Home

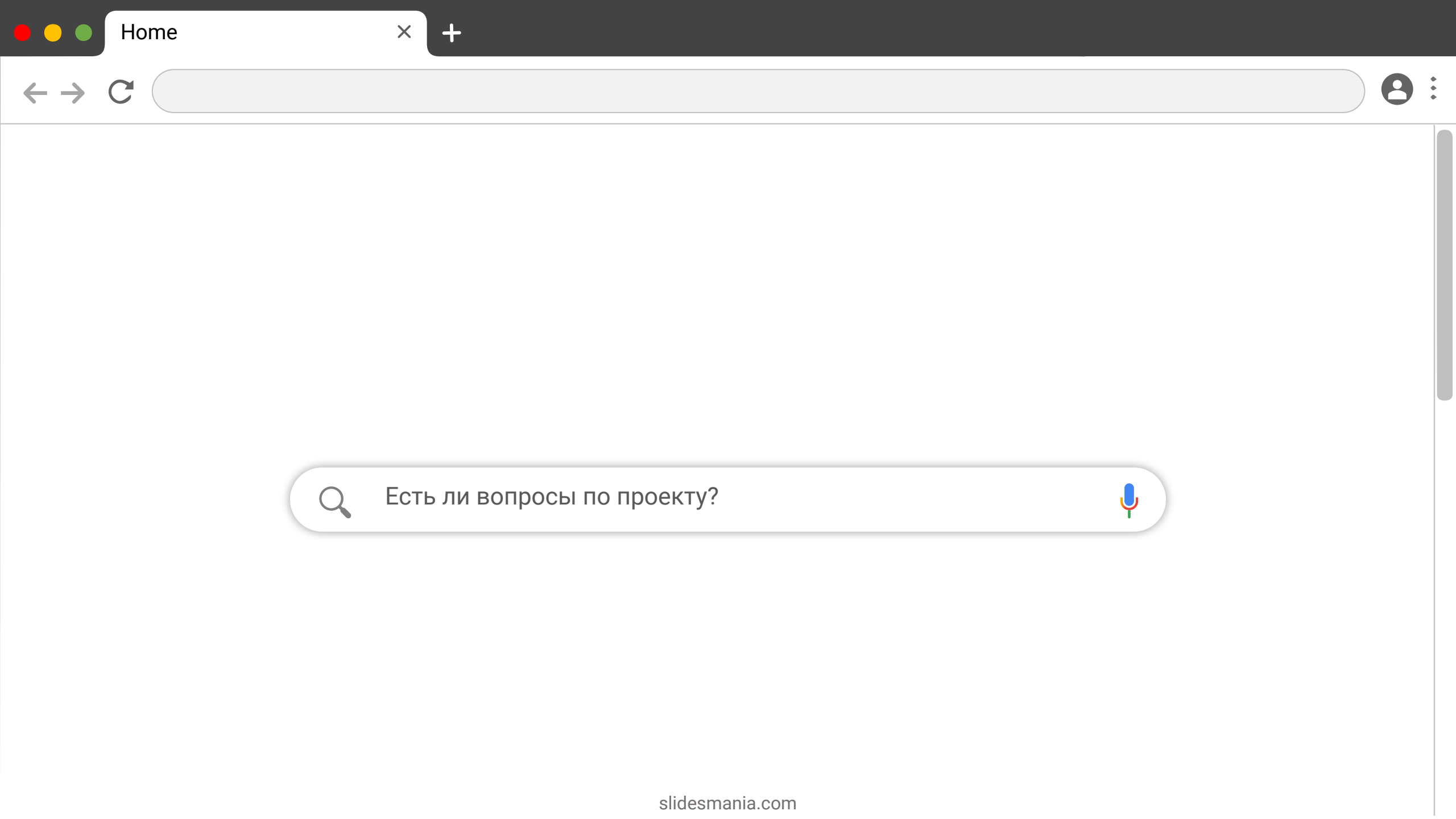
Информационный поиск

Лекция 7. Теория графов (и немного knowledge retrieval)

