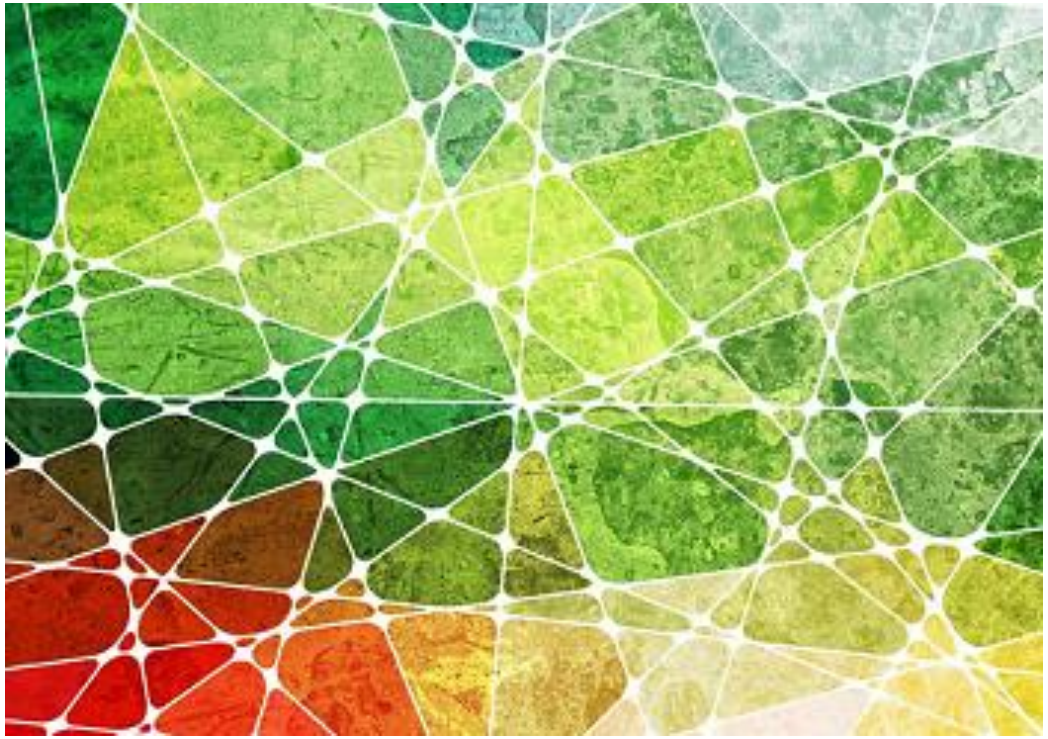
C#
VII
Collections

Plan

1. Абстрактные типы данных
2. Структуры данных



Definitions

Тип данных — множество значений и операций над этими значениями (например, болт можно вкрутить)

Абстрактный тип данных (обобщенные коллекции)

- ❑ интерфейс для работы с данными
- ❑ скрывает подробности хранения данных в памяти и управления ими
  - Примитивные типы
  - Список
  - Словарь
  - Стек
  - Очередь
  - Множество
- ❑ что писать в коде

Структуры данных

- ❑ описывают как данные организованы
- ❑ как к ним получить доступ в памяти компьютера
  - Массив
  - Связный список
  - Двусвязный список
  - Дерево
  - Граф
  - Хеш-таблица
- ❑ как есть на самом деле

List<T> vs Array

Список - похож на массив, наиболее часто используемая коллекция

```
List<Student> std = new List<Student>();
std.Add(new Student());          // добавить элемент в конец списка
int countStd = std.Count;        // количество элементов в списке
std.RemoveAt(0);                 // удалить элемент в указанной позиции
Student student = std[0];        // получить элемент по индексу
std[0] = new Student();          // установить элемент в указанную позицию
std.Sort();                      // отсортировать список
```

Произвольные:

- Извлечение
- Вставка (у массива нет)
- Удаление (у массива нет)

