" остовное дерево.

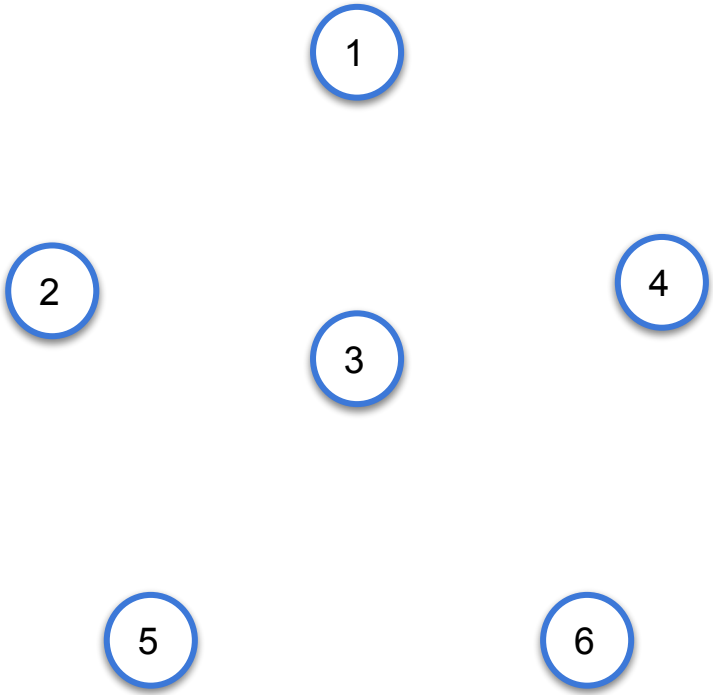
Алгоритм Прима (Prim)

Пусть **V = {1, 2, …, n}**. Сначала **U= {1}**. На каждом шаге алгоритма находится ребро наименьшей стоимости ***(u, v)*** такое, что $u \in U$ и $v \in V \setminus U$, затем вершина **v** переносится из множества $V \setminus U$ в множество U. Этот процесс продолжается до тех пор, пока множество U не станет равным множеству V.

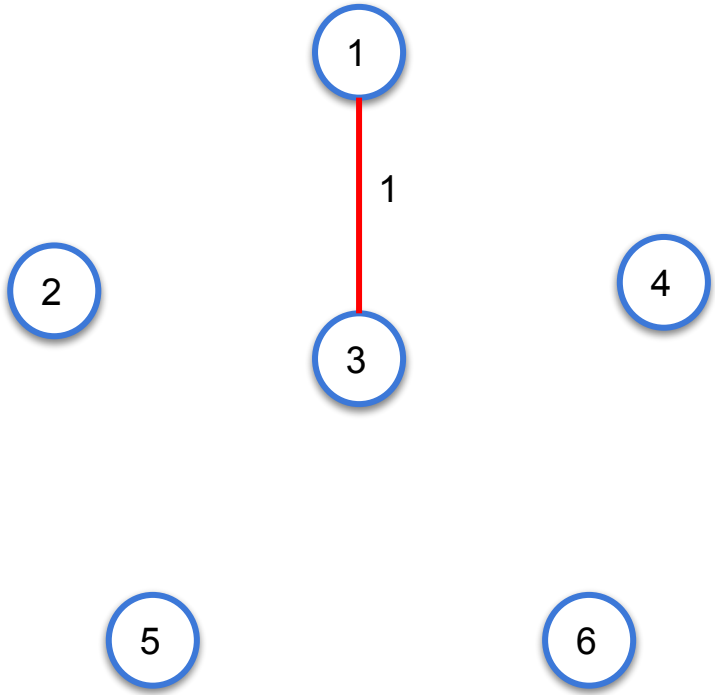
Алгоритм Прима (Prim)



