扑克牌（poker）

【题目描述】

小 P 从同学小 Q 那儿借来一副 n 张牌的扑克牌。

本题中我们不考虑大小王，此时每张牌具有两个属性：花色和点数。花色共有 4 种：方片、草花、红桃和黑桃。点数共有 13 种，从小到大分别为 `A 2 3 4 5 6 7 8 9 T J Q K`。注意：点数 `10` 在本题中记为 `T`。

我们称一副扑克牌是完整的，当且仅当对于每一种花色和每一种点数，都恰好有一张牌具有对应的花色和点数。由此，一副完整的扑克牌恰好有 $4 \times 13 = 52$ 张牌。以下图片展示了一副完整的扑克牌里所有的 52 张牌。

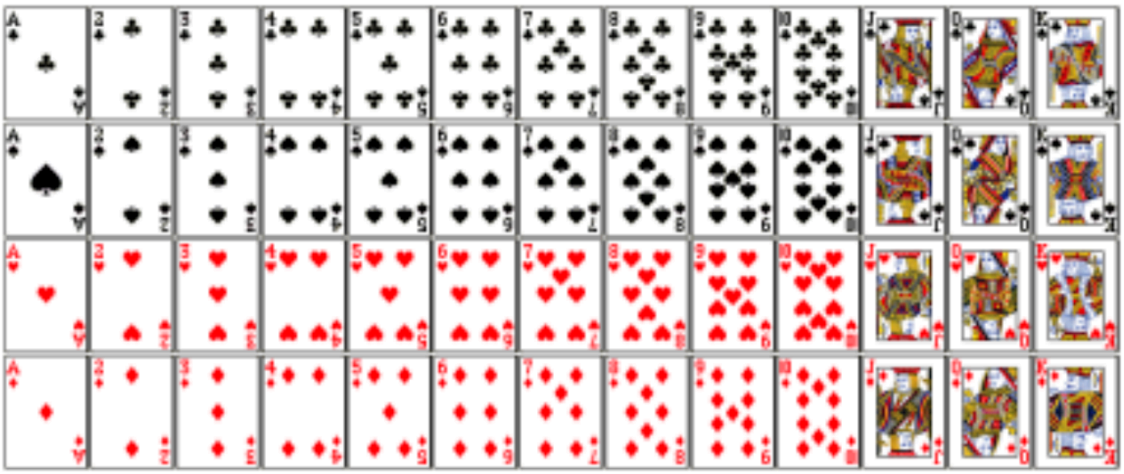


图 1: 一副完整的扑克牌

小 P 借来的牌可能不是完整的，为此小 P 准备再向同学小 S 借若干张牌。可以认为小 S 每种牌都有无限张，因此小 P 可以任意选择借来的牌。小 P 想知道他至少得向小 S 借多少张牌，才能让从小 S 和小 Q 借来的牌中，可以选出 52 张牌构成一副完整的扑克牌。

为了方便你的输入，我们使用字符 `D` 代表方片，字符 `C` 代表草花，字符 `H` 代表红桃，字符 `S` 代表黑桃，这样每张牌可以通过一个长度为 2 的字符串表示，其中第一个字符表示这张牌的花色，第二个字符表示这张牌的点数，例如 `CA` 表示草花 `A`，`ST` 表示黑桃 `T`（黑桃 `10`）。

题目大意

一副扑克牌共有4种花色13种点数，给你n张牌，请你求出还缺少哪些牌。

解题思路

根据输入的字符串，直接判断有多少张不同的牌，然后输出52减去已有的牌数即可。**判断不同的牌可以使用双重循环和字符串比较，也可以使用map、桶、unique等方法实现。**方法多样，自由发挥即可，没有特殊要考虑的坑点。

难度

入门

参考代码

```
#include <bits/stdc++.h>
using namespace std;
string s[55];
int main()
{
    int n;
    cin >> n;
    int cnt=0;
    for(int i=0; i<n; ++i){
        cin >> s[i];
        bool flag = true;
        for(int j=0;j<i;j++)if(s[j]==s[i])flag=false;
        if(flag)cnt++;
    }
    cout << 52-cnt;
    return 0;
}
```

