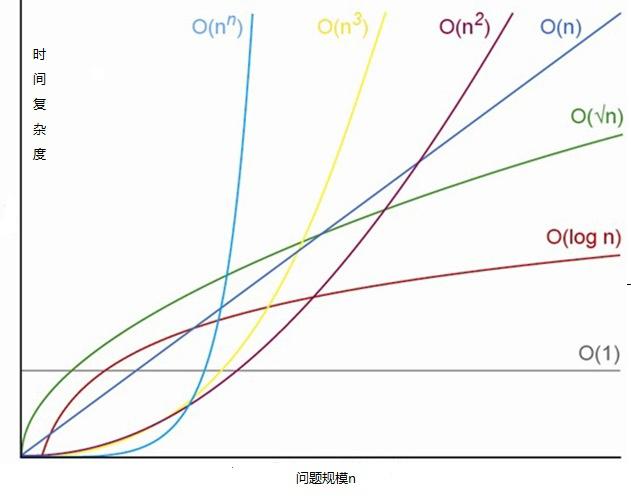
00[*To All, only love is the best teacher, it is far better than the responsibility.*](http://www.juzimi.com/ju/734409)

------Albert Einstein

# 算法复杂度



# 递归与分治法

## 递归法

### 基本特征

在方法中调用自身的编程技术。

1. 能转换为一个或多个结构相同，规模较小的问题，然后从这些小问题可以方便的构造出大问题的解；
2. 递归调用次数必须是有限的，也就是必须有终止条件。



### 汉诺塔

#include <iostream>

using namespace std;

//目的：把n个盘子从柱子a移动到柱子c

//1. 把前（n-1）当一整体，从a转到b

//2. 把第个从a移动到c;

//3. 把前（n-1）当一整体，从b转到c

//parameter: get n from a to c by b;

void hanooi(int n, char a, char b, char c){

if(n == 1){

cout<<"move "<<n<<" plate from "<<a<<" to "<<c<<endl;

return;

}

hanooi(n-1, a, c, b);

cout<<"move "<<n<<" plate from "<<a<<" to "<<c<<endl;

hanooi(n-1, b, a, c);

}

int main()

{

int n = 4;

cout<<n<<" layer Hannoi tower"<<endl;

hanooi(n, 'a', 'b', 'c');

system("pause");

return 1;

}

### 经典到爆的斐波那契数列

数列：0 1 1 2 3 5 8 13 ...

序号：1 2 3 4 5 6 ...

#### 递归法

#include <iostream>

using namespace std;

int fib(int n){

if(n == 1) return 0;

if(n == 2) return 1;

return fib(n-1)+fib(n-2);

}

int main()

{

int n;

while(1){

cin>>n;

cout<<fib(n)<<endl;

}

system("pause");

return 1;

}

#### 循环法？

#include <iostream>

using namespace std;

int fib(int n){

if(n == 1) return 0;

if(n == 2) return 1;

int f1 = 0, f2 = 1, result;

for(int i=3; i<=n; i++){

result = f1+f2;

f1 = f2;

f2 = result;

}

return result;

}

int main(){

int n;

while(1){

cin>>n;

cout<<fib(n)<<endl;

}

system("pause");

return 1;

}

#### 神奇的数学法

xxxx

### 八皇后

#### 描述：

不说了

#### 递归法

原来八皇后问题递归起来如此简单，乐死我了。。。

#include <iostream>

using namespace std;

//static the number of result

int resultNum;

//chess[i]: 把棋子放在第i行的第chess[i]列，棋盘从1数到8；

int chess[9];

//检查A步的赋值check[n]=i;是否可行

int check(int n){

for(int i=1; i<n; i++){

//第n行的棋子与第i行的chess, 同一列 || 右下角 || 左下角

if(chess[n]==chess[i] || chess[n]==chess[i]+(n-i) || chess[n]==chess[i]-(n-i))

return 0;

}

return 1;

}

void showChess(){

cout<<"Result - "<<++resultNum<<endl;

for(int i=1; i<=8; i++)

cout<<i<<":"<<chess[i]<<"\t";

cout<<endl;

}

//method: recursion || backtrack

//n is the row of chessboard;

void putChess(int n){

if(n > 8){

showChess();

return;

}

for(int i=1; i<=8; i++){

//A在当前这个n行放第i列

chess[n] = i;

//如果A步可行

if(check(n)){

//n这个棋子放的可行 && n<8 行, 那就继续下一个吧

putChess(n+1);

