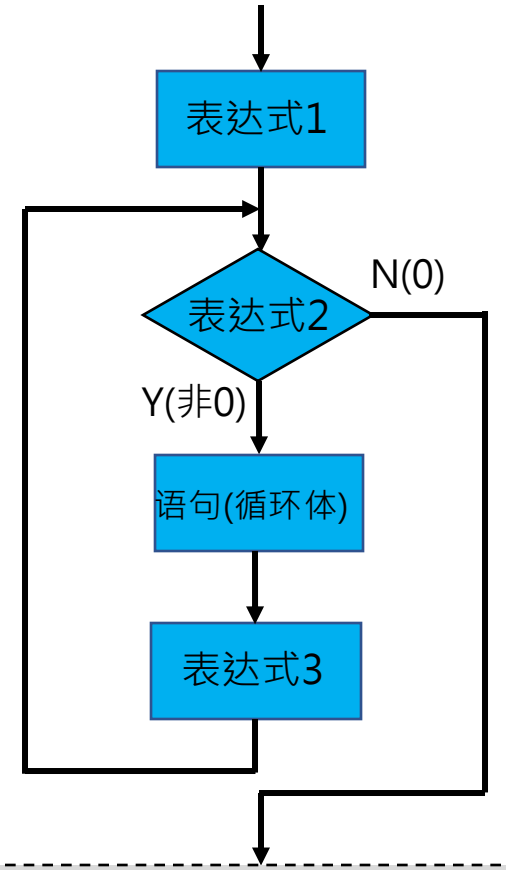
CS 1501 程序设计思想与方法 (C++)
Principles and Methodology in Programming
Chapter 4. 循环程序设计-2

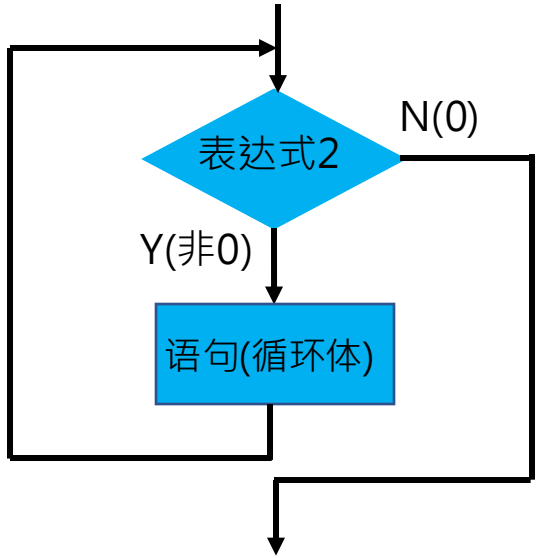
陈东尧
上海交通大学
2025年秋季

本章课程流程

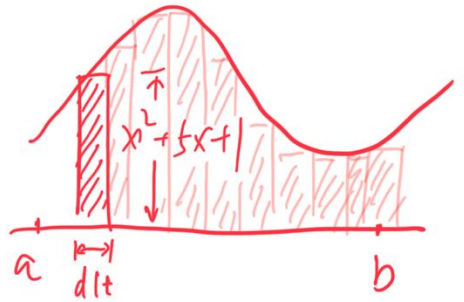
1. 重复 N 次循环
2. while 循环
3. do … while 循环
4. 循环的中途退出
5. 枚举法
6. 贪婪法

本章内容回顾





`for（初始化; 条件; 循环步长）`



计算定积分的实例

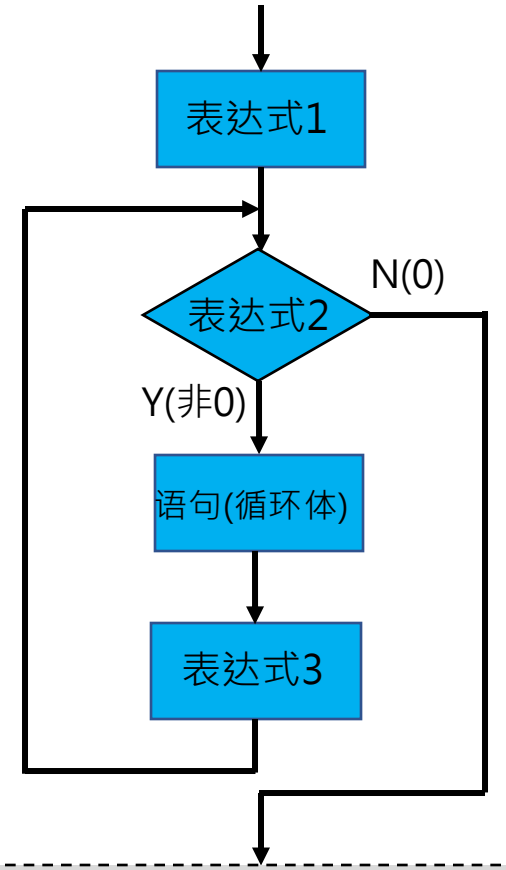
```
for(表达式1; 表达式2; 表达式3)
{
        语句;
}
```

```
while(表达式)
{
        语句;
}
```

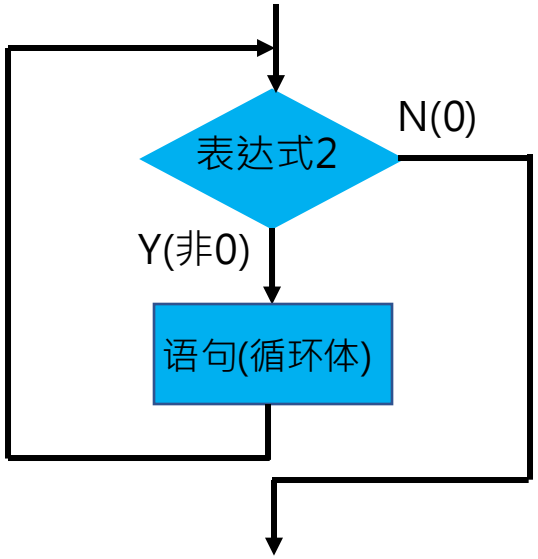
本章课程流程

1. 重复 N 次循环
2. while 循环
3. do ... while 循环
4. 循环的中途退出
5. 枚举法
6. 贪婪法

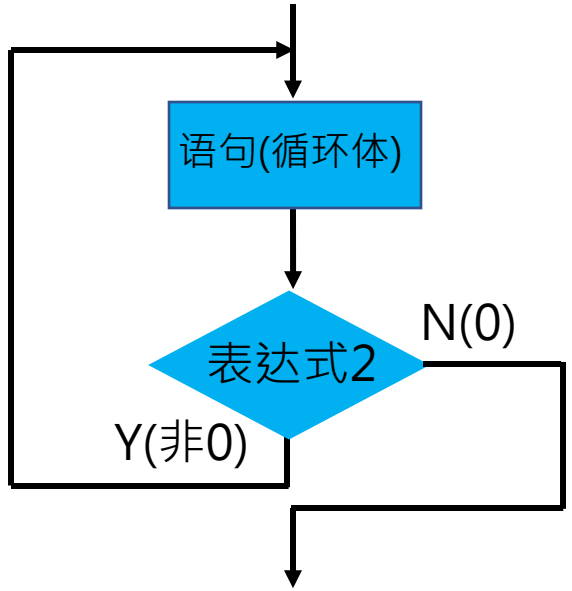
本章内容回顾




```
for(表达式1; 表达式2; 表达式3)
{
        语句;
}
```