地图探险（explore）

【题目描述】

小 A 打算前往一片丛林去探险。丛林的地理环境十分复杂，为了防止迷路，他先派遣了一个机器人前去探路。

丛林的地图可以用一个 n 行 m 列的字符表来表示。我们将第 i 行第 j 列的位置的坐标记作 $(i,j)(1 \le i \le n, 1 \le j \le m)$。如果这个位置的字符为 `x`，即代表这个位置上有障碍，不可通过。反之，若这个位置的字符为 `.`，即代表这个位置是一片空地，可以通过。

这个机器人的状态由位置和朝向两部分组成。其中位置由坐标 $(x,y)(1 \le x \le n, 1 \le y \le m)$ 刻画，它表示机器人处在地图上第 x 行第 y 列的位置。而朝向用一个 $0 \sim 3$ 的整数 d 表示，其中 $d = 0$ 代表向东，$d = 1$ 代表向南，$d = 2$ 代表向西，$d = 3$ 代表向北。

初始时，机器人的位置为 (x_0, y_0)，朝向为 d_0。保证初始时机器人所在的位置为空地。接下来机器人将要进行 k 次操作。每一步，机器人将按照如下的模式操作：

1. 假设机器人当前处在的位置为 (x,y)，朝向为 d。则它的方向上的下一步的位置 (x',y') 定义如下：若 $d = 0$，则令 $(x',y') = (x,y+1)$，若 $d = 1$，则令 $(x',y') = (x+1,y)$，若 $d = 2$，则令 $(x',y') = (x,y-1)$，若 $d = 3$，则令 $(x',y') = (x-1,y)$。
2. 接下来，机器人判断它下一步的位置是否在地图内，且是否为空地。具体地说，它判断 (x',y') 是否满足 $1 \le x' \le n, 1 \le y' \le m$，且 (x',y') 位置上是空地。如果条件成立，则机器人会向前走一步。它新的位置变为 (x',y')，且朝向不变。如果条件不成立，则它会执行“向右转”操作。也就是说，令 $d' = (d+1) \bmod 4$（即 $d+1$ 除以 4 的余数），且它所处的位置保持不变，但朝向由 d 变为 d'。

小 A 想要知道，在机器人执行完 k 步操作之后，地图上所有被机器人经过的位置（包括起始位置）有几个。

题目大意

在一个n行m列的二维棋盘上有障碍物，机器人初始在(x0，y0)位置，初始方向为d0。机器人要执行k次操作，如果前方可以移动，则移动一步；否则向右转。

解题思路

直接模拟即可，严格按照题目的意思来。题目中已经给出了每个方向对应的坐标变化，这是方向数组。**直接模拟k次操作即可。**

难度

普及-

参考代码

```
#include <bits/stdc++.h>
using namespace std;
int dx[]={0,1,0,-1}, dy[]={1,0,-1,0};
char mp[1005][1005];
bool used[1005][1005];
int main()
{
    int T;
    cin >> T;
    while(T--)
    {
        memset(used, 0, sizeof(used));
        int n, m, k, x, y, d;
        cin >> n >> m >> k >> x >>y >> d;
        used[x][y]=1;
        for(int i=1; i<=n; ++i){
            for(int j=1;j<=m;j++){
                cin>>mp[i][j];
            }
        }
        for(int i=1;i<=k;i++)
        {
            int xx = x+dx[d];
            int yy = y+dy[d];
            if(!(xx>=1&&xx<=n&&yy>=1&&yy<=m&&mp[xx][yy]=='.'))
            {
                d=(d+1)%4;
            }else{
                x=xx;
                y=yy;
            }
```

直接模拟即可，严格按照题目的意思来。题目中已经给出了每个方向对应的坐标变化，这是方向数组。**直接模拟k次操作即可。**

难度

普及-

参考代码

```
#include <bits/stdc++.h>
using namespace std;
int dx[]={0,1,0,-1}, dy[]={1,0,-1,0};
char mp[1005][1005];
bool used[1005][1005];
int main()
{
    int T;
    cin >> T;
    while(T--)
    {
        memset(used, 0, sizeof(used));
        int n, m, k, x, y, d;
        cin >> n >> m >> k >> x >>y >> d;
        used[x][y]=1;
        for(int i=1; i<=n; ++i){
            for(int j=1;j<=m;j++){
                cin>>mp[i][j];
            }
        }
        for(int i=1;i<=k;i++)
        {
            int xx = x+dx[d];
            int yy = y+dy[d];
            if(!(xx>=1&&xx<=n&&yy>=1&&yy<=m&&mp[xx][yy]=='.'))
            {
                d=(d+1)%4;
            }else{
                x=xx;
                y=yy;
            }
            used[x][y]=1;
        }
        }
        int ans=0;
        for(int i=1; i<=n; ++i)
        {
            for(int j=1; j<=m; ++j)
            {
                if(used[i][j])ans++;
            }
        }
        cout << ans << endl;
    }
    return 0;
}
```

