#include <assert.h>

#include <ctype.h>

#include <limits.h>

#include <math.h>

#include <stdbool.h>

#include <stddef.h>

#include <stdint.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MOD 1000000007

char\* readline();

char\* ltrim(char\*);

char\* rtrim(char\*);

char\*\* split\_string(char\*);

int parse\_int(char\*);

/\*

 \* Complete the 'legoBlocks' function below.

 \*

 \* The function is expected to return an INTEGER.

 \* The function accepts following parameters:

 \*  1. INTEGER n

 \*  2. INTEGER m

 \*/

long long powmod(long long base,long long exp){

    long long res=1;

    while(exp>0){

        if(exp & 1) res = (res\* base)% MOD;

        base=(base\*base)%MOD;

        exp>>=1;

    }

    return res;

}

int legoBlocks(int n, int m) {

static long long row[1001];

static long long total[1001];

static long long solid[1001];

static int initialized=0;

if(!initialized){

    row[0]=1;

    for(int w=1;w<=1000;w++){

        row[w]=0;

        for(int b=1;b<=4;b++){

            if(w-b>=0)

            row[w]=(row[w]+row[w-b])%MOD;

        }

    }

    initialized=1;

}

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

    total[i]=powmod(row[i],n);

}

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

    solid[i]=total[i];

    for(int j=1;j<i;j++){

        solid[i]=(solid[i]-(solid[j]\*total[i-j])%MOD+MOD)%MOD;

    }

}

return (int)solid[m];

}

int main()

{

    FILE\* fptr = fopen(getenv("OUTPUT\_PATH"), "w");

    int t = parse\_int(ltrim(rtrim(readline())));

    for (int t\_itr = 0; t\_itr < t; t\_itr++) {

        char\*\* first\_multiple\_input = split\_string(rtrim(readline()));

        int n = parse\_int(\*(first\_multiple\_input + 0));

        int m = parse\_int(\*(first\_multiple\_input + 1));

        int result = legoBlocks(n, m);

        fprintf(fptr, "%d\n", result);

    }

    fclose(fptr);

    return 0;

}

char\* readline() {

    size\_t alloc\_length = 1024;

    size\_t data\_length = 0;

    char\* data = malloc(alloc\_length);

    while (true) {

        char\* cursor = data + data\_length;

        char\* line = fgets(cursor, alloc\_length - data\_length, stdin);

        if (!line) {

            break;

        }

        data\_length += strlen(cursor);

        if (data\_length < alloc\_length - 1 || data[data\_length - 1] == '\n') {

            break;

        }

        alloc\_length <<= 1;

        data = realloc(data, alloc\_length);

        if (!data) {

            data = '\0';

            break;

        }

    }

    if (data[data\_length - 1] == '\n') {

        data[data\_length - 1] = '\0';

        data = realloc(data, data\_length);

        if (!data) {

            data = '\0';

        }

    } else {

        data = realloc(data, data\_length + 1);

        if (!data) {

            data = '\0';

        } else {

            data[data\_length] = '\0';

        }

    }

    return data;

}

char\* ltrim(char\* str) {

    if (!str) {

        return '\0';

    }

    if (!\*str) {

        return str;

    }

    while (\*str != '\0' && isspace(\*str)) {

        str++;

    }

    return str;

}

char\* rtrim(char\* str) {

    if (!str) {

        return '\0';

    }

    if (!\*str) {

        return str;

    }

    char\* end = str + strlen(str) - 1;

    while (end >= str && isspace(\*end)) {

        end--;

    }

    \*(end + 1) = '\0';

    return str;

}

char\*\* split\_string(char\* str) {

    char\*\* splits = NULL;

    char\* token = strtok(str, " ");

    int spaces = 0;

    while (token) {

        splits = realloc(splits, sizeof(char\*) \* ++spaces);

        if (!splits) {

            return splits;

        }

        splits[spaces - 1] = token;

        token = strtok(NULL, " ");

    }

    return splits;

}

int parse\_int(char\* str) {

    char\* endptr;

    int value = strtol(str, &endptr, 10);

    if (endptr == str || \*endptr != '\0') {

        exit(EXIT\_FAILURE);

    }

    return value;

}