5-1 CLRS chap 11.
11.3-5.

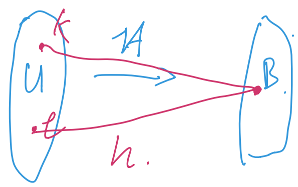
11.3-5 ★

Define a family $\mathcal{H}$ of hash functions from a finite set U to a finite set B to be ***ϵ-universal*** if for all pairs of distinct elements k and l in U,

$$\Pr\{h(k) = h(l)\} \leq \epsilon ,$$

where the probability is over the choice of the hash function h drawn at random from the family $\mathcal{H}$. Show that an ϵ-universal family of hash functions must have

$$\epsilon \geq \frac{1}{|B|} - \frac{1}{|U|} .$$



$\leq \varepsilon$.

网上有很多解释与证.

map $|U| \rightarrow |B|$

$E(e \text{ in } B \text{ from } \# n) = \frac{|U|}{|B|}$

for. k, $E(\frac{|U|}{|B|} - 1) \Rightarrow$ same.

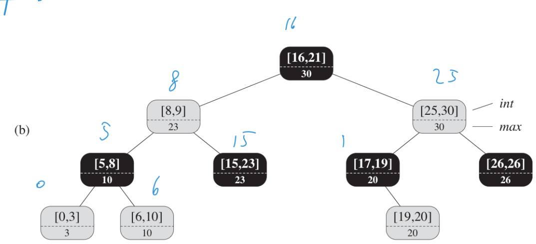
$\therefore$ 概 $h(k) = h(l) \geq \frac{\frac{|U|}{|B|} - 1}{|U|} = \frac{1}{|B|} - \frac{1}{|U|}$ (basic idea)

Exercise 5-3. Read CLRS, Chapter 14.

Exercise 5-4. Exercise 14.3-3.

14.3-3

Describe an efficient algorithm that, given an interval i, returns an interval overlapping i that has the minimum low endpoint, or $T.nil$ if no such interval exists.




```
SearchMin(T, i)
  x = T.root
  while x != nil
      if x.left != nil and x.left.max >= x.low
            x = x.left.
      else if . i interlap x,
            break
      else . x = x.right.

  return x.
```

Exercise 5-5. Read CLRS, Chapter 15. DP: optimization problem.

Exercise 5-6. Exercise 15.1-4.

Exercise 5-7. Exercise 15.2-4.

Exercise 5-8. Exercise 15.4-5.

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, typically in a bottom-up fashion.
4. Construct an optimal solution from computed information.

Bottom up: "reverse topologic sort" | top-down: DFS with memo.

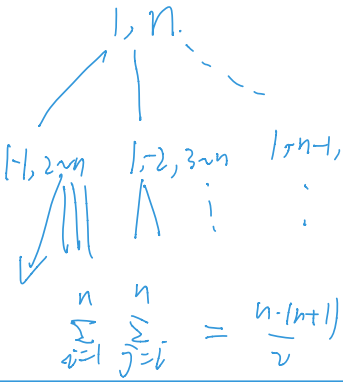
15.1-4

Modify MEMOIZED-CUT-ROD to return not only the value but the actual solution, too.

```
EX-MEMOIZED-CUT-ROD(p, n, r, s).
  if r[n] >= 0, return r[n]
  if n == 0
      q = 0.
  else q = -∞.
      for i = 1 to n
          if q < p[i] + EXTENDED-CUT-ROD(p, n-i, r, s).
              q = p[i] + EXTENDED-CUT-ROD(p, n-i, r, s).
              s[j] = i.
  r[n] = q.
  return q.
```

15.2-4

Describe the subproblem graph for matrix-chain multiplication with an input chain of length n. How many vertices does it have? How many edges does it have, and which edges are they?



$$\sum_{i}^{n}\sum_{j} R(i,j) = \frac{n^3-n}{3}$$?