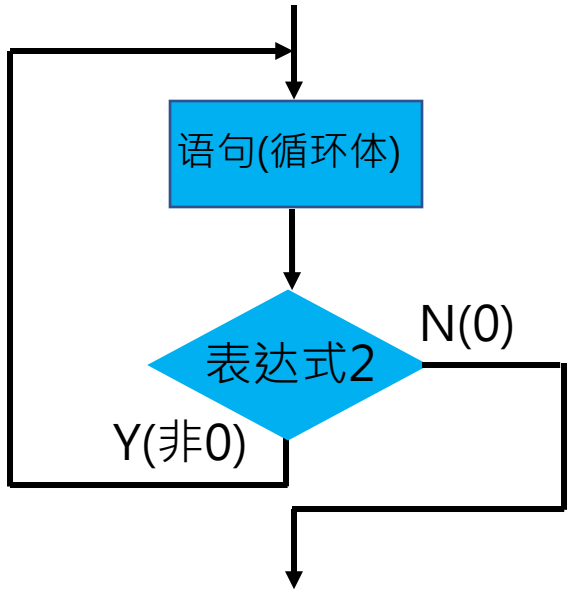



```
while(表达式)
{
        语句;
}
```




```
do
{
        语句;
} while(表达式);
```

do-while循环




```
do
{
        语句;
} while(表达式);
```

- 执行过程：它先执行循环体中的语句，然后再判断条件是否为真，如果为真则继续循环；如果为假，则终止循环。

do … while 循环语句

- 格式： do 语句 while (表达式)
- 执行过程：**先**执行循环体，**然后**判断循环条件。如条件成立，继续循环，直到条件为假

do … while 循环语句

- 格式： do 语句 while (表达式)
- 执行过程：**先**执行循环体，**然后**判断循环条件。如条件成立，继续循环，直到条件为假
- 如将若干个输入数相加，直到输入0为止。

```
total = 0;
do { cout << " ? ";
     cin >> value ;
   total += value;
} while (value != 0);
```

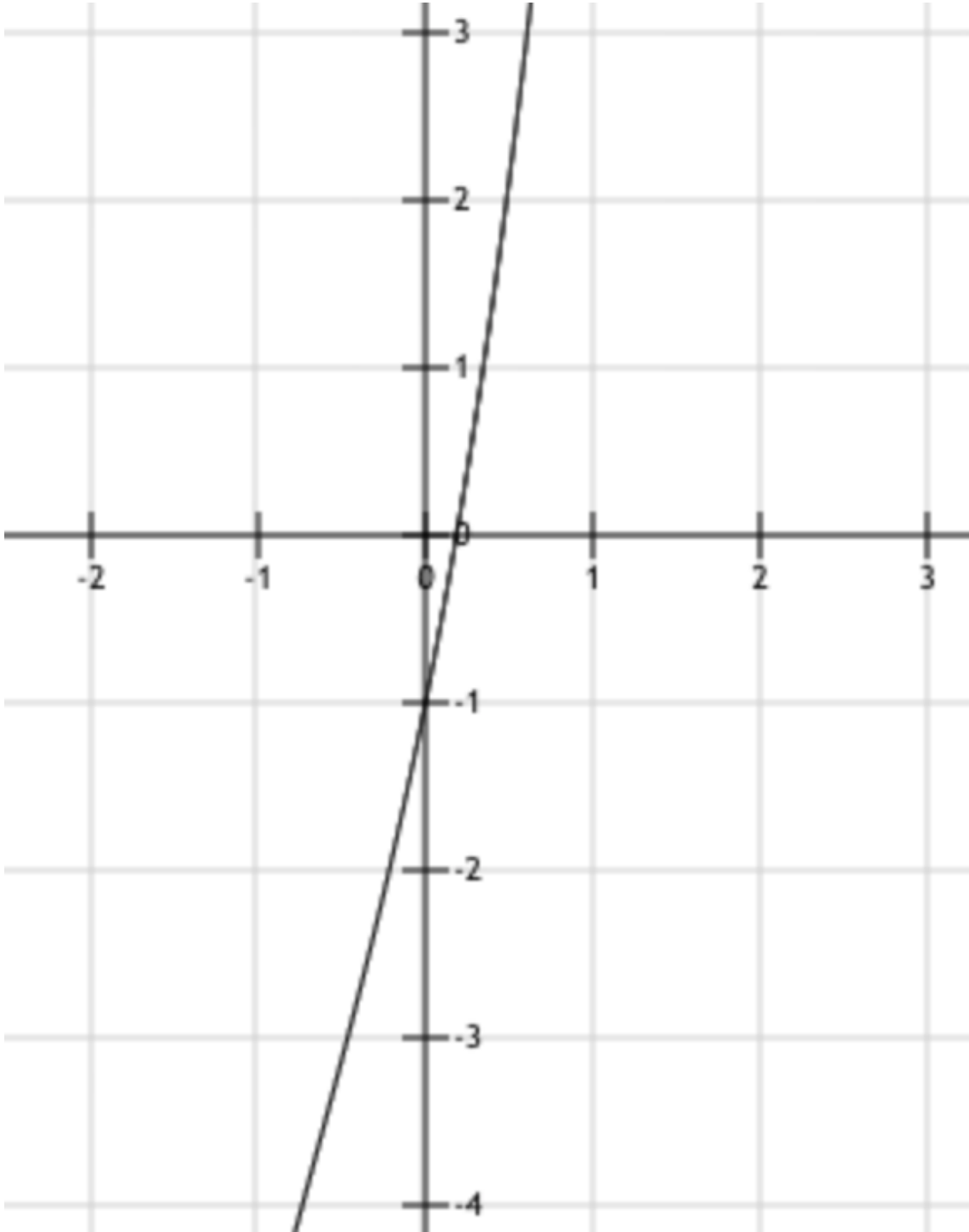
编程实例

- 计算方程 f(x)=0 在区间[a, b]之间的根 。

  前提：f(a)和f(b)必须异号

- 假设方程为 $x^3 + 2x^2 + 5x - 1 = 0$

  ， 区间为[-1, 1]



