- 作业讲解
  - TC第12.1节练习2、5
  - TC第12.2节练习5、8、9
  - TC第12.3节练习5
  - TC第12章问题1
  - TC第13.1节练习5、6、7
  - TC第13.2节练习2
  - TC第13.3节练习1、5
  - TC第13.4节练习1、2、7

TC第12.1节练习2

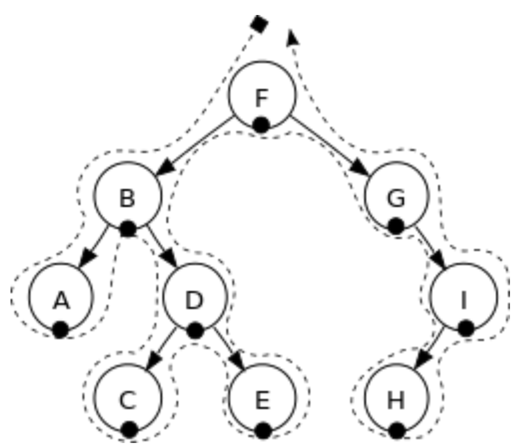
- BST的性质
  - ≥左子节点，≤右子节点，这样对吗？
  - ≥左子树中的节点，≤右子树中的节点

TC第12.1节练习5

- Any comparison-based algorithm for constructing a binary search tree from an arbitrary list of n elements takes Ω(nlgn) time in the worst case.
  - 反证法：假设只需o(nlgn)，则comparison-based sorting只需o(nlgn)。

TC第12.2节练习8

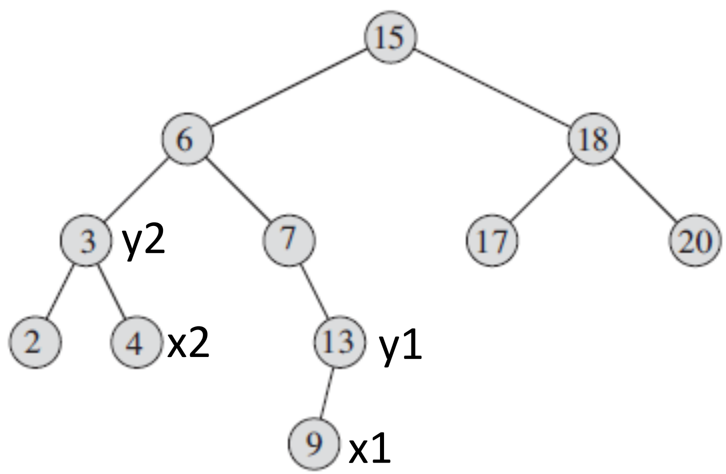
- k successive calls to TREE-SUCCESSOR take O(k+h) time.
  - 其实是在做中序遍历
  - 运行时间即经过顶点的总次数，分两种情况
    - 向上到达
      - 在首顶点向上走到根的路径上（<=h）
      - 其它每次向上到达必然伴随着一次向下到达，不影响渐进时间
    - 向下到达
      - k个后继（=k）
      - 其它都是花费在那些key更大的顶点上，只存在于末顶点到根的路径上（<=h）



