week9

递归 - 函数自己调用自己

```
public void recur(int level, int param) {

    // terminator
    if (level > MAX_LEVEL) {
      // process result
      return;
    }

    // process current logic
    process(level, param);

    // drill down
    recur(level: level + 1, newParam);

    // restore current status

}
```

1)

思维

1. 人肉递归低效、很累

2. 找到最近最简方法，将其拆解成可重复解决的问题

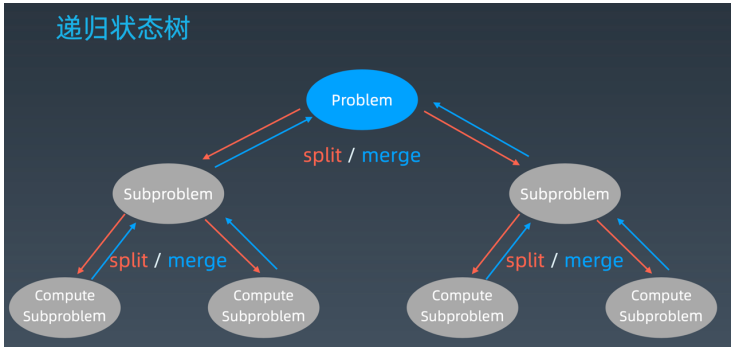
3. 数学归纳法思维

本质：寻找重复性 —> 计算机指令集

2)

3)

递归状态树



4)

关键点

动态规划 和 递归或者分治 没有根本上的区别（关键看有无最优的子结构）

拥有共性：找到重复子问题

差异性：最优子结构、中途可以淘汰次优解

遍历字符串

- Python:

```
for ch in “abbc”:
  print(ch)
```

- Java:

```
String x = “abbc”;
for (int i = 0; i < x.size(); ++i) {
  char ch = x.charAt(i);
}
for ch in x.toCharArray() {
  System.out.println(ch);
}
```

- C++:

```
string x(“abbc”);
for (int i = 0; i < s1.length(); i++) {
  cout << x[i];
}
```

5)

6)

字符串匹配算法

1. 暴力法（brute force） - O(mn)

2. Rabin-Karp 算法

3. KMP 算法

- 课后了解：
  Boyer-Moore 算法：https://www.ruanyifeng.com/blog/2013/05/boyer-moore_string_search_algorithm.html
  Sunday 算法：https://blog.csdn.net/u012505432/article/details/52210975

7)

Rabin-Karp 算法

在朴素算法中，我们需要挨个比较所有字符，才知道目标字符串中是否包含子串。那么， 是否有别的方法可以用来判断目标字符串是否包含子串呢?

答案是肯定的，确实存在一种更快的方法。为了避免挨个字符对目标字符串和子串进行比较， 我们可以尝试一次性判断两者是否相等。因此，我们需要一个好的哈希函数（hash function）。 通过哈希函数，我们可以算出子串的哈希值，然后将它和目标字符串中的子串的哈希值进行比较。 这个新方法在速度上比暴力法有显著提升。

Rabin-Karp 算法

Rabin-Karp 算法的思想：

1. 假设子串的长度为 M (pat)，目标字符串的长度为 N (txt)
2. 计算子串的 hash 值 hash_pat
3. 计算目标字符串txt中每个长度为 M 的子串的 hash 值（共需要计算 N-M+1 次）
4. 比较 hash 值：如果 hash 值不同，字符串必然不匹配; 如果 hash 值相同，还需要使用朴素算法再次判断

8)

KMP 算法

KMP算法（Knuth-Morris-Pratt）的思想就是，当子串与目标字符串不匹配时，其实你已经知道了前面已经匹配成功那 一部分的字符（包括子串与目标字符串）。以阮一峰的文章为例，当空格与 D 不匹配时，你其实 知道前面六个字符是“ABCDAB”。KMP 算法的想法是，设法利用这个已知信息，不要把“搜索位置” 移回已经比较过的位置，继续把它向后移，这样就提高了效率。

https://www.bilibili.com/video/av11866460?from=search&seid=17425875345653862171

http://www.ruanyifeng.com/blog/2013/05/Knuth%E2%80%93Morris%E2%80%93Pratt_algorithm.html