思路

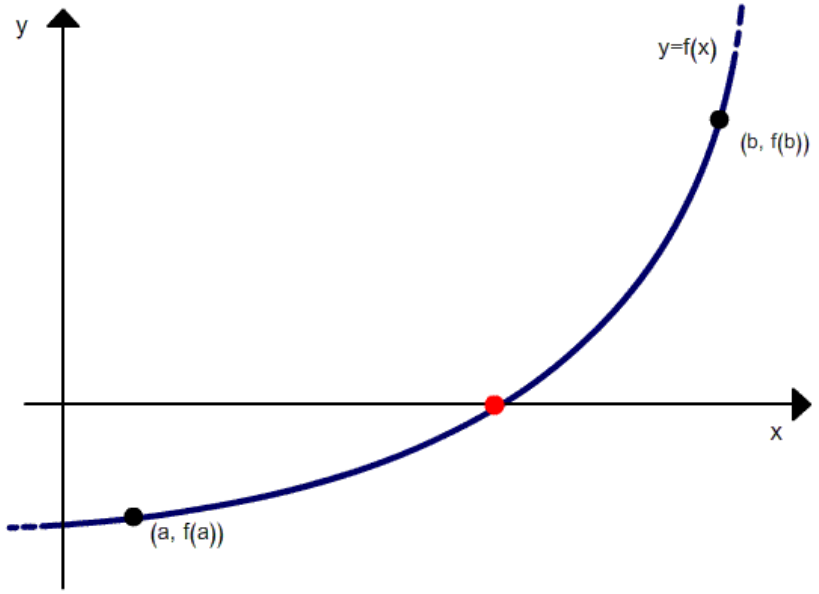
- 计算方程 f(x)=0 在区间[a, b]之间的根 。 前提：f(a)和f(b)必须异号
- 思考一种可以利用**循环**的解题方法

- 割线法（Secant method）

  割线法由以下的递推关系定义：

$$x_{n+1} = x_n - \frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})} f(x_n)$$

  从上式中可以看出，割线法需要两个初始值x_0和x_1，它们离函数的根越近越好。



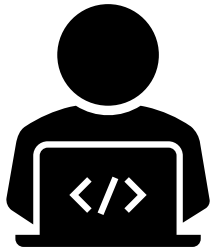
具体地

- 令x1 = a, x2 = b
- 连接(x1, f(x1))和(x2, f(x2))的弦交与x轴的坐标点可用如下公式求出：

$$x = \frac{x_1 f(x_2) - x_2 f(x_1)}{f(x_2) - f(x_1)}$$

- 若f(x)与f(x1)同符号，则方程的根在(x, x2)之间，将x作为新的x1。否则根在(x1, x)之间，将x设为新的x2。
- 重复步骤(2)和(3)，直到f(x)小于某个指定的精度为止。此时的x为方程f(x) = 0的根。

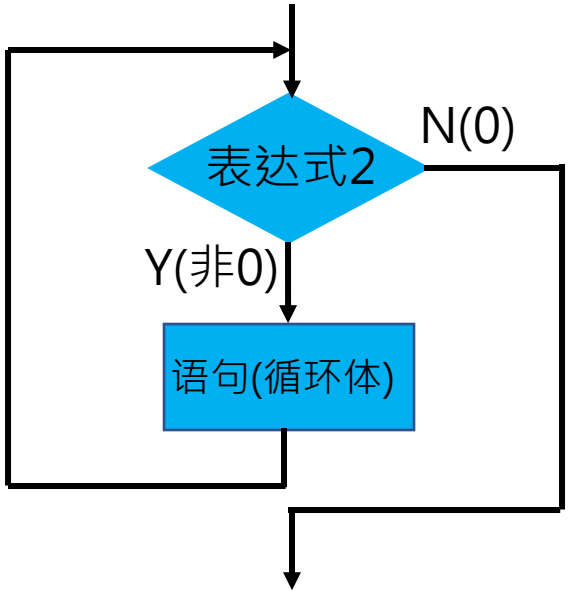
```cpp
#include <iostream>
#include <math.h>
using namespace std;
int main()
{
    double x, x1 = -1, x2 = 1, fx1, fx2, fx;
    int i = 0;
    do {
        cout<<i++<<endl;
        fx1 = x1 * x1 * x1 + 2 * x1 * x1 + 5 * x1 -1;
        fx2 = x2 * x2 * x2 + 2 * x2 * x2 + 5 * x2 -1;

        x = (x1 * fx2 - x2 * fx1) / (fx2 - fx1); // 切线法
        fx = x * x * x + 2 * x * x + 5 * x -1;

        if (fx * fx1 > 0) {
                x1 = x; }
        else {
                x2 = x;
        }
    } while (fabs( fx ) > 1e-7);
    cout << "方程的根为：  " << x << endl;
    return 0;
}
```

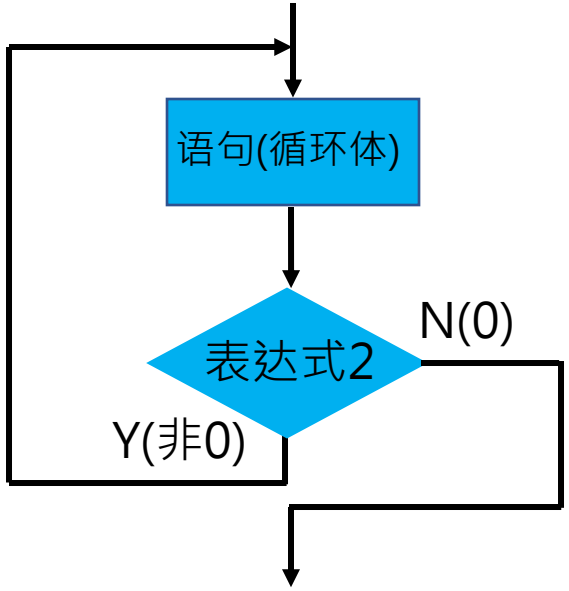


代码演示

chapter4-dowhile-1.cpp

While 和 do-while的区别





先执行循环体中的语句，然后再判断条件是否为真

当输入i为1和11时，输出分别是什么？

```cpp
#include <iostream>
using namespace std;

int main()
{
    int sum = 0;
    int i;
    cin>>i;
    do
    {
        sum += i;
        i++;
    }while(i <= 10);
    cout << "sum= " << sum;
    return 0;
}
```

```cpp
#include <iostream>
using namespace std;

int main()
{
    int sum = 0;
    int i;
    cin >> i;
    while(i <= 10)
    {
        sum += i;
        i++;
    }
    cout << "sum= " << sum;
    return 0;
}
```

