- 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

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3

- 注意：链表无法进行二分查找

TC第11.2节练习5

- 鸽巢原理

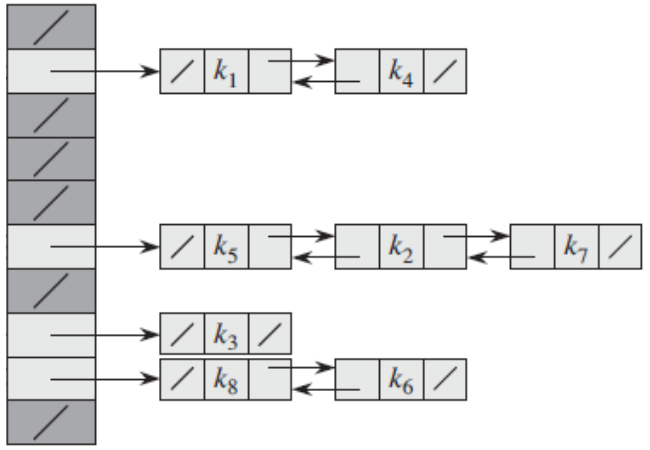
TC第11.2节练习6

- 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

- 教材讨论
  - TC第15章

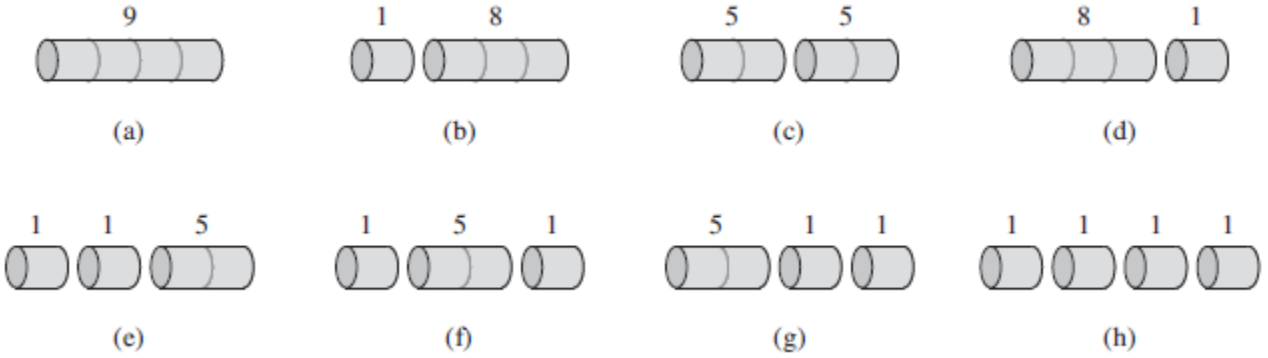
问题1：dynamic programming的基本概念

- 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.

- 广义上决定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2：dynamic programming的实例

- 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.



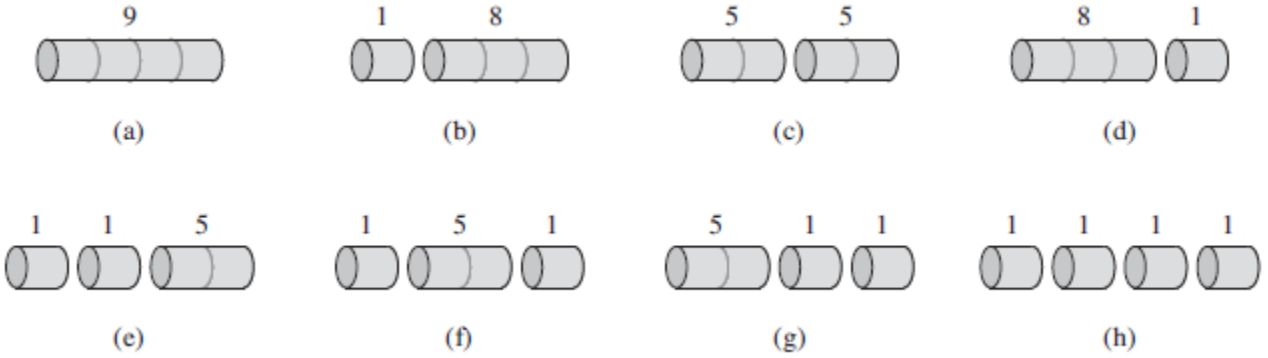
length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

问题2：dynamic programming的实例

- 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.

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

```
PRINT-CUT-ROD-SOLUTION(p, n)
1  (r, s) = EXTENDED-BOTTOM-UP-CUT-ROD(p, n)
2  while n > 0
3      print s[n]
4      n = n − s[n]
```



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2：dynamic programming的实例 (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2：dynamic programming的实例 (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.

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

```
PRINT-OPTIMAL-PARENS(s, i, j)
1  if i == j
2      print "A"_i
3  else print "("
4      PRINT-OPTIMAL-PARENS(s, i, s[i, j])
5      PRINT-OPTIMAL-PARENS(s, s[i, j] + 1, j)
6      print ")"
```

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

问题2：dynamic programming的实例 (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.

$S_1 = \texttt{ACCGGTCGAGTGCGCGGAAGCCGGCCGAA}$

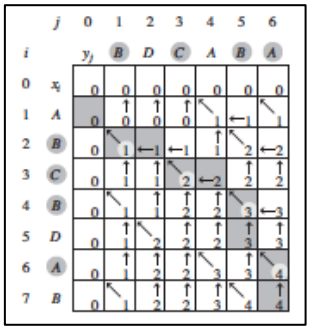
$S_2 = \texttt{GTCGTTCGGAATGCCGTTGCTCTGTAAA}$

$\texttt{GTCGTCGGAAGCCGGCCGAA}$

问题2：dynamic programming的实例 (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.

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



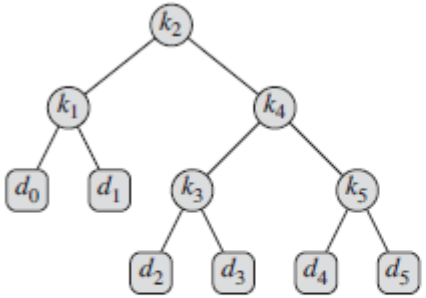
S_1 = ACCGGTCGAGTGCGCGGAAGCCGGCCGAA

S_2 = GTCGTTCGGAATGCCGTTGCTCTGTAAA

GTCGTCGGAAGCCGGCCGAA

问题2：dynamic programming的实例 (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.

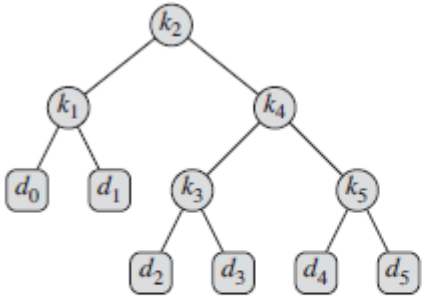


i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

问题2：dynamic programming的实例 (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.

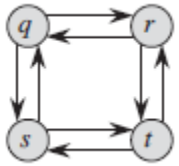
$$e[i,j] = \begin{cases} q_{i-1} & \text{if } j = i-1, \\ \min\limits_{i \le r \le j} \{e[i,r-1] + e[r+1,j] + w(i,j)\} & \text{if } i \le j. \end{cases}$$



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2：dynamic programming的实例 (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2：dynamic programming的实例 (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2：dynamic programming的实例 (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2：dynamic programming的实例 (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2：dynamic programming的实例 (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).