- 作业讲解
  - CS第5.5节问题8、11、14
  - TC第11.2节练习3、5、6
  - TC第11.3节练习3、4
  - TC第11.4节练习2、3
  - TC第11章问题1、2

  - 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

CS第5.5节问题8

- hash n items into k locations
  - (a) probability that all n items hash to different locations
    - n>k: 0
    - n≤k: $\frac{A_k^n}{k^n}$
  - (b) probability that the i-th item is the first collision
    - $\frac{A_k^{i-1}}{k^{i-1}} \cdot \frac{i-1}{k}$
  - (c) expected number of items you hash until the first collision
    - $\sum_{i=2}^{\min(n,k+1)} i\left(\frac{A_k^{i-1}}{k^{i-1}} \cdot \frac{i-1}{k}\right)$

CS第5.5节问题14

- expected number of empty slots when you hash 2k items into a hash table with k slots
  - 定理5.15 $$k\left(1-\frac{1}{k}\right)^{2k}$$
- expected fraction of empty slots when k is reasonably large
  - $$\lim_{k\to+\infty}\frac{k\left(1-\frac{1}{k}\right)^{2k}}{k}=\left(\lim_{k\to+\infty}\left(1-\frac{1}{k}\right)^{k}\right)^{2}=\frac{1}{e^{2}}$$

TC第11.2节练习5

- 鸽巢原理

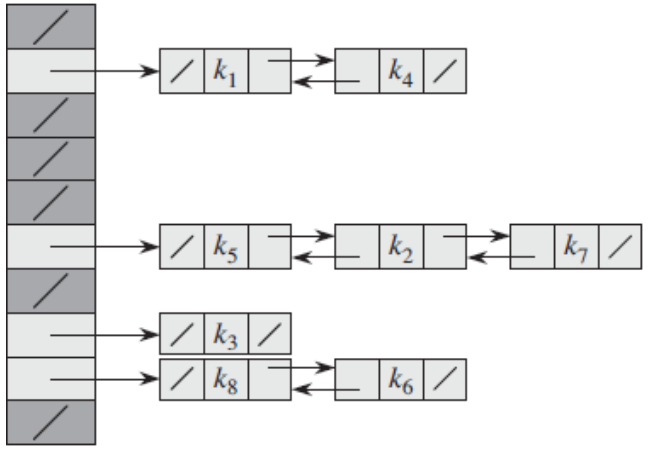
TC第11.2节练习6

- Selects a key uniformly at random from among n keys in the hash table of size m and returns it in expected time O(L(1+1/α)).
  - 方法1：随机等概率地选一个chain，再从中随机等概率地选一个key，这样可以吗？
  - 方法2：随机等概率地选一个序号，再找到对应的key
    - 先找chain，期望运行时间：

$$\sum_{i=1}^{m} \frac{L_i}{n} i = \cdots$$

    - 再在chain中找key，期望运行时间：

$$\sum_{i=1}^{m} \frac{L_i}{n} \frac{1+L_i}{2} = \cdots$$



TC第11.3节练习3

- character string interpreted in radix 2^p
  - $x_n \ldots x_1 x_0 \rightarrow \sum x_i (2^p)^i$
  - $\left(x_i \left(2^p\right)^i + x_j \left(2^p\right)^j\right) - \left(x_j \left(2^p\right)^i + x_i \left(2^p\right)^j\right)$
    $= \left(x_i - x_j\right)\left(\left(2^p\right)^i - \left(2^p\right)^j\right)$
    $= \left(x_i - x_j\right)\left(2^p - 1\right)\cdots$

TC第11章问题1

- (d) Show that the expected length E[X] of the longest probe sequence is O(lgn).

$$
\begin{aligned}
E[X] &= \sum_{i=1}^{n} i \Pr(X = i) = \sum_{i=1}^{2\lg n} i \Pr(X = i) + \sum_{i=2\lg n+1}^{n} i \Pr(X = i) \\
&\le 2\lg n \sum_{i=1}^{2\lg n} \Pr(X = i) + n \sum_{i=2\lg n+1}^{n} \Pr(X = i) \\
&= 2\lg n \Pr(X \le 2\lg n) + n \Pr(X > 2\lg n) \\
&\le 2\lg n + n\mathrm{O}\left(\frac{1}{n}\right) \\
&= \mathrm{O}(\lg n)
\end{aligned}
$$

