记忆化搜索

01背包：https://www.lanqiao.cn/problems/19681/learning

dp求解组合数：https://www.lanqiao.cn/problems/19948/learning

数组分割：https://www.lanqiao.cn/problems/3535/learning

安全序列：https://www.lanqiao.cn/problems/3423/learning

选数异或：https://www.lanqiao.cn/problems/3711/learning

在对于有重复子问题的题目中，通过搜索的方式往往会导致大量重复计算，而记忆化搜索与动态规划算法的目的均是通过记录重复子问题，来避免重复计算。他们的时间复杂度是一致的。

1. **定义记忆化数组**
   根据问题的需求，定义一个数组（或其他数据结构）用于存储已计算子问题的结果。数组的维度通常与问题的状态空间一致。
2. **初始化记忆化数组**
   将记忆化数组初始化为一个特殊值（如 `-1`），用于表示该状态尚未被计算过。需要注意的是，在某些问题中（如数位 DP），状态值为 `0` 也可能是有效答案，因此初始化时需要根据具体问题选择合适的初始值。
3. **实现搜索函数**
   编写递归函数，在函数中首先检查当前状态是否已经被计算过（即记忆化数组中是否存储了有效值）。如果已经计算过，则直接返回存储的结果；否则，进行递归计算，并将结果保存到记忆化数组中，以便后续使用。
4. **调用搜索函数并返回结果**
   从初始状态调用搜索函数，最终返回记忆化搜索的结果。

记忆化搜索本质上还是考察搜索，因此回溯法一定要学好！

斐波拉契序列

```cpp
#include<bits/stdc++.h>
using namespace std;
int f[50];
int f(int n){
    if(n==1||n==2) return 1;
    if(f[n]!=-1) return f[n];
    return f[n]=f(n-1)+f(n-2);
}
int main(){
    memset(f,-1,sizeof(f));
    cout<<f(10);
}
```

```java
import java.util.Arrays;

public class Main {
    static int[] f = new int[50]; // 定义记忆化数组

    static int fib(int n) {
        if (n == 1 || n == 2) return 1; // 基础情况
        if (f[n] != -1) return f[n]; // 如果已经计算过，直接返回
```

```
        f[n] = fib(n - 1) + fib(n - 2); // 计算并保存结果
        return f[n];
    }

    public static void main(String[] args) {
        Arrays.fill(f, -1); // 初始化记忆化数组
        System.out.println(fib(10)); // 输出结果
    }
}
```

```
f = [-1] * 50  # 定义记忆化数组

def fib(n):
    if n == 1 or n == 2:
        return 1  # 基础情况
    if f[n] != -1:
        return f[n]  # 如果已经计算过，直接返回
    f[n] = fib(n - 1) + fib(n - 2)  # 计算并保存结果
    return f[n]

# 初始化记忆化数组（Python 中不需要显式初始化）
print(fib(10))  # 输出结果
```

01背包

```
#include<bits/stdc++.h>
using namespace std;
const int N=1010;
int w[N],v[N];
int dp[N][N];
int n,m;
//1.定义记忆化数组，参数和递归函数参数一样
//2.初始化记忆化数组为 -1
//3.搜索的时候记忆
//  - 在递归边界下判断当前状态是否访问过，访问过，直接返回
//  - 没访问过，正常搜索，记忆下来
// 时间复杂度=动态规划时间复杂度，空间不一定
int dfs(int u,int sum){
  //u代表当前选择的是第几个物品
  //sum表示当前的总体积
  //递归函数返回值代表最大价值
  if(u>n){
    return 0;
  }
  if(dp[u][sum]!=-1) return dp[u][sum];
  int ans1=0,ans2=dfs(u+1,sum);
  if(sum+v[u]<=m) ans1=dfs(u+1,sum+v[u])+w[u];
  return dp[u][sum]=max(ans1,ans2);
}
int main(){
    memset(dp,-1,sizeof(dp));
    cin>>n>>m;
    for(int i=1;i<=n;i++) cin>>v[i]>>w[i];
    cout<<dfs(1,0);
}
```

```java
import java.util.Scanner;

public class Main {
    static final int N = 1010; // 最大物品数量
    static int[] w = new int[N]; // 物品价值数组
    static int[] v = new int[N]; // 物品体积数组
    static int[][] dp = new int[N][N]; // 记忆化数组
    static int n, m; // 物品数量和背包容量

    // 记忆化搜索函数
    static int dfs(int u, int sum) {
        if (u > n) {
            return 0; // 递归边界：超出物品数量
        }
        if (dp[u][sum] != -1) {
            return dp[u][sum]; // 如果已经计算过，直接返回
        }
        int ans1 = 0, ans2 = dfs(u + 1, sum); // 不选当前物品
        if (sum + v[u] <= m) {
            ans1 = dfs(u + 1, sum + v[u]) + w[u]; // 选当前物品
        }
        return dp[u][sum] = Math.max(ans1, ans2); // 记忆化并返回结果
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        n = sc.nextInt(); // 输入物品数量
        m = sc.nextInt(); // 输入背包容量
        for (int i = 1; i <= n; i++) {
            v[i] = sc.nextInt(); // 输入物品体积
            w[i] = sc.nextInt(); // 输入物品价值
        }
        // 初始化记忆化数组
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++) {
                dp[i][j] = -1;
            }
        }
        System.out.println(dfs(1, 0)); // 输出结果
    }
}
```

```python
N = 1010  # 最大物品数量
w = [0] * N  # 物品价值数组
v = [0] * N  # 物品体积数组
dp = [[-1 for _ in range(N)] for _ in range(N)]  # 记忆化数组

# 记忆化搜索函数
def dfs(u, sum):
    if u > n:
        return 0  # 递归边界：超出物品数量
    if dp[u][sum] != -1:
        return dp[u][sum]  # 如果已经计算过，直接返回
    ans1, ans2 = 0, dfs(u + 1, sum)  # 不选当前物品
    if sum + v[u] <= m:
        ans1 = dfs(u + 1, sum + v[u]) + w[u]  # 选当前物品
    dp[u][sum] = max(ans1, ans2)  # 记忆化并返回结果
```

```python
    return dp[u][sum]

# 主程序
if __name__ == "__main__":
    n, m = map(int, input().split())  # 输入物品数量和背包容量
    for i in range(1, n + 1):
        v[i], w[i] = map(int, input().split())  # 输入物品体积和价值
    print(dfs(1, 0))  # 输出结果
```