Ещё существует `SortedList<>`

Элементы всегда хранятся в упорядоченном виде =>

нельзя вставить элемент в произвольную позицию

Dictionary <TKey, TValue>

```
var avgMark = new Dictionary<string, double>();
avgMark.Add("Ivanov", 4.6);              // добавление
avgMark["Petrov"] = 3;                   // добавление
double avgIvanov = avgMark["Ivanov"];    // получение
avgMark["Ivanov"] = 5;                   // изменение
// обход всего списка
foreach (var item in avgMark)
   Write ($"{item.Key} - {item.Value}, "); Ivanov - 5,Petrov - 3,
int count = avgMark.Count;  // количество элементов в коллекции
avgMark.Remove("Ivanov");   // удаление из коллекции
```

Словарь – используется для хранения соответствий между двумя объектами:

- ключом - key
- значением - value

Key - уникален, нельзя добавить две записи с одинаковым ключом

Queue <T>

Очередь – список элементов по принципу FIFO (FirstInFirstOut)

```
Queue<Child> kindergarten = new Queue<Child>();
kindergarten.Enqueue(new Child("Vasya")); // добавить в конец очереди
kindergarten.Enqueue(new Child("Petya")); // добавить в конец очереди
kindergarten.Enqueue(new Child("Ivan")); // добавить в конец очереди
Child child = kindergarten.Dequeue();     // удалить из начала очереди
WriteLine (child.Name);                   Vasya
WriteLine(kindergarten.Count);            2
```

Очередь с приоритетом – это когда в детский садик в первую очередь возьмут ребенка, у которого приоритет выше Eneque(child, priority).

В FCL (Framework Class Library)встроенного типа для очереди с приоритетом обнаружить не удалось.

Stack <T>

Стек – список элементов по принципу LIFO (LastInFirstOut)

```
Stack<Card> cards = new Stack<Card>();

cards.Push(new Card("7 piki"));
cards.Push(new Card("Valet chervi"));
cards.Push(new Card("Tuz kresti"));

Card card = cards.Pop();
WriteLine(card.Name);          Tuz kresti
cards.Pop();
WriteLine(cards.Count);        1
```

Set

Множество – неупорядоченные группы уникальных элементов подобные математическим множествам.

- порядок элементов не важен
- важна уникальность элементов в группе

Стандартный набор операций для множества

add(e) – добавление элемента

list() – перечисление всех элементов

delete(e) – удаление элемента

В FCL встроенного типа для очереди с приоритетом обнаружить не удалось.

Массив

Выделение единого пространства в памяти и последовательная запись в него элементов

Содержимое памяти

12 13 14 15 16 17 18 19

Адреса ячеек памяти

- Все элементы массива занимают один и тот же объем
- К элементам массива можно обращаться напрямую

$$a = s + (b \cdot i)$$

s – адрес начала массива

b – размер каждого элемента (в байтах)

i – индекс элемента массива

a – адрес ***i***-го элемента массива

Проблема: ?

Нецелесообразно, тяжело, не всегда возможно выделить большой непрерывный блок памяти под массив

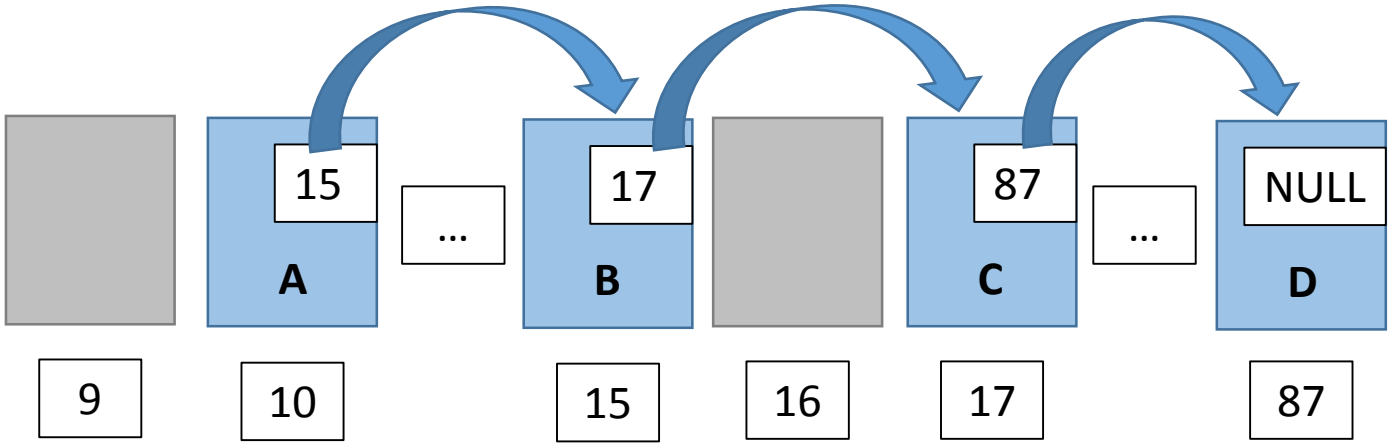
Linked List (связный список)