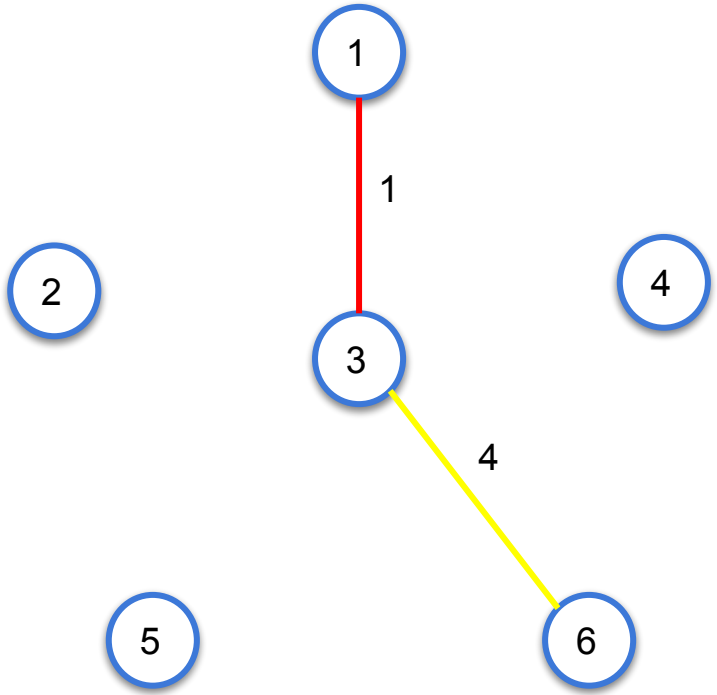
Алгоритм Прима (Prim)



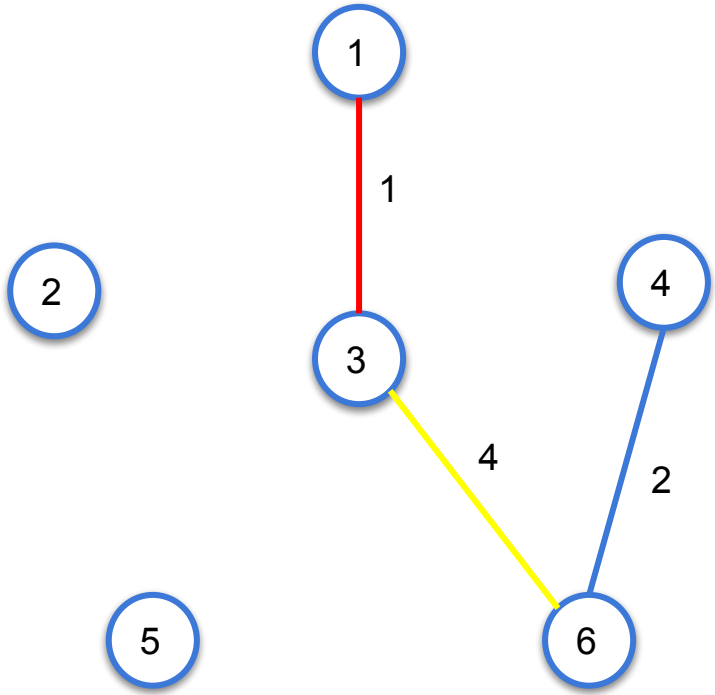
Алгоритм Прима (Prim)



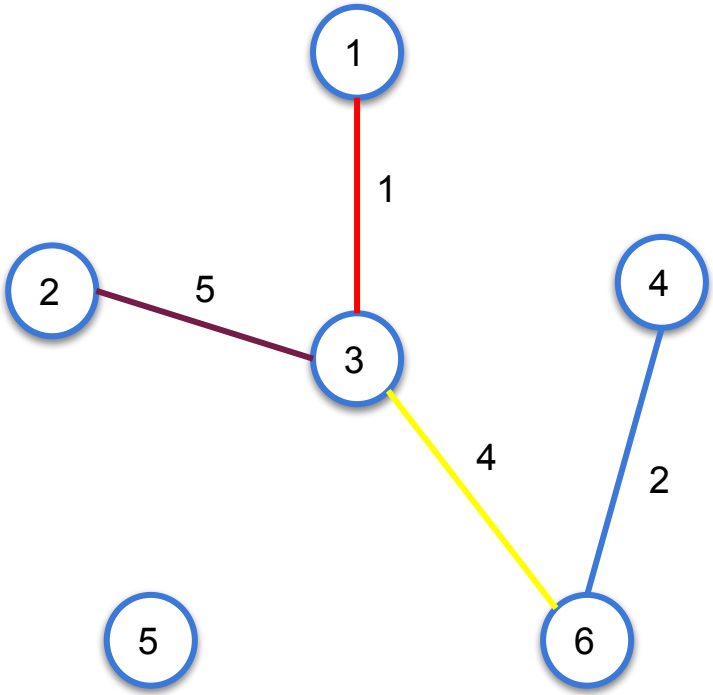
Алгоритм Прима (Prim)



