递归2

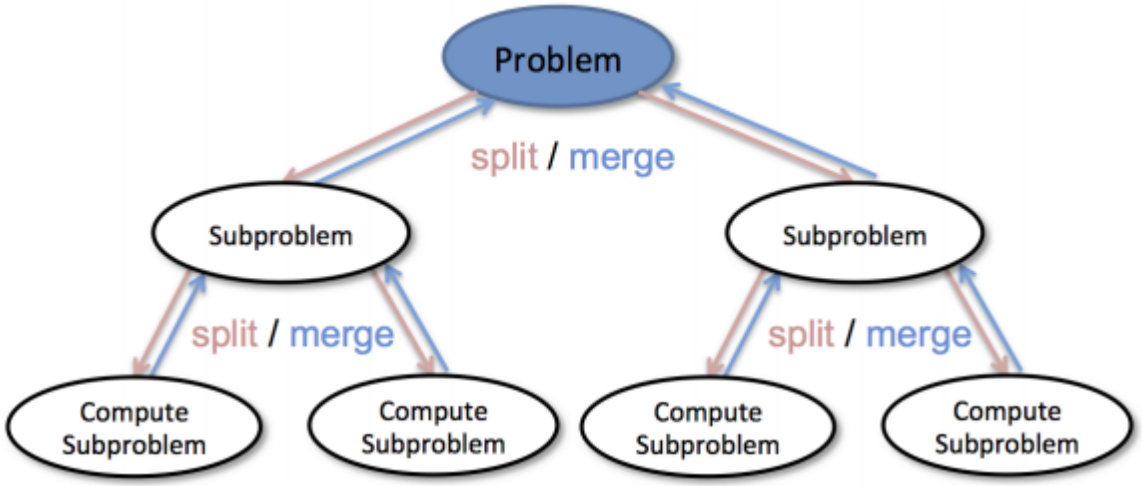
Divide-and-conquer strategy and recursion
分治策略与递归

divide-and-conquer strategy
分治策略

Typical strategy to solve a problem: divide and conque
解决问题的典型策略：分而治之

The problem is divided into several smaller-scale parts, by solving each small-scale part problem and summarizing the results to obtain the solution of the original problem

将问题分为若干更小规模的部分，通过解决每一个小规模部分问题，并将结果汇总得到原问题的解



Recursive algorithm and divide-and-conquer strategy
递归算法与分治策略

Three laws of recursion:
递归三定律：

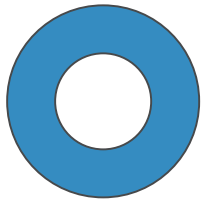
- Basic end conditions, to solve the minimum scale problem
  基本结束条件，解决最小规模问题
- Reduce the scale, to the basic end conditions evolution
  缩小规模，向基本结束条件演进
- Call itself to solve the same scaled-down problem
  调用自身来解决已缩小规模的相同问题

It reflects the strategy of divide-and-conquer
体现了分治策略

- Problem solving relies on several downsizing problems
  问题解决依赖于若干缩小了规模的问题
- Summarize the solution of the original problem
  汇总得到原问题的解

This application is quite widespread
应用相当广泛

- Sort, find, traverse, evaluate, etc
  排序、查找、遍历、求值等等



Optimize problems and greedy strategies
优化问题和贪心策略

optimization problem
优化问题

Many algorithms in computer science are designed to find the optimal solutions to optimal problems

计算机科学中许多算法都是为了找到某些问题的最优解

For example, the shortest path between two points;

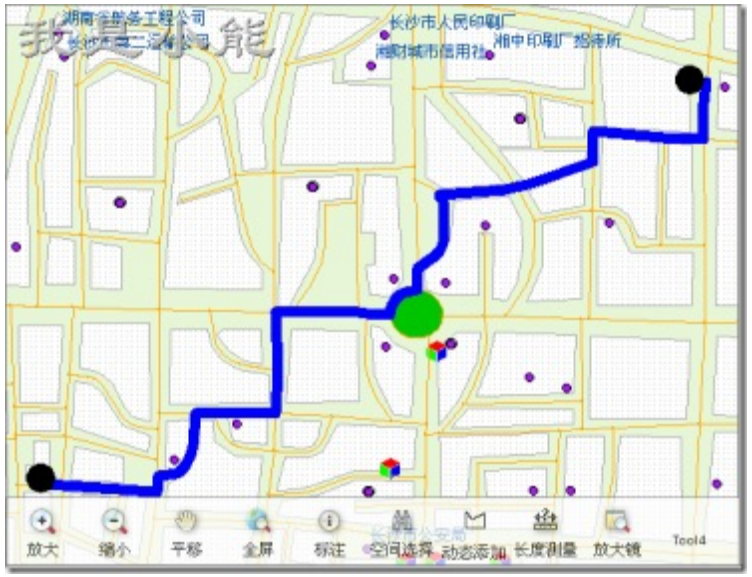
例如，两个点之间的最短路径；

A straight line that best matches a series of points;