- Ячейки, в которых хранятся данные, могут находиться в произвольных участках памяти.

  Не обязательно последовательно друг за другом

- Память для ячеек выделяется по мере необходимости

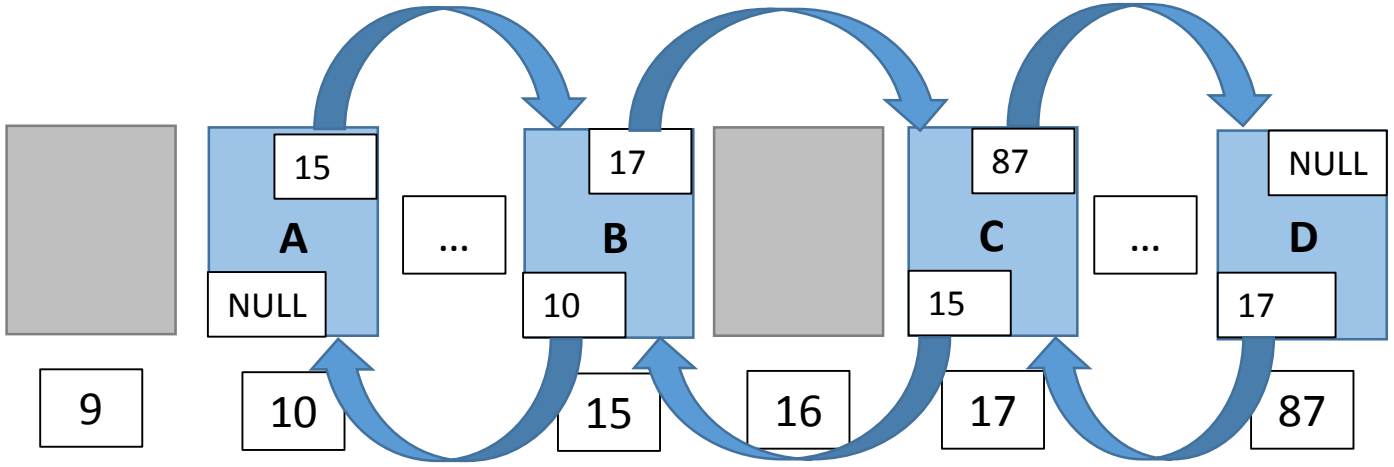
- Каждая ячейка имеет указатель, сообщающий об адресе следующей ячейке в цепи



- Размер ограничен имеющейся памятью
- Легкое добавление и удаление
- Долгий доступ к произвольному элементу необходимо перемещаться по указателям с начала списка

Double Linked List (двусвязный список)

- Связный список, где ячейки имеют два указателя:
  - на предыдущую ячейку
  - на следующую ячейку



можем двигаться в обоих направлениях

требуется больше памяти для хранения (ещё один указатель)

Array vs Linked List

Array

- Произвольный доступ к данным
- Быстрый доступ к элементам
- Число элементов не изменяется во время выполнения программы, благодаря чему легко выделить непрерывное пространство памяти