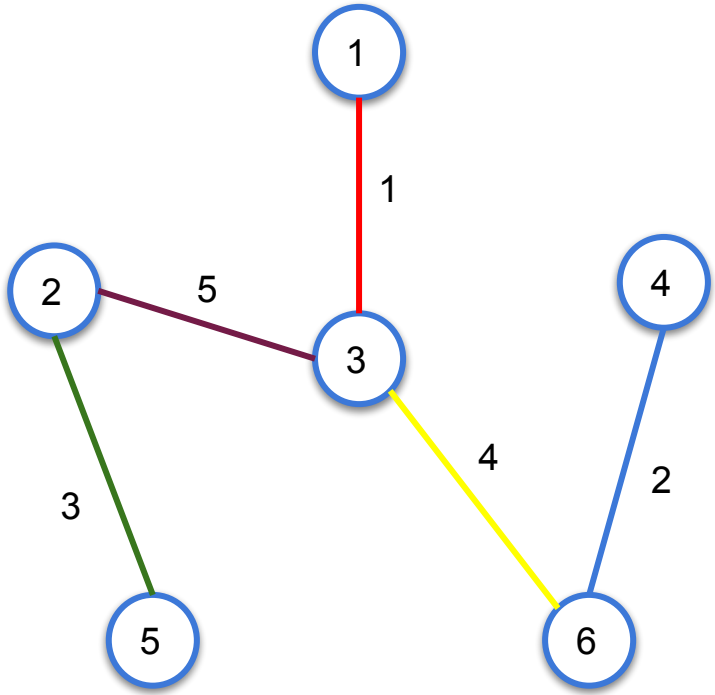
Алгоритм Прима (Prim)



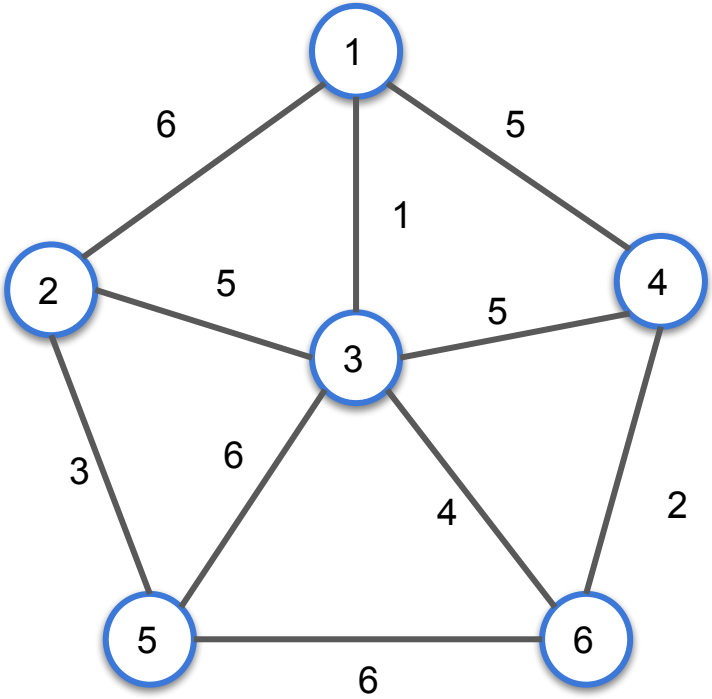
Алгоритм Прима (Prim)