TC第11章问题2

- (b) Show that $P_k \le nQ_k$.
  - P_k=Pr(最多的一个恰为k)

    =Pr(存在一个恰为k且其余均≤k)

    ≤Pr(存在一个恰为k)

    ≤∑Pr(第i个恰为k)

    =∑Q_k

    =nQ_k

TC第12.1节练习2

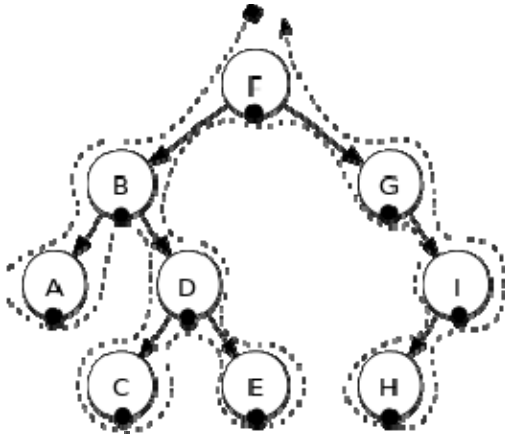
- BST的性质
  - ≥左子节点，≤右子节点，这样对吗？
  - ≥左子树中的节点，≤右子树中的节点

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

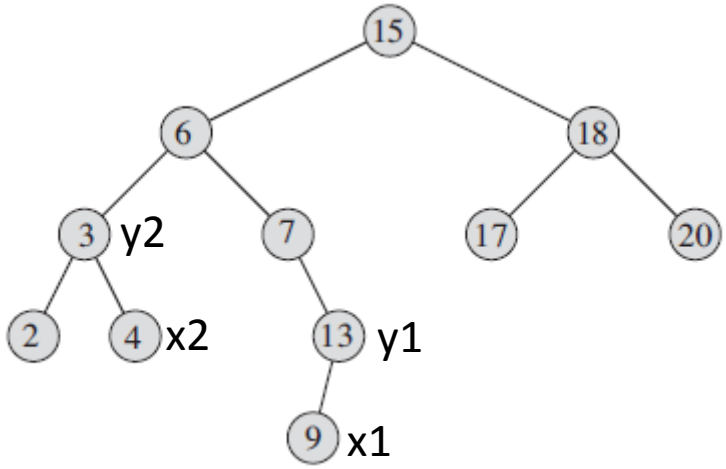
- k successive calls to TREE-SUCCESSOR take O(k+h) time.
  - 其实是在做中序遍历
  - 运行时间即经过顶点的总次数，分两种情况
    - 向上到达
      - 在首顶点向上走到根的路径上（<=h）
      - 其它每次向上到达必然伴随着一次向下到达，不影响渐进时间
    - 向下到达
      - k个后继（=k）
      - 其它都是花费在那些key更大的顶点上，只存在于末顶点到根的路径上（<=h）