LinkedList

- Не требуется произвольный доступ к данным
- Быстро выполняются операции вставки, удаления
- Количество элементов заранее не известно (растет и / или уменьшается по ходу работы программы)

vs

Tree. Definitions

Связный список в виде ветвящейся структуры

Узел - ячейка

Ребро – указатель из одной ячейки на другую

Корень – самый первый узел, который не имеет родите

Все узлы, кроме корня, имеют строго одного родителя

Братские узлы – два узла с общим родителем

Предки узла – его родитель, прародитель, … корень

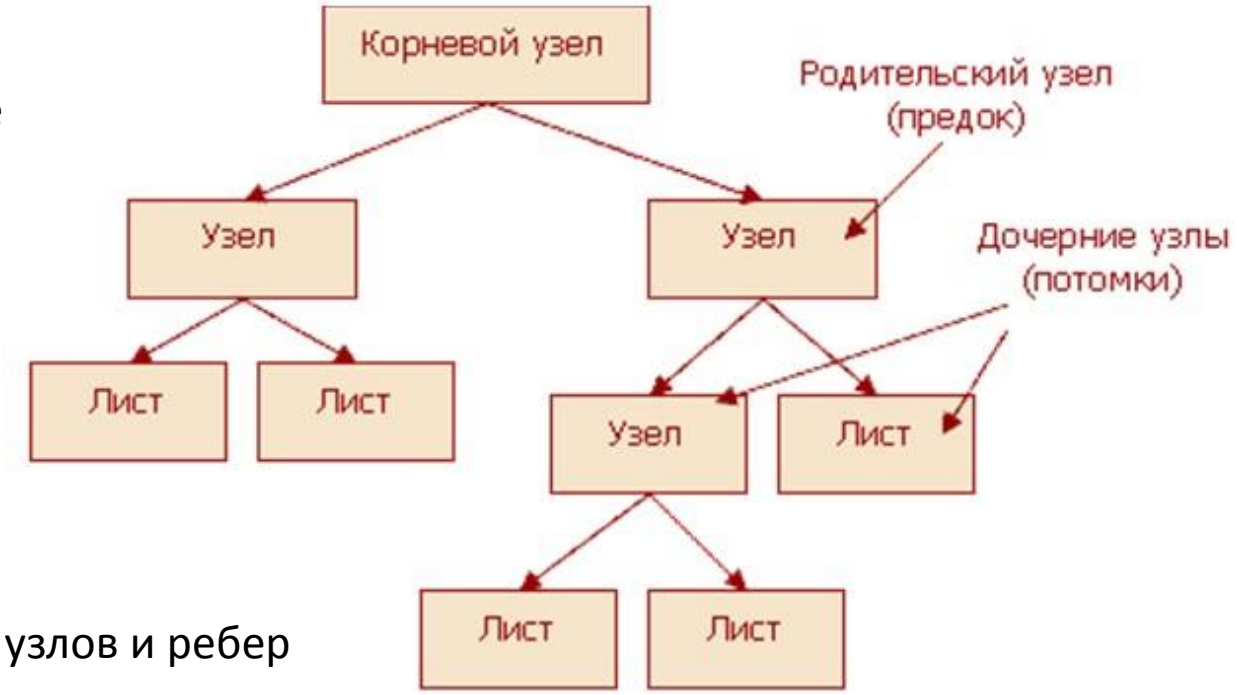
Потомки узла - его дочерние узлы, внуки и т.д.