TC第12.2节练习9

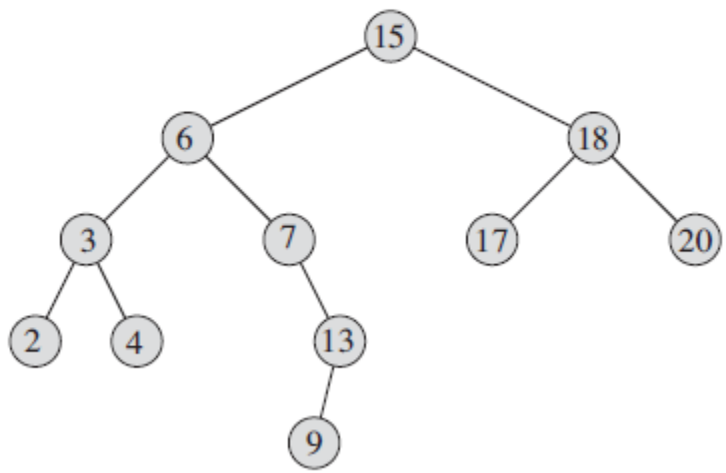
- 为什么y1一定是x1的后继？
- 为什么y2一定是x2的前驱？

- 注意：讨论的范围不能限于以y为根的子树。



TC第12.3节练习5

- Instead of x.p, keeps x.succ.
  - 实现getParent函数
  - 注意维护受影响顶点的succ

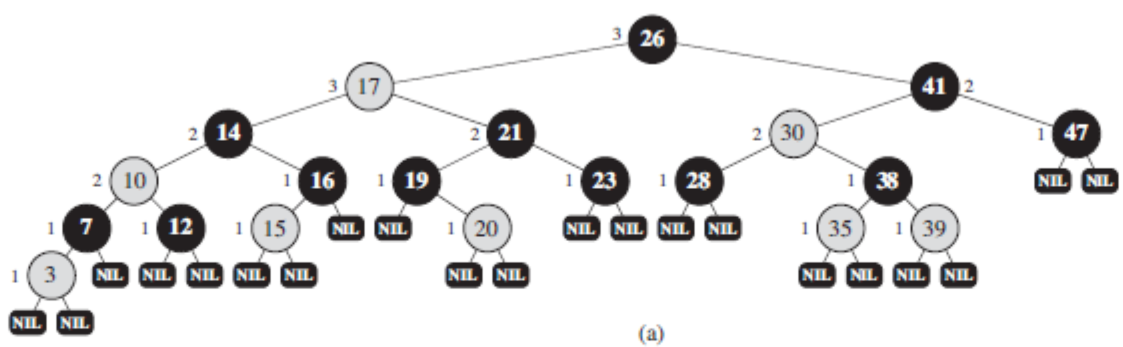


TC第12章问题1

- (a) insert n items with identical keys.
  - n^2
- (b) alternates between x.left and x.right.
  - nlgn
- (c) list
  - n
- (d) randomly between x.left and x.right.
  - Worst-case: n^2
  - Expected: nlgn

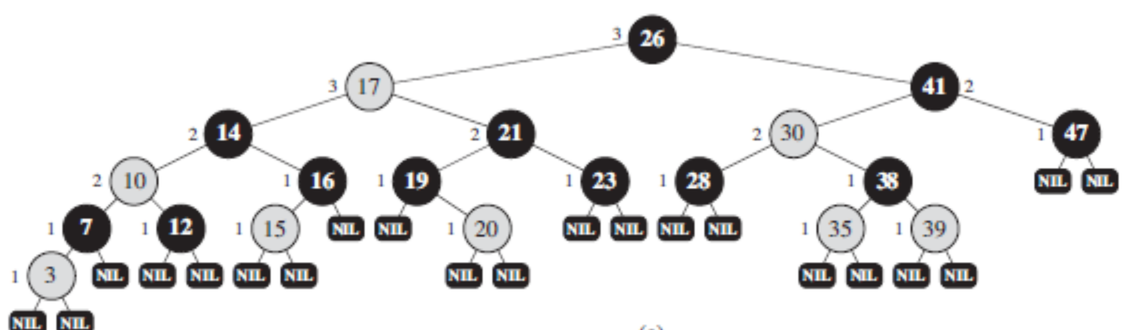
TC第13.1节练习6

- Number of internal nodes with black-height k?
  - Largest: 2^{2k}-1，不是2^{2k+2}-1（P309: from, but not including, a node…）
  - Smallest: 2^{k}-1，不是k（ P308: We shall regard these NILs as…）



TC第13.1节练习7

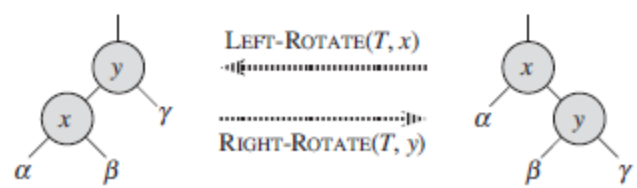
- Ratio of red internal nodes to black internal nodes.
  - Largest: 2
  - Smallest: 0



(a)