数组分割

```cpp
#include <bits/stdc++.h>
using namespace std;
const int N=1e3+10;
int a[N];
int dp[N][2];
int n;
const int mod=1000000007;
int dfs(int u,int sum){//前u个数中选出奇偶性为1/0
    if(u>n){
        return sum==0;
    }
    if(dp[u][sum]!=-1) return dp[u][sum];
    int ans=0;
    if(a[u]%2==1){
        ans=ans+dfs(u+1,sum^1)+dfs(u+1,sum);
    }else{
        ans=ans+dfs(u+1,sum)+dfs(u+1,sum);
    }//可以合二为一，直接写成dfs(u+1,sum^(a[u]%2))+dfs(u+1,sum);
    return dp[u][sum]=ans%mod;
}
void solve(){
    int sum=0;
    scanf("%d",&n);
    for(int i=1;i<=n;i++) dp[i][0]=dp[i][1]=-1;
    for(int i=1;i<=n;i++){
        int x;
        scanf("%d",&x);
        x%=2;
        a[i]=x;
        sum+=a[i];
    }
    if(sum%2==1){
        printf("0\n");
        return;
    }
    printf("%d\n",dfs(1,0));
}
int main() {
    int t;
    scanf("%d",&t);
    while(t--)  solve();
    return 0;
}
```

```java
import java.util.Scanner;
```

```java

public class Main {
    static final int N = 1010; // 最大范围
    static final int MOD = 1000000007; // 定义模数
    static int[] a = new int[N]; // 存储数组
    static int[][] dp = new int[N][2]; // 记忆化数组
    static int n; // 数组长度

    // 记忆化搜索函数
    static int dfs(int u, int sum) {
        if (u > n) {
            return sum == 0 ? 1 : 0; // 递归边界：前 n 个数中选出奇偶性为 sum
        }
        if (dp[u][sum] != -1) {
            return dp[u][sum]; // 如果已经计算过，直接返回
        }
        int ans = (dfs(u + 1, sum ^ (a[u] % 2)) + dfs(u + 1, sum)) % MOD;
// 计算并记忆化
        return dp[u][sum] = ans;
    }

    static void solve() {
        Scanner sc = new Scanner(System.in);
        n = sc.nextInt(); // 输入数组长度
        for (int i = 1; i <= n; i++) {
            dp[i][0] = dp[i][1] = -1; // 初始化记忆化数组
        }
        int sum = 0;
        for (int i = 1; i <= n; i++) {
            int x = sc.nextInt() % 2; // 输入数字并取模 2
            a[i] = x;
            sum += x;
        }
        if (sum % 2 == 1) {
            System.out.println("0"); // 如果总和为奇数，直接输出 0
            return;
        }
        System.out.println(dfs(1, 0)); // 输出结果
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int t = sc.nextInt(); // 输入测试用例数量
        while (t-- > 0) {
            solve(); // 处理每个测试用例
        }
    }
}
```

```python
MOD = 1000000007  # 定义模数
N = 1010  # 最大范围

# 记忆化搜索函数
def dfs(u, sum):
    if u > n:
        return 1 if sum == 0 else 0  # 递归边界：前 n 个数中选出奇偶性为 sum
    if dp[u][sum] != -1:
```

```
        return dp[u][sum]  # 如果已经计算过，直接返回
    ans = (dfs(u + 1, sum ^ (a[u] % 2)) + dfs(u + 1, sum)) % MOD  # 计算并记忆化
    dp[u][sum] = ans
    return dp[u][sum]

def solve():
    global n, a, dp
    n = int(input())  # 输入数组长度
    a = [0] * (n + 1)  # 存储数组
    dp = [[-1 for _ in range(2)] for _ in range(n + 1)]  # 初始化记忆化数组
    sum_val = 0
    nums = list(map(int, input().split()))  # 输入数组
    for i in range(1, n + 1):
        x = nums[i - 1] % 2  # 输入数字并取模 2
        a[i] = x
        sum_val += x
    if sum_val % 2 == 1:
        print("0")  # 如果总和为奇数，直接输出 0
        return
    print(dfs(1, 0))  # 输出结果

if __name__ == "__main__":
    t = int(input())  # 输入测试用例数量
    for _ in range(t):
        solve()  # 处理每个测试用例
```