Telegram

GitHub

Ника Зыкова
2023/10/11

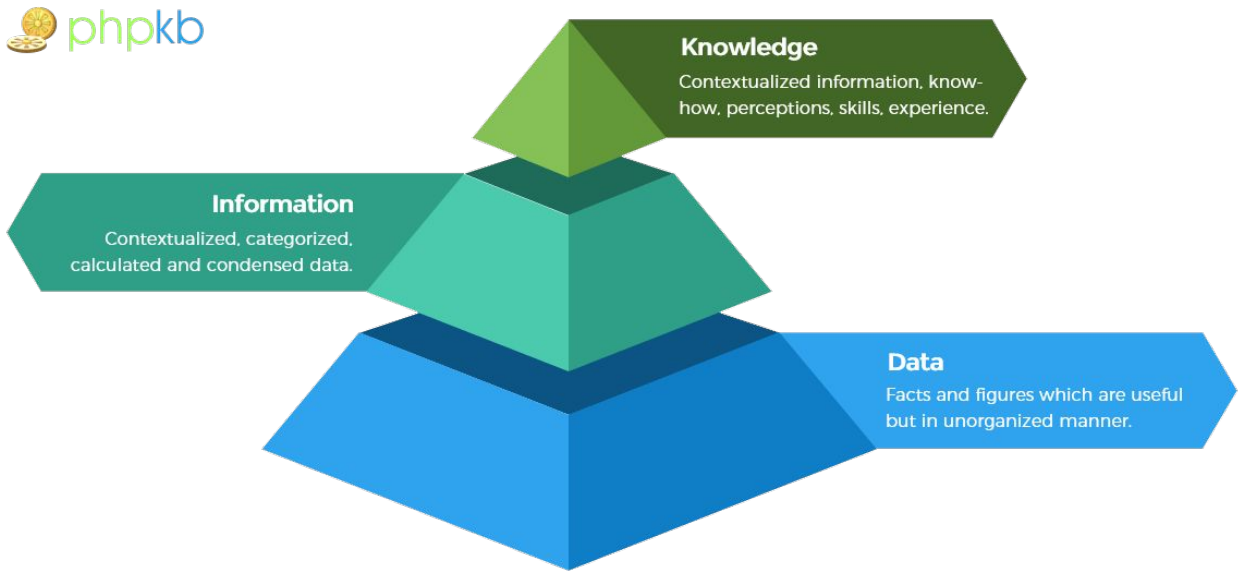


Постановка задачи

В рамках задачи knowledge retrieval необходимо извлечь из текста (в нашем случае) структурированную информацию в **форме, удобной для усвоения человеком**.

Список документов - это неудобная форма!

Насколько быстрее вы учитесь по учебнику, чем по списку страниц из гугла?



Отличия IR и KR

Information retrieval	**Knowledge retrieval**
Информацию нужно найти	Из найденной информации необходимо извлечь структурированную форму
Четкая постановка задачи, общая для большинства решений	Различные постановки задачи под разные цели и ситуации
Легкая формализация	Нужен свой способ оценки качества под каждую задачу