TC第12.2节练习9

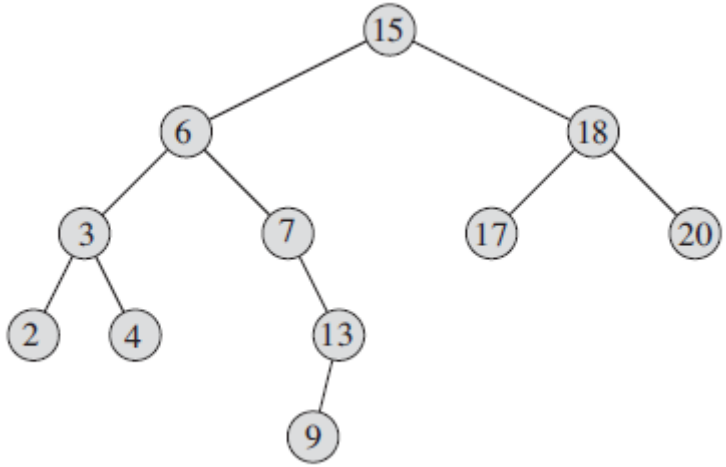
- 为什么y1一定是x1的后继？
- 为什么y2一定是x2的前驱？

- 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

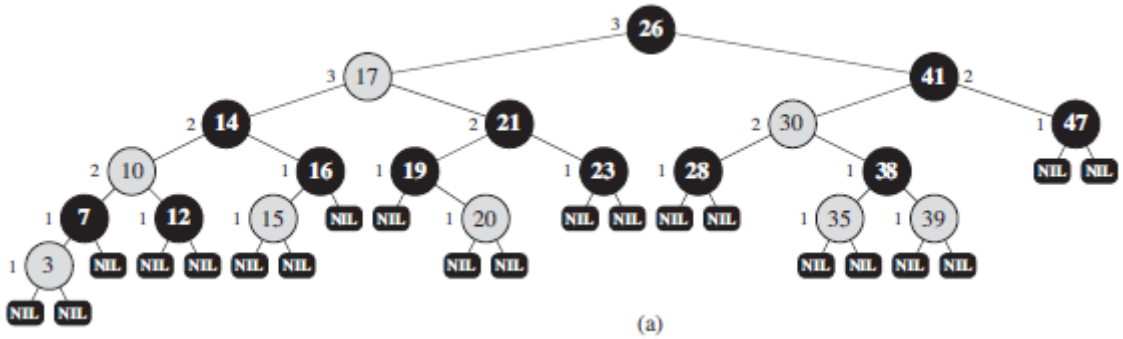


TC第12章问题1

- (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

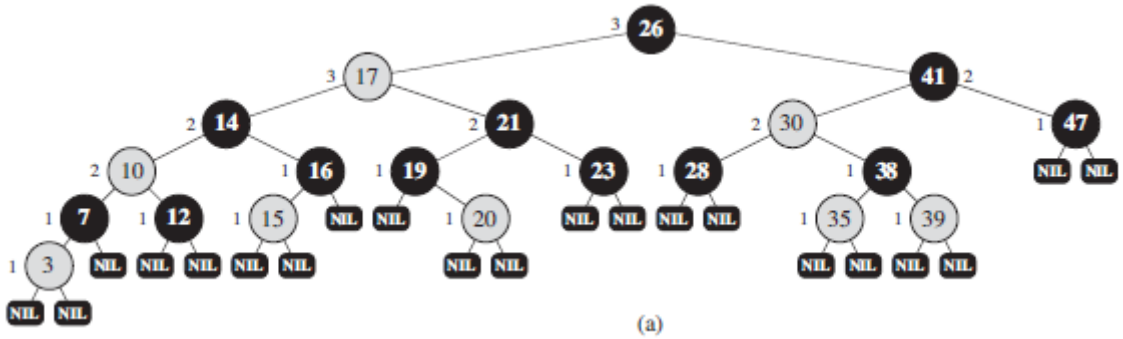
TC第13.1节练习6

- 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…）



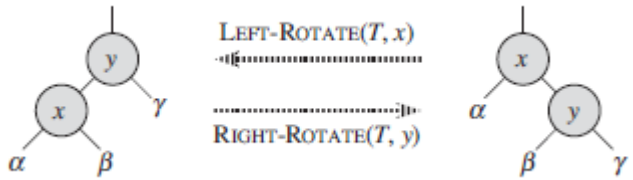
TC第13.1节练习7

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



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15章
  - TC第16章第1、2、3节

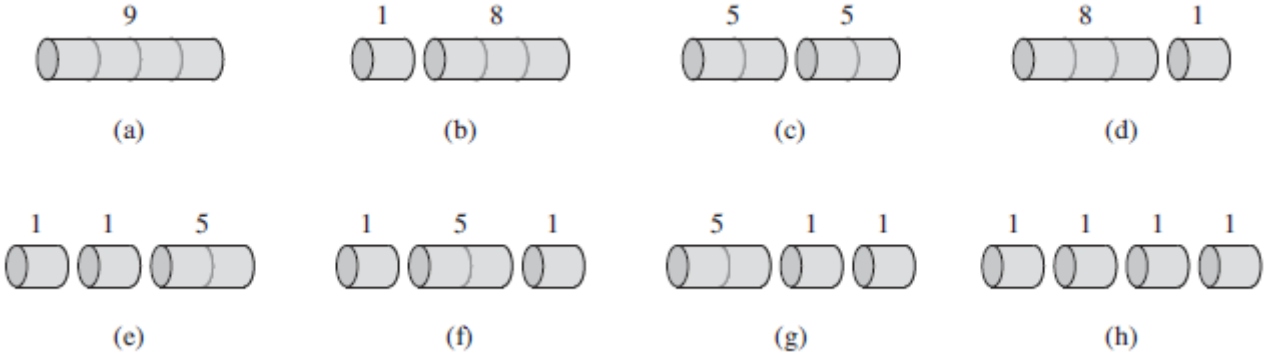
问题1：dynamic programming

- 什么样的问题可以使用dynamic programming来求解？它高效的根本原因是什么？付出了什么代价？
- 你理解dynamic programming的四个步骤了吗？
  1. Characterize the structure of an optimal solution.
  2. Recursively define the value of an optimal solution.
  3. Compute the value of an optimal solution.
  4. Construct an optimal solution from computed information.

- 决定dynamic programming运行时间的要素是哪两点？
- top-down with memorization和bottom-up method哪个更快？

问题1：dynamic programming (续)

- 你能说明求解rod cutting的四个步骤吗？
  1. Characterize the structure of an optimal solution.
