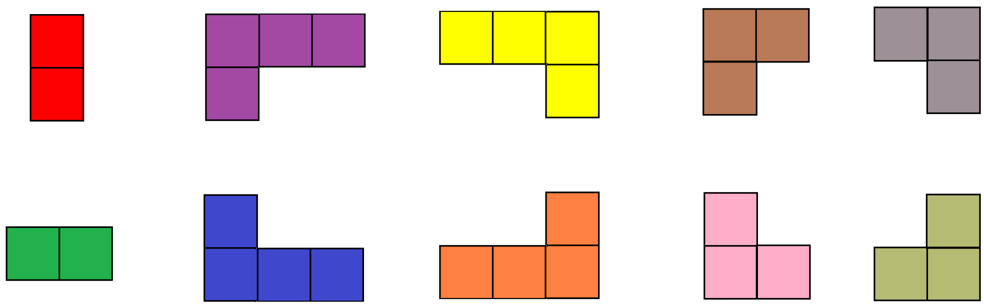
E2.B 比较简单的铺瓷砖

题目描述

有一个宽度为 2，长度为 n 的地板需要贴上瓷砖，有以下 10 种形状的瓷砖可供选择：



瓷砖不能**旋转、翻转、重叠、切割**，且必须恰好把地板**铺满**，可以结合输入输出样例理解。

the_ignorant 想知道一共有多少种不同的铺瓷砖方案，两种方案不同当且仅当存在一个位置上的瓷砖颜色不同。由于答案可能很大，你只需要输出答案对 998244353 取模后的结果。