15.4-5

Give an $O(n^2)$-time algorithm to find the longest monotonically increasing subsequence of a sequence of n numbers.

```
f(n, i, last)
  int re.
  if n[i] > last.
      re = max[f(n, i+1, n[i]) + 1, f(n, i+1, last)].
  else
      re = f(n, i+1, last).
  return re.
```

or. sort the n numbers A get A'.
solve LCS of A and A'
$\Theta(n \lg n) + \Theta(n^2) = O(n^2)$

Problem Set 5

This problem set is due **at 11:59pm** on **Friday, March 20, 2015.**

Please turn in your solution to all problems in a single pdf file. Your submitted solution should start with your name, the course number, your recitation section, the date, and the names of any students with whom you collaborated.

Exercise 5-1. Read CLRS, Chapter 11.

Exercise 5-2. Exercise 11.3-5.

Exercise 5-3. Read CLRS, Chapter 14.

Exercise 5-4. Exercise 14.3-3.

Exercise 5-5. Read CLRS, Chapter 15.

Exercise 5-6. Exercise 15.1-4.

Exercise 5-7. Exercise 15.2-4.

Exercise 5-8. Exercise 15.4-5.

Problem 5-1. New Operations for Skip Lists [25 points]

This problem will demonstrate that skip lists, and some augmentations of skip lists, can answer some queries about "nearby" elements efficiently. In a dynamic-set data structure, the query FINGER-SEARCH(x, k) is given a node x in the data structure and a key k, and it must return a node y in the data structure that contains key k. (You may assume that such a node y in fact appears in the data structure.) The goal is for FINGER-SEARCH to run faster when nodes x and y are nearby in the data structure.

(a) [12 points] Write pseudocode for FINGER-SEARCH(x, k) for skip lists. Assume all keys in the skip list are distinct. Assume that the given node x is in the level-0 list, and the operation should return a node y that stores k in the level-0 list.

Your algorithm should run in $O(\lg m)$ steps with high probability, where $m = 1 + |\text{rank}(x.key) - \text{rank}(k)|$. Here, $\text{rank}(k)$ refers to the rank (index) of a key k in the sorted order of the dynamic set (when the procedure is invoked). "High probability" here is with respect to m; more precisely, for any positive integer α, your algorithm

5-1 finger-search (x, k):

if x.value = k, return go down x to level 0.

else.

x = x.up.

if x.right.value < k.

elesif x.left.value > k.

finger-search (x, k) // k > key(x)

z = x

while:

if z.up ≠ nil and z.up.next.key ≤ k

z = z.up

else if z.next.next.key ≤ k.

z = z.next

else: break.

while

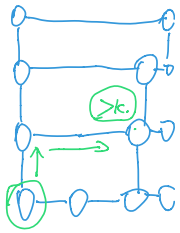
if z.next.key ≤ k.

z = z.next

elesif z.level ≠ 0.

z.= z.down

else return z.



y在x右边.

对称for 左边.

5-1. b. rank(x) + r.

count(x). the #elements in level 0

that have keys k with key(x) < k ≤ x.next.key

5-1. c.

it maintains the invariant: count[z] ≤ rem.

Rank-Search (x, r)

z = x

rem = r.

while:

if z.up ≠ NIL and z.up.count ≤ rem.

z = z.up.

else if. z.count + z.next.count ≤ rem

rem = rem - z.count.

z = z.next

else break. // z.count ≤ rem < z.count + z.next.count.

(2). while.

if z.count ≤ rem.

rem = rem - z.count

z = z.next

elesif z.level ≠ 0.

z = z.down

else return z.

should run in $O(\lg m)$ time with probability at least $1 - \frac{1}{m^\alpha}$. (The constant implicit in this O may depend on α.) Analyze your algorithm carefully.

In writing your code, you may assume that the implementation of skip lists stores both a $-\infty$ at the front of each level as well as a $+\infty$ as the last element on each level.

Another query on a dynamic set is RANK-SEARCH(x, r): given a node x in the data structure and positive integer r, return a node y in the data structure that contains the key whose rank is $\text{rank}(x) + r$. You may assume that such a node appears in the data structure. Rank search can be implemented efficiently in skip lists, but this requires augmenting the nodes of the skip list with new information.

