2020 蓝桥杯省赛 B 组模拟赛（五）题解

结果填空：数字操作

答案：996

按题意模拟。

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
    int n, k;
    cin >> n >> k;
    while (k--)
    {
        if (n % 10 == 0)
            n /= 10;
        else
            n--;
    }
    cout << n << endl;
}
```

结果填空：字符串操作

答案：$zzzzzzywusqpnmljd$

操作一等价于可以任意调整字符