

# Презентация к дипломной работе

Разработка многопрофильной  
системы информационного поиска

# Основные компании

- Amazon
- Google
- Яндекс
- Amazon
- Twitter
- Microsoft



# Характеристики сложноструктурированных данных

1. Внутренняя интерпретация.
2. Наличие внутренней структуры связей.
3. Шкалирование.
4. Погружение в пространство с семантической метрикой.
5. Наличие активности.

# Используемые алгоритмы

- PageRank
- DBScan
- Rock
- Наивный байесовский классификатор
- Семантические сети



# Области применения системы информационного поиска

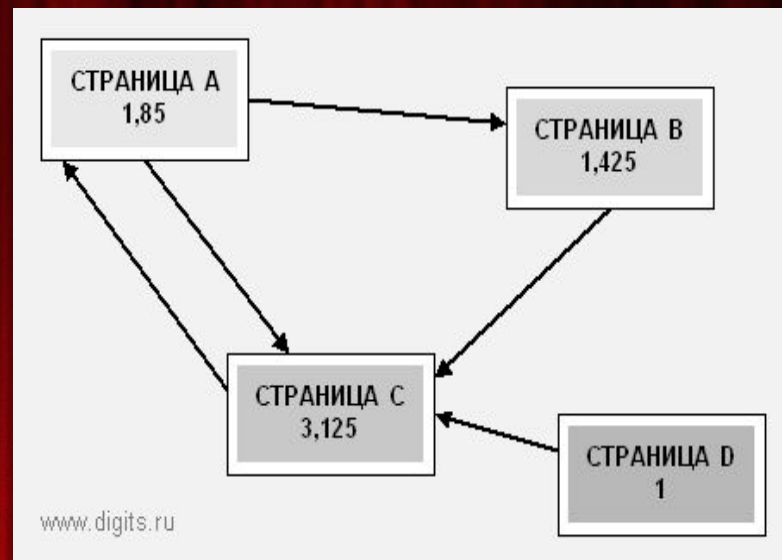
- Поиск информации.
- Формирование рекомендаций.
- Установление авторства.
- Проверка на плагиат.
- Автоматическая генерация текстов для SEO (поисковой оптимизации).
- Лингвистический анализ литературных текстов.
- Корректировка текстов и исправление опечаток.

# Алгоритм PageRank

1. Каждой странице присваиваем вес равной единице.
2. Подсчитываем количество исходящих связей для каждой страницы.

3. 
$$pageRank(A) = 1 - d + d \cdot \left( \frac{pageRank(T1)}{C(T1)} + \dots + \frac{PageRank(Tn)}{C(Tn)} \right)$$

Вычисляем ранг каждой страницы с помощью формулы. Где A – страница, ранг которой необходимо найти, C(T1) – количество исходящих ссылок, d – коэффициент затухания.





Место для блок-схемы

# Алгоритм ROCK

Procedurecluster (S, k)

Begin

1. link := compute-links (S) //Вычисляем связи в множестве точек S
  2. for each s from S do
  3.  $q[s]$  := build-local-hear (link,S) //Из каждой точки множества S на основе связей формируем кластер
  4.  $Q$ :=build-global-hear (S,q) //Содержит список всех кластеров множества S
  5. whilesize (Q) >kdo { //Формируем кластеры, точки, которых имеют максимальное число связей до тех пор, пока не получим желаемое число кластеров
  6.  $u$  := extract-max (Q)
  7.  $v$  := max ( $q[u]$ )
  8. delete (Q,v)
  9.  $w$ := merge (u,v)
  10. for each x from ( $q[u]$  or  $q[v]$ ) do {
  11. link [ $x,w$ ] := link [ $x,u$ ] + link [ $x,v$ ]
  12. delete ( $q[x],u$ ); delete ( $q[x],v$ )
  13. insert ( $q[x],w,g(x,w)$ ); insert ( $q[w],x,g(x,w)$ );
  14. update (Q,x, $q[x]$ )
  15. }
  16. insert (Q,w, $q[w]$ ) //Добавляем кластер в список всех кластеров
  17. deallocate ( $q[u]$ ); deallocate ( $q[v]$ );
  18. }
- end.



# Алгоритм DBSCAN

```
public List<Cluster> cluster() {  
    int clusterId = getNextClusterId();  
    for(DataPoint p : points) {  
        if(isUnclassified(p) ) { //Проверяем классифицировали ли мы данную точку.  
            boolean isClusterCreated = createCluster(p, clusterId);  
            //Создаем кластер для каждой точки  
            if( isClusterCreated ) {  
                clusterId = getNextClusterId();  
            }  
        }  
    }  
    List<Cluster> allClusters = new ArrayList<Cluster>();  
    for(Map.Entry<Integer, Set<DataPoint>> e : clusters.entrySet()) {  
        String label = String.valueOf(e.getKey()); //Создаем кластер и имя для него  
        Set<DataPoint> points = e.getValue();  
        if( points != null && !points.isEmpty() ) {  
            Cluster cluster = new Cluster(label, e.getValue());  
            allClusters.add(cluster);  
        }  
    }  
    return allClusters; //Возвращаем список всех кластеров, которые были созданы  
}
```

# Алгоритм DBSCAN

```
private boolean createCluster(DataPoint p, Integer clusterId){
    Set<DataPoint> nPoints = findNeighbors(p, eps);
    if( nPoints.size() < minPoints ) {
        assignPointToCluster(p, CLUSTER_ID_NOISE); //Если количество точек окружности меньше, чем minPoints,
        присваиваем точке значение «Шум»
        isClusterCreated = false;
    } else {
        assignPointToCluster(nPoints, clusterId); //Иначе добавляем точку в кластер
        nPoints.remove(p); //Удаляем точку из рассмотрения
        while(nPoints.size() > 0 ) { //Просматриваем все точки, если нашли точку, которую уже рассматривали
            то ставим ей статус пограничной, добавляем в кластер и удаляем из рассмотрения
            DataPoint nPoint = nPoints.iterator().next();
            Set<DataPoint> nnPoints = findNeighbors(nPoint, eps);
            if( nnPoints.size() >= minPoints ) {
                for(DataPoint nnPoint : nnPoints ) {
                    if( isNoise(nnPoint) ) {
                        assignPointToCluster(nnPoint, clusterId); //Добавляем точку в кластер
                    } else if( isUnclassified(nnPoint) ){
                        nPoints.add(nnPoint);
                        assignPointToCluster(nnPoint, clusterId);}
                }
            }
            nPoints.remove(nPoint); //Удаляем точку из рассмотрения
        }
        isClusterCreated = true;
    }
    return isClusterCreated;
}
```



# Наивный байесовский классификатор

- Место для блок-схемы.