能最好匹配一系列点的直线；

Or a minimum set that satisfies certain conditions

或者满足一定条件的最小集合



Exchange problem
找零兑换问题

A classic case is the problem of exchanging the least number of coins
一个经典案例是兑换最少个数的硬币问题

Suppose you program a vending machine manufacturer, and the vending machine has to give the customer a minimum number of coins each time;
假设你为一家自动售货机厂家编程序，自动售货机要每次找给顾客最少数量硬币；

If a customer puts in a $1 and buys 37¢, aiming to exchange for ¢63, the minimum number is: 2 quarter (¢25), 1 dime (¢10), and 3 penny (¢1), a total of 6
假设某次顾客投进$1纸币，买了¢37的东西，要找¢63，那么最少数量就是：2个quarter(¢25)、 1个dime(¢10)和3个penny(¢1)，一共6个

Greedy strategy to solve the problem of exchange
贪心策略解决找零兑换问题

People will use a variety of strategies to solve these problems, such as the most intuitive "greedy strategy". Generally we do so:

人们会采用各种策略来解决这些问题，例如最直观的“贪心策略”一般我们这么做：

Start with the largest denomination, with as much balance as possible, then to the next largest denomination, and with as much coins as possible, until penny (¢1)

从最大面值的硬币开始，用尽量多的数量有余额的，再到下一最大面值的硬币，还用尽量多的数量，一直到penny(¢1)为止

Greedy Strategy Greedy Method
贪心策略GreedyMethod

Greedy strategy
贪心策略

Because every time we try to solve the largest part of the problem corresponding to the coin exchange problem, which is to quickly reduce the zero face value using the largest number of coins each time

因为我们每次都试图解决问题的尽量大的一部分对应到兑换硬币问题，就是每次以最多数量的最大面值硬币来迅速减少找零面值

The "greedy strategy" solves the exchange problem and performs well under the coin system of the dollar or other currencies

“贪心策略”解决找零兑换问题，在美元或其他货币的硬币体系下表现尚好

Greedy strategy fails
贪心策略失效

But if your boss decides to export the vending machine to Elbonia, things get a bit complicated

但如果你的老板决定把自动售货机出口到Elbonia，事情就会有点复杂

(Series *Dilbert* fabricated version of the country) Because this strange country in addition to three face values above, there is a coin of [¢21]!

(系列漫画Dilbert里杜撰的国家)因为这个古怪的国家除了上面3种面值之外，还有一种【¢21】的硬币！

Greedy strategy fails
贪心策略失效

According to the "greedy strategy", in Elboia, 63 is the original 6 coins

按照“贪心策略”，在Elbonia，¢63还是原来的6个硬币

¢63=¢25*2+¢10*1+¢1*3

But the optimal solution is three coins whose face value is ¢21!

但实际上最优解是3个面值¢21的硬币！

¢63=¢21*3

The "greedy strategy" has failed

“贪心策略”失效了

InflationinElbonia

PROBLEMSOLVED

Recrecursive solution to exchange problem

找零兑换问题的递归解法

Exchange problem: Recursive solution method

找零兑换问题：递归解法

We will find a way to definitely find the optimal solution

我们来找一种肯定能找到最优解的方法

Whether the greedy strategy is effective depends on the specific coin system

贪心策略是否有效依赖于具体的硬币体系

The first is to determine the basic end conditions. The simple and most direct situation of the coin exchange is that the face value of the change needs to be changed is exactly equal to a certain coin

首先是确定基本结束条件，兑换硬币这个问题最简单直接的情况就是，需要兑换的找零，其面值正好等于某种硬币

If the change is 25 points, the answer is 1 coin!