Листья - узлы без дочерних узлов

Путь - между двумя узлами определяется множеством узлов и ребер

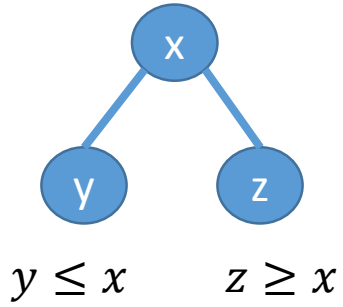
Уровень узла - длина пути от него до корневого узла

Высота дерева – уровень самого глубокого узла



Tree. Binary Search Tree

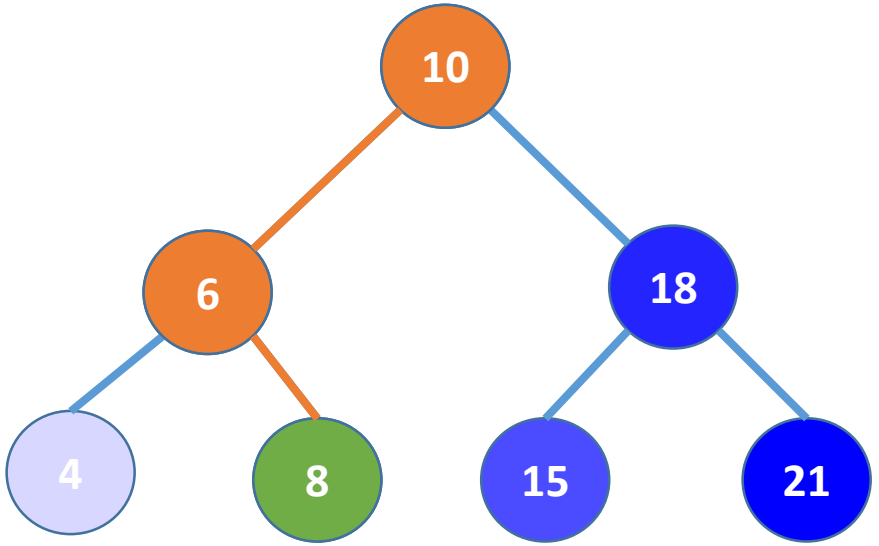
- узлы могут иметь не более двух дочерних узлов
- узлы располагаются согласно их значению:
  - слева меньше
  - справа больше



