- 作业讲解
  - TC第15.1节练习1、3
  - TC第15.2节练习2、4
  - TC第15.3节练习3、5、6
  - TC第15.4节练习3、5
  - TC第15.5节练习1
  - TC第15章问题4

TC第15.1节练习1

- 要重视数学归纳法的结构：
  - n=0时……
  - 假设n≤k时成立，则n=k+1时……

TC第15.2节练习2

- Give a recursive algorithm that actually performs the optimal matrix-chain multiplication.

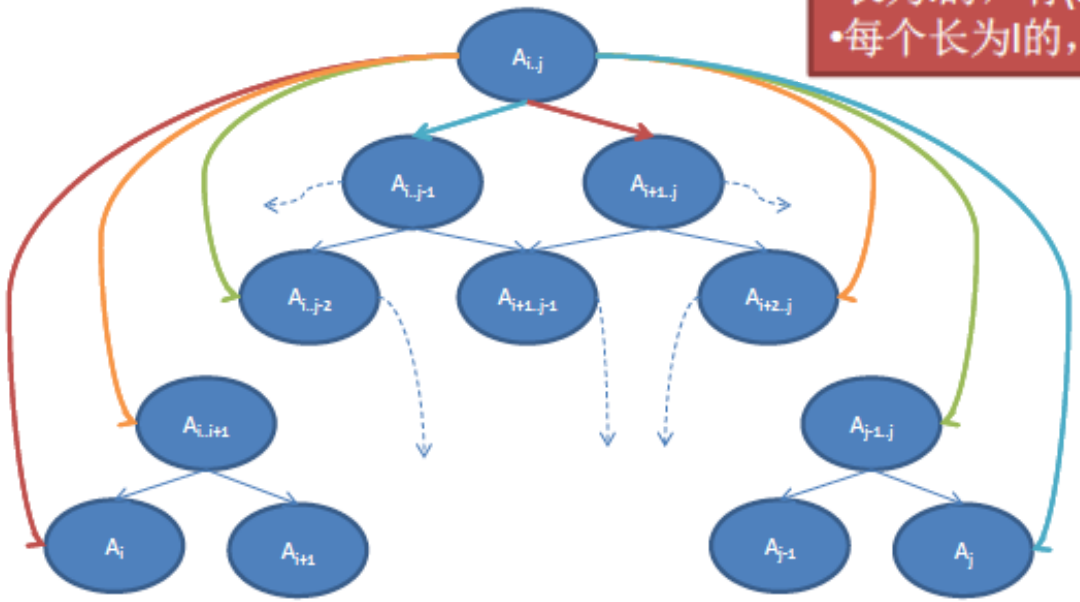
```
MATRIX-CHAIN-MULTIPLY (A, s, i, j) {
   if (i==j)
      return Ai;
   else if ((i+1)==j)
      return Ai x Aj; //这步需要吗?
   else
      return MATRIX-CHAIN-MULTIPLY(A, s, i, s[i, j]) x MATRIX-CHAIN-MULTIPLY(A, s, s[i, j]+1, j);
}
```

TC第15.2节练习4

- Subproblem graph
  - Vertex: Subproblem.
  - Edge (x, y): Need a solution to y when solving x.

= n+…+1 = n(n+1)/2

= ∑l(n+1-l)2(l-1)
•长为l的，有(n+1-l)个
•每个长为l的，出度为2(l-1)



TC第15.3节练习6

- A sequence of trades may entail a commission, which depends on the number of trades you make.
  - c_k：总共兑换k次的费用

TC第15.4节练习5

- 方法1
  - $m(i) = \max_{1 \le j < i}\{m[j]\}+1,\ \text{s.t. } x[j] \le x[i]$
- 方法2
  - Y ← X;
  - SORT(Y);
  - LCS-LENGTH(X, Y);

