T1 天才拆分

子任务 1~5　(50分)

$k=1$，所以只需要预处理n以内的各位是9的数字，判断读入的数是否在里面即可

子任务 6（100分)

观察得到，如果一个数是 k 阶天才数，那么它一定是 k 个 9 的倍数。比如 2 阶天才数 9999，$9999=9900+99=101*99$，同理$999999=990000+9900+99=x*99$。所以只需要判断n是不是 k 个 9 的倍数即可。

```
#include <bits/stdc++.h>
using namespace std;
long long t, k, n, p[15] = {0, 9, 99, 999, 9999,
99999, 999999, 9999999, 99999999, 999999999,
9999999999, 99999999999};

int main() {
    cin >> t;
    while (t--) {
        cin >> k >> n;
        if (n % p[k] == 0)
            cout << "aya\n";
        else
```

```
            cout << "baka\n";
    }
    return 0;
}
```

T2 自学

子任务 1 (10分)

$m = 1$的时候，每个任务只够一天学的(否则答案为0)，所以遍历n个课程，取最小的$max(A_i, B_i)$即可。时间复杂度$O(N)$

子任务 2 (25分)

$A_i = B_i$，所以可以看成是全部科目自学。用优先队列维护n个科目当前的理解度，每次取出最小值更新即可。时间复杂度$O(NM \log N)$

子任务 3 (27分)

- 假如$A_i < B_i$, 则上课改为自学更好。所以我们让$A_i = \max(A_i, B_i)$，默认上课的效率是优于自学的，简化该问题。这样我们就需要优先上课，再看是否需要安排时间自学。

- 做法类似子任务2，使用优先队列维护当前科目的理解度，每次取出最小的科目，如果这个科目已经上的天数$< m$，则理解度加上A_i，否则理解度加上B_i。
- 时间复杂度$O(NM \log N)$

子任务 4 (29分)

$A_i = B_i$，全部科目都可以看成自学，无需考虑每周上课科目限制。最小值最大，考虑二分答案x，科目i需要$\lceil \frac{x}{B_i} \rceil$天。判定$\sum \lceil \frac{x}{B_i} \rceil \leq NM$即可。
时间复杂度$O(N \log(MB_{\max}))$

子任务 5 (100分)

- 考虑子任务3和子任务4的做法结合
- 另$A_i = \max(A_i, B_i)$，这样就可以默认上课效率高于自学。
- 二分答案x，对于科目i，假如$MA_i \leq x$，则需要$\lceil \frac{x}{A_i} \rceil$天，否则需要$\lceil \frac{x-MA_i}{B_i} \rceil$天。

```
#include<bits/stdc++.h>
using namespace std;
using ll = long long;

int main() {
    ios::sync_with_stdio(false);
    cin.tie(0);
    int n, m;
```

```
    cin >> n >> m;
    vector<int> A(n + 1), B(n + 1);
    for (int i = 1; i <= n; ++i) {
        cin >> A[i];
    }
    for (int i = 1; i <= n; ++i) {
        cin >> B[i];
        A[i] = max(A[i], B[i]);
    }
    auto check = [&](ll x) {
        ll day = 0;
        for (int i = 1; i <= n; ++i) {
            if (x <= (ll) A[i] * m) {
                day += (x + A[i] - 1) / A[i];
            } else {
                day += m + (x - (ll) A[i] * m + B[i]
- 1) / B[i];
            }
            if (day > (ll)n * m)
                return 0;
        }
        return 1;
    };
    ll l = 0, r = 1e18, ans = 0;
    while (l <= r) {
        ll mid = l + r >> 1;
        if (check(mid)) {
            l = mid + 1;
            ans = mid;
        } else {
            r = mid - 1;
        }
    }
```

```
    cout << ans;
    return 0;
}
```