  2. Recursively define the value of an optimal solution.
  3. Compute the value of an optimal solution.
  4. Construct an optimal solution from computed information.

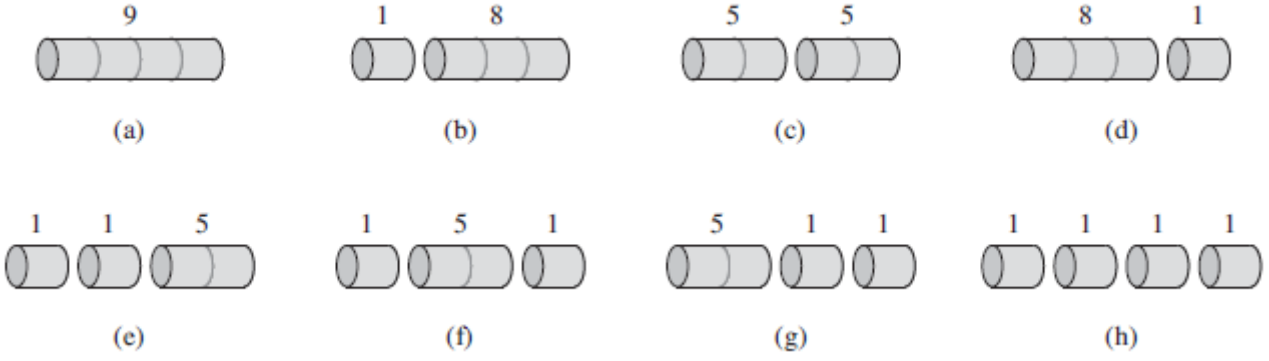


length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

问题1：dynamic programming (续)

- 你能说明求解rod cutting的四个步骤吗？
  1. Characterize the structure of an optimal solution.
  2. Recursively define the value of an optimal solution.
  3. Compute the value of an optimal solution.
  4. Construct an optimal solution from computed information.

$$r_n = \max_{1 \le i \le n} (p_i + r_{n-i})$$



length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

问题1：dynamic programming (续)

- 你能说明求解matrix-chain multiplication的四个步骤吗？
  1. Characterize the structure of an optimal solution.
  2. Recursively define the value of an optimal solution.
  3. Compute the value of an optimal solution.
  4. Construct an optimal solution from computed information.

$$
\begin{aligned}
&(A_1(A_2(A_3A_4)))\,,\\
&(A_1((A_2A_3)A_4))\,,\\
&((A_1A_2)(A_3A_4))\,,\\
&((A_1(A_2A_3))A_4)\,,\\
&(((A_1A_2)A_3)A_4)\,.
\end{aligned}
$$