输入格式

第一行一个正整数 t（$1 \le t \le 10^6$），表示数据组数。

对于每组数据，一行一个正整数 n（$1 \le n \le 10^6$），表示地板的长度。

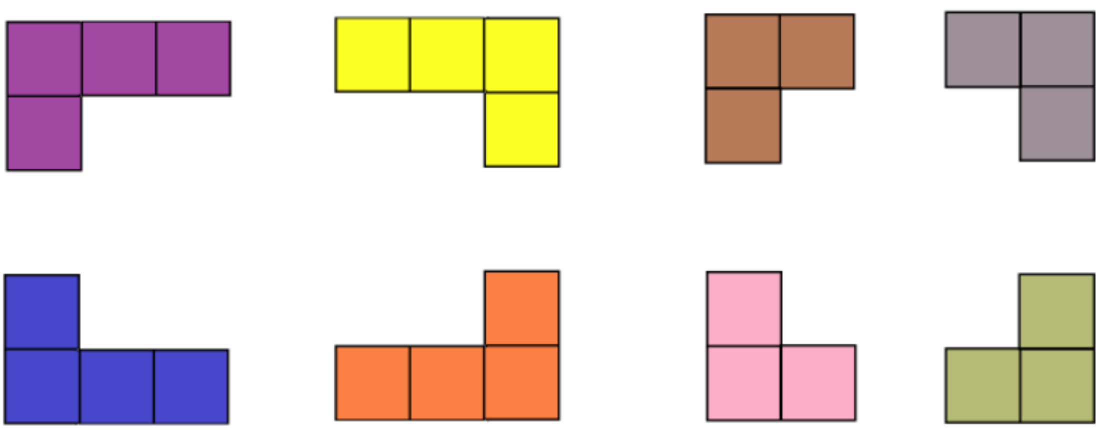
输出格式

对于每组数据，输出一行一个非负整数，表示贴满瓷砖的方案数对 998244353 取模后的结果。

题目解析

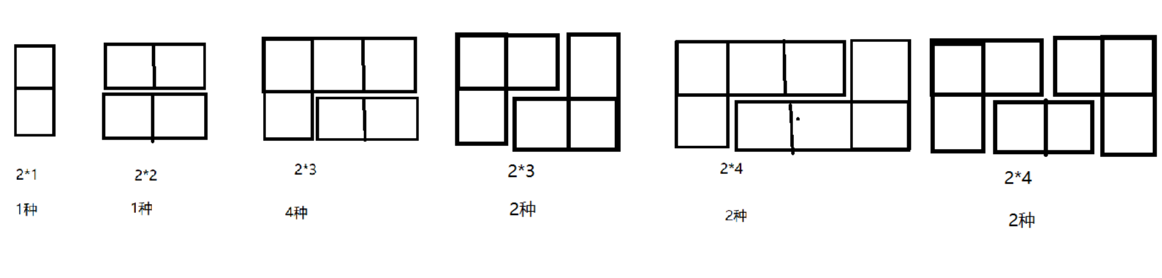
本题要求贴满瓷砖的方案数，为计数问题，题目提示算法为“递推”。尽管并不涉及最优化，但依旧可以使用动态规划的思想进行求解。分析如下：

题目要求将2×n的地板铺满，不能留存缝隙，但题给的以下八种瓷砖都缺了一个角，在铺设时必须被其他瓷砖的突出部“插进来”，以填满缺角带来的缝隙。



不难想到如下方法：我们先将瓷砖都**补成2×m的基本组合**（1 ≤ m ≤ n）（基本组合是指在任意位置竖着切一刀，都不会切断瓷砖的矩形组合），将这些组合作为铺瓷砖的基本单元，如此一来，我们不必再考虑瓷砖具体形状带来的困扰。同时问题转化为：求**在长为n的一维地板上铺设基本单元的方案种数**，我们只需要做一位的递推。

接下来的工作是确定基本单元的形状和种数。在考场上很容易想到如下基本组合：

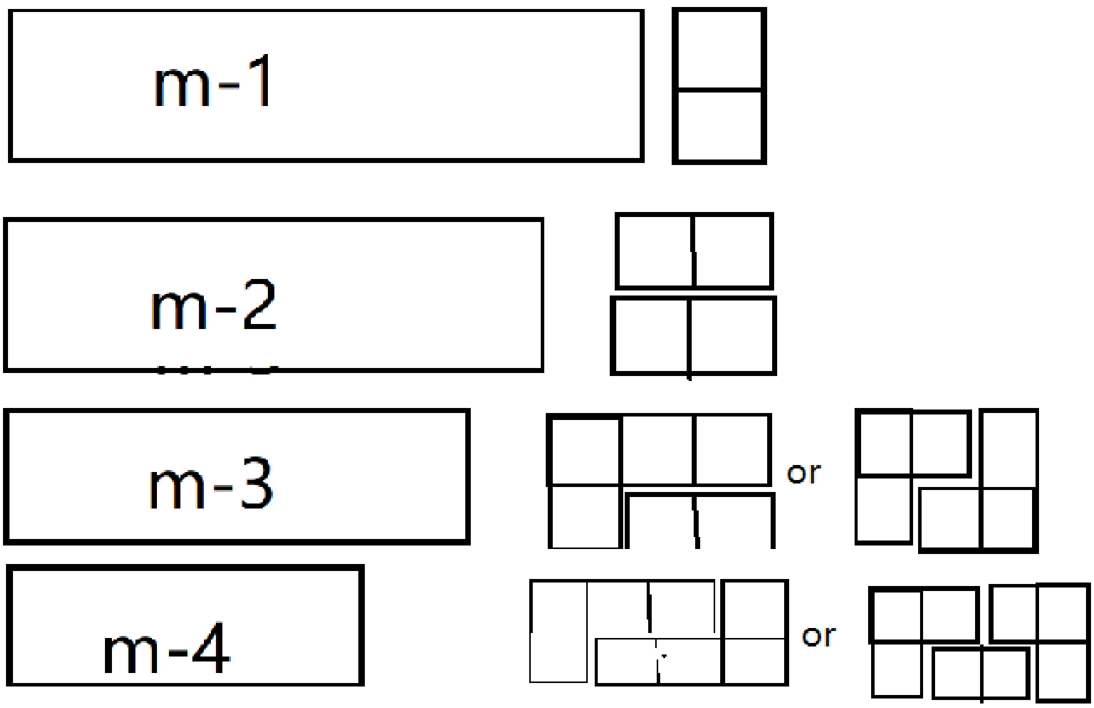


有了这些2×m基本单元，我们就可以开始愉快地铺瓷砖了：

设铺满2×n的地板有f_n种方案，如果暴力枚举所有可能的场合，时间复杂度显然是指数级别的。考虑到题目的数据范围，根据做题的经验以及题目提示，我们最好在$O(n)$的时间内求解$f_1, f_2, \ldots\ldots, f_n$。