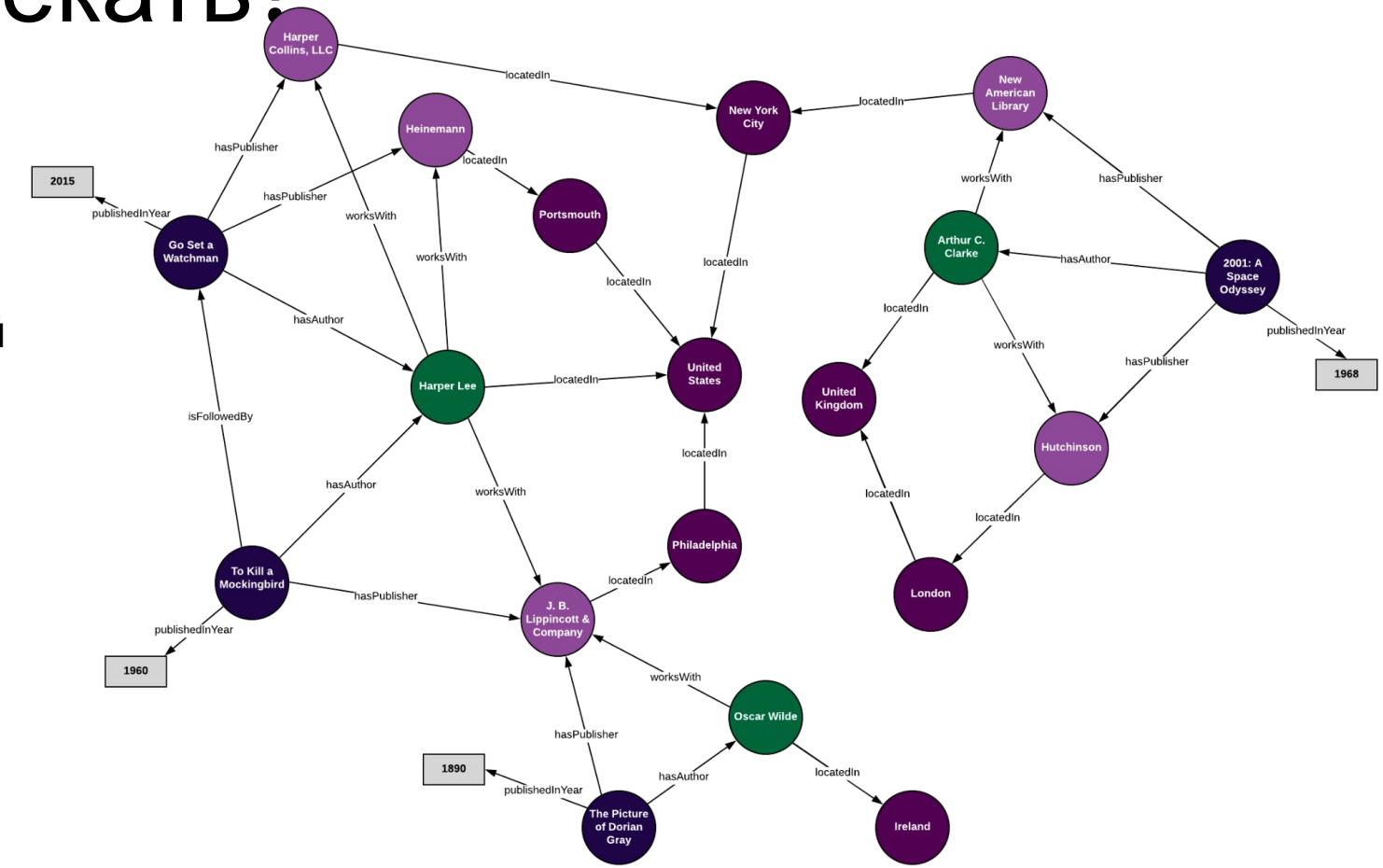
Home | Recent Tabs | Current Page

You can type something here...

Что можно извлекать?

- Таблицы
- Таксономии
- Онтологии/графы знаний
- Графы социальных отношений

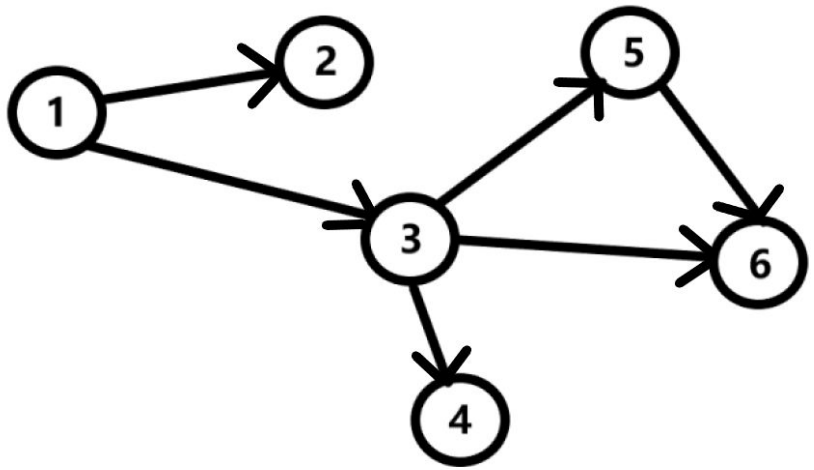
Три из четырех - графы по своей структуре!