1	5	2	4	9	7	8	5

TC第15章问题4

- Printing neatly
  - 子问题和最优子结构是什么？

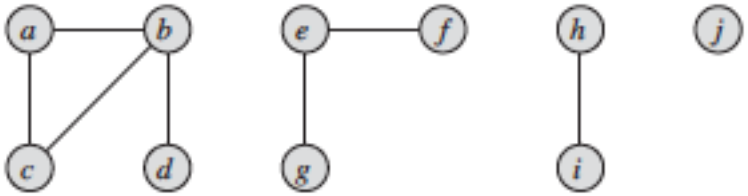
Aaa bbb aaa bbb Aaa bbb aaa bbb Aaa bbb aaa bbb a
bbb aaa bbb bbb aaa bbb Aaa bbb aaa bbb Aaa bb aaa
bbb Aaa bbb aaa bbb Aaa bbb aaa bbb Aaa bb aaa bb
Aaa bbb aaa bbb Aaa bbb aaa bbb Aaa bbb aaa bbb Aaa
bbb aaa bbb Aaa bbb aaa bbb Aaa bbb aaa bbb aa bbb
aaa bbb Aaa bbb aaa bbb Aaa bbb aaa bbb Aaa bbb aaa
bbb Aaa bbb aaa bbb

- 教材讨论
  - TC第21章
  - SB第2章

问题1：connected components

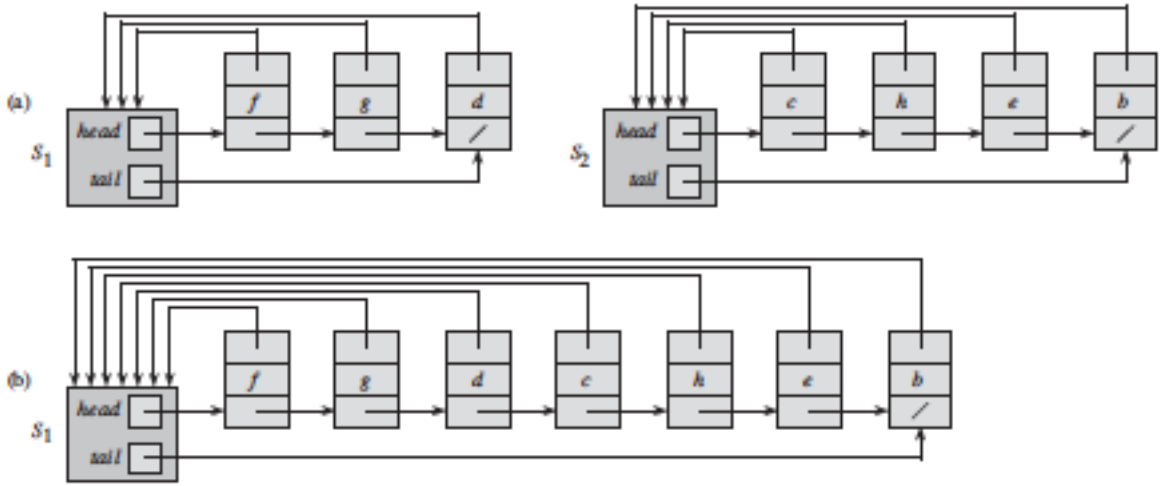
- 你能否借助这个例子，解释Union-Find的三个操作？
  - MAKE-SET(x)
  - UNION(x,y)
  - FIND-SET(x)
- 你会证明CONNECTED-COMPONENTS的正确性吗？

```
CONNECTED-COMPONENTS(G)
1  for each vertex v ∈ G.V
2      MAKE-SET(v)
3  for each edge (u, v) ∈ G.E
4      if FIND-SET(u) ≠ FIND-SET(v)
5          UNION(u, v)
```