T3 打 ACM 最快乐的就是滚榜读队名了

- 比较友好的小模拟题
- $m = 1$时，直接输出一个队名即可，期望得分10分。
- $n = 1$与正解区别不大，直接讲正解。
- 首先我们需要记录队伍的名称、提交记录中的出现编号、通过的题数、罚时、用以排名。此外还需要记录每道题之前的提交次数和滚榜时的时间贡献。
- 对于每条提交记录
  - 如果通过
    - 封榜前：更新罚时，标记题目通过。
    - 封榜后：标记，记录罚时贡献，滚榜再更新
  - 没通过，该题目的提交数++。
- 处理完所有提交记录后，丢进堆里，每次取出队伍，输出名称，按题目顺序计算每题罚时，直到榜单发生变化，就丢回去堆里，结束操作。
- 时间复杂度$O(\min(Klogm, mnlogm))$

```cpp
#include<bits/stdc++.h>
using namespace std;
struct Q{
    string name;    // 队伍名
    int id, sum, ti, st;    // 第一次提交编号，过题数，罚时，滚榜的起点
    int to[28]; // 用于滚榜的通过的题目罚时
    int cnt[28];    // 每道题的提交次数
    bool operator < (const Q &x) const {
        if (sum != x.sum)
            return sum > x.sum;
        else if (ti != x. ti)
            return ti < x.ti;
        else
            return id < x.id;
    }
} q[1005];
map<string, int> mp;
int num;
int main() {
    // freopen("down/3.in", "r", stdin);
    // freopen("donw/3.ans", "w", stdout);
    ios::sync_with_stdio(false);
    cin.tie(0);
    int n, m, k;
    cin >> n >> m >> k;
    for (int i = 1; i <= k; ++i) {
        string a, b, c, d;
        cin >> a >> b >> c;
        getline(cin, d);
        int xx = a[0] - '0', yy = stoi(a.substr(2, 2)), zz = stoi(a.substr(5, 2));  // 得到时间
        int p = b[0] - 'A' + 1; // 提交编号和题目编号
```

```
        if (mp.find(c) == mp.end()) {   // 第一次提交记录
            mp[c] = ++num;
            q[num] = {c, num, 0, 0, 1};
        }
        int id = mp[c];
        if (q[id].to[p])    // 已经通过的就不管
            continue;
        int T = xx * 60 + yy;
        if (d == " Accepted") { // 假如通过了
            q[id].to[p] = 1;    // 1标记通过了，> 1偏移记录罚时
            if (T > 240 || (T == 240 && zz)) {   // 封榜后通过的话
                q[id].to[p] = T + 20 * q[id].cnt[p] + 1;
            } else {    // 封榜前通过的话
                q[id].ti += T + 20 * q[id].cnt[p];  // 更新罚时
                q[id].sum++;    // 通过数++
            }
        } else {    // 没通过次数++
            q[id].cnt[p]++;
        }
    }
    priority_queue<Q> qu;
    for (int i = 1; i <= num; ++i) {
        qu.push(q[i]);
    }
    while (qu.size() >= 2) {
        Q fi = qu.top();
        qu.pop();
        cout << fi.name << '\n';
```

```
            Q se = qu.top();
            for (int i = fi.st; i <= n; ++i) {
                if (fi.to[i] > 1) { // 滚榜后才通过
                    fi.ti += fi.to[i] - 1;  // 更新罚时
                    fi.sum++;   // 题数通过
                    if (fi < se) {  // 一旦上升就滚动
                        fi.st = i + 1;
                        qu.push(fi);
                        break;
                    }
                }
            }
        }
        cout << qu.top().name;
}
```

T4 找爸爸

声明：来自 CodePlus 2017 11 月赛，清华大学计算机科学与技术系学生算法与竞赛协会 荣誉出品。

Credit：idea/卢政荣 命题/卢政荣 验题/何昊天

Git Repo：https://git.thusaac.org/publish/CodePlus201711

感谢腾讯公司对此次比赛的支持。

首先添加完空格后的序列长度不会超过 $n+m$，因为如果超过 $n+m$ 一定会有两个序列在某个位置都是空格，显然删掉这个位置答案更优。

下面是std的写法

测试点1 (10分)

只有两种情况，直接匹配或各加一个空格，if判断即可。

测试点2 (20分)

暴力枚举空格插在什么位置。复杂度 $O(2^n n)$

测试点3-4 (40分)

考虑动态规划，$f(i,j)$ 表示只考虑第一个串的前 i 个字符和第 j 个串的前 j 个字符的答案。

如果第一个串的第 i 个字符和第二个串的第 j 个字符匹配，那答案就是 $f(i-1,j-1)+d\left(a_i,b_j\right)$。

如果第一个串的第i个字符和第二个串的第 k（$k<j$）个字符匹配，那答案就是$f(i,k)-B-A(j-k-1)$。

如果第一个串的第 k（$k<i$） 个字符和第二个串的第 j 个字符匹配，那答案就是$f(k,j)-B-A(i-k-1)$。

三种情况取 $\max$ 即可，枚举 i,j,k，复杂度$O(n^3)$。

测试点5 (10分)

要么直接给短的插一段空格匹配，要么全部错位。

测试点6-10 (100分)

- 在测试点3-4的基础上优化一下，可以令$mxi(i,j)$表示 i 必须和空格匹配的答案，$mxj(i,j)$表示 j 必须和空格匹配的答案，则 $mxi(i,j)=\max\left(dp(i-1,j)-A,mxi(i-1,j)-B\right)$，$mxj(i,j)$ 类似，那么dp方程就是