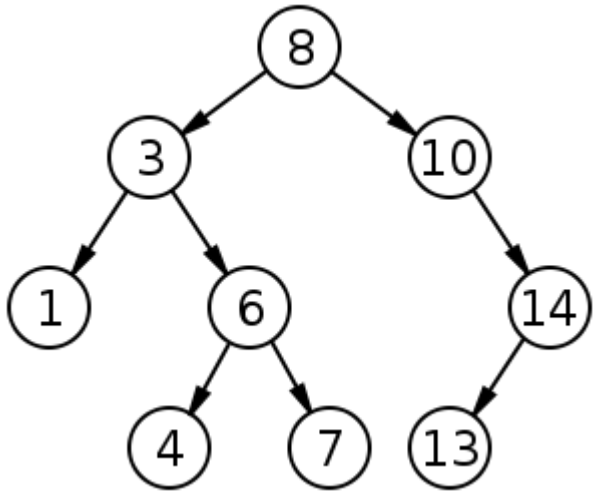


Граф и дерево

Граф - это совокупность непустого множества вершин и набора упорядоченных/неупорядоченных пар из множества вершин

Дерево - связный ациклический граф (часто ориентированный)





Графы: где используются

- Представление молекулярной структуры в биологии
- Графы вычислений и автоматы в информатике
- Граф в основе географии в онлайн-картах
- Онтологии и таксономии
- Графы социальных отношений

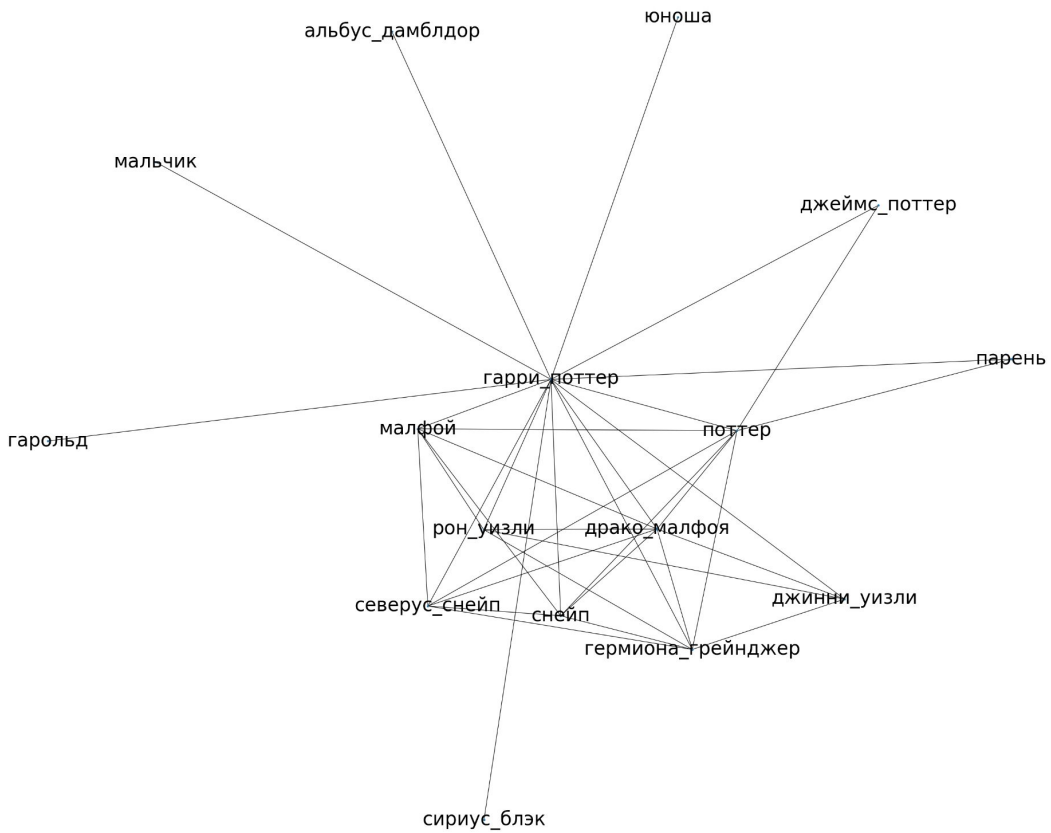
В целом, с помощью графа можно представить любую структуру, если ее можно перевести в формат сущности-связи!