БЫСТРЫЙ ПОИСК

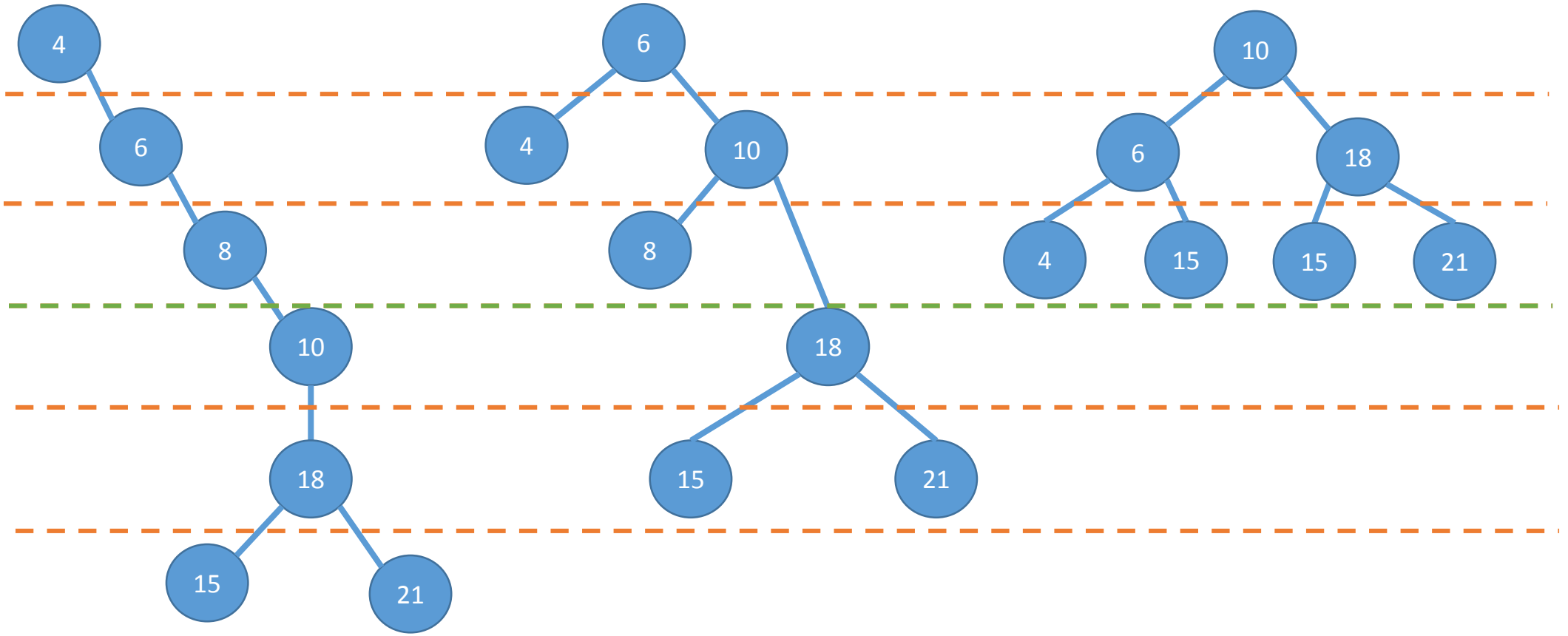
Пример поиска: пусть *X = 8*

1. X > 10 нет => идем влево
2. X > 6 да => идем вправо
3. X = 8 да ВСЁ!



Tree. Balance

Балансировка – уменьшение высоты дерева



плохая

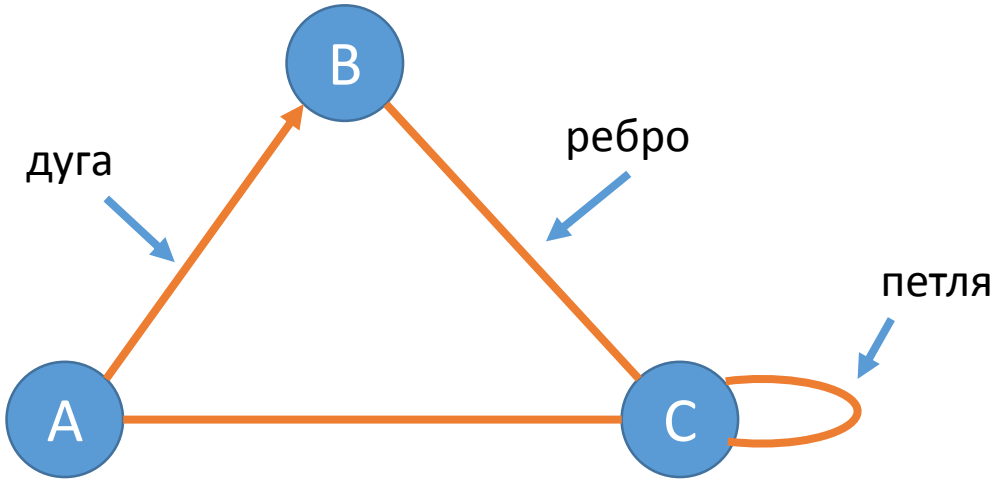
средняя

идеальная

БЫСТРЫЙ ПОИСК

Graph

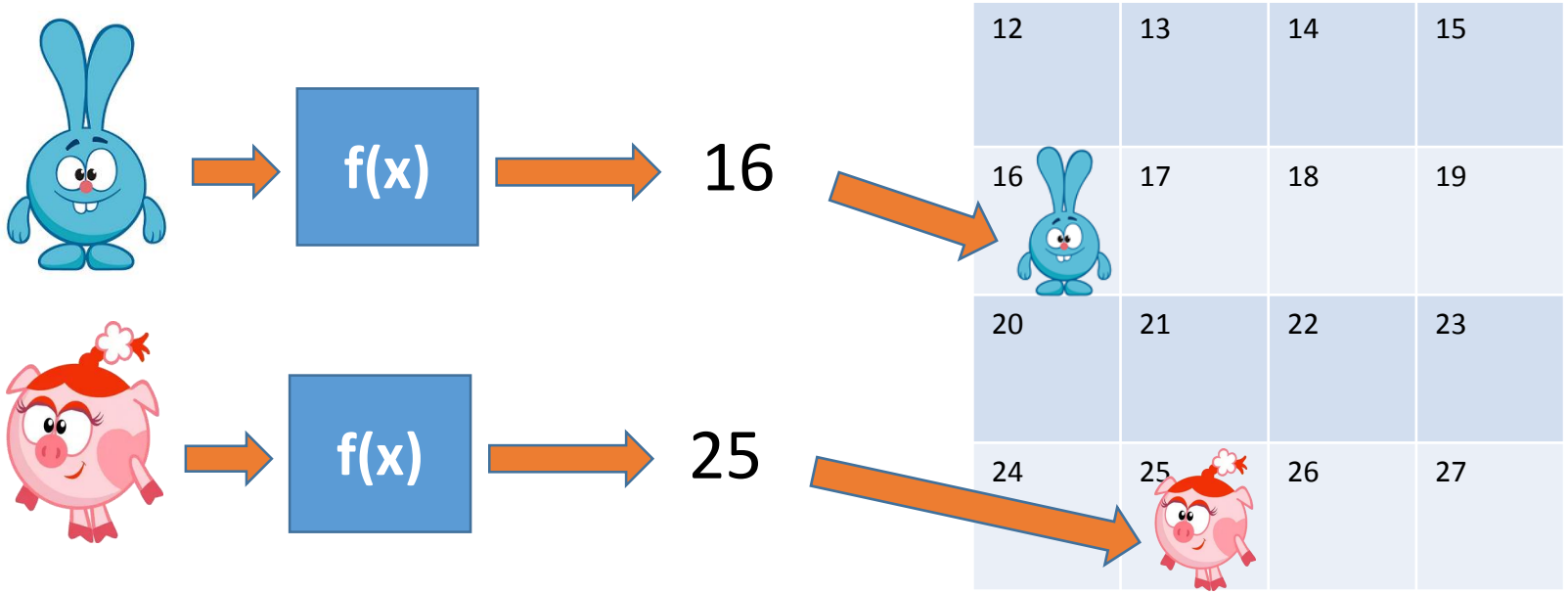
- Дерево без дочерних, родительских и корневого узлов.
- Данные свободно организованы в виде узлов (вершин) и дуг (ребер)
- Любой узел может иметь произвольное число входящих и исходящих ребер
- Самая гибкая из всех структур
- Пример: узлы люди, а ребра – дружеские связи



Hash Table

Хеш-функция (hash — «превращать в фарш», «мешанина»), или функция свёртки — функция, осуществляющая преобразование входных данных произвольной длины в выходную строку установленной длины, выполняемое определённым алгоритмом

Хеш-коллизия – одинаковый хэш код для разных входных данных (проблемика)



- Используются для реализации словарей и множеств.
- Вставка и удаление элементов быстрее, чем в деревьях
- Требуется выделение очень большого блока непрерывной памяти

Итоги

- Структуры данных определяют конкретные способы организации элементов в памяти компьютера.
- Разные структуры требуют разных операций для хранения, удаления, поиска и обхода хранящихся данных
- Каждый раз необходимо выбирать, какую структуру данных использовать в соответствии с текущей ситуацией
- Любой узел может иметь произвольное число входящих и исходящих ребер

- Вместо структур данных, лучше иметь дело с АТД (абстрактными типами данных). Они освобождают код от деталей, связанных с обработкой данных.
- Не изобретаем велосипед, пользуемся Generic Collections (обобщенными коллекциями)

Итоги

В какой структуре данных хранить:

- Вещи, которые Лосяш собирается взять в поход — List
- Тарелки, у Совуньи на кухне — Stack
- Сверла у Пина в коробке (все разного размера) — Set
- Товарищи, с которыми Нюша собирается погулять — Queue
- Поле для морского боя Бараша — Array
- Стоимость музыкальных инструментов Кар-Карыч — Dictionary


Спасибо за внимание!