# 动态规划

## 算法思想

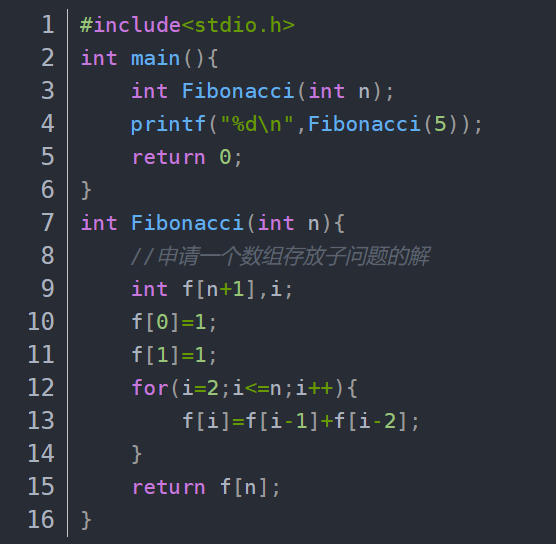
**动态规划**：算法通常用于求解某种具有最优性质的问题。其思想和分治法类似，基本思想时将待求解问题分解成若干个子问题，先求解子问题，然后从这些子问题的解中得到原有问题的解，类似于数学归纳法。与分治的不同的是，动态规划分解后得到子问题往往是相互联系的。

步骤：

1. 划分：按照问题的特征，把问题分为若干阶段。注意：划分后的阶段一定是有序的或者可排序的
2. 确定状态和状态变量：将问题发展到各个阶段时所处的各种不同的客观情况表现出来。状态的选择要满足无后续性
3. 确定决策并写出状态转移方程：状态转移就是根据上一阶段的决策和状态来导出本阶段的状态。根据相邻两个阶段状态之间的联系来确定决策方法和状态转移方程
4. 边界条件：状态转移方程是一个递推式，因此需要找到递推终止的条件

## 算法案例

斐波拉契数列：



## leetcode算法题演示

[118. 杨辉三角 - 力扣（LeetCode）](https://leetcode.cn/problems/pascals-triangle/description/?envType=study-plan-v2&envId=top-100-liked)

[198. 打家劫舍 - 力扣（LeetCode）](https://leetcode.cn/problems/house-robber/description/?envType=study-plan-v2&envId=top-100-liked)

## 适用场景或面试题

背包问题

问题描述：  
给定一组物品，每个物品有一个重量和一个价值。我们需要选择一些物品放入一个背包中，使得背包的总重量不超过一个给定的限制，并且所选物品的总价值最大化。

解决方法：  
这个问题可以使用动态规划来解决，具体步骤如下：

1、定义状态：设dp[i][j]表示在前i个物品中，背包容量为j时可以获得的最大价值。

2、初始化状态：将dp数组初始化为0，即dp[i][0] = 0，dp[0][j] = 0。

3、状态转移方程：对于第i个物品，有两种选择：放入背包或不放入背包。

如果选择放入背包，那么背包的容量减去当前物品的重量，背包中的总价值为dp[i-1][j-w[i]] 加上当前物品的价值v[i]。

如果选择不放入背包，背包中的总价值为dp[i-1][j]。

4、综合两种选择，取价值较大的情况作为dp[i][j]的值：dp[i][j] = max(dp[i-1][j-w[i]] + v[i], dp[i-1][j])。

5、递推计算：根据状态转移方程，使用两层循环遍历物品和背包容量，计算dp数组的值。

返回结果：最终的最大价值为dp[n][W]，其中n为物品的数量，W为背包的容量。

# 贪心算法

## 算法思想

贪心算法：贪心算法是一种基于贪心策略的算法，即在每一步选择中都采取当前最优的选择，希望得到全局最优的解。

步骤：

1. 建立数学模型，把问题分成若干个子问题。
2. 对每个子问题求解局部最优解，然后合成原问题的一个解。

贪心算法的关键：贪心策略的选择必须具有无后效性，即前面的选择不影响后面的选择。

贪心算法的难点：如何划分问题、如何从当前小问题进入下一步小问题。

## 算法案例

零钱找回问题：

假设1元、2元、5元、10元、20元、50元、100元的纸币分别有n1,n2,n3,n4,n5,n6,n7张

用这些钱支付K元，至少用多少张纸币。

采用贪心算法：每次增加一张已拥有的数额最大的纸币，如果大了就换一张小的，小了继续重复增加一张已拥有的数额最大的纸币，不考虑全局，只考虑当前情况。

## leetcode算法题演示

[55. 跳跃游戏 - 力扣（LeetCode）](https://leetcode.cn/problems/jump-game/description/?envType=study-plan-v2&envId=top-100-liked)