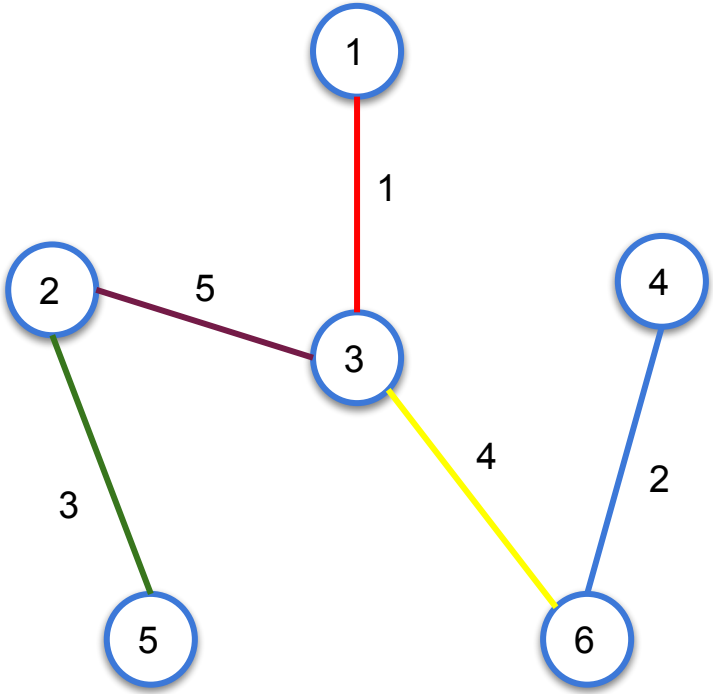


Алгоритм Прима (Prim)



Алгоритм Прима (Prim)





Алгоритм Прима (Prim)

Оцените сложность алгоритма?

Алгоритм Прима (Prim)

```
func prim(V, E):
   U = {1}
   T = {}
   while U != V do
      select (u, v) ∈ E, u ∈ U and v ∈ V and
c(u, v) minimum
      T = T + {(u, v)}
      U = U + {v}
```

Алгоритм Крускала (Kruskal)

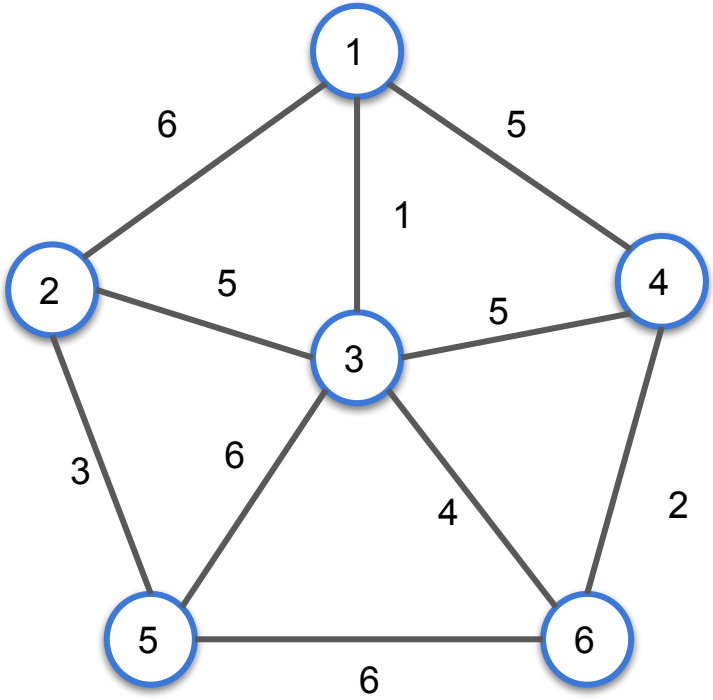
Алгоритм Крускала (Kruskal) - построение остовного дерева минимальной стоимости для графа ***G=<V, E>*** начинается с графа ***T = <V, {∅}>***, состоящего только из всех вершин графа ***G*** и не имеющего ребер. Таким образом, каждая вершина является связной (с самой собой) компонентой. В процессе выполнения алгоритма мы имеем набор связных компонент, постепенно объединяя которые формируем остовное дерево.

