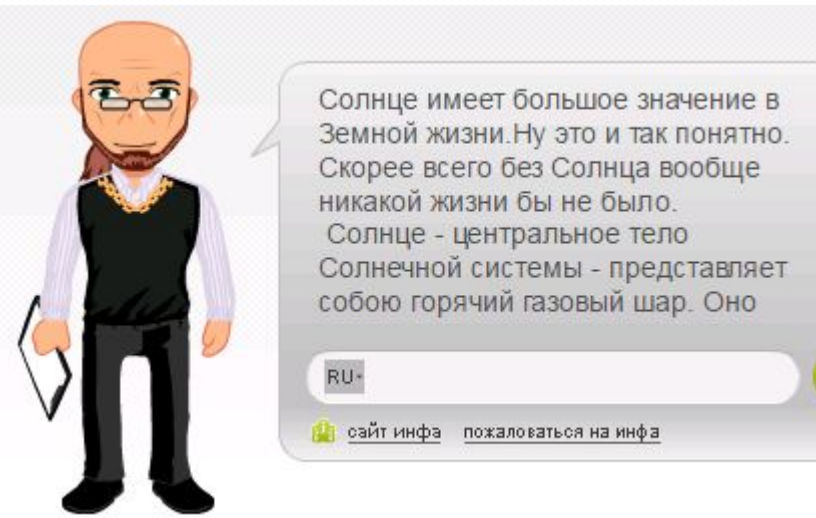
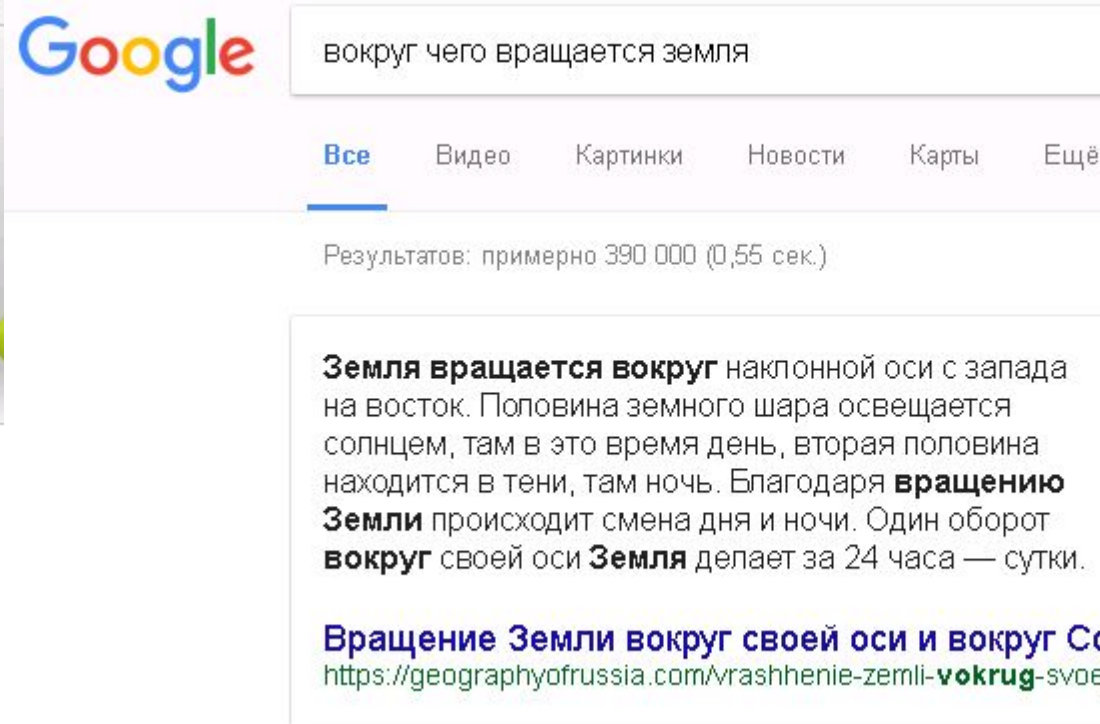


# **Диалог с информационной системой**

# Примеры интеллектуальных диалоговых систем



Инфы – интеллектуальные слуги (iii.ru)



Семантический поиск Google

# Принципы архитектуры

## Выполняются 2 этапа:

- Наполнение базы знаний (индексация)
- Поиск подходящего ответа в базе знаний

## Необходимы 3 компонента:

- База знаний
- Модуль индексирования
- Модуль диалога с пользователем

# Индексация

- Перевод слов в начальную форму (инфинитив глаголов и именительный падеж в единственном числе именных частей речи)
- Выделение частей (ключевых объектов и типов связей)

# Индексация вручную

(обработка исходного текста)

~~Из этого следует вывод, что~~

для превращения воды в пар

требуется

приток теплоты, подобно тому как это  
имеет место при превращении кристалла

~~(льда)~~ в жидкость

Лед является кристаллом

# Индексация вручную

(запись в текстовый массив)

```
string[,] knowledge = {  
    {"для превращения воды в пар",  
     "требуется",  
     "приток теплоты, подобно тому как это имеет место " +  
     "при превращении кристалла в жидкость"},  
    {"лед", "является", "кристаллом"}  
};
```

