

Лекция 11.

Вероятностный анализ.

Рандомизированные алгоритмы



Даниил Михайлович Берлизов

Старший преподаватель Кафедры вычислительных систем СибГУТИ

E-mail: sillyhat34@gmail.com

Курс «Структуры и алгоритмы обработки данных»
Осенний семестр, 2021 г.

Задача о найме

[CLRS, глава 5]

- Имеется n кандидатов
- Требуется нанять «лучшего» сотрудника
- Стоимость одного собеседования — c денежных единиц
- Стоимость процедуры найма — h денежных единиц



```
function HireAssistant()  
    best = 0  
    for i = 1 to n do  
        if IsFirstAssistantBetter(i, best) then  
            best = i  
            HireAssistant(i)  
        end if  
    end for  
end function
```

- Худший случай: претенденты приходят в упорядоченной последовательности, от наименее квалифицированного до «гуру» — **после каждого из собеседований нанимаем нового сотрудника**

$$T_{\text{Worst}} = O(nc + nh)$$

Задача о найме

```
function HireAssistant()  
    best = 0  
    for i = 1 to n do  
        if IsFirstAssistantBetter(i, best) then  
            best = i  
            HireAssistant(i)  
        end if  
    end for  
end function
```

- ♦ **Вероятностный анализ** (*probabilistic analysis*) — анализ алгоритмов, при котором делается предположение о распределении вероятности поступления входных данных и оценивается математическое ожидание числа операций

Задача о найме

```
function HireAssistant()  
    best = 0  
    for i = 1 to n do  
        if IsFirstAssistantBetter(i, best) then  
            best = i  
            HireAssistant(i)  
        end if  
    end for  
end function
```

- Пусть все претенденты приходят на собеседование в случайном порядке, их квалификации случайны
- Затраты на процедуру найма сотрудников — $nc + mh$
- Оценить затраты в среднем — вычислить математическое ожидание числа наймов сотрудников

Задача о найме

```
function HireAssistant()  
    best = 0  
    for i = 1 to n do  
        if IsFirstAssistantBetter(i, best) then  
            best = i  
            HireAssistant(i)  
        end if  
    end for  
end function
```

- ♦ Кандидаты приходят в случайном порядке
- ♦ Любой из первых i кандидатов может оказаться лучшим
- ♦ Первый всегда лучший
- ♦ Второй лучше первого с вероятностью $1/2$
- ♦ Третий лучше двух предыдущих с вероятностью $1/3$
- ♦ ...

Задача о найме

```
function HireAssistant()  
    best = 0  
    for i = 1 to n do  
        if IsFirstAssistantBetter(i, best) then  
            best = i  
            HireAssistant(i)  
        end if  
    end for  
end function
```

- Вероятность того, что квалификация претендента i выше квалификации претендентов $1, 2, \dots, i-1$, равна $1/i$, тогда математическое ожидание числа наймов новых сотрудников равно:

$$\sum_{i=1}^n \frac{1}{i} = 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n} \approx \ln n + \gamma$$

- $\gamma = 0,5772\dots$ — постоянная Эйлера
- Затраты на организацию найма в среднем случае равны $O(nc + h \ln n)$

Задача о найме

```
function HireAssistant()  
    best = 0  
    for i = 1 to n do  
        if IsFirstAssistantBetter(i, best) then  
            best = i  
            HireAssistant(i)  
        end if  
    end for  
end function
```

- Как добиться того, чтобы худший случай ($O(nc + nh)$) не возникал и преобладал средний случай входных данных ($O(nc + h\ln n)$)?

Задача о найме

```
function HireAssistant()  
  best = 0  
  for i = 1 to n do  
    if IsFirstAssistantBetter(i, best) then  
      best = i  
      HireAssistant(i)  
    end if  
  end for  
end function
```

- Если заранее известен список кандидатов, мы можем выполнить его *рандомизацию* — реализовать случайную перестановку, чтобы избежать возникновения худшего случая
- Возникновение худшего случая не исключено, если будет «плохой» генератор случайных чисел

Задача о найме

```
function RandomizedHireAssistant()  
    PermuteRandom()      /* Перемешать список кандидатов */  
    best = 0  
    for i = 1 to n do  
        if IsFirstAssistantBetter(i, best) then  
            best = i  
            HireAssistant(i)  
        end if  
    end for  
end function
```