Таксономия и онтология

Таксономия	**Онтология**
В большинстве случаев иерархия	Модель произвольных отношений
Чаще всего дерево	Чаще всего граф (могут быть циклы, кратные ребра и т.д.)
Один тип связи: отношения родитель-потомок, общее-частное и т.д.	Множество вариантов связи в одной структуре
Одна предметная область	Любой вариант масштаба (метапредметные графы знаний)

Граф социальных отношений

- Вершины - сущности, способные взаимодействовать между собой (люди, организации, персонажи книг)
- Ребра - типы и сила взаимодействия между сущностями
- Чаще всего используются в сфере DH и социального анализа



Сложности

- Не только найти информацию, но и структурировать ее
- Попытка автоматизировать творческую деятельность: нет четкого алгоритма для систематизации произвольной информации
- Часто построение такой структуры требует знаний, выходящих за рамки области
- Задача, где может не быть правильного решения (особенно в графах)
- Задача, сложная для решения человеком

А зачем?

Граф - математический объект, это дает возможность применять весь разработанный для него математический аппарат в конкретных задачах.

Что есть для графов:

- Алгоритмы обхода и поиска, оптимальные для конкретных типов графа и задач
- Алгоритмы выделения плотных групп - кластеров, сообществ
- Множество метрик, позволяющих оценить структуру в целом
- Богатая типология, с выделенными для каждой группы свойствами

Как хранить графы: определение

Мы можем хранить граф так, как нам предписывает определение: множество (список) вершин и множество (упорядоченных) пар из них.

Множество вершин: {1, 2, 3, 4, 5}

Множество ребер: {(1, 3), (2, 5), (4, 3), (1, 5), (2, 3)}

Иногда можно не хранить отдельно множество вершин. Когда? А когда нельзя?

Как хранить графы: матрица смежности

Матрица смежности графа числом вершин n — это квадратная матрица размера n, где по столбцам и строкам расположены вершины, а в ячейках - числовые характеристики ребер (есть|нет, кол-во ребер, вес ребра)

Матрица:

	1	2	3	4	5
1			1		1
2			1		1
3					
4			1		
5					

ИЛИ

	1	2	3	4	5
1			1		1
2			1		1
3	1	1		1	
4			1		
5	1	1			

Как хранить графы: словарь

У матрицы смежности те же проблемы, что и у частотного индекса: множество нулей зря занимают память. Способ решения такой же: храним данные в виде словаря.

Словарь:

```
{
    1: {3: 1, 5: 1},
    2: {5: 1, 3: 1},
    3: {},
    4: {3: 1},
    5: {}
}
```

И есть еще множество других способов хранения.

Как хранить деревья: прямая польская запись

Теоретически, для деревьев применимы все способы хранения, существующие для графов, однако они часто избыточны. Почему?

Для деревьев разработаны свои способы хранения. Первый из них - прямая польская запись: запись в форме списка, где каждый элемент - список из вершины и списка ее дочерних вершин.

Пример:

[ROOT,
[root,
[VERB,
[obl, [NOUN, [case, [ADP]]]],
[nsubj, [NOUN]]]]]

Как хранить деревья: обратная польская запись

Еще есть обратная польская запись - сначала выводятся потомки, а потом сама вершина. Такой формат записи чуть менее человекочитаемый, но за пределами деревьев зависимостей очень распространен.

Пример:

[[[[[[[ADP], case], NOUN], obl],
[[NOUN], nsubj],
VERB],
root],
ROOT]

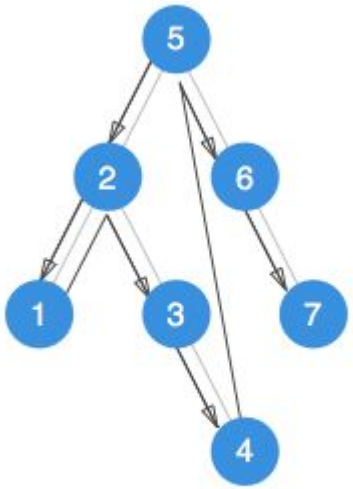
Алгоритмы обхода: в глубину

Часто требуется перебрать все вершины графа: например, чтобы найти конкретный элемент или совершить операцию со всеми вершинами.