本章课程流程

1. 重复 N 次循环
2. while 循环
3. do ... while 循环
4. 循环的中途退出
5. 枚举法
6. 贪婪法

循环的中途退出

- 考虑一个读入数据直到读到标志值就停止的问题。例如字母表“abcdefghijklmnopqrstuvwxyz”读到g就退出
- 该结构由以下步骤组成：
  - 读入一个值
  - 如果读入值与标志值相等，则退出循环
  - 执行在读入那个特定值情况下需要执行的语句

循环的中途退出

- 基于标志的循环结构被改为：

```
读入一个值
While （读入值与标志值不相等）
    {
        执行在读入那个特定值情况下需要执行的语句      //例如读到“g”则退出并存储已读数据
        读入一个值
    }
```

- 当一个循环中有一些操作必须在条件测试之前执行时，称为循环的中途退出问题。

循环的中途退出

- 基于标志的循环结构被改为：

  读入一个值

  While （读入值与标志值不相等）

  {

  执行在读入那个特定值情况下需要执行的语句 //例如读到“g”则退出并存储已读数据

  读入一个值

  }

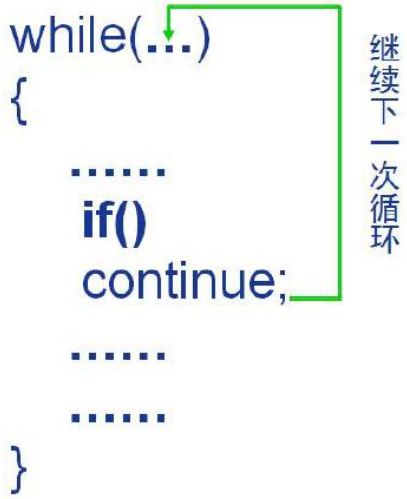
- 当一个循环中有一些操作必须在条件测试之前执行时，称为循环的中途退出问题。

用Switch语句重写百分制转换

score>=90	A
90>score>=80	B
80>score>=70	C
70>score>=60	D
score<60	E

```
switch( score/10 )
{
        case 10:
        case  9: cout << "A"; break;
        case  8: cout << "B"; break;
        case  7: cout << "C"; break;
        case  6: cout << "D"; break;
        default: cout << "E";
}
```

Break vs. Continue (跳出 vs. 继续)




```
while(x++ < 10){
    if(x == 3)
        break;

    cout << "x = " << x;
}
```

```
while(x++ < 10){
    if(x == 3)
        continue;

    cout << "x = " << x;
}
```

Break vs. Continue (跳出 vs. 继续)

- break 语句：跳出循环
- 上述问题可以用下列方案解决：

```
while (true) {
        读入一个值
        if（读入值==标志) break;
        根据数据作出处理
    }
```

vs

```
读入一个值
While （读入值与标志值不相等）
  {
     执行语句
     读入一个值
  }
```

- continue 语句：跳出当前循环周期

本章课程流程

1. 重复 N 次循环
2. while 循环
3. do ... while 循环
4. 循环的中途退出
5. 枚举法
6. 贪婪法

枚举法（Enumeration Method）

- 对所有可能的情况一种一种去尝试，直到找到正确的答案。
- 用**题目中给定的检验条件**判定**哪些是无用的，哪些是有用的**。能使命题成立，即为其解。

- 枚举法的实现基础是循环。
  - 为什么？
  - 循环用在哪里？

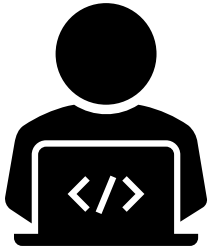
枚举法示例1

- 有个人有一千块钱，打算买一百只鸡。到市场一看，公鸡一只30元,母鸡一只50元,小鸡3只10元（3只不拆开卖）,试求用1000元买100只鸡,各为多少才合适?

【参考文章】https://blog.csdn.net/linj_m/article/details/17393031

枚举法示例1

- 根据题意我们可以得到方程组：
- $$30X + 50Y + 10Z/3 = 1000;$$
  $$X + Y + Z = 100;$$
  $$(100 > X, Y, Z > 0, Z\%3 == 0)$$
- 根据这两个公式，我们可以写出最为简单的代码，一一列举所有解进行枚举



代码演示

chapter4-chicken.cpp

```
for (x=0; x < 100; x++){
        for (y=0; y < 100; y++){
                for (z = 0; z < 100; z+=3){
                        if (x+y+z == 100 && 30*x + 50*y + 10*z/3==1000){
                                cout <<x<<", "<<y<<", "<<z<<endl;
                        }
                }
        }
}
```

枚举法示例1

- 可否优化？加快运行速度 ➔ **减少枚举次数！**
- 三种鸡的和是固定的我们只要枚举二种鸡（x,y），第三种鸡就可以根据约束条件求得（z = 100 - x - y），这样就缩小了枚举范围：

  4X + 7Y = 100；

  X + Y + Z = 100；

  （X，Y，Z > 0, Z%3 == 0），=>> 0 <= x < = 25因此代码可以优

```
for (x=0; x <=25; x++){
        y = 100 - 4*x;
        if(y%7 == 0 && y >= 0){
                y /= 7;
                z = 100 - x - y;
                if (z%3 == 0 && 30*x + 50*y + 10*z/3==1000){
                        cout <<x<<", "<<y<<", "<<z<<endl;
                }
        }
```

枚举法示例2: 四大湖问题

- 上地理课时，四个学生回答我国四大湖的大小时分别说：
  - ① 甲：洞庭最大，洪泽最小，鄱阳第三
  - ② 乙：洪泽最大，洞庭最小，鄱阳第二，太湖第三
  - ③ 丙：洪泽最小，洞庭第三
  - ④ 丁：鄱阳最大，太湖最小，洪泽第二，洞庭第三