dp求解组合数

```
#include <bits/stdc++.h>
using namespace std;
const int mod=1e9+7;
const int N=3010;
int C(int n,int m){//记忆化搜索
//3步
//1.定义记忆化数组：参数=dfs参数
//2.初始化记忆化数组，将数组中所有的值赋值成-1，一般只要初始化一次
//3.对于dfs程序，判断是否访问过 访问过直接返回记忆化数组，否则正常搜索，
//结果存入记忆化
  if(n==m||m==0) return 1;
  return (C(n-1,m)+C(n-1,m-1))%mod;
}
int main()
{
  int n,m;
  cin>>n>>m;
  cout<<C(n,m);
  return 0;
}
```

```
import java.util.Scanner;

public class Main {
    static final int MOD = (int) 1e9 + 7; // 定义模数

    // 计算组合数 C(n, m)
```

```java
    static int C(int n, int m) {
        if (n == m || m == 0) {
            return 1; // 递归边界：C(n, n) = C(n, 0) = 1
        }
        return (C(n - 1, m) + C(n - 1, m - 1)) % MOD; // 递归计算
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt(); // 输入 n
        int m = sc.nextInt(); // 输入 m
        System.out.println(C(n, m)); // 输出结果
    }
}
```

```python
MOD = 10**9 + 7  # 定义模数

# 计算组合数 C(n, m)
def C(n, m):
    if n == m or m == 0:
        return 1  # 递归边界：C(n, n) = C(n, 0) = 1
    return (C(n - 1, m) + C(n - 1, m - 1)) % MOD  # 递归计算

# 主程序
if __name__ == "__main__":
    n, m = map(int, input().split())  # 输入 n 和 m
    print(C(n, m))  # 输出结果
```

安全序列

```cpp
#include <bits/stdc++.h>
using namespace std;
const int N=1e6+10;
int n,k;
const int mod=1000000007;
int dfs(int u){
    if(u>n){
        return 1;
    }
    int ans=dfs(u+1)+dfs(u+k+1);
    return ans%mod;
}
void solve(){
    scanf("%d%d",&n,&k);
    printf("%d",dfs(1));
}
int main() {
    int t=1;
    while(t--)  solve();
    return 0;
}
```

```java
import java.util.Scanner;

public class Main {
```

```java
    static final int MOD = 1000000007; // 定义模数
    static int n, k; // 输入参数

    // 递归函数
    static int dfs(int u) {
        if (u > n) {
            return 1; // 递归边界：超出范围
        }
        int ans = (dfs(u + 1) + dfs(u + k + 1)) % MOD; // 递归计算
        return ans;
    }

    static void solve() {
        Scanner sc = new Scanner(System.in);
        n = sc.nextInt(); // 输入 n
        k = sc.nextInt(); // 输入 k
        System.out.println(dfs(1)); // 输出结果
    }

    public static void main(String[] args) {
        int t = 1; // 测试用例数量
        while (t-- > 0) {
            solve(); // 处理每个测试用例
        }
    }
}
```

```python
MOD = 10**9 + 7  # 定义模数

# 递归函数
def dfs(u):
    if u > n:
        return 1  # 递归边界：超出范围
    ans = (dfs(u + 1) + dfs(u + k + 1)) % MOD  # 递归计算
    return ans

def solve():
    global n, k
    n, k = map(int, input().split())  # 输入 n 和 k
    print(dfs(1))  # 输出结果

if __name__ == "__main__":
    t = 1  # 测试用例数量
    for _ in range(t):
        solve()  # 处理每个测试用例
```