Первый способ - обход в глубину. Он заключается в следующей логике шагов:

1. Обрабатываем текущую вершину
2. Если у нее есть потомки, то поочередно выбираем их как текущую вершину и повторяем пункты 1-2
3. Если потомков нет, то поднимаемся на уровень выше

Как называются такие алгоритмы?



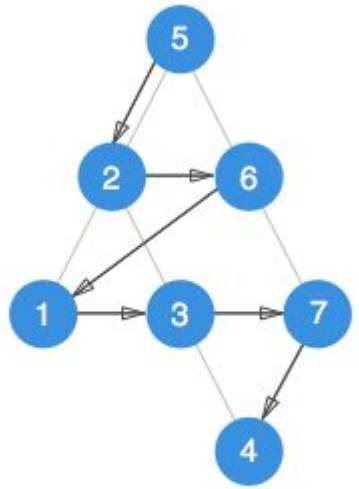
Depth-first traversal

Алгоритмы обхода: в ширину

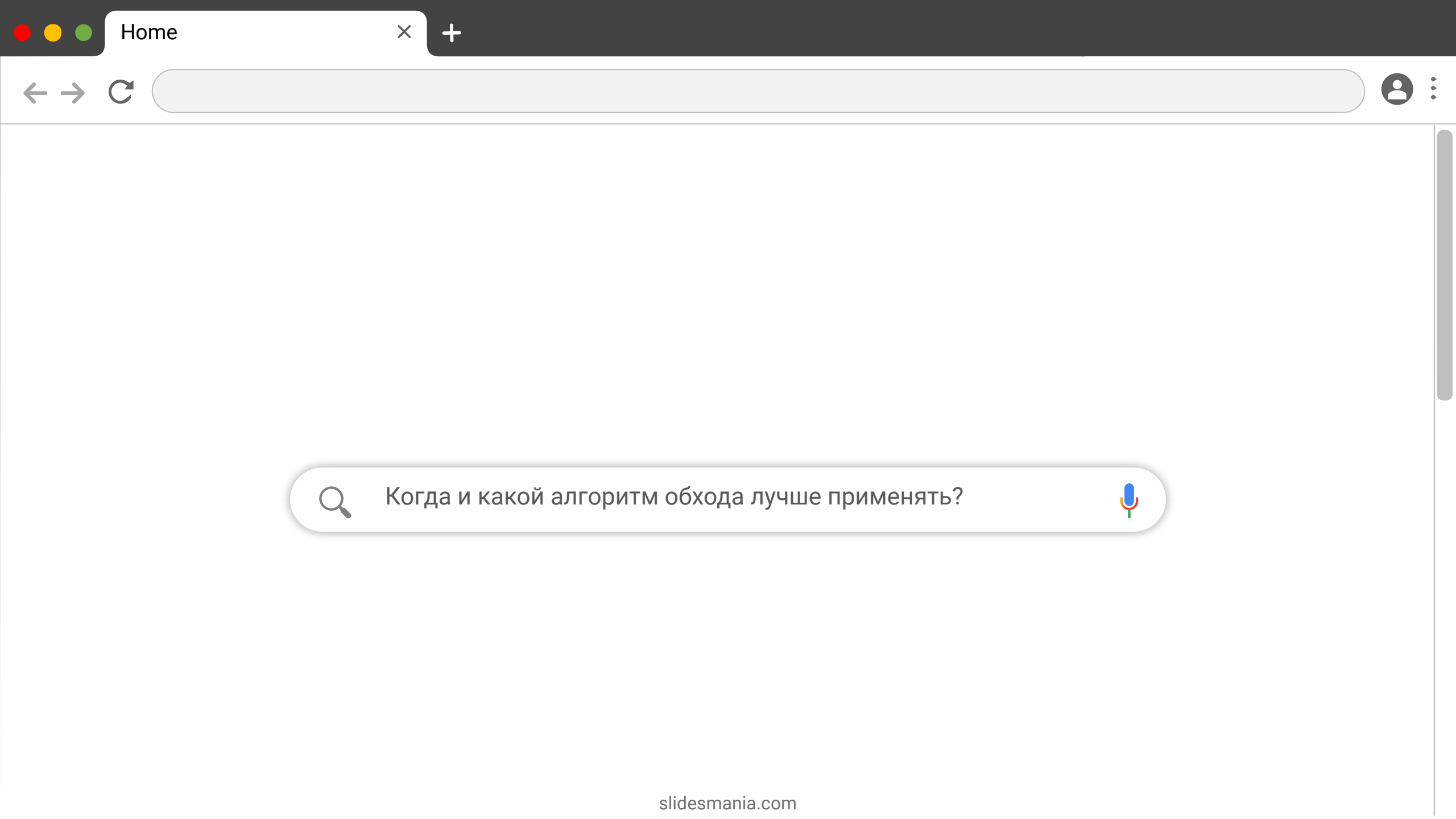
Второй способ - обход в ширину. Логика шагов следующая:

1. Обрабатываем текущую вершину
2. Если у на текущем уровне есть другие вершины, то обрабатываем их
3. Если вершин на уровне больше нет, то спускаемся на уровень ниже и повторяем пункты 1-3

Какая здесь может быть сложность с точки зрения реализации в коде?



Breadth-first traversal



Метрики графов

1. Кол-во вершин и ребер
2. Степень вершин: максимальная, средняя, распределение
3. Ребра: распределение весов
4. Пути: распределение длин и стоимостей
5. Degree, closeness, betweenness centrality
6. Кластерность

Degree centrality

Доля вершин, с которыми соединена текущая.

Можно считать:

- Для конкретной вершины
- Среднее и медианное значение
- Распределение значений
- Зависимость от остальных метрик в графе (или внешних для графа метрик - метаинформации вершин и ребер)

Betweenness centrality

Показывает, какую долю от всех кратчайших путей в графе составляют кратчайшие пути, проходящие через текущую вершину.

$$c_B(v) = \sum_{s,t \in V} \frac{\sigma(s,t|v)}{\sigma(s,t)}$$

где

V - мноджество вершин в графе

s, t - какие-то вершины

σ(s, t) - кол-во кратчайших путей от s до t

σ(s, t|v) - кол-во кратчайших путей от s до t, проходящих через v

Closeness centrality

Величина, обратная среднему кратчайшему пути от вершины до всех достижимых из нее вершин.