}

}

}

int main(){

putChess(1);

system("pause");

return 1;

}

## 分治法

### 基本思想

将一个难以直接解决的大问题，分割成一些规模较小的相同问题，以便各个击破，分而治之。

Divide-and-Conquer(P){

for(i=1; i<=K; i++){

//从子问题Pi中获得解yi

yi = Divide-and-Conquer(Pi);

//合并k个解得到原问题解；

T = merge(y1, y2, … , yk);

}

}

### 二分法求方程近似解

方程 f(x) = x^3 + x^2 – 1 = 0 在[0, 1]上的近似解；精确度为0.01

#include <iostream>

using namespace std;

double fun(double x){

return pow(x, 3)+x\*x-1;

}

double findRoot(double left, double right){

double mid;

while(right-left >= 0.01){

mid = (left+right)/2;

if(fun(mid) == 0)

return mid;

//mid 与left在两边

if(fun(mid)\*fun(left) < 0)

right = mid;

else

left = mid;

}

return left;

}

int main(){

cout<<"The root: "<<findRoot(0, 1)<<endl;

system("pause");

return 1;

}

# 贪心法

## 基本思路

每次取当前情况下最有利的解，“只看眼前，不管未来”，典型的乐观派；但是人无远虑，就不能保证最终解释全局最优的，但大多数情况下还是满足要求的；

## 部分背包

所有装入的物品不一定是整体，可以切片；所以嘛，我们就先对单价排个序，然后从性价比高的挑，直到满载而归；

/\*\*

function:

这是部分背包，里面装的东西跟盼盼小面包一样，是可以切片滴

steps：

1. 求出单价，并排序

2. 从单价最大的开始放入，接着下一个，如果下一个会爆掉容量，则放入部分

date: 2016-4-12

\*/

#include <iostream>

using namespace std;

int weight[5] = {1, 2, 3, 4, 5};

int value[5] = {3, 10, 6, 3, 5};

double ratio[5];

int capacity = 10;

void sortRaito(){

for(int i=0; i<5; i++)

ratio[i] = (double)value[i]/weight[i];

for(int i=0; i<4; i++){

for(int j=0; j<4-i; j++)

if(ratio[j+1] > ratio[j]){

swap(ratio[j+1], ratio[j]);

swap(weight[j+1], weight[j]);

swap(value[j+1], value[j]);

}

}

}

double computeProfit(){

double profit = 0;

int tempW = 0, i=0;

while(tempW<=capacity && i<5){

if(tempW+weight[i] <= capacity){

tempW += weight[i];

profit += value[i];

//不够装了就装部分吧, 要判断一下是否够装

}else if(tempW+weight[i] >= capacity){

int leftW = capacity - tempW;

profit += ratio[i]\*leftW;

break;

}

i++;

}

return profit;

}

int main(){

sortRaito();

cout<<computeProfit()<<endl;

system("pause");

return 1;

}

## 哈夫曼编码

## 单元最短路径

## 最小生成树

### Kruskal 算法

### Prim算法

## 穷举法

比如有n个元素，就设置一个vector<int> select(n, 0);

select[i] = 0 || select[i] =1; 2^n的复杂度，然后保留每次的0110… 这样的组合（select）

# 回溯法

## 基本思想

回溯可以利用状态图来理解：

A）for (int i=index; i<=n ; i++) {

X[i]=1

subSet2(i+1);

//不加i，for会往下走到下一个元素

x[i] = 0;

}

1. x[index] = 1;

subSet(index+1);

//不加index，往下走

x[index] = 0;

subSet(index+1);

## 小老鼠走迷宫

迷宫8\*8，为了方便，两端都设置成墙，通为0，不通为1；

从起点开始，将其入栈，上右下左四个方向依次试探，如果有下一个为0，则把下一个入栈保存，如果四方无路了，则当前结点出栈，如果栈空了，那说明此迷宫无路可走；

### 代码：

#include <iostream>

using namespace std;

#define maxSize = 100

#define N 10

#define M 10

//maze only be: 8\*8, 两端是wall

int mg[M][N] = {

{1,1,1,1,1,1,1,1,1,1},

{1,0,0,1,0,0,0,1,0,1},

{1,0,0,1,0,0,0,0,0,1},

{1,0,0,0,0,1,1,0,0,1},

{1,0,1,1,1,0,0,0,0,1},

{1,0,0,0,1,0,0,0,0,1},

{1,0,1,0,0,0,1,0,0,1},

{1,1,1,1,1,0,1,1,0,1},

{1,0,0,0,0,0,0,0,0,1},

{1,1,1,1,1,1,1,1,1,1}

};

class stac{

public:

//迷宫矩阵的行和列

int i; int j;

//标志当前是望哪个方向走可以有下一个；0,1,2,3: 上右下左四个方向

int dir;

};

stac stack[100];

//record the nums in stacks

int top = 0;

void maze(){

int ii, jj, di, find;

//入口

stack[top].i = 1;

stack[top].j = 1;

stack[top].dir = -1;

//mg[i][j] = -1表示这个点暂时确定是可行的，打个标记已访问；如果后来发现它没法下一个了，mg = 0

mg[1][1] = -1;

//管它呢，先试探性的走，走的通就进栈，发现四个方向都没路了，则把栈顶给退了，返回上一步

//如果所有的元素都退栈了，也就是top = -1, 那就无路可走了哈

while(top > -1){

ii = stack[top].i;

jj = stack[top].j;

di = stack[top].dir;

//if(ii==8 && jj==8) 已经到达出口了哈

if(ii==M-2 && jj==N-2){

cout<<"迷宫通路："<<endl;

for(int k=0; k<=top; k++){

cout<<"("<<stack[k].i<<","<<stack[k].j<<")";

if((k+1)%5 == 0)

cout<<endl;

}

cout<<endl;

return;

}

find = 0;

//试探性换着四个方向进行下一步，如果没墙就 find = 1

while(di<4 && find==0){

di++;

switch(di){

case 0: ii=stack[top].i-1; jj = stack[top].j; break;

case 1: ii=stack[top].i; jj = stack[top].j+1; break;

case 2: ii=stack[top].i+1; jj = stack[top].j; break;

case 3: ii=stack[top].i; jj = stack[top].j-1; break;

}

if(mg[ii][jj] == 0)

find = 1;

}

//如果所谓的下一步是可行的，那么记录下哈

if(find == 1){

stack[top].dir = di;

top++;

stack[top].i = ii;

stack[top].j = jj;

cout<<"进栈:（"<<ii<<","<<jj<<")"<<endl;

stack[top].dir = -1;

mg[ii][jj] = -1;

}else{

cout<<"出栈: "<<stack[top].i<<","<<stack[top].j<<endl;

mg[stack[top].i][stack[top].j] = 0;

top--;

}

}

cout<<"没有路走了"<<endl;

}

void printMaze(){

for(int i=0; i<M; i++){

for(int j=0; j<N; j++){

if(mg[i][j] == -1)

cout<<"\*";

else

cout<<mg[i][j];

cout<<" ";

}

cout<<endl;

}

}

int main(){

maze();

cout<<endl;

printMaze();

system("pause");

return 1;

}

### 点评

这个例题也是绝了，充分表现了回溯的思想，但是它却没有用到两种基本框架，所有它未对所有路径进行遍历，只是求出了一个可行解；

## 求子集合

遍历所有集合可能, include empty subSet;

### 代码

#include <iostream>

using namespace std;

int a[] = {3, 5, 7, 9};

//传说中回溯法要的访问标记

int x[5];

int n = 4;

int nCount = 0;

void printArray(){

nCount++;

for(int i=0; i<n; i++)

if(x[i] == 1)

cout<<a[i]<<" ";

cout<<endl;

}

//回溯方法一

void subSet2(int index){

if(index >= n){

printArray();

return;

}

//从index开始往后直到n循环，不是n-1哦

//why? 把i<=n改为i<n就可以计数了

for(int i=index; i<=n; i++){

//加i，往下走

x[i] = 1;

subSet2(i+1);

//不加i，for会往下走到下一个元素

x[i] = 0;

}

}

// 回溯法二

void subSet(int index){

if(index == n){

printArray();

return;

}

//加index，往下走

x[index] = 1;

subSet(index+1);

//不加index，往下走

x[index] = 0;

subSet(index+1);

}

int main(){

subSet(0);

cout<<"nCount: "<<nCount<<endl;

cout<<endl;

nCount = 0;

subSet2(0);

cout<<"nCount: "<<nCount<<endl;

system("pause");

return 0;

}

### 问题

框架2计数不可用？why?