Алгоритм Крускала (Kruskal)

При построении связных, постепенно возрастающих компонент поочередно проверяются ребра из множества ***E*** в порядке возрастания их стоимости. Если очередное ребро связывает две вершины из разных компонент, тогда оно добавляется в граф ***T***. Если это ребро связывает две вершины из одной компоненты, то оно отбрасывается, **ПОЧЕМУ?**

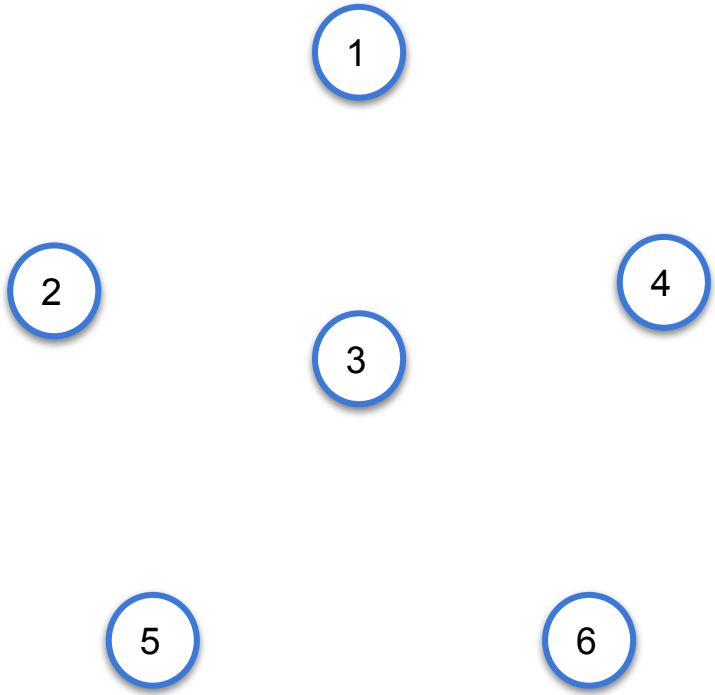
Когда все вершины графа ***G*** будут принадлежать одной компоненте, построение остовного дерева минимальной стоимости ***T*** для этого графа заканчивается.

Алгоритм Крускала (Kruskal)



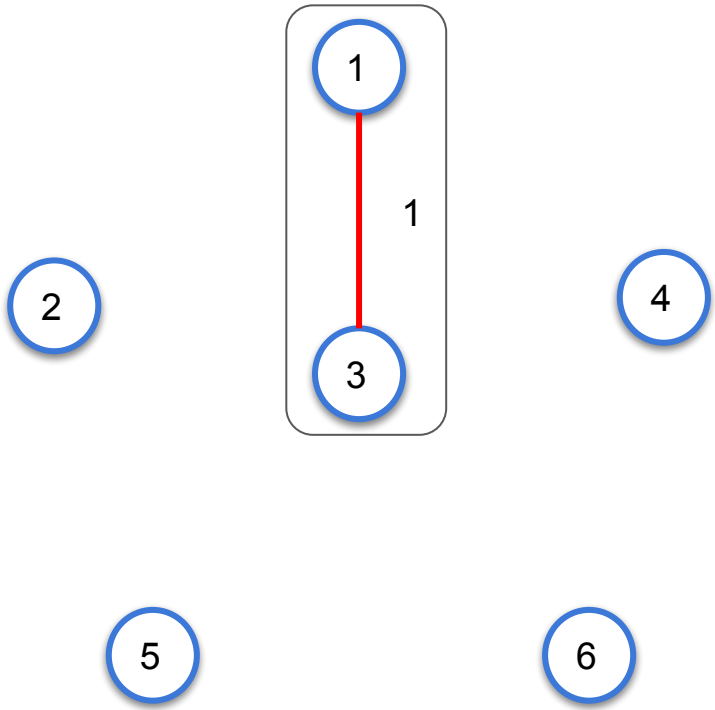
Алгоритм Крускала (Kruskal)