- 已知对于每个湖的大小，每个人仅答对了一个
- 设计一程序让计算机通过这些信息去判别四个湖的大小。

四大湖问题：解题思路

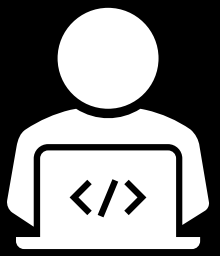
- 如果用a,b,c,d分别表示四个湖。a表示洞庭湖， b表示洪泽湖， c表示鄱阳湖，d表示太湖。那么， abcd的排序可表示他们的大小关系。
- 假设：洞庭最大， 洪泽第二， 鄱阳第三， 太湖第四， 然后检查每位同学是否都讲对了一个。
- 如果不是， 再尝试下一种情况：洞庭最大， 洪泽第二， 鄱阳第四， 太湖第三，再检查每位同学是否都讲对了一个。
- 尝试所有可能的情况， 直到满足每位同学都讲对一个为止。

四大湖问题：解题思路

- 为了尝试所有情况， 我们需要假设洞庭湖可能是最大， 也可能是第二、第三或第四。
- 因此， a的值可能从1变到4。同样， b, c ,d的值也都可能从1变到4。
- 我们需要一个循环控制结构， 使a, b, c, d的值能自动从1变到4。

循环出现了！

```
int a = 0, b = 0, c = 0, d = 0; // a:洞庭湖；  b: 洪泽湖；  c: 鄱阳湖；  d: 太湖

for (a=1; a<=4; ++a){
        for (b=1; b<=4; ++b){
                if ( a == b) continue; // 湖的面积不等
                for (c=1; c<=4; ++c){
                                if (c==a||c==b)
                                                continue;
                                else {
                                                d = 10 - a - b - c;
                                        if (((a==1)+(b==4)+(c==3))==1
                                        &&((b==1)+(a==4)+(c==2)+(d==3))==1
                                        &&((b==4)+(a==3))==1
                                        &&((c==1)+(d==4)+(b==2)+(a==3))==1)
                                        cout << a << b << c << d;
                                }
                }
        }
}
```



代码演示

chapter4-fourlakes.cpp

- 我们意识到了湖的大小面积不等。
- 能否继续提升?

```
int a, b, c, d; bool flag = false;
for (a=1; a<=4; ++a) {
    for (b=1; b<=4; ++b) {
        if ( a == b) continue;
        for (c=1; c<=4; ++c){
            if (c==a||c==b)
                continue;
            else {
                d = 10 - a - b - c;
                if (((a==1)+(b==4)+(c==3))==1
                &&((b==1)+(a==4)+(c==2)+(d==3))==1
                &&((b==4)+(a==3))==1
                &&((c==1)+(d==4)+(b==2)+(a==3))==1) {
                    cout << a << b << c << d << endl;
                    flag = true; break;
                } //if
            }//else
        }
        if (flag) break;
    }//for
    if (flag) break;
}//for
```

注意break 和continue的用法

- **湖的面**积不等
- **找到答案就**结束!