## 数组子元素求和为某个数

#include <iostream>

using namespace std;

int a[] = {1, 2, 3, 4};

int x[4];

int n = 4;

int sum = 0;

int sumX = 5; //总和

void printArray(){

for(int i=0; i<n; i++)

if(x[i] == 1)

cout<<a[i]<<" ";

cout<<endl;

}

void subSet(int index){

if(sum == sumX){

printArray();

return;

}

if(sum>sumX || index == n)

return;

//select a[index];

x[index] = 1;

sum += a[index];

subSet(index+1);

//don't select a[index]

x[index] = 0;

sum -= a[index];

subSet(index+1);

}

void subSet2(int index){

if(sum == sumX){

printArray();

return;

}

if(sum>sumX || index == n)

return;

//搞不懂为什么，求sum和这里是i<n, 不能用i<=n

for(int i=index; i<=n; i++){

x[i] = 1;

sum += a[i];

subSet2(i+1);

sum -= a[i];

x[i] = 0;

}

}

int main(){

subSet(0);

cout<<endl;

sum = 0;

memset(x, 0, sizeof(x));

subSet2(0);

system("pause");

return 0;

}

### 问题

框架2中i<n,不能加等号 上题求子集合要i<=n？why?

## 变体 - 组合不重复

给上题来个变体，依旧从数组中找组合为某个数，但是组合不要重复

<http://blog.chinaunix.net/uid-22753395-id-1769851.html>

给定一个数t，以及n个整数组成的集合，在这集合中找到和为t的所有组合，

例如t=4，n=6，这6个数为[4,3,2,2,1,1]。可以输出4个不同的组合，它们的和为4: 4, 3+1, 2+2, and 2+1+1。

**解题思路**

先将数据按从大到小进行排序，然后使用回溯法遍历所有可能。注意去掉重复的结果。

#include <iostream>

#include <algorithm>

using namespace std;

int a[]={4,3,1,2,1,2};

bool x[6];//标记第i个元素是否已经使用

int N = 6;//元素个数

int t = 4;//目标和

int sum = 0;//当前和

void printArray(){

for(int i=0; i<N; i++)

if(x[i] == 1)

cout<<a[i]<<" ";

cout<<endl;

}

void backtrace(int n)

{

if(sum==t){//当前和等于t,输出结果

printArray();

return;

}

if(sum>t || n==N)//和大于t or 超过n层

return ;

for(int i=n;i<N;++i){

x[i]=true;

sum+=a[i];

backtrace(i+1);

x[i]=false;

sum -=a[i];

//跳过相同的，i<N-1是为了防止地址越界

while(i<N-1 && a[i]==a[i+1])

i++;

}

}

void subSet(int index){

if(sum == t){

printArray();

return;

}

if(sum>t || index == N)

return;

//select a[index];

x[index] = 1;

sum += a[index];

subSet(index+1);

//don't select a[index]

x[index] = 0;

sum -= a[index];

subSet(index+1);

//用这种方法，如何跳过相同数字呢？

int i = index;

while(i<N-1 && a[i]==a[i+1])

i++;

}

int main()

{

sort(a, a+N);

backtrace(0);

cout<<endl;

memset(x, 0, sizeof(x));

subSet(0);

system("pause");

return 0;

}

## 八皇后问题

与递归法中的是一毛一样的，递归是一种手段，回溯是一种思想；

## 01背包

### 代码

#include <iostream>

using namespace std;

int p[] = {9, 10, 7, 4}, w[] = {3, 5, 2, 1};

int n = 4, c = 7;

//最终的最大价值，当前价值，当前重量

int maxP, cp, cw;

//这个就是传说中的 回溯法必备的访问标记数组， 最终版的访问标记

int\* x = new int[n];

//每次回溯一种情况，临时的访问标记

int\* tempX = new int[n];

void backTrack0(int i){

if(i == n){

if(cp > maxP){

maxP = cp;

for(int i=0; i<n; i++)

x[i] = tempX[i];

}

return;

}

//要第i个物品，但是之前要判断下是否重量超标了哈

if(cw + w[i] <= c){

cw += w[i]; cp += p[i];

tempX[i] = 1;

backTrack0(i+1);

cw -= w[i]; cp -= p[i];

}

//不要第i个物品

tempX[i] = 0;

backTrack0(i+1);

}

//对backTrack0的简写，使用的是同一种框架， 便于理解

void backTrack(int i){

if(cw > c)

return;

if(i == n){

if(cp > maxP){

maxP = cp;

memcpy(x, tempX, sizeof(int)\*n);

}

return;

}

//要第i个物品

cw += w[i]; cp += p[i];

tempX[i] = 1;

backTrack(i+1);

cw -= w[i]; cp -= p[i];