- **Рандомизированный алгоритм** (*randomized algorithm*) — алгоритм, некоторые шаги которого основаны на случайном правиле (датчик псевдослучайных чисел)
- Рандомизированные алгоритмы являются недетерминированными
- Случайную перестановку можно реализовать разными способами, например, за время $O(n \log n)$ на основе сортировки [CLRS, С. 151]

Случайная перестановка на месте

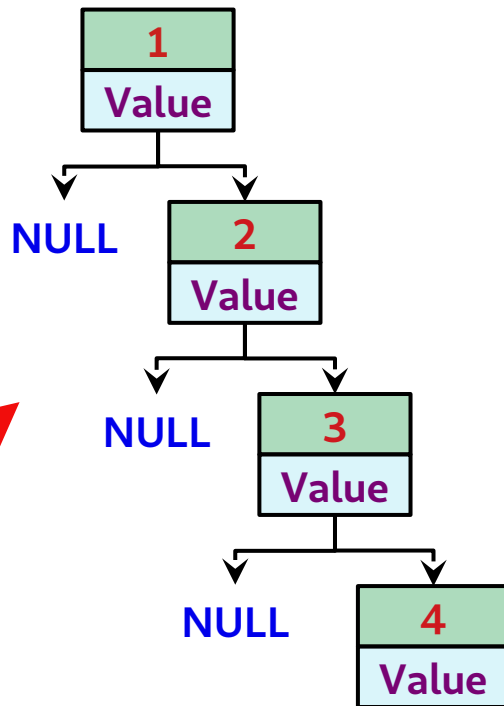
```
function PermuteRandomInPlace(a[1:n], n)
  for i = 1 to n do
    temp = a[i]
    r = Random(i, n)
    a[i] = a[r]
    a[r] = temp
  end for
end function
```

$T = O(n)$

Двоичные деревья поиска

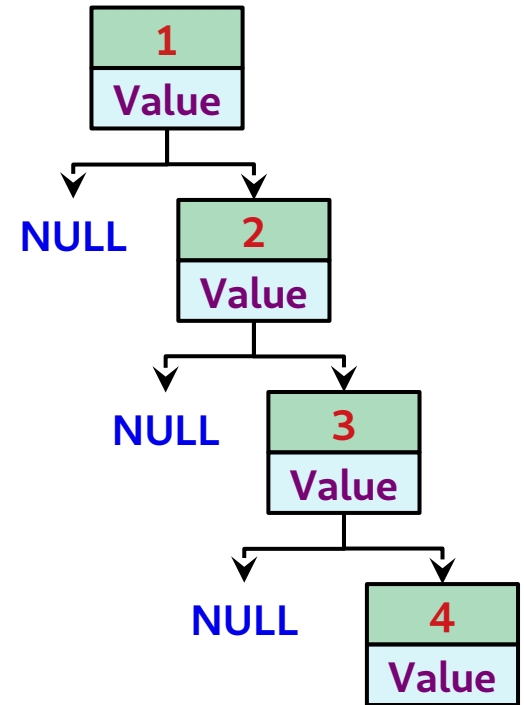
- Операции над двоичным деревом имеют трудоёмкость, пропорциональную высоте h дерева
- В среднем случае высота дерева $O(\log n)$
- В худшем случае элементы добавляются по возрастанию (убыванию) ключей — дерево вырождается в список длины $O(n)$

```
bstree_add(1, value)  
bstree_add(2, value)  
bstree_add(3, value)  
bstree_add(4, value)
```



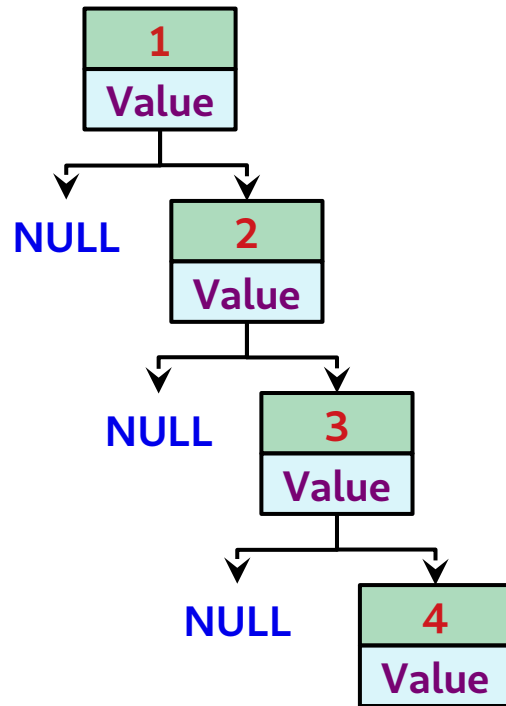
Двоичные деревья поиска

- Как минимизировать вероятность возникновения худшего случая?