本章课程流程

1. 重复 N 次循环
2. while 循环
3. do … while 循环
4. 循环的中途退出
5. 枚举法
6. 贪婪法

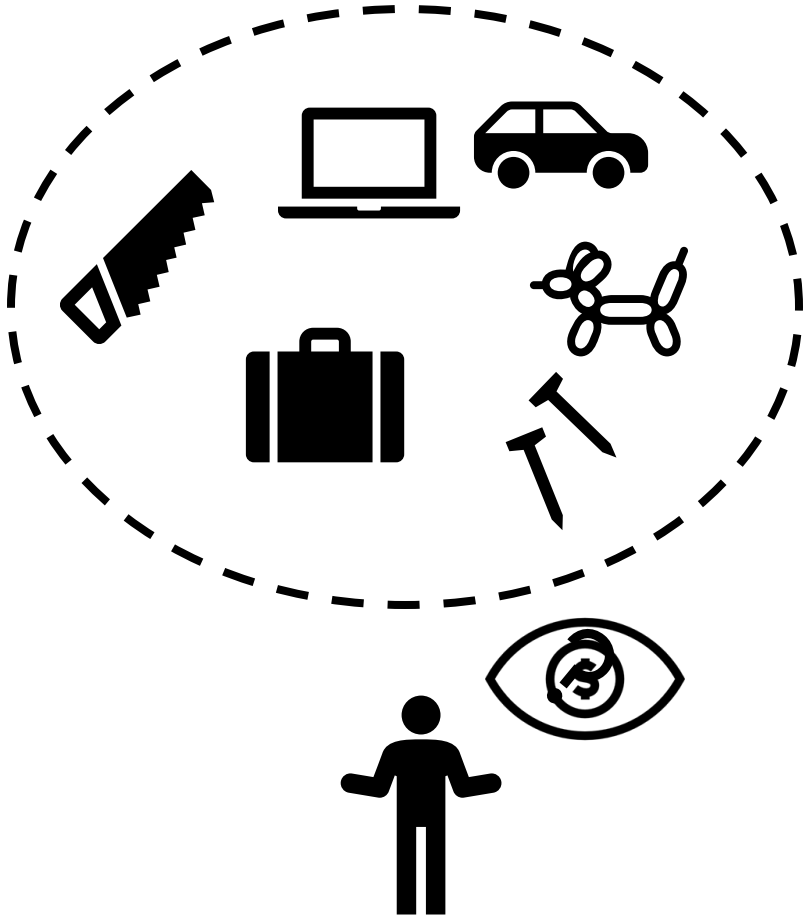
Greedy Algorithm

贪婪算法

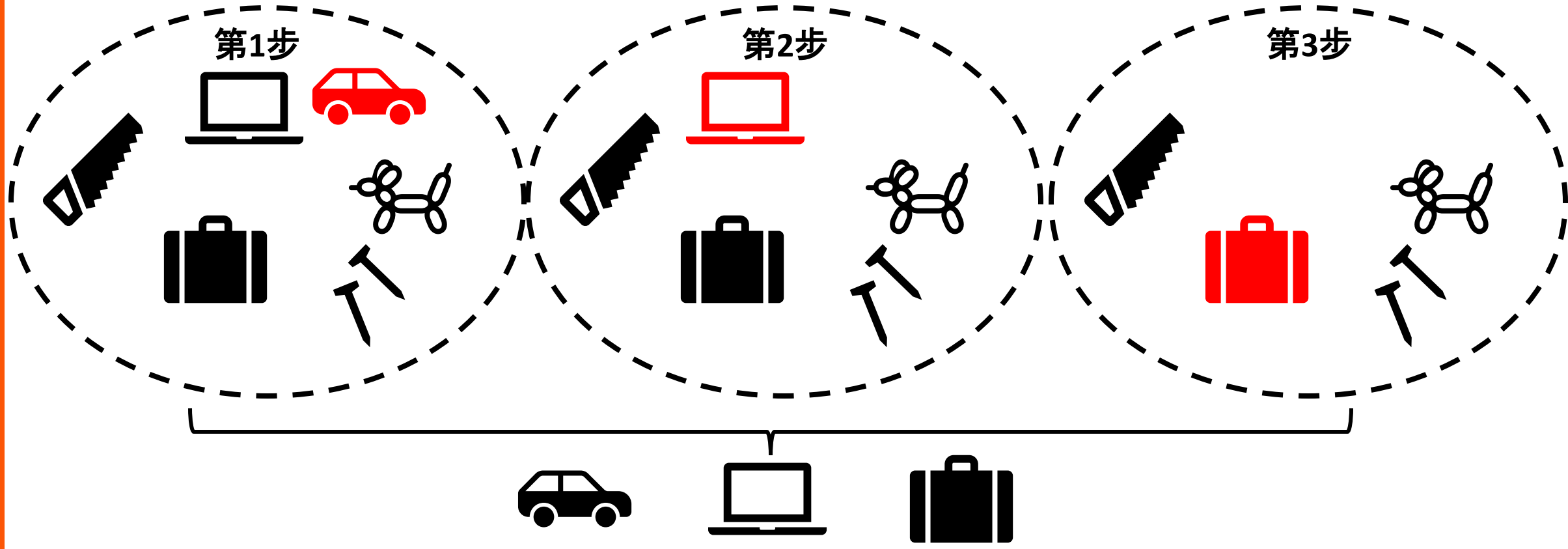
贪婪（贪心）算法的基本思想

- 问题：
  - 小明在超市活动中喜中大奖，他可以在超市内任意选择k件商品带回家。现在小明知道超市有n件(n>=k)商品，第i件(1<=i<=n)商品的价值为vi，请问小明如何选择可以使带回家的商品价值总和最大。
- 贪婪法：

每次从剩下的商品中选价值最大的商品，选k次。



贪婪（贪心）算法的基本思想



在求解过程的每一步都选取一个局部最优（local optimal）的策略，把问题规模缩小，最后把每一步的结果合并起来形成一个全局解。

贪婪（贪心）算法的基本思想

- 在求解过程的每一步都选取一个局部最优的策略，把问题规模缩小，最后把每一步的结果合并起来形成一个全局解。
- 基本步骤：
  - 从某个初始解出发
  - 采用迭代的过程，当可以向目标前进一步时，就根据局部最优策略，得到一个部分解，缩小问题规模。
  - 将所有解综合起来

- 贪心算法在数据科学领域被广泛应用，特别是金融工程。其中一个贪心算法例子就是Ensemble method。

硬币找零问题（Coin Change Problem）

- 对于一种货币，有面值为**1分**, **2分**, **5分**和**1角**的硬币，**最少**需要多少个硬币来找出K分钱的零钱。

硬币找零问题（Coin Change Problem）

- 对于一种货币，有面值为**1分**, **2分**, **5分**和**1角**的硬币，**最少**需要多少个硬币来找出K分钱的零钱。

- 不断地使用面值最大的硬币。如要找零的值小于最大的硬币值，则尝试第二大的硬币。依此类推。
- 不断尝试的过程就是循环

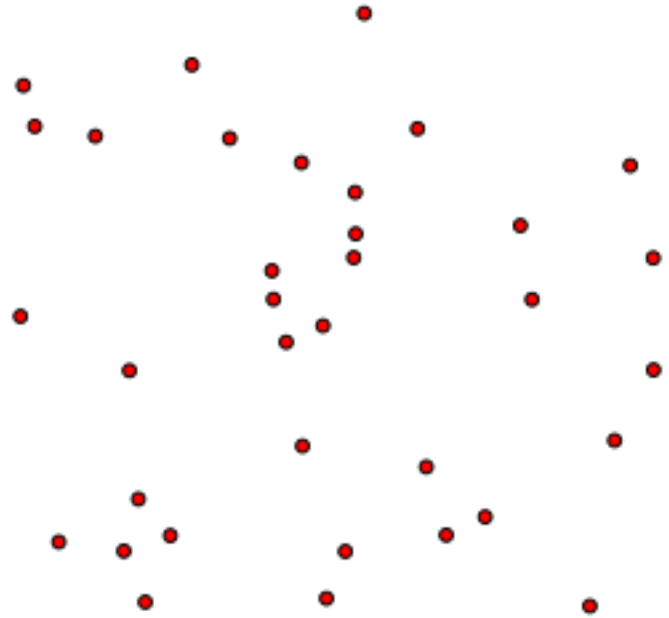
```cpp
int main()
{
    int money;
    int onefen = 0, twofen = 0, fivefen = 0, onejiao = 0;
    cout << "输入要找零的钱（以分为单位）： ";
    cin >> money;

    //不断尝试每一种硬币
    while (money >= 10) {onejiao++; money -= 10;}
    while (money >= 5) {fivefen++; money -= 5;}
    while (money >= 2) {twofen++; money -= 2;}
    while (money >= 1) {onefen++; money -= 1;}

    //输出结果
    if(onejiao == 0 && onefen == 0 && twofen == 0 && fivefen == 0){
        cout<<"无解";
    }else{
        cout << "1角硬币数： " << onejiao << endl;
        cout << "5分硬币数： " << fivefen << endl;
        cout << "2分硬币数： " << twofen << endl;
        cout << "1分硬币数： " << onefen << endl;
    }
    return 0;
}
```