T =<V, {∅}>



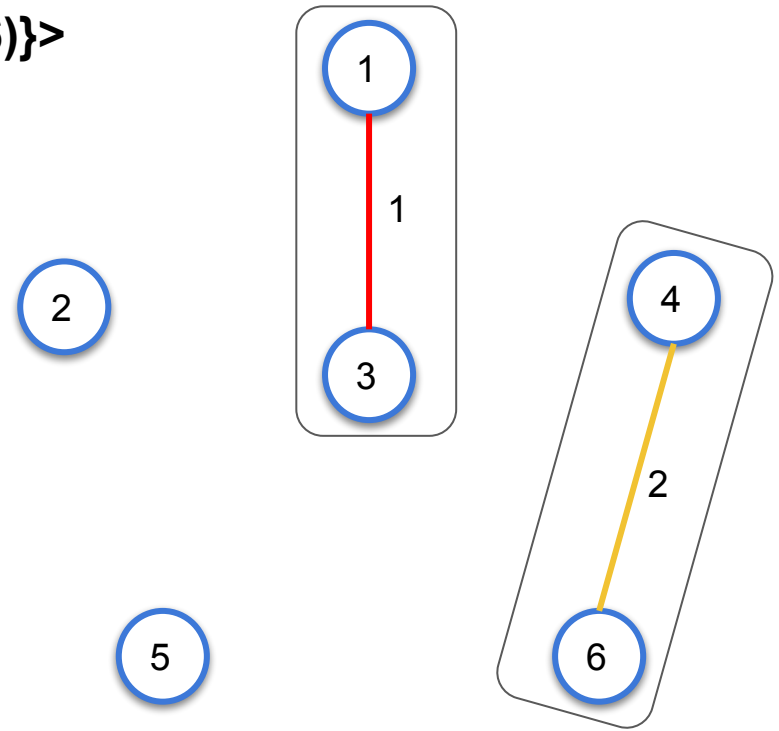
Алгоритм Крускала (Kruskal)

T =<V, {(1,2)}>



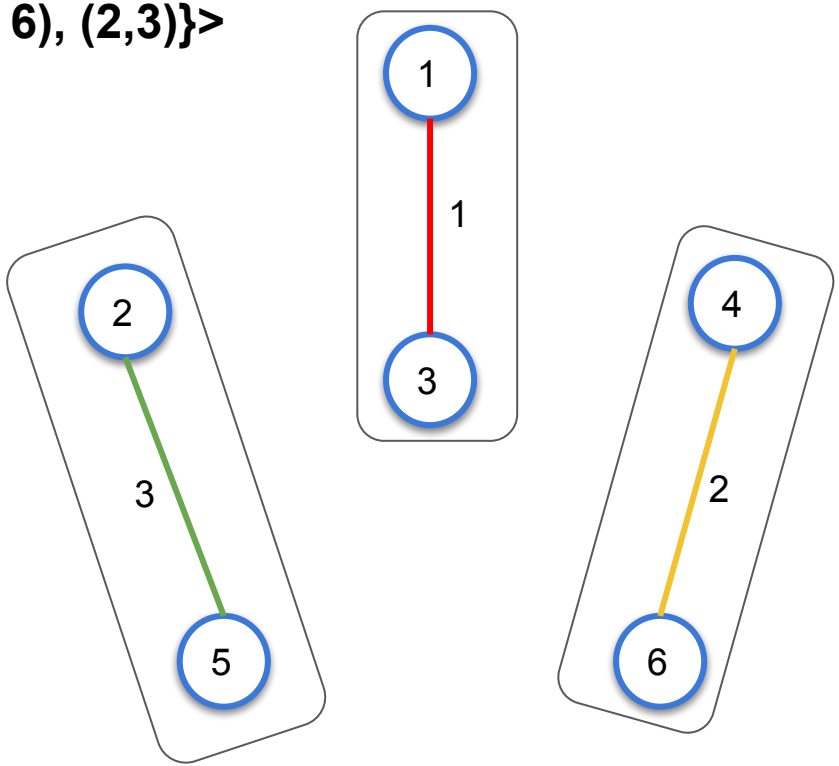
Алгоритм Крускала (Kruskal)