保存整页为图片 >

小木棍（sticks）

【题目描述】

小 S 喜欢收集小木棍。在收集了 n 根长度相等的小木棍之后，他闲来无事，便用它们拼起了数字。用小木棍拼每种数字的方法如下图所示。



图 2: 每种数字的小木棍拼法

现在小 S 希望拼出一个正整数，满足如下条件：

- 拼出这个数恰好使用 n 根小木棍；
- 拼出的数没有前导 0；
- 在满足以上两个条件的前提下，这个数尽可能小。

小 S 想知道这个数是多少，可 n 很大，把木棍整理清楚就把小 S 折腾坏了，所以你需要帮他解决这个问题。如果不存在正整数满足以上条件，你需要输出 -1 进行报告。

题目大意

T组询问，每次询问恰好使用n根火柴能拼出的最小的不含前导0的数字，如果不行，输出-1。

解题思路

为了使数字尽可能小，应当位数尽可能少。**数字8会消耗7根火柴，因此应尽量使用数字8。**根据n % 7的结果，补充不同的数字。可以通过打表或观察部分分中的描述来发现规律。

规律：

- 1 -> 2
- 10 -> 8 % 7 = 1
- 2 -> 5
- 20 -> 11 % 7 = 4
- 200 -> 17 % 7 = 3
- 6 -> 6

对于小范围的n，可以直接使用数位分离的技巧进行判断。

难度

普及

参考代码

```
#include<iostream>
#include<cstring>
using namespace std;
```

参考代码

```
#include<iostream>
#include<cstring>
using namespace std;
int num[110000];
int l[10]={6,2,5,5,4,5,6,3,7,6};
int calc(int x){
    if(x==0)return l[0];
    int ans = 0;
    while(x){
        ans += l[x%10];
        x/=10;
    }
    return ans;
}
int main(){
    int n;
    memset(num,-1,sizeof(num));
    for(int i=1;i<=100000;i++){
        num[i]=calc(i);
    }
    int T;
    cin>>T;
    while(T--){
        cin >> n;
        if(n<=30){
            bool flag = true;
            for(int i=1;i<=100000;i++){
                if(num[i]==n){
                    cout<<i<<endl;
                    flag=false;
                    break;
                }
            }
            if(flag)cout<<-1<<endl;
        }else{
            if(n%7==1){
                cout << 10;
                n-=8;
            }else if(n%7==2){
                cout<<1;
```

```
    memset(num,-1,sizeof(num));
    for(int i=1;i<=100000;i++){
        num[i]=calc(i);
    }
    int T;
    cin>>T;
    while(T--){
        cin >> n;
        if(n<=30){
            bool flag = true;
            for(int i=1;i<=100000;i++){
                if(num[i]==n){
                    cout<<i<<endl;
                    flag=false;
                    break;
                }
            }
            if(flag)cout<<-1<<endl;
        }else{
            if(n%7==1){
                cout << 10;
                n-=8;
            }else if(n%7==2){
                cout<<1;
                n-=2;
            }else if(n%7==3){
                cout<<200;
                n-=17;
            }else if(n%7==4){
                cout<<20;
                n-=11;
            }else if(n%7==5){
                cout<<2;
                n-=5;
            }else if(n%7==6){
                cout<<6;
                n-=6;
            }
            for(int i=1;i*7<=n;i++)cout<<8;
            cout << endl;
        }
    }
}
```

保存整页为图片 >

接龙（chain）

【题目描述】

在玩惯了成语接龙之后，小 J 和他的朋友们发明了一个新的接龙规则。

总共有 n 个人参与这个接龙游戏，第 i 个人会获得一个整数序列 S_i 作为他的词库。一次游戏分为若干轮，每一轮规则如下：

- n 个人中的某个人 p 带着他的词库 S_p 进行接龙。若这不是游戏的第一轮，那么这一轮进行接龙的人不能与上一轮相同，但可以与上上轮或更往前的轮相同。
- 接龙的人选择一个长度在 $[2, k]$ 的 S_p 的连续子序列 A 作为这一轮的**接龙序列**，其中 k 是给定的常数。若这是游戏的第一轮，那么 A 需要以元素 1 开头，否则 A 需要以上一轮的接龙序列的最后一个元素开头。
  - 序列 A 是序列 S 的连续子序列当且仅当可以通过删除 S 的开头和结尾的若干元素（可以不删除）得到 A。