如找零25分，答案就是1个硬币！

exchange problem: Recursive solution method

找零兑换问题：递归解法

The second is to reduce the size of the problem. We should try each coin once, such as the dollar coin system:

其次是减小问题的规模，我们要对每种硬币尝试1次，例如美元硬币体系：

After the change minus 1 cent, seek the minimum number of exchange coins (recursive call itself);

找零减去1分(cent)后，求兑换硬币最少数量(递归调用自身)；

The change minus 5 cents (nikel)，exchange the minimum number of coins

找零减去5分(nikel)后，求兑换硬币最少数量

After the change minus 10 cents(dime), exchange the minimum number of coins

找零减去10分(dime)后，求兑换硬币最少数量

minus 25 cents (quarter)，exchange the minimum number of coins

找零减去25分(quarter)后，求兑换硬币最少数量

The smallest one of the above 4 items was selected.

上述4项中选择最小的一个。

$$numCoins = min \begin{cases} 1 + numCoins(originalamount - 1) \\ 1 + numCoins(originalamount - 5) \\ 1 + numCoins(originalamount - 10) \\ 1 + numCoins(originalamount - 25) \end{cases}$$

Exchange problem: recursive solution code
找零兑换问题：递归解法代码

```
def recMC(coinValueList,change):
    minCoins = change
    if change in coinValueList:
      return 1
    else:
       for i in [c for c in coinValueList if c <= change]:
          numCoins = 1 + recMC(coinValueList,change-i)
          if numCoins < minCoins:
             minCoins = numCoins
    return minCoins

print(recMC([1,5,10,25],63))
```

Minimum size, straight back
最小规模，直接返回

Call yourself
调用自身

Reduce scale: Select a minimum number of one coin denominations minus each time
减小规模：每次减去一种硬币面值挑选最小数量

Exchange problem: recursive solution method analysis

找零兑换问题：递归解法分析

Although the recursive solution method can solve the problem, but its biggest problem is: extreme!low!effect!

递归解法虽然能解决问题，但其最大的问题是：极！其！低！效！

For the issue of 63 cents, 67,716,925 recursion call is required!

对63分的兑换硬币问题，需要进行67,716,925次递归调用！

It took 40 seconds on my laptop to get the problem solved: 6 coins

在普通笔记本电脑上花费了40秒时间得到解：6个硬币

```
import time

print(time.clock())
print(recMC([1, 5, 10, 25], 63))
print(time.clock())
```

```
/Library/Frameworks/Python.framework
0.050163
6
40.899408
```

Exchange problem: recursive solution method analysis

找零兑换问题：递归解法分析

For 26 exchange coins, look at the recursive calling process (a fraction of 377 times of recursions)

以26分兑换硬币为例，看看递归调用过程(377次递归的一小部分)

One big secret that we find is that we repeat the calculations too much!

我们发现一个重大秘密，就是重复计算太多！

For example, when exchanging for 15 cents, it appeared 3 times! But it eventually goes with 52 recursive calls

例如找零15分的，出现了3次！而它最终解决还要52次递归调用

It is clear that the fatal disadvantage of this algorithm is repeated calculation

很明显，这个算法致命缺点是重复计算

chapter summary
本章小结

In this chapter we study several recursive algorithms and show that recursion is an effective technique for solving certain complex problems with self-similarity.

在本章我们研究了几种递归算法，表明了递归是解决某些具有自相似性的复杂问题的有效技术。

The "three laws" of recursive algorithm:

递归算法“三定律”：

①The recursive algorithm must have the basic end condition

递归算法必须具备基本结束条件

②The recursive algorithm must reduce the size, change the state, and evolve to the basic end condition

递归算法必须要减小规模，改变状态，向基本结束条件演进

③The recursive algorithm has to call itself

递归算法必须要调用自身

chapter summary
本章小结

- In some cases, recursion can replace iterative cycle.
- Recursion algorithm can naturally fit with the expression of the problem, but is not always the most appropriate algorithm.
- Sometimes recursive algorithm can trigger a huge amount of repeated calculation, "memory / function value cache" can be through additional storage space record intermediate calculation results to effectively reduce repeated calculation.
- If the optimal solution of a problem includes smaller optimal solution of the same problem, it can be solved by dynamic planning.

某些情况下，递归可以代替迭代循环，递归算法通常能够跟问题的表达自然契合，递归不总是最合适的算法，有时候递归算法会引发巨量的重复计算，“记忆化/函数值缓存”可以通过附加存储空间记录中间计算结果来有效减少重复计算