[45. 跳跃游戏 II - 力扣（LeetCode）](https://leetcode.cn/problems/jump-game-ii/description/?envType=study-plan-v2&envId=top-100-liked)

## 适用场景或面试题

贪心适用于求解最优化问题如：背包问题、最小生成树、最短路径。

柠檬水问题

问题描述：  
柠檬水摊贩每杯售价为 5 美元。顾客排队购买柠檬水，每位顾客只会购买一杯柠檬水，并且顾客只能支付 5、10 或 20 美元。柠檬水摊贩开始时没有任何零钱。我们需要判断摊贩能否给每位顾客正确找零，即摊贩手中是否有足够的零钱满足所有顾客的找零需求。

解决方法：  
这个问题可以使用贪心算法来解决，具体步骤如下：

初始化两个变量，分别表示手中拥有的 5 美元和 10 美元的数量。

遍历每位顾客，根据其支付金额进行处理：

如果顾客支付 5 美元，不需要找零，将手中的 5 美元数量加一。

如果顾客支付 10 美元，需要找零 5 美元，此时手中必须有至少一张 5 美元，将手中的 5 美元数量减一，将手中的 10 美元数量加一。

如果顾客支付 20 美元，需要找零 15 美元，优先使用一张 10 美元和一张 5 美元找零，如果不存在这样的组合，则尝试使用三张 5 美元找零。如果无法找零成功，返回 false。

遍历结束后，如果所有顾客的找零需求都被满足，返回 true；否则，返回 false。

**经常使用的Fun.c函数：**

**#ifndef \_\_WD\_FUNC\_H**

**#define \_\_WD\_FUNC\_H**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <string.h>**

**#include <stdbool.h>**

**#include <errno.h>**

**#include <error.h>**

**#include <sys/mman.h>**

**#include <sys/wait.h>**

**#include <time.h>**

**#include <unistd.h>**

**#include <sys/stat.h>**

**#include <sys/types.h>**

**#include <signal.h>**

**#include <dirent.h>**

**#include <sys/socket.h>**

**#include <netinet/in.h>**

**#include <arpa/inet.h>**

**#include <netdb.h>**

**#include <sys/epoll.h>**

**#include <assert.h>**

**#include <fcntl.h>**

**#include <sys/ioctl.h>**

**#include <pthread.h>**

**#define SIZE(a) (sizeof(a)/sizeof(a[0]))**

**typedef void (\*sighandler\_t)(int);**

**#endif**

执行命令vim func.h

sudo mv ~/func.h /usr/include/

后形成我们现在每次.c的头文件

执行命令 find ~ -name “sni\*.c”

vim 上条指令找到的文件路径

复制

**#include <func.h>**

**int main(int argc, char\* argv[])**

**{**

**printf("Hello world\n");**

**return 0;**

**}**

形成了我们当前采用的.c格式

别名文件：命令alias,可定义别名

Bash的配置文件：~/.bashrc

最后一行添加这些

if [ -n "$NVIM\_LISTEN\_ADDRESS" ]; then

export PS1='\[\033[01;32m\]\u@\h\[\033[00m\]:\[\033[01;34m\]\w\[\033[00m\]>> '

else

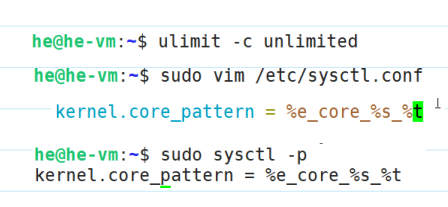
export PS1='\[\033[01;32m\]\u@\h\[\033[00m\]:\[\033[01;34m\]\w\[\033[00m\]\$ '

Fi

修改后执行source ~/.bashrc

形成老师一样的颜色界面

core文件（报错文件记录）的使用



Makefile文件