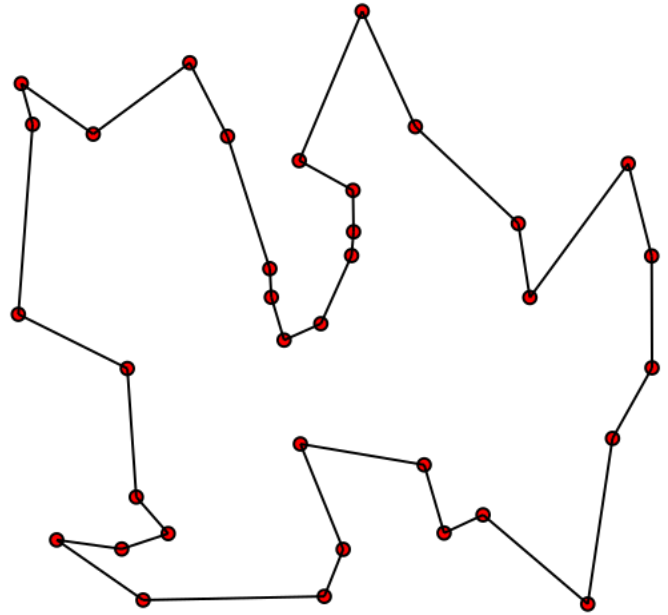
代码演示

chapter4-greedy.cpp

Travelling salesman problem

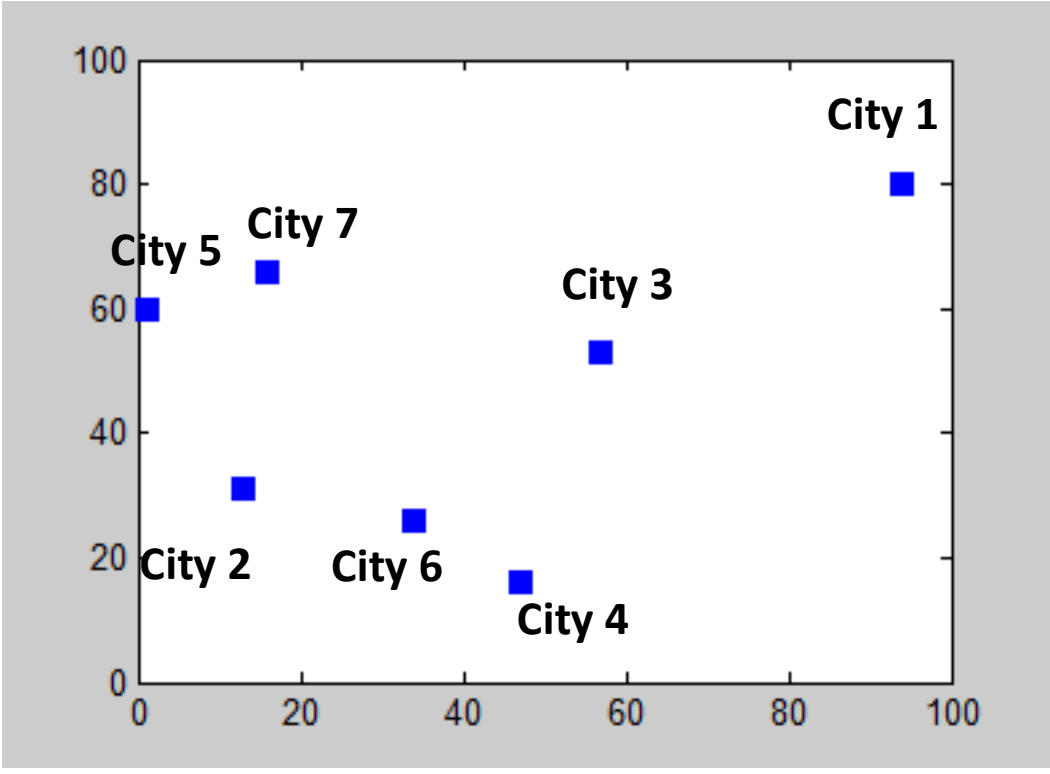
- 给定一系列城市和每对城市之间的距离，求解访问每座城市一次并回到起始城市的最短回路。
  - 该问题是NP hard的，是理论计算机、运筹学中的重要问题





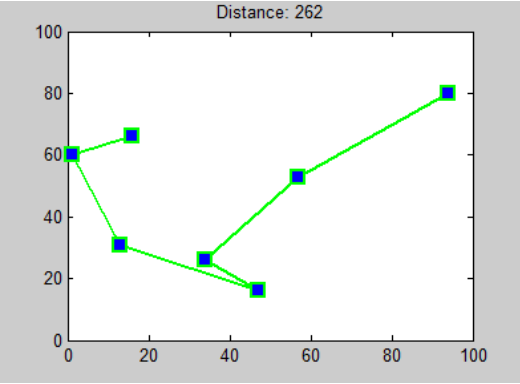
Travelling salesman problem

- 贪婪法给出的解：**最近邻算法**，即让推销员选择最近的未访问城市作为下一步行动。
  - 该算法可以快速产生有效的短路线。对于随机分布在平面上的 N 个城市，该算法平均产生的路径仅比最短路径长 25%