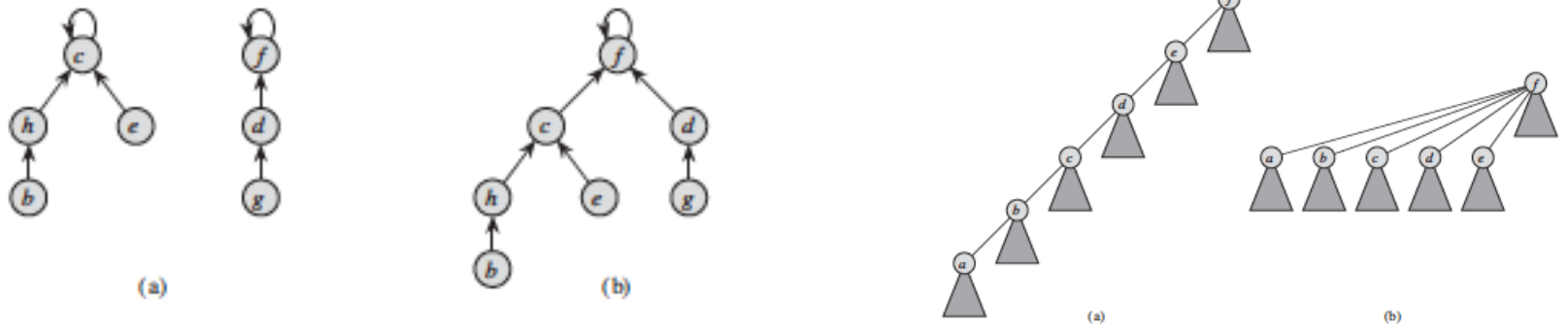
问题2：linked-list representation

- 请简要解释这种实现方式
  - Union-Find的三个操作如何实现？
  - 运行时间分别是？其中的瓶颈是？
    - 如何利用weighted-union heuristic加以改进？
    - 能改进到O(1)吗？（需要付出什么其它代价？）
  - head和tail能不能省掉一个？（不改变运行时间）



问题3：disjoint-set forests

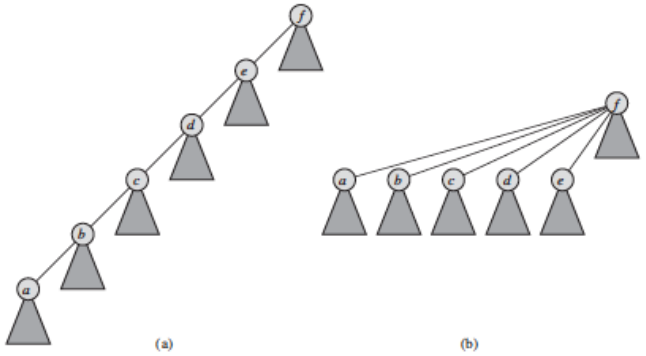
- 请简要解释这种实现方式
  - Union-Find的三个操作如何实现？
  - 运行时间分别是？其中的瓶颈是？
    - 如何利用union by rank加以改进？
    - 如何利用path compression加以改进？



问题3： disjoint-set forests (续)

- 你能解释FIND-SET的原理吗？
- 你会证明FIND-SET的正确性吗？

```
FIND-SET(x)
1  if x ≠ x.p
2        x.p = FIND-SET(x.p)
3  return x.p
```



问题4：how many tables

- Today is Ignatius' birthday. He invites a lot of friends. Now it's dinner time. Ignatius wants to know how many tables he needs at least. You have to notice that not all the friends know each other, and all the friends do not want to stay with strangers.

问题5：the suspects

- Severe acute respiratory syndrome (SARS), an atypical pneumonia of unknown etiology, was recognized as a global threat in mid-March 2003. To minimize transmission to others, the best strategy is to separate the suspects from others.
- In the Not-Spreading-Your-Sickness University (NSYSU), there are many student groups. Students in the same group intercommunicate with each other frequently, and a student may join several groups. To prevent the possible transmissions of SARS, the NSYSU collects the member lists of all student groups, and makes the following rule in their standard operation procedure (SOP).
  - Once a member in a group is a suspect, all members in the group are suspects.
- However, they find that it is not easy to identify all the suspects when a student is recognized as a suspect. Your job is to write a program which finds all the suspects.

问题6：partition refinement

- In the design of algorithms, **partition refinement** is a technique for representing a partition of a set as a data structure that allows the partition to be refined by splitting its sets into a larger number of smaller sets. In that sense it is **dual** to the union-find data structure, which also maintains a partition into disjoint sets but in which the operations merge pairs of sets together.
- 请给出partition refinement的一种实现，并分析refinement的运行时间。
- 利用你的实现，解决distinguishing prefixes问题：
  Given a set of strings *S*, the distinguishing prefix of a string *s* in *S* is the shortest prefix of *s* that is a prefix of no other string in *S*. Determine the distinguishing prefixes of all the strings in *S*.