问题1：dynamic programming (续)

- 你能说明求解matrix-chain multiplication的四个步骤吗？
  1. Characterize the structure of an optimal solution.
  2. Recursively define the value of an optimal solution.
  3. Compute the value of an optimal solution.
  4. Construct an optimal solution from computed information.

$$m[i,j] = \begin{cases} 0 & \text{if } i = j\ , \\ \min\limits_{i \le k < j} \{m[i,k] + m[k+1,j] + p_{i-1} p_k p_j\} & \text{if } i < j\ . \end{cases}$$

$$\begin{array}{l} (A_1(A_2(A_3A_4)))\ , \\ (A_1((A_2A_3)A_4))\ , \\ ((A_1A_2)(A_3A_4))\ , \\ ((A_1(A_2A_3))A_4)\ , \\ (((A_1A_2)A_3)A_4)\ . \end{array}$$

问题1：dynamic programming (续)

- 你能说明求解longest common subsequence的四个步骤吗？
  1. Characterize the structure of an optimal solution.
  2. Recursively define the value of an optimal solution.
  3. Compute the value of an optimal solution.
  4. Construct an optimal solution from computed information.

$S_1 = \texttt{ACCGGTCGAGTGCGCGGAAGCCGGCCGAA}$

$S_2 = \texttt{GTCGTTCGGAATGCCGTTGCTCTGTAAA}$

$\texttt{GTCGTCGGAAGCCGGCCGAA}$

问题1：dynamic programming (续)

- 你能说明求解longest common subsequence的四个步骤吗？
  1. Characterize the structure of an optimal solution.
  2. Recursively define the value of an optimal solution.
  3. Compute the value of an optimal solution.
  4. Construct an optimal solution from computed information.

$$c[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0\,, \\ c[i-1,j-1]+1 & \text{if } i,j > 0 \text{ and } x_i = y_j\,, \\ \max(c[i,j-1], c[i-1,j]) & \text{if } i,j > 0 \text{ and } x_i \neq y_j\,. \end{cases}$$

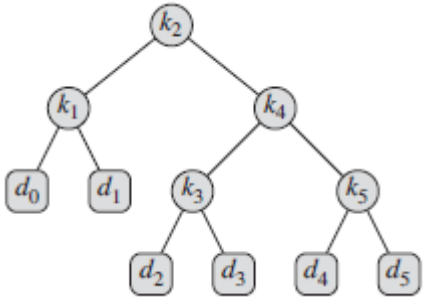
$S_1 =$ ACCGGTCGAGTGCGCGGAAGCCGGCCGAA

$S_2 =$ GTCGTTCGGAATGCCGTTGCTCTGTAAA

GTCGTCGGAAGCCGGCCGAA

问题1：dynamic programming (续)

- 你能说明求解optimal binary search trees的四个步骤吗？
  1. Characterize the structure of an optimal solution.
  2. Recursively define the value of an optimal solution.
  3. Compute the value of an optimal solution.
  4. Construct an optimal solution from computed information.

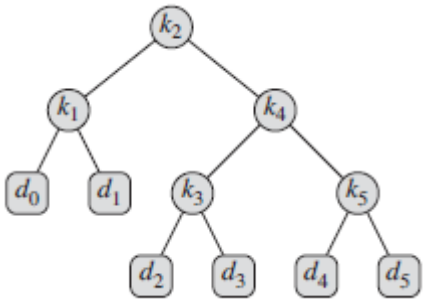


i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

问题1：dynamic programming (续)

- 你能说明求解optimal binary search trees的四个步骤吗？
  1. Characterize the structure of an optimal solution.
  2. Recursively define the value of an optimal solution.
  3. Compute the value of an optimal solution.
  4. Construct an optimal solution from computed information.