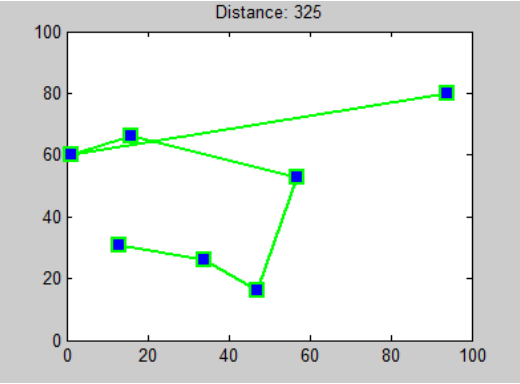


Travelling salesman problem

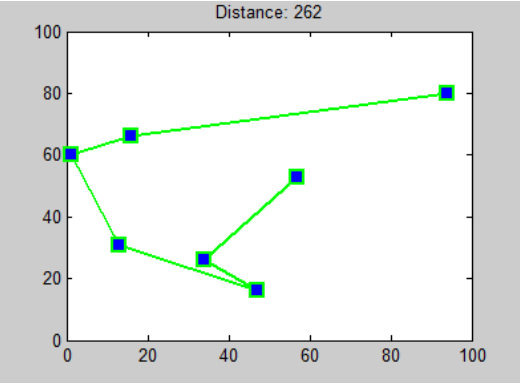
Starting from **City 1**

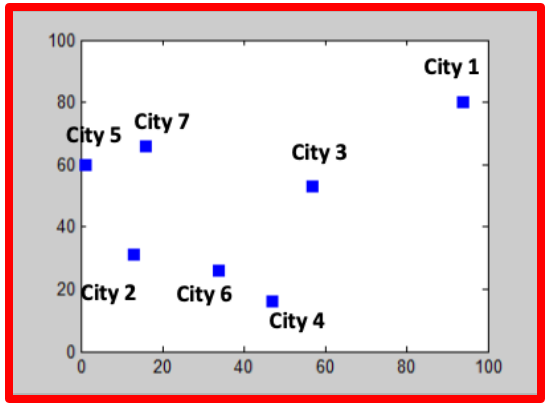


Starting from **City 2**

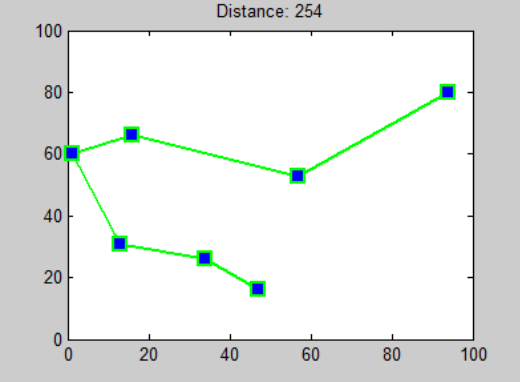


Starting from **City 3**

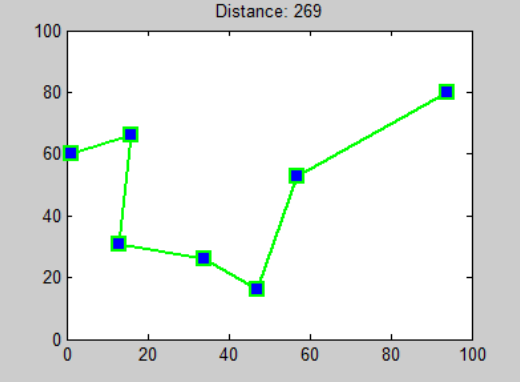




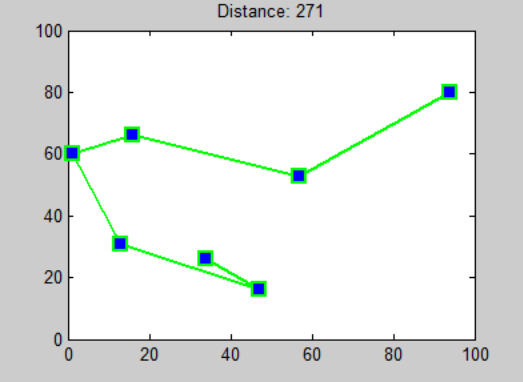
Starting from **City 4**



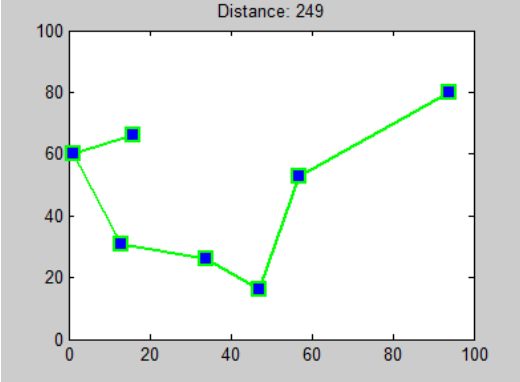
Starting from **City 5**



Starting from **City 6**



Starting from **City 7**



贪婪法的失效

贪婪法的失效

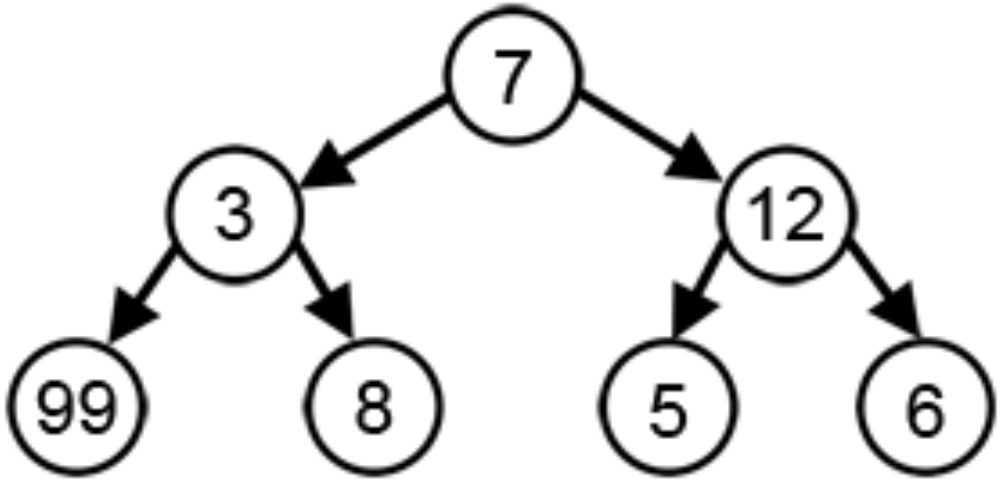
核心问题：局部最优不为全局最优！

贪婪法的失效

- 假设一个国家的货币系统中只有1分，3分，9分和1角四种硬币，最少需要多少个硬币来找出K分钱的零钱。
- 贪婪法吗？

- 当K=12时，按照贪婪法得到的解是12=10+1+1，即3。
- 实际上可以12=9+3，即正解为2。

贪婪法的失效



- 寻找最大加和
  - 贪婪法会限制于局部最优的步骤，故在第二步会选12而不选3. 这导致了最后漏掉了99这个明显的全局最优解

告诉了我们：贪婪虽有效，远见更重要！

贪婪法的失效

- 问题所在：局部最优不为全局最优！

- 应该怎么做？

  令f[k]表示k分钱时需要的最少硬币数

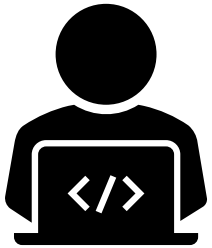
  如果取1分钱，则问题变为求f[k-1]

  如果取3分钱，则问题变为求f[k-3]

  …

  f[k]=min(f[k-1]+1,f[k-3]+1,f[k-9]+1,f[k-10]+1)

  感兴趣的同学可以自己动手实现一下。



代码演示

coins_dynamic_programming.cpp

第四章小结练习（1）

1. 下面一段代码，功能为循环的读取一个整数并输出，如果整数为奇数，则停止读入，否则继续循环。

请从A-D中选取代码片段填到横线内，使得代码功能正确

```
int n;
while (cin >> n) {
   cout << n << endl;
   if (_________) {
         _________;
   }
}
```

A. n%2 != 0　　B. n%2==0　　C. continue　　D. break

第四章小结练习（1）

1. 下面一段代码，功能为循环的读取一个整数并输出，如果整数为奇数，则停止读入，否则继续循环。

请从A-D中选取代码片段填到横线内，使得代码功能正确

```
int n;
while (cin >> n) {
   cout << n << endl;
   if (___A_____) {
         ___D_____;
   }
}
```

A. n%2 != 0　　B. n%2==0　　C. continue　　D. break

第四章小结练习（2）

- 选出以下错误的一项：

A. 贪婪法可用于求问题的最优解。

B. 枚举法可用于在寻找某个问题的所有的可能的方案，从中找出可行解。

C. 贪婪法一定可以求得问题的最优解。

D. 枚举法可以使用循环结构实现。

第四章小结练习（2）

- 选出以下错误的一项：C

A. 贪婪法可用于求问题的最优解。

B. 枚举法可用于在寻找某个问题的所有的可能的方案，从中找出可行解。

C. 贪婪法一定可以求得问题的最优解。

D. 枚举法可以使用循环结构实现。