//不要第i个物品

tempX[i] = 0;

backTrack(i+1);

}

//回溯法经典的框架2： for循环

void backTrack2(int i){

if(cw > c)

return;

if(i == n){

if(cp > maxP){

maxP = cp;

memcpy(x, tempX, sizeof(int)\*n);

}

return;

}

//搞不懂为什么，求sum和这里是i<n, 不能用i<=n

for(int j=i; j<=n; j++){

//要物品i

cw += w[j]; cp += p[j];

tempX[j] = 1;

backTrack2(j+1);

cw -= w[j]; cp -= p[j];

//不要物品i

tempX[j] = 0;

}

}

void printResult(){

cout<<"The max value you can put is: "<<maxP<<endl;

cout<<"The object are: ";

for(int i=0; i<n; i++){

if(x[i] == 1)

cout<<i+1<<" ";

}

cout<<endl;

}

int main()

{

//backTrack0(0);

//backTrack(0);

backTrack2(0);

printResult();

system("pause");

return 0;

}

### 军哥点评：

这里只是运气好，使用两种框架都爆破了，但是=还是<=的问题来日再来探究。o(^▽^)o

# 动态规划

## 多源最短路径

### 解析：

[最短路径—Dijkstra算法和Floyd算法\_华山大师兄\_cnblogs](https://www.cnblogs.com/biyeymyhjob/archive/2012/07/31/2615833.html)

## 01背包问题；

<http://zdr53227.blog.163.com/blog/static/26080889201041593347721/>

背包问题的精彩总结详见<<你的背包>>

**输入：**

背包重量，物品数

w1, v1

w2, v2

…

wn, vn

**输出：**

最大价值

### 输入样例：

10 3

3 4

4 5

5 6

样例输出：

11

### 分析：

见题下网址：精辟；

方便计算第一行为空，所以行：物品个数+1；

背包容量从empty到weight, 所以列：背包重量+1；

新建数组c[物品个数+1][背包重量+1]

1. c[i][j]: 最大value值；

i表示只从1到i号物体中挑选；所以每个c[i][:]就面临两种情况，i物品的选与不选？

j: 重量最大为j;

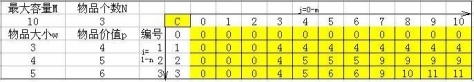
1. 分两种情况；

一定不选i号物体, 背包重量为j，能达到的最大价值：c[i-1][j];

一定选上第i号物体，且背包重量最大为j：c[i-1][j-weight(i-1)]+value(i-1);

注意不能是：c[i][j-weight(i-1)]+value(i-1)，想一想这是为什么呢？

Weight(i-1), value(i-1)☞的是第i号物品的重量和价值；



和最大公共子序列同为二维数组求解

|  |  |  |
| --- | --- | --- |
|  | 最大公共子序列 | 01背包 |
| I,j | 两个字符串的位置  V[i][j]: I,j位置最大序列长度 | I: 物品编号；由w[i]可以取物品大小，v[i]取物品价值；  J: 背包重量  V[i][j]: 物品只能从1到i中选，背包重量最大为j的情况下，背包最大价值 |
| C[i][j]赋值 | 比较相邻的  If (s1[i] == s2[j]){  C[i][j] = c[i-1][j-1] + 1;  } else {  max(c[i][j-1], c[i-1][j])  } | 思维跳动太多，完全不是相邻的了  //1. 选i号物品  V1 = c[i-1][j-第i物品的重量] + 第i物品的价值；  //2. 一定不要i号物品  V2 = c[i-1][j]  C[i][j] = max(v1, v2); |

### 代码1：二维数组 – C++版本

#include<iostream>

#include<iomanip>

#include<vector>

usingnamespace std;

int M, N;

void printarray(vector<vector<int>>c){

for(int i=0; i<N+1; i++){

for(int j=0; j<M+1; j++)

cout<<setw(3)<<c[i][j];

cout<<endl;

}

}

void findDetail(vector<vector<int>>c, vector<vector<int>>wv){

cout<<"maxValue: "<<c[N][M]<<endl;

while(N!=0 && M!=0){

//说明添加了第w行，也就是第w-1号物体

if(c[N][M] >c[N-1][M]){

cout<<"object "<<N-1<<": "<<wv[N-1][0]<<", "<<wv[N-1][1]<<endl;

M -= wv[N-1][0];

}

//else 就没选w行呗

N -= 1;

}

}

int main()

{

cout<<"Please input the capcity M and the number of items N"<<endl;

cin>>M>>N;