T =<V, {(1,2), (4, 6)}>



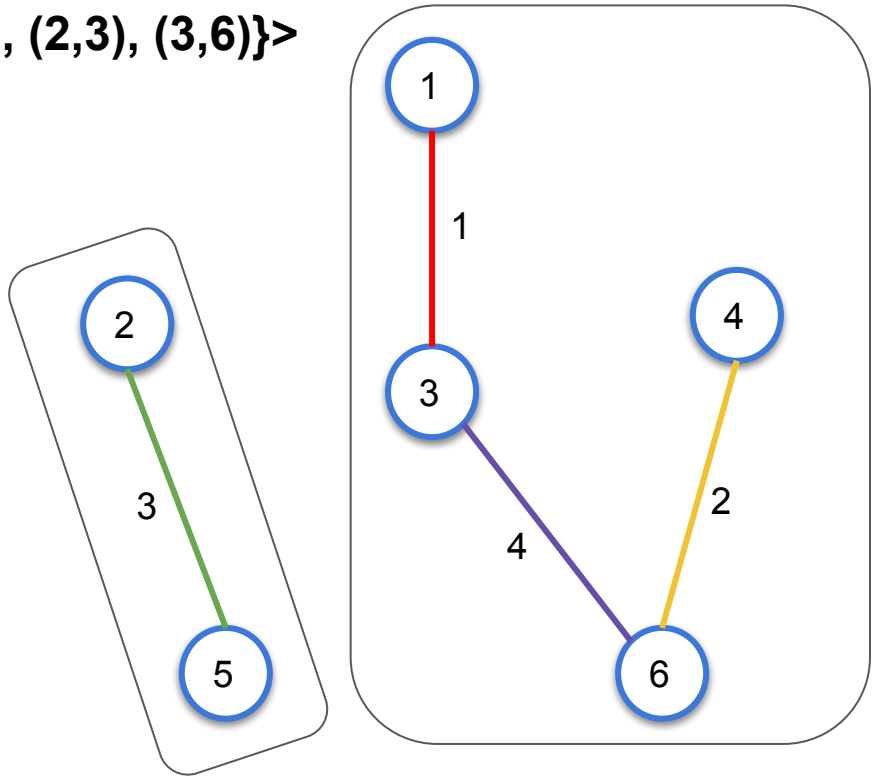
Алгоритм Крускала (Kruskal)

T =<V, {(1,2), (4, 6), (2,3)}>



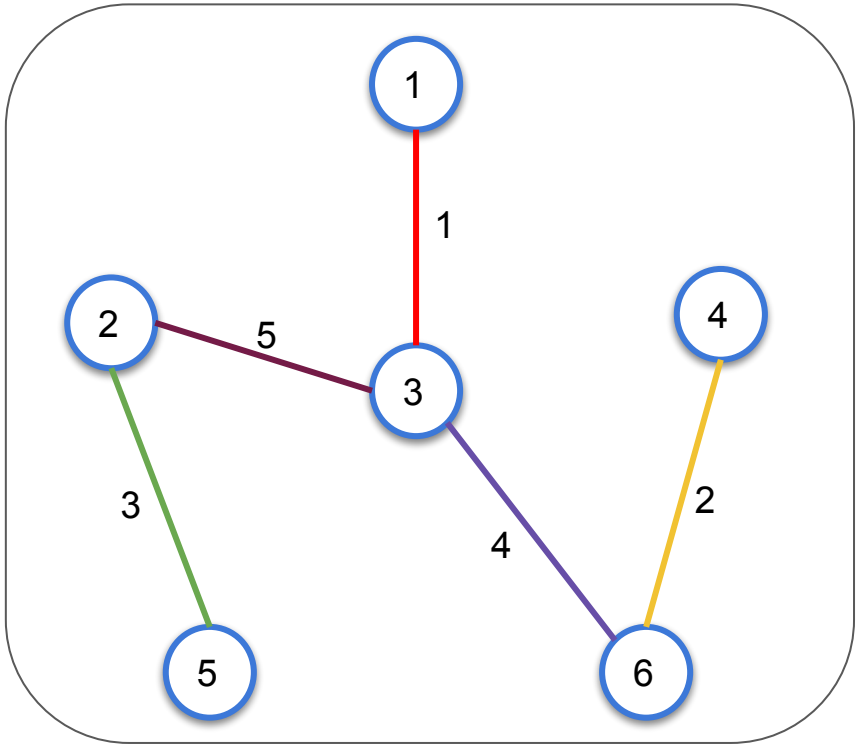
Алгоритм Крускала (Kruskal)

T =<V, {(1,2), (4, 6), (2,3), (3,6)}>



Алгоритм Крускала (Kruskal)

T =<V, {(1,2), (4, 6), (2,3), (3,6), (3,2)}>



Алгоритм Крускала (Kruskal)

```
func kruskal(V, E)
   T = {component:V, edges:{}}
   while T[component] != 1 do
      get e = (u, v) from E where c(e) is min and
(component(u) != component(v))
      T.edges += e
      new_component = component(u) merge
component(v)
      T[component] del component(u)
      T[component] del component(v)
      T[component] +=  new_component
```

Алгоритм Крускала (Kruskal)

Оцените сложность алгоритма?