  $dp(i,j)=\max\left\{dp(i-1,j-1)+d\left(a_i,b_j\right),mxi(i,j),mxj(i,j)\right\}$

  复杂度$O\left(n^2\right)$。

```
#include <bits/stdc++.h>
using namespace std;
int d[5][5],A,B;
char s[5005];
int a[5005],b[5005];
int dp[3005][3005],fr[3005][3005];
int mxi[3005][3005],mxj[3005][3005];
int change(char x)
{
    if(x=='A') return 0;
    if(x=='T') return 1;
    if(x=='G') return 2;
    return 3;
}
int main()
{
    //freopen("tt.in","r",stdin);
    scanf("%s",s);
    int n=strlen(s);
    for(int i=0;i<n;i++) a[i+1]=change(s[i]);
    scanf("%s",s);
    int m=strlen(s);
```

```
    for(int i=0;i<m;i++) b[i+1]=change(s[i]);
    for(int i=0;i<4;i++)
        for(int j=0;j<4;j++)
            scanf("%d",&d[i][j]);
    cin>>A>>B;
    A=-A;B=-B;
    memset(dp,0x80,sizeof(dp));
    memset(mxi,0x80,sizeof(mxi));
    memset(mxj,0x80,sizeof(mxj));
    dp[0][0]=0;
    for(int i=1;i<=n;i++) dp[i][0]=A+B*(i-1);
    for(int i=1;i<=m;i++) dp[0][i]=A+B*(i-1);
    for(int i=1;i<=n;i++)
    {
        for(int j=1;j<=m;j++)
        {
            dp[i][j]=dp[i-1][j-1]+d[a[i]][b[j]];
            fr[i][j]=1;
            mxi[i][j]=max(mxi[i-1][j]+B,dp[i-1]
[j]+A);
            mxj[i][j]=max(mxj[i][j-1]+B,dp[i][j-
1]+A);
            if(dp[i][j]<mxi[i][j])
            {
                dp[i][j]=mxi[i][j];
                fr[i][j]=2;
            }
            if(dp[i][j]<mxj[i][j])
            {
                dp[i][j]=mxj[i][j];
                fr[i][j]=3;
            }
        }
```

```
    }
    cout<<dp[n][m]<<endl;
}
```

另一个类似的solution

- 定义$f[i][j][0/1/2]$表示原来串匹配到A的前i个，B的前j个字母，且当前的串A',B'末尾都没有空格/A'有空格，B'没/B'有空格，A'没。这里显然同时有空格是不优的。
- 对于一段连续的空格，我们定义第一个空格的贡献是$-A$，剩余的都是$-B$，分类讨论转移即可。
- $f_{i,j,0}=\max f_{i-1,j-1,0/1/2}+d_{x_i,y_i}$
  - 两边末尾都不是空格，直接让$A[i]$和$B[j]$匹配，显然是从上次的三个中取最大值，然后加上现在这个位置的价值。
- $f_{i,j,1}=\max\{f_{i,j-1,1}-b,f_{i,j-1,0}-a,f_{i,j-1,2}-a\}$
  - A'末尾是空格，B'不是，如果上一次转移的A'末尾有空格，则是$-b$，否则就是$-a$的贡献。
- $f_{i,j,2}=\max\{f_{i-1,j,2}-b,f_{i-1,j,0}-a,f_{i-1,j,2}-a\}$
  - 同上

时间复杂度$O(n^2)$

```
#include<bits/stdc++.h>
using namespace std;
using ll = long long;
const int N = 3010;
```

```
const ll INF = -1e18;
int n, m, a, b;
int x[N], y[N], mp[256];
int d[4][4];
ll dp[N][N][3];
char s[N];

int main() {
    mp['A'] = 0;
    mp['T'] = 1;
    mp['G'] = 2;
    mp['C'] = 3;
    cin >> (s + 1);
    n = strlen(s + 1);
    for (int i = 1; i <= n; ++i)
        x[i] = mp[s[i]];
    cin >> (s + 1);
    m = strlen(s + 1);
    for (int i = 1; i <= m; ++i)
        y[i] = mp[s[i]];
    for (int i = 0; i < 4; ++i)
        for (int j = 0; j < 4; ++j)
            cin >> d[i][j];
    cin >> a >> b;
    for (int i = max(n, m); i; --i) {
        dp[0][i][0] = dp[i][0][0] = dp[0][i][2] =
dp[i][0][1] = INF;
        dp[0][i][1] = dp[i][0][2] = -a - b * (i - 1);
    }
    dp[0][0][1] = dp[0][0][2] = INF;
    for (int i = 1; i <= n; ++i)
        for (int j = 1; j <= m; ++j) {
```

```
            dp[i][j][0] = *max_element(dp[i - 1][j -
1], dp[i - 1][j - 1] + 3) + d[x[i]][y[j]];
            dp[i][j][1] = max({dp[i][j - 1][0] - a,
dp[i][j - 1][1] - b, dp[i][j - 1][2] - a});
            dp[i][j][2] = max({dp[i - 1][j][0] - a,
dp[i - 1][j][2] - b, dp[i - 1][j][1] - a});
        }
    cout << *max_element(dp[n][m], dp[n][m] + 3);
    return 0;
}
```