# Поиск в базе знаний

## Инструменты для поиска:

- Функции работы со строками языка C#
- Построение и использование регулярных выражений

# Поиск с помощью регулярных выражений

1. Разбиение вопроса на слова
2. Поиск сказуемого в вопросе
3. Формирование регулярного выражения для сказуемого
4. Формирование регулярного выражения для подлежащего
5. Поиск по двум регулярным выражениям
6. Если результата нет, попытка поиска только по подлежащему



# Разбиение вопроса на слова

```
//функция, которая делает первую букву маленькой
string small1(string str)
{
return str.Substring(0, 1).ToLower() + str.Substring(1);
}

//главная функция, обрабатывающая запросы клиентов
protected void Page_Load(object sender, EventArgs e)
{
    //знаки препинания
    char[] separators = "'\",.!?()[\\".ToCharArray();
    //получение текста из параметра запроса
    string txt = small1(Request["txt"]);
    //добавление пробелов перед знаками препинания
    for (int i = 0; i < separators.Length; i++)
        txt = txt.Replace("" + separators[i], " " + separators[i]);
    //массив слов и знаков препинания
    List<string> words = new List<string>(txt.Split());
}
```

# Псевдоокончания для поиска сказуемых

```
//массив пар нужных псевдоокончаний и их тегированных замен
string[,] endings = {
{"ет", "(ет|ут|ют) "}, {"ут", "(ет|ут|ют) "}, {"ют", "(ет|ут|ют) "},
{"ит", "(ит|ат|ят) "}, {"ат", "(ит|ат|ят) "}, {"ят", "(ит|ат|ят) "},
{"ется", "(ет|ут|ют) ся"}, {"утся", "(ет|ут|ют) ся"}, {"ются", "(ет|ут|ют) ся"},
{"ится", "(ит|ат|ят) ся"}, {"атся", "(ит|ат|ят) ся"}, {"ятся", "(ит|ат|ят) ся"},
{"ен", "ен"}, {"ена", "ена"}, {"ено", "ено"}, {"ены", "ены"},
{"ан", "ан"}, {"ана", "ана"}, {"ано", "ано"}, {"аны", "аны"},
{"жен", "жен"}, {"жна", "жна"}, {"жно", "жно"}, {"жны", "жны"};
//черный список слов, попадающих по ошибке
string[] blacklist = { "замена", "замены", "атрибут", "маршрут",
                      "член", "нет" };
,
```

# Поиск сказуемого в вопросе

```
//функция определения соответствующих псевдоокончаний
private int getEnding(string word)
{
    //проверка по черному списку
    if (blacklist.Contains(word)) return -1;
    //перебор псевдоокончаний
    for (int j = 0; j < endings.Length / 2; j++)
    {
        //проверка, оканчивается ли i-ое слово на j-ое псевдоокончание
        if (word.EndsWith(endings[j, 0]))
        {
            return j;    //возврат номера псевдоокончания
        }
    }
    return -1;    //если совпадений нет - возврат -1
}
```

# Формирование регулярных выражений

```
using System.Text.RegularExpressions;

//замена псевдоокончания на набор возможных окончаний
words[i] = words[i].Substring(0, words[i].Length -
    endings[ending, 0].Length) + endings[ending, 1];
//создание регулярного выражения для поиска по сказуемому из вопроса
Regex predicate = new Regex(words[i]);
//для кратких прилагательных захватываем следующее слово
if (endings[ending, 0] == endings[ending, 1])
{
    predicate = new Regex(words[i] + " " + words[i + 1]);
    i++;
}
//создание регулярного выражения для поиска по подлежащему из вопроса
Regex subject = new Regex(".*" +
    string.Join(".*", words.ToArray().Skip(i + 1)) +
    ".*");
```

# Поиск по двум регулярным выражениям

```
//поиск совпадений с шаблонами среди связей семантической сети
for (int j = 0; j < knowledge.Length / 3; j++)
{
    if (predicate.IsMatch(knowledge[j, 1]) &&
        (subject.IsMatch(knowledge[j, 0]) ||
         subject.IsMatch(knowledge[j, 2])))
    {
        //создание простого предложения из семантической связи
        Response.Write(big1(knowledge[j, 0] + " " +
            knowledge[j, 1] + " " + knowledge[j, 2] + ". "));
        result = true;
    }
}
```

# Поиск по подлежащему, если предыдущий поиск не дал результатов

```
//если совпадений с двумя шаблонами нет,  
if (result == false)  
    //поиск совпадений только с шаблоном подлежащего  
    for (int j = 0; j < knowledge.Length / 3; j++)  
    {  
        if ((subject.IsMatch(knowledge[j, 0]) ||  
            subject.IsMatch(knowledge[j, 2])))  
        {  
            //создание простого предложения из семантической связи  
            Response.Write(big1(knowledge[j, 0] + " " +  
                knowledge[j, 1] + " " + knowledge[j, 2] + ". "));  
            result = true;  
        }  
    }  
}
```