为了强调合作，小 J 给了 n 个参与游戏的人 q 个任务，第 j 个任务需要这 n 个人进行一次游戏，在这次游戏里进行恰好 r_j 轮接龙，且最后一轮的接龙序列的最后一个元素恰好为 c_j。为了保证任务的可行性，小 J 请来你判断这 q 个任务是否可以完成的，即是否存在一个可能的游戏过程满足任务条件。

题目大意

有n个人，每个人有一个数字序列，q次询问，每次询问这n个人能否完成r轮接龙，并且最后一轮接龙的最后一个数字是c。接龙规则为每个人拿出自己数字序列中的一个连续子序列，要求开头和上一个序列的结尾一致，且不能同一个人连续接龙，选出的子序列长度在2到k之间。

解题思路

首先，q次询问每次r轮和数字c结尾，显然不能每次询问单独做，**所以需要有一个存答案的数组bool ans[r][c]来表示。**接下来考虑如何求出ans数组。第r轮的结果依赖于上一轮选择的是谁，因此可以采用动态规划的方法来求解。

定义dp[i][j]表示前i轮以第j个数字结尾是否可行。求出dp数组后再利用dp数组去求ans数组。这里同时处理题目的一个特殊情况，每个人的数字序列长度之和最多为2e5，**因此可以使用一个一维数组存储出，另外记录一个一维数组from分别表示每个数字是第几个人的手牌。**在转移dp[i][j]时，需要通过第j张牌前面的2到k张范围内是否有上一轮可以到达的情况，这个使用前

2e5，**因此可以使用一个一维数组存储出，另外记录一个一维数组from分别表示每个数字是第几个人的手牌。**在转移dp[i][j]时，需要通过第j张牌前面的2到k张范围内是否有上一轮可以到达的情况，这个使用前缀和的方法处理。另外，当一张牌只有一个人可以接龙，那么下一轮不能让他接龙；若有两人以上可以接龙，则每个人都可以下一轮接龙。

难度

提高

参考代码

```
#include<iostream>
#include<cstring>
using namespace std;
bool dp[105][200005];
bool ans[105][200005];
int from[200005], a[200005], pre[200005], s[200005];
int main()
{
    int T;
    cin >> T;
    while(T--)
    {
        memset(dp, 0, sizeof(dp));
        memset(s, 0, sizeof(s));
        s[1]=1e9;
        memset(ans, 0, sizeof(ans));
        int n, k, q, tot=0;
        cin >> n >> k >> q;
        for(int i=1; i<=n; ++i)
        {
            int t;
            cin >> t;
            t--;
            cin >> a[++tot];
            if(a[tot]==1) dp[0][tot]=1;
            from[tot]=tot;
            while(t--)
            {
                cin >> a[++tot];
                if(a[tot]==1) dp[0][tot]=1;
                from[tot]=from[tot-1];
            }
        }
        for(int i=1; i<=100; ++i)
        {
```

```
            if(a[tot]==1) dp[0][tot]=1;
            from[tot]=tot;
            while(t--)
            {
                cin >> a[++tot];
                if(a[tot]==1) dp[0][tot]=1;
                from[tot]=from[tot-1];
            }
        }
        for(int i=1; i<=100; ++i)
        {
            for(int j=1; j<=tot; ++j)
            {

                pre[j]=pre[j-1]+(s[a[j]] && s[a[j]]!=from[j]);
            }
            memset(s, 0, sizeof(s));
            for(int j=1; j<=tot; ++j)
            {
                dp[i][j]=(pre[j-1]-pre[max(from[j], j-k+1)-1]!=0);
                if(dp[i][j])
                {
                    ans[i][a[j]] = 1;
                    if(!s[a[j]]) s[a[j]]=from[j];
                    else if(s[a[j]]!=from[j]) s[a[j]]=1e9;
                }
            }
        }
        while(q--)
        {
            int r, c;
            cin >> r >> c;
            cout << ans[r][c] << endl;
        }
    }
    return 0;
}
```