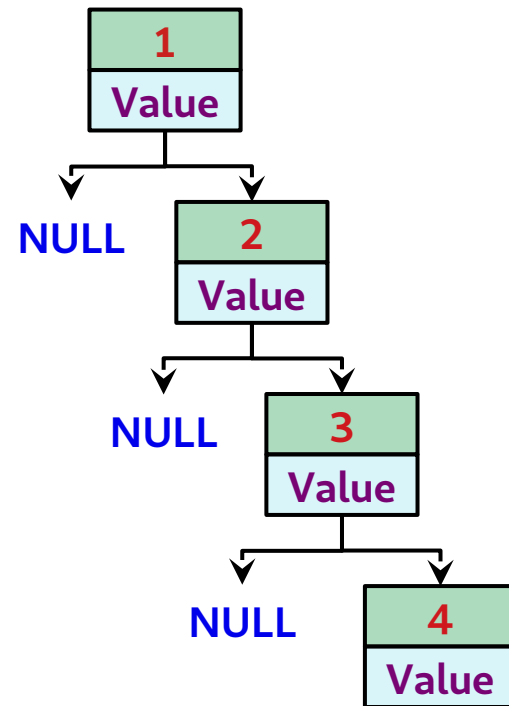
Двоичные деревья поиска

- Как минимизировать вероятность возникновения худшего случая?
- **Offline-решение**
 - Заранее известен набор из n ключей
 - Перемешиваем ключи в случайном порядке и вставляем их в дерево
 - Перемешивание требует $O(n)$ операций



Двоичные деревья поиска

- Как минимизировать вероятность возникновения худшего случая?
- Online-решение
 - Ключи поступают динамически
 - Как реализовать рандомизацию?



Рандомизированные деревья поиска

- **Рандомизированное бинарное дерево поиска** (*randomized binary search tree, RBST*) — бинарное дерево поиска, в котором ключи с некоторой вероятностью вставляются в корень
- Любой вставляемый ключ с вероятностью $1 / n$ может оказаться корнем — заменить его

Рандомизированные деревья поиска

- Вставка ключа в RBST, содержащее n узлов
- С вероятностью $1 / (n + 1)$ делаем новый ключ корнем

```
function RBSTInsert(tree, key)
  r = Random(0, n)
  if r = n then
    tree = RBSTInsertAtRoot(tree, key)  /* Замена корня */
  if key < tree.key then
    tree = RBSTInsert(tree.left, key)
  else
    tree = RBSTInsert(tree.right, key)
  end if
  return tree
end function
```

Рандомизированные деревья поиска

```
function RBSTInsertAtRoot(tree, key, left, right)
    RBSTSplit(tree, key, left, right)
    root = RBSTCreate()
    root.key = key
    root.left = left
    root.right = right
    return root
end function

function RBSTSplit(tree, key, left, right)
    if n = 0 then
        left = RBSTCreate()
        right = RBSTCreate()
    else if key < tree.key then
        right = tree
        RBSTSplit(tree.left, key, left, right.left)
    else
        left = tree
        RBSTSplit(tree.right, key, left.right, right)
    end if
end function
```

Дальнейшее чтение

- Прочитать в [CLRS, 3ed.] главу 5 «Вероятностный анализ и рандомизированные алгоритмы»
- Разобрать операцию удаления элемента из рандомизированного бинарного дерева поиска



ご清聴ありがとうございました!



Даниил Михайлович Берлизов

Старший преподаватель Кафедры вычислительных систем СибГУТИ

E-mail: sillyhat34@gmail.com

Курс «Структуры и алгоритмы обработки данных»

Осенний семестр, 2021 г.