Алгоритм Крускала (Kruskal)

Как можно реализовать поиск ребра наименьшей стоимости более эффективно?

Указания для задач

Для замера работы функции нужно использовать метод now() класса datetime модуля datetime

Указания для задач

```
array = [0] * N
array.insert(N,0)
```

```
import datetime


def main():
    t1 = datetime.datetime.now()
    #You'r code here
    ...
    print(datetime.datetime.now() - t1)

if __name__ == '__main__':
    main()
```

Указания для задач

```
import numpy as np


...
# Generate numpy Array with N random numbers
array = np.random.rand(N)
#Sort Array by quick sort
sorted(array)
...
```

Указания для задач

jupyter MyTests Last Checkpoint: a minute ago (unsaved changes)

File Edit View Insert Cell Kernel Help Python 2

Code CellToolbar

In [6]:
```
import datetime
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
```

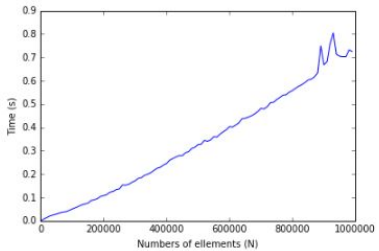
In [7]:
```
times = []
for N in xrange(1000, 1000000, 10000):
    array = np.random.rand(N)
    t1 = datetime.datetime.now()
    sorted(array)
    t_res = datetime.datetime.now() - t1
    times.append([N,t_res])
y = [t[1].total_seconds() for t in times]
x = [t[0] for t in times]
```

In [8]:
```
%matplotlib inline
```

In [9]:
```
plt.plot(x, y)

plt.xlabel('Numbers of ellements (N)')
plt.ylabel('Time (s)')

plt.savefig("test.png")
plt.show()
```



Задачи

- Реализовать алгоритм перемножения квадратных матриц. Матрицы могут задаваться как список списков. Считывать можно из файла потока ввода, или задавать случайным образом (используя функцию np.random.rand(N)). Оценить временную и ассимптотическую сложность алгоритма, построить график.
- Найти все пифагоровы тройки ($c^2 = a^2 + b^2$) для заданного интервала. Интервал задается парой чисел через пробел считанных из входного потока (например: 10 100) помните, что верхняя грань отрезка должна быть больше нижней. Если задано одно число, то считаем, что ограничение снизу равно по умолчанию 1. Оценить временную и ассимптотическую сложность алгоритма, построить график.
- Реализовать алгоритм факторизации числа (разложение числа как произведение двух других чисел). Оценить временную и ассимптотическую сложность алгоритма, построить график.
- Реализовать алгоритм рассчитывающий сочетания и размещения.
- Факториал довольно емкостная функция, при расчете которого для больших значений может случится переполнение (т.е. полученное число будет больше чем максимально возможное число в вашей системе). Подумайте как преодолеть эту проблему.

Задачи

Написать оболочку для работы с графами:

- создавать графы
- Выводить граф (в виде таблицы смежности)
- Удалять ребра
- Ищет путь в графе для заданных вершин
  - Флойда-Уоршела
  - Форда-Беллмана
  - Дейкстра

Задачи

1. Скачать файл https://goo.gl/z7H7DU
2. Файл содержит карту препятствия обозначены символом ‘%’ клетки, по которым можно передвигаться обозначены ‘-’, при этом каждая клетка по которой можно двигаться имеет вес 1.
3. Робот начинает движение в клетке обозначенной буквой ‘P’ и движется в клетку обозначенной буквой ‘T’.
4. Нужно рассчитать оптимальную траекторию пути робота с помощью алгоритма A*.
5. Выведите траекторию в отдельный файл.

Задачи

1. Реализуйте функцию DFS
2. С помощью вашей функции реализуйте алгоритм разбиение графа на компоненты связности.
3. Реализуйте алгоритм проверки орграфа на цикличность
4. Реализуйте алгоритм Крускала/Прима для поиска минимального остовного дерева взвешенного графа.

Спасибо за внимание!