(b) [6 points] Define a way of augmenting skip lists that will support efficient rank search, and show that your augmentation does not increase the usual high-probability order-of-magnitude time bounds for the SEARCH, INSERT, and DELETE operations.

(c) [7 points] Now write pseudocode for RANK-SEARCH(x, r) for skip lists. Again assume that all keys in the skip list are distinct. Assume that the given node x is in the level-0 list, and the operation should return a node y in the level-0 list.

Your algorithm should run in $O(\lg m)$ steps with high probability with respect to $m = r+1$. More precisely, for any positive integer α, your algorithm should run in $O(\lg m)$ time with probability at least $1 - \frac{1}{m^\alpha}$. Analyze your algorithm carefully.

Problem 5-2. Choosing Prizes [25 points]

In this problem, you are presented with a collection of n ***prizes*** from which you are allowed to select at most m prizes, where $m < n$. Each prize p has a nonnegative integer ***value***, denoted $p.value$. Your objective is to maximize the total value of your chosen prizes.

The problem has several variations, described in parts (a)–(d) below. In each case, you should give an efficient algorithm to solve the problem, and analyze your algorithm's time and space requirements.

In parts (a)–(c), the prizes are presented to you as a sequence $P = \langle p_1, p_2, \ldots, p_n \rangle$, and your algorithm must output a *subsequence* S of P. In other words, the selected prizes S ($|S| = m$) must be listed in the same order as they are in P.

(a) [4 points] Give an algorithm that returns a subsequence $S = \langle s_1, s_2, \ldots \rangle$ of P of length at most m, for which $\sum_j s_j.value$ is maximum. Analyze your algorithm in terms of n and m.

(b) [7 points] Now suppose there are two types of prizes, type A and type B. Each prize's type is given as an attribute $p.type$. Give an algorithm that returns a subsequence $S = \langle s_1, s_2, \ldots \rangle$ of P of length at most m, for which $\sum_j s_j.value$ is maximum, subject to the new constraint that, in S, *all the prizes of type A must precede all the prizes of type B*. Analyze your algorithm in terms of n and m.

a). output the mth largest value in n number. $(n-m+1)$

select m. $\Theta(n)$. $\Theta(1)$

(c) [7 points] As in part (a), there is only one type of prize. Give an algorithm that returns a subsequence $S = \langle s_1, s_2, \ldots \rangle$ of P of length at most m, for which $\sum_j s_j.value$ is maximum, subject to the new constraint that, in S, *the values of the prizes must form a non-decreasing sequence*. Analyze your algorithm in terms of n and m.

In part (d), the prizes are represented by a *rooted binary tree* T, with root vertex r, where each vertex u has an associated prize, $u.prize$. Let P be the set of prizes in the tree. As before, each prize p has a nonnegative integer attribute $p.value$.

(d) [7 points] Give an algorithm that returns a set S of at most m prizes for which $\sum_{s \in S} s.value$ is maximum, subject to the new constraint that, for any $s \in S$ that is associated with a non-root node u of T, the prize at node $u.parent$ is also in S. (This implies that the selected prizes must be associated with nodes that form a connected subtree of T rooted at r.)

c. $C(i, last, k)$

if $P_i.value \geq last$.

$C(i, last, k) = \max(C(i+1, P_i.value, k-1)$ 和 $C(i+1, last, k))$

else.

$C(i, last, k) = C(i+1, last, k)$.

$(n+1)(m+1)$ 个 subproblem. n. n. m

$\Theta(n^2 \cdot m)$., space $\Theta(n \cdot n^2)$.

---

d. $C(u, k) = \max_{0 \leq j \leq k-1} (u.prize.value + C(u.left, j) + C(u.right, k-1-j))$.

这左右在分配 j., max.

n 个 node, 每个 node m 种可能 (分配分数 select).

$n \cdot m$ 个 subprob, solve one $\Theta(m)$

$\Rightarrow \Theta(nm^2)$