

动态规划

动态规划一般均为求值问题，核心为穷举，存在最优子结构。子问题中含有大量的重复计算。

斐波那契数列

暴力求解：

```
int fib(int N) {
    if(1 == N || 2 == N) {
        return 1;
    }

    return fib(N-1) + fib(N-2);
}
```

备忘录方式：

```
int fib(int N){
    if(0 > N) {
        return -1;
    }
    std::vector<int> memo(N+1,0);
    return helper(memo, N);
}

int helper(std::vector<int> &memo, int N){
    // base case
    if(1 == N || 2 == N) {
        return 1;
    }

    if(0 != memo[N]) {
        return memo[N];
    }
    memo[N] = helper(memo, N-1) + helper(memo, N-2);
    return memo[N];
}
```

从备忘录方法可以推导出动态规划方法：

```
int fib(int N) {
    if(1 == N || 2 == N) {
        return 1;
    }
}
```

```
std::vector<int> dp(N+1, 0);  
// base case  
dp[0] = dp[1] = 1;  
for(int i = 2; i <= N; ++i) {  
    dp[i] = dp[i+1] + dp[i+2];  
}  
  
return dp[N];  
}
```

凑零钱问题

给你k种面值的硬币，面值分别为c1, c2 ... ck，每种硬币的数量无限，再给一个总金额amount，问你最少需要几枚硬币凑出这个金额，如果不可能凑出，算法返回 -1。

算法框架：

```
// coins 中是可选硬币面值，amount 是目标金额  
int coinChange(std::vector<int> &coins, int amount);
```

暴力求解：

```
int coinChange(std::vector<int> &coins, int amount) {  
    // amount == 0  
    if(0 == amount) {  
        return 0;  
    }  
  
    // amount < 0  
    if(0 > amount) {  
        return -1;  
    }  
  
    int res = INT_MAX;  
    for(auto coin : coins) { // 每次都是从整体的递归数组中获取余下的钱数  
        int subproblem = coinChange(coins, amount - coin);  
        if(-1 == subproblem) {  
            continue;  
        }  
  
        res = std::min(res, 1 + subproblem);  
    }  
  
    return res == INT_MAX ? -1 : res;  
}
```

备忘录方式：

```

int coinChange(std::vector<int> &coins, int amount) {
    if(amount < 0 ) {
        return -1;
    }
    std::vector<int> memo(amount+1, -1);
    return helper(coins, memo, amount);
}

int helper(std::vector<int> &coins, std::vector<int> &memo, int amount){

    if(-1 != memo[amount]) {
        return memo[amount];
    }

    // amount == 0
    if(0 == amount) {
        return 0;
    }

    // amount < 0
    if(0 > amount) {
        return -1;
    }

    int res = INT_MAX;
    for(auto coin : coins) {
        int subproblem = coinChange(coins, amount-coin);
        if(-1 == subproblem) {
            continue;
        }

        res = std::min(res, 1 + subproblem);
    }

    return memo[amount] = (res == INT_MAX ? -1 : res);
}

```

动态规划解法：

```

int coinChange(std::vector<int> &coins, int amount) {
    if(amount < 0 ) {
        return -1;
    }
    std::vector<int> dp(amount+1, amount+1);
    dp[0] = 0;
    for(int i=0;i<dp.size();++i){
        for(auto coin : coins){
            if(i - coin < 0) {
                continue;
            }
            dp[i] = std::min(dp[i], 1+dp[i-coin]);
        }
    }
}

```

```

    }
}
return dp[amount] == amount+1 ? -1:dp[amount];
}

```

凑零钱方法数

给定数组arr，arr中的数据都为正数且不重复，每个值代表一种货币面值，每种货币面值可以使用任意张，
在给定一个整数表示要找的零钱数。求找零钱的方法。

递归方法：

```

#include <vector>

class Solution {
public:
    // 递归解法
    int coinNum(std::vector<int> &nums, int aim) {
        int len = nums.size();
        if (0 == len || aim == 0) {
            return 0;
        }

        return coinNum(nums, aim, len, 0);
    }

private:
    int coinNum(std::vector<int> &nums, int aim, int len, int index) {
        int res = 0;

        if (index == len) {
            res = aim == 0 ? 1 : 0;
        } else {
            for (int i = 0; nums[index] * i <= aim; i++) {
                res += coinNum(nums, aim - nums[index] * i, len, index + 1);
            }
        }

        return res;
    }
};

```

备忘录优化：

```

#include <vector>

class Solution {
public:
    // 备忘录解法
    int coinNum(std::vector<int> &nums, int aim) {
        int len = nums.size();
        if (0 == len || aim == 0) {
            return 0;
        }

        std::vector<std::vector<int>> memo =
            std::vector<std::vector<int>>(len + 1, std::vector<int>(aim + 1,
-1));

        return coinNum(nums, aim, len, 0, memo);
    }

private:
    int coinNum(std::vector<int> &nums,
                int aim,
                int len,
                int index,
                std::vector<std::vector<int>> &memo) {
        int res = 0;

        if (index == len) {
            res = aim == 0 ? 1 : 0;
        } else {
            for (int i = 0; nums[index] * i <= aim; i++) {
                if (-1 != memo[index + 1][aim]) {
                    return memo[index + 1][aim];
                }
                res += coinNum(nums, aim - nums[index] * i, len, index + 1, memo);
            }
        }

        memo[index][aim] = res == 0 ? -1 : res;

        return res;
    }
};

```

动态规划解法：

```

class Solution {
public:
    int change(int amount, std::vector<int>& coins) {
        int len = coins.size();

        std::vector<std::vector<int>> memo(len + 1,

```

```
std::vector<int>(amount + 1, 0));

// base case
for (int i = 0; i <= len; i++) {
    memo[i][0] = 1;
}

// dp
for (int i = 1; i <= len; i++) {
    for (int j = 1; j <= amount; j++) {
        if (j - coins[i - 1] >= 0) {
            memo[i][j] = memo[i - 1][j] + memo[i][j - coins[i - 1]];
        } else {
            memo[i][j] = memo[i - 1][j];
        }
    }
}

return memo[len][amount];
};
```