容易看出$f_1, f_2, \ldots\ldots, f_n$都可以独立求解，因此可以考虑利用动态规划的思想进行递推：**在求解f_m时，我们假设$f_1, f_2, \ldots\ldots, f_{m-1}$都已经求解完毕，并考虑如何利用$f_1, f_2, \ldots\ldots, f_{m-1}$来表示$f_m$**，即状态转移方程。

一次只允许铺设一个基本单元（否则会出现重复计数），不难发现状态m可以由以下方式转移而来：



于是得到状态转移方程：

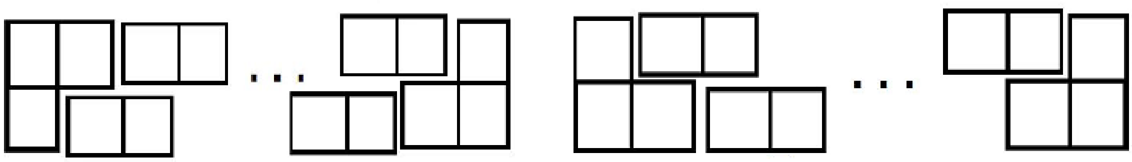
$$f_m = f_{m-1} + f_{m-2} + 6 * f_{m-3} + 4 * f_{m-4}$$

同时规定边界条件$f_k = 0 (k < 0), f_0 = 0$。

于是我们连样例都没过（当然有的同学可能又多想了几种基本情况，过了样例但是WA）。我们得到的答案偏小，这说明我们漏掉了一些情况。

有些部分同学在此题浪费了很多时间，尤其是未使用递推，而是使用记忆化搜索的同学（虽然这两者本质的逻辑是一样的，而且如果难以写出状态转移方程时，用搜索可能更有利于快速解题），他们怀疑是自己的代码实现出现了漏洞，而非算法有问题。事实上，当在考场上遇到TLE或WA等情况时，除非当天做题状态真的很差，否则在短暂debug后，我们应当相信自己的C语言基础，确定是算法出现了问题，然后冷静且大胆地去重新思考，查找算法的漏洞，甚至是推翻重来。更何况这是B题，我们有理由相信自己的能力足以拿下。

事实上，我们的基本单元还可以是如下组成：



2*k (k>=5且为奇数)

2种

2*k (k>=6且为偶数)

2种

于是状态转移方程修改为：

$$f_m = f_{m-1} + f_{m-2} + 6 * f_{m-3} + 4 * f_{m-4} + 2 * f_{m-5} + 2 * s_{m-5}$$

即：

$$f_m = f_{m-1} + f_{m-2} + 4 * f_{m-3} + 2 * f_{m-4} + 2 * s_{m-3}$$

其中$s_m = \sum_0^m f_i$

算法时间复杂度为$O(n)$。

代码实现

一种可能的代码实现如下：

```
#include<stdio.h>
#define ll long long int
#define M 1000100

ll MOD=998244353;
ll result[M]={1, 1, 2, 9, 21, 48};
ll sum[M]={1, 2, 4, 13, 34, 82};
int main()
{
    int t, n, i;
    scanf("%d",&t);
    for(i=6; i<M; i++)
    {
        result[i] = (result[i-1] + result[i-2] + 4*result[i-3] + 2*result[i-4]
+2*sum[i-3])%MOD;
        sum[i] = (sum[i-1] + result[i])%MOD;
    }
    for(i=0;i<t;i++)
    {
        scanf("%d",&n);
```

```
        printf("%lld\n",result[n]);
    }
    return 0;
}
```

本题难度不是太高，随便分析下就给方程的话真的就没啥可说的了，因此较为详细地展示了思考和debug的全过程

AUTHOR: 20375331