TC第13.2节练习2

- Exactly n-1 possible rotations.
  - 每个rotation都将一个顶点提到了其父顶点的位置
  - 每个非根顶点对应一种被提的rotation，总共n-1种



- 教材讨论
  - TC第16章第1、2、3节
  - TC第17章

问题1：greedy algorithms

- 你怎么理解greedy algorithms的两个重要性质？
  - greedy-choice property
  - optimal substructure
- 你能不能结合activity-selection problem解释为什么这两个性质缺一不可？
- 为什么greedy algorithms比dynamic programming快？

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

问题1：greedy algorithms

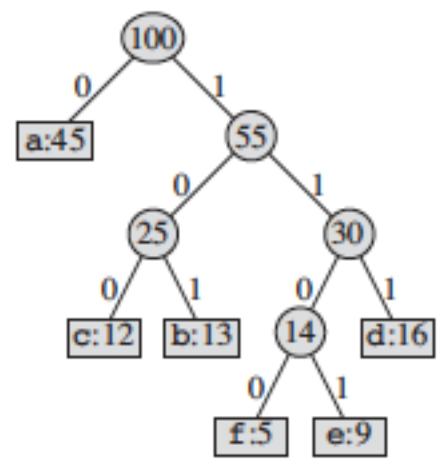
- 你怎么理解greedy algorithms的两个重要性质？
  - greedy-choice property
  - optimal substructure
- 你能不能结合activity-selection problem解释为什么这两个性质缺一不可？
- 为什么greedy algorithms比dynamic programming快？
  - making the first choice before solving any subproblems
  - making one greedy choice after another
    reducing each given problem instance to a smaller one

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

问题1：greedy algorithms (续)

- 关于Huffman codes的greedy algorithm
  - greedy choice是什么？
  - greedy-choice property是什么？
  - optimal substructure是什么？

	a	b	c	d	e	f
Frequency (in thousands)	45	13	12	16	9	5
Fixed-length codeword	000	001	010	011	100	101
Variable-length codeword	0	101	100	111	1101	1100



问题1：greedy algorithms (续)

Describe an efficient algorithm that, given a set $\{x_1, x_2, \ldots, x_n\}$ of points on the real line, determines the smallest set of unit-length closed intervals that contains all of the given points. Argue that your algorithm is correct.

问题1：greedy algorithms (续)

Describe an efficient algorithm that, given a set $\{x_1, x_2, \ldots, x_n\}$ of points on the real line, determines the smallest set of unit-length closed intervals that contains all of the given points. Argue that your algorithm is correct.

Sol: First we sort the set of n points $\{x_1, x_2, ..., x_n\}$ to get the set $Y = \{y_1, y_2, ..., y_n\}$ such that $y_1 \leq y_2 \leq ... \leq y_n$. Next, we do a linear scan on $\{y_1, y_2, ..., y_n\}$ started from y_1. Everytime while encountering y_i, for some $i \in \{1, ..., n\}$, we put the closed interval $[y_i, y_i + 1]$ in our optimal solution set S, and remove all the points in Y covered by $[y_i, y_i + 1]$. Repeat the above procedure, finally output S while Y becomes empty. We next show that S is an optimal solution.

We claim that there is an optimal solution which contains the unit-length interval $[y_1, y_1 + 1]$. Suppose that there exists an optimal solution S^* such that y_1 is covered by $[x', x'+1] \in S^*$ where $x' < 1$. Since y_1 is the leftmost element of the given set, there is no other point lying in $[x', y_1)$. Therefore, if we replace $[x', x'+1]$ in S^* by $[y_1, y_1+1]$, we will get another optimal solution. This proves the claim and thus explains the greedy choice property. Therefore, by solving the remaining subproblem after removing all the points lying in $[y_1, y_1 + 1]$, that is, to find an optimal set of intervals, denoted as S', which cover the points to the right of $y1 + 1$, we will get an optimal solution to the original problem by taking union of $[y_1, y_1 + 1]$ and S'.

问题1：greedy algorithms (续)

- 建议阅读打星号的16.4节

问题2： amortized analysis

- amortized analysis和average-case analysis有什么异同？

问题2： amortized analysis

- amortized analysis和average-case analysis有什么异同？
  - per operation vs. per algorithm
  - worst-case vs. average-case

问题2： amortized analysis (续)

- 这些问题的分析难在哪儿？
  amortized analysis能带来什么好处？
  - stack operations

PUSH(S, x) pushes object x onto stack S.