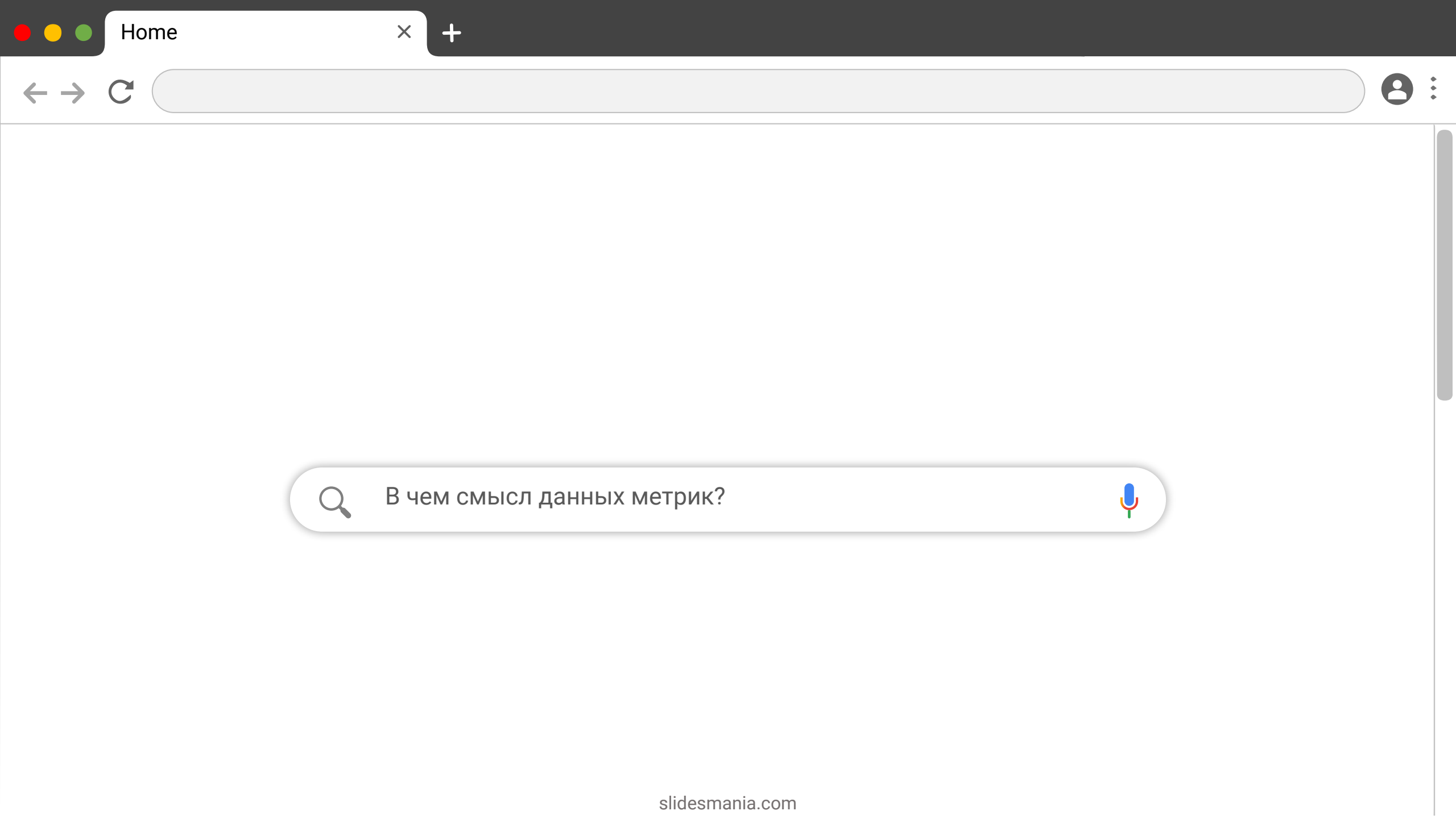
$$C(u) = \frac{n-1}{\sum_{v=1}^{n-1} d(v,u)},$$

где

u - текущая вершина

d(u, v) - кратчайший путь от вершины u до вершины v

n - 1 - кол-во вершин, достижимых из u



Home | Recent Tabs | Current Page

You can type something here...

Кластеризация

Вообще, это одна из задач обучения без учителя, где необходимо выделить в данных группы похожих по признакам объектов.

Однако, часто кластеризация воспринимается не как задача, а как инструмент для исследования данных. Тогда используются хорошо зарекомендовавшие себя алгоритмы.

Например:

- K-means
- DBscan
- Louvain Method

K-means

Самый алгоритмически простой, но довольно неточный метод.

Идея: (случайно) разбрасываем по пространству k центров кластеров, стараемся итеративно минимизировать расстояние от элементов кластера до его центра

Плюсы:

- Легкий в реализации
- Чаще всего быстро обучается

Минусы:

- Требуется вручную задать кол-во кластеров
- Очень чувствителен к изначальному расположению центров
- Не умеет отсеивать шум и промежуточные положения

DBscan

Идея: если между двумя точками протянулась плотная группа (тут есть параметр плотности и размера группы) то мы считаем их одной группой; точки на краях или между присоединяем к ближайшей группе; если точка совсем далеко, то она выброс

Плюсы:

- Кол-во кластеров определяется само
- Есть гиперпараметры, которые можно подбирать под конкретную задачу

Минусы:

- Плохо работает с нечеткими границами и не очень плотными кластерами
- Может быть излишне сложным для простых случаев

Louvain Method

Идея: изначально все вершины в отдельных кластерах; потом, максимизируя функцию модулярности, перемещаем вершины в кластеры; сжимаем кластера в одну вершину и повторяем процесс

Модулярность сравнивает число рёбер внутри кластера с ожидаемым числом рёбер для случайного графа с тем же кол-вом вершин