//convient to compute, the first row should be all zero;

//capcity should from empty(0) to M

vector<vector<int>> c(N+1, vector<int>(M+1));

vector<vector<int>> wv(N, vector<int>(2));

cout<<"please input "<<N<<" groups of Weight and value:"<<endl;

for(int i=0; i<N; i++)

cin>>wv[i][0]>>wv[i][1];

for(int i=1; i<N+1; i++){

int w = wv[i-1][0];

for(int j=1; j<M+1; j++){

if(w<=j && c[i-1][j-w]+wv[i-1][1]>c[i-1][j])

c[i][j] = c[i-1][j-w]+wv[i-1][1];

else

c[i][j] = c[i-1][j];

}

}

printarray(c);

findDetail(c, wv);

system("pause");

return 1;

}

### 代码2：二维数组- python版本

*# Author: shujun  
# Date: 2020-03-22***class** Solution:  
  
 **def** findMaxValue(self):  
 W, N = input(**"请输入背包最大重量和物品数："**).split(**" "**)  
 W = int(W)  
 N = int(N)  
 print(W, **", "**, N)  
 ws = []  
 vs = []  
 **for** i **in** range(N):  
 w, v = input(str(i)+**" 请输入物品重量和价值："**).split(**" "**)  
 ws.append(int(w))  
 vs.append(int(v))  
  
 *# 常用套路，从1开始真正计算，index=0处默认填充* values = [[0 **for** i **in** range(W+1)] **for** i **in** range(N+1)]  
 **for** i **in** range(1, N+1):  
 **for** j **in** range(1, W+1):  
 **if** j >= ws[i-1]:  
 *# 1. abort object i  
 # v1 = values[i-1][j]  
 # 2. add object i  
 # print(j, " ", ws[i-1])* addiValue = values[i-1][j-ws[i-1]] + vs[i-1]  
 values[i][j] = max(values[i-1][j], addiValue)  
 **else**:  
 values[i][j] = values[i-1][j]  
 **return** values[N][W]  
  
  
solution = Solution()  
result = solution.findMaxValue()  
print(result)

### 代码3：一维数组

//f[v]=max{f[v],f[v-w[i]]+v[i]};

#include<iostream>

usingnamespace std;

int main()

{

int value[1000], weight[1000];

int f[1000];

int N, V;

cin>>V>>N;

memset(f, 0, sizeof(f));

for(int i=1; i<=N; i++){

cin>>weight[i]>>value[i];

}

for(int i=1; i<=N; i++){

//note: 此处是逆序输出哈

//如果是顺序就是不对的，因为f[j]越加越多

//逆序才能保证f[j-weight[i]]是上一个物品

//for(int j=1; j<=V; j++){

for(int j=V; j>=weight[i]; --j){

if(f[j-weight[i]]+value[i]>f[j])

f[j] = f[j-weight[i]] + value[i];

}

}

cout<<f[V]<<endl;

//system("pause");

return 0;

}

## 军哥点评：

强大的动态规划，高深佩服！！！！

1. 也不知道什么时候有这么一个用自己的语言记录学习的想法，貌似是为了15年算法竞赛，从2015年11月份？每天早晨开始一个小时（8-9）记录一点点，不过参加应该也木有用，算法这条路真是路漫漫其修远兮 ~ ~；
2. 回头看看这些记录，实在太有用了，试想如果我从头看公共子序列，看导弹拦截，看01背包，必定找资料花很长时间，即便找到了通俗的资料，很多也写的很啰嗦，不够简洁；即便有简洁的，也没有自己的语言这么符合思维，可以瞬间回顾到。
3. 最大公共子序列的求解有点类似于01背包的二维度求解法，不过应该和那个中间有路障，从一个点到另一个点总共可以有多少路径的更为相似。导弹拦截和01背包的一维度解法

# 分支限界算法设计

五虎上将的最后一关，过五关

## 基本思想：

## 0-1背包

### FIFO分支限界法

程序很长，待续。。。。

### 优先队列式分支限界法

## 旅行售货员问题

精彩待续。。。

# 搜索问题

## For循环遍历

/random/\_01\_01\_sumOf2Num.py

/hot100/ \_02\_15\_threesum.py

/hot100/ \_10\_39\_combinationSum.py

## 深度搜索

递归，一颗深度优先遍历树

递归的算法复杂度是多少？

## 广度搜索

队列，一颗广度优先遍历树

# References

算法分析与设计 石志国 刘冀伟等

Leecode