POP(S) pops the top of stack S and returns the popped object. Calling POP on an empty stack generates an error.

```
MULTIPOP(S, k)
while not STACK-EMPTY(S) and k > 0
    POP(S)
    k = k - 1
```

  - incrementing a binary counter

Counter value	A[7]	A[6]	A[5]	A[4]	A[3]	A[2]	A[1]	A[0]	Total cost
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	1
2	0	0	0	0	0	0	1	0	3
3	0	0	0	0	0	0	1	1	4
4	0	0	0	0	0	1	0	0	7
5	0	0	0	0	0	1	0	1	8
6	0	0	0	0	0	1	1	0	10
7	0	0	0	0	0	1	1	1	11
8	0	0	0	0	1	0	0	0	15
9	0	0	0	0	1	0	0	1	16
10	0	0	0	0	1	0	1	0	18
11	0	0	0	0	1	0	1	1	19
12	0	0	0	0	1	1	0	0	22
13	0	0	0	0	1	1	0	1	23
14	0	0	0	0	1	1	1	0	25
15	0	0	0	0	1	1	1	1	26
16	0	0	0	1	0	0	0	0	31

问题2： amortized analysis (续)

- aggregate analysis
  - 该方法的基本思路是什么？
  - 如何用来解决这两个问题？
    - 如果增加一个DECREMENT，结果又如何？
  - 它在使用上有什么局限吗？

PUSH(S, x) pushes object x onto stack S.

POP(S) pops the top of stack S and returns the popped object. Calling POP on an empty stack generates an error.

```
MULTIPOP(S, k)
while not STACK-EMPTY(S) and k > 0
    POP(S)
    k = k - 1
```

Counter value	A[7]	A[6]	A[5]	A[4]	A[3]	A[2]	A[1]	A[0]	Total cost
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	1
2	0	0	0	0	0	0	1	0	3
3	0	0	0	0	0	0	1	1	4
4	0	0	0	0	0	1	0	0	7
5	0	0	0	0	0	1	0	1	8
6	0	0	0	0	0	1	1	0	10
7	0	0	0	0	0	1	1	1	11
8	0	0	0	0	1	0	0	0	15
9	0	0	0	0	1	0	0	1	16
10	0	0	0	0	1	0	1	0	18
11	0	0	0	0	1	0	1	1	19
12	0	0	0	0	1	1	0	0	22
13	0	0	0	0	1	1	0	1	23
14	0	0	0	0	1	1	1	0	25
15	0	0	0	0	1	1	1	1	26
16	0	0	0	1	0	0	0	0	31

问题2：amortized analysis (续)

- accounting method
  - 该方法的基本思路是什么？
    右侧这个式子是什么含义？ $\sum_{i=1}^{n} \hat{c}_i \geq \sum_{i=1}^{n} c_i$
  - 如何用来解决这两个问题？
    上述式子是如何保证成立的？
    - 如果增加一个RESET，结果又如何？
      (Keep a pointer to the high-order 1.)

PUSH(S, x) pushes object x onto stack S.

POP(S) pops the top of stack S and returns the popped object. Calling POP on an empty stack generates an error.

MULTIPOP(S, k)

```
while not STACK-EMPTY(S) and k > 0
    POP(S)
    k = k - 1
```

Counter value	A[7]	A[6]	A[5]	A[4]	A[3]	A[2]	A[1]	A[0]	Total cost
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	1
2	0	0	0	0	0	0	1	0	3
3	0	0	0	0	0	0	1	1	4
4	0	0	0	0	0	1	0	0	7
5	0	0	0	0	0	1	0	1	8
6	0	0	0	0	0	1	1	0	10
7	0	0	0	0	0	1	1	1	11
8	0	0	0	0	1	0	0	0	15
9	0	0	0	0	1	0	0	1	16
10	0	0	0	0	1	0	1	0	18
11	0	0	0	0	1	0	1	1	19
12	0	0	0	0	1	1	0	0	22
13	0	0	0	0	1	1	0	1	23
14	0	0	0	0	1	1	1	0	25
15	0	0	0	0	1	1	1	1	26
16	0	0	0	1	0	0	0	0	31