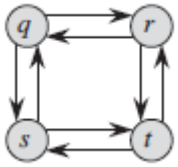
$$e[i,j] = \begin{cases} q_{i-1} & \text{if } j = i-1\,, \\ \min\limits_{i \le r \le j} \{e[i,r-1] + e[r+1,j] + w(i,j)\} & \text{if } i \le j\,. \end{cases}$$



i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

问题1： dynamic programming (续)

- unweighted longest simple path为什么不具有最优子结构？
- unweighted shortest simple path为什么不存在这个问题？



问题1：dynamic programming (续)

- 你能说明求解longest palindrome subsequence的四个步骤吗？
  1. Characterize the structure of an optimal solution.
  2. Recursively define the value of an optimal solution.
  3. Compute the value of an optimal solution.
  4. Construct an optimal solution from computed information.

A *palindrome* is a nonempty string over some alphabet that reads the same forward and backward. Examples of palindromes are all strings of length 1, `civic`, `racecar`, and `aibohphobia` (fear of palindromes).

Give an efficient algorithm to find the longest palindrome that is a subsequence of a given input string. For example, given the input `character`, your algorithm should return `carac`.

问题1：dynamic programming (续)

- 你能说明求解longest palindrome subsequence的四个步骤吗？
  1. Characterize the structure of an optimal solution.
  2. Recursively define the value of an optimal solution.
  3. Compute the value of an optimal solution.
  4. Construct an optimal solution from computed information.

```
longest(i,j)= j-i+1 if j-i<=0,
              2+longest(i+1,j-1) if x[i]==x[j]
              max(longest(i+1,j),longest(i,j-1)) otherwise
```

A *palindrome* is a nonempty string over some alphabet that reads the same forward and backward. Examples of palindromes are all strings of length 1, `civic`, `racecar`, and `aibohphobia` (fear of palindromes).

Give an efficient algorithm to find the longest palindrome that is a subsequence of a given input string. For example, given the input `character`, your algorithm should return `carac`.

问题1：dynamic programming (续)

- 你能说明求解edit distance的四个步骤吗？
  1. Characterize the structure of an optimal solution.
  2. Recursively define the value of an optimal solution.
  3. Compute the value of an optimal solution.
  4. Construct an optimal solution from computed information.

Insertion of a single symbol. If $a = uv$, then inserting the symbol x produces uxv. This can also be denoted $\varepsilon \to x$, using ε to denote the empty string.
Deletion of a single symbol changes uxv to uv $(x \to \varepsilon)$.
Substitution of a single symbol x for a symbol $y \neq x$ changes uxv to uyv $(x \to y)$.

The Levenshtein distance between "kitten" and "sitting" is 3. The minimal edit script that transforms the former into the latter is:

1. **k**itten → **s**itten (substitution of "s" for "k")
2. sitt**e**n → sitt**i**n (substitution of "i" for "e")
3. sittin → sitting (insertion of "g" at the end).

问题1：dynamic programming (续)

- 你能说明求解edit distance的四个步骤吗？
  1. Characterize the structure of an optimal solution.
  2. Recursively define the value of an optimal solution.
  3. Compute the value of an optimal solution.
  4. Construct an optimal solution from computed information.

$$d_{ij} = \begin{cases} d_{i-1,j-1} & \text{for } a_j = b_i \\ \min \begin{cases} d_{i-1,j} + w_{\text{del}}(b_i) \\ d_{i,j-1} + w_{\text{ins}}(a_j) \\ d_{i-1,j-1} + w_{\text{sub}}(a_j, b_i) \end{cases} & \text{for } a_j \neq b_i \end{cases} \qquad \text{for } 1 \leq i \leq m, 1 \leq j \leq n.$$

Insertion of a single symbol. If $a = uv$, then inserting the symbol x produces uxv. This can also be denoted $\varepsilon \to x$, using ε to denote the empty string.
Deletion of a single symbol changes uxv to uv $(x \to \varepsilon)$.
Substitution of a single symbol x for a symbol $y \neq x$ changes uxv to uyv $(x \to y)$.

The Levenshtein distance between "kitten" and "sitting" is 3. The minimal edit script that transforms the former into the latter is:

1. **k**itten → **s**itten (substitution of "s" for "k")
2. sitt**e**n → sitt**i**n (substitution of "i" for "e")
3. sittin → sitting (insertion of "g" at the end).

问题2：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2：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2：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？

问题2： 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2：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2： greedy algorithms (续)

- 建议阅读打星号的16.4节