问题2： amortized analysis (续)

- potential method
  - 该方法的基本思路是什么？右侧这组式子是什么含义？
  - 如何用来解决这两个问题？以及一个新问题：
    Implement a queue with two stacks.
    The amortized cost of ENQ and DEQ is O(1).

$$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) .$$

$$\begin{aligned} \sum_{i=1}^{n} \hat{c}_i &= \sum_{i=1}^{n} (c_i + \Phi(D_i) - \Phi(D_{i-1})) \\ &= \sum_{i=1}^{n} c_i + \Phi(D_n) - \Phi(D_0) . \end{aligned}$$

$$\Phi(D_n) \ge \Phi(D_0)$$

Counter value	A[7]	A[6]	A[5]	A[4]	A[3]	A[2]	A[1]	A[0]	Total cost
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	1
2	0	0	0	0	0	0	1	0	3
3	0	0	0	0	0	0	1	1	4
4	0	0	0	0	0	1	0	0	7
5	0	0	0	0	0	1	0	1	8
6	0	0	0	0	0	1	1	0	10
7	0	0	0	0	0	1	1	1	11
8	0	0	0	0	1	0	0	0	15
9	0	0	0	0	1	0	0	1	16
10	0	0	0	0	1	0	1	0	18
11	0	0	0	0	1	0	1	1	19
12	0	0	0	0	1	1	0	0	22
13	0	0	0	0	1	1	0	1	23
14	0	0	0	0	1	1	1	0	25
15	0	0	0	0	1	1	1	1	26
16	0	0	0	1	0	0	0	0	31

PUSH(S, x) pushes object x onto stack S.

POP(S) pops the top of stack S and returns the popped object. Calling POP on an empty stack generates an error.

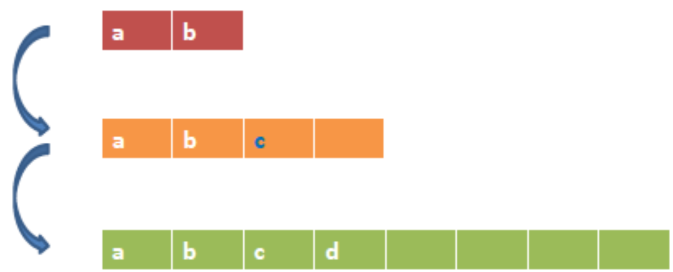
```
MULTIPOP(S, k)
while not STACK-EMPTY(S) and k > 0
    POP(S)
    k = k - 1
```

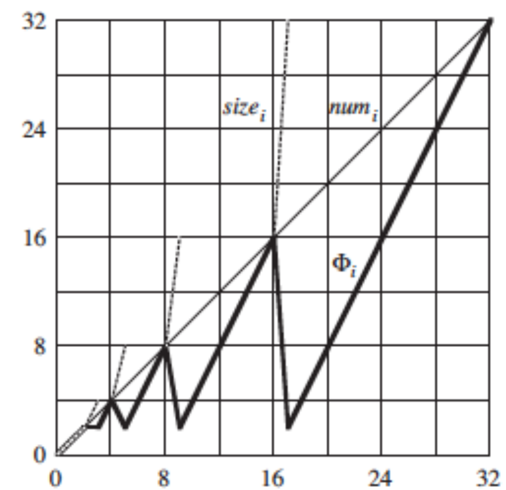
问题3： dynamic tables

- 对于table expansion， 你能解释aggregate和accounting的分析过程吗？

$$c_i = \begin{cases} i & \text{if } i-1 \text{ is an exact power of 2 ,} \\ 1 & \text{otherwise .} \end{cases}$$

$$\begin{aligned} \sum_{i=1}^{n} c_i &\le n + \sum_{j=0}^{\lfloor \lg n \rfloor} 2^j \\ &< n + 2n \\ &= 3n , \end{aligned}$$





- 对于potential function $\Phi(T) = 2 \cdot T.num - T.size$
你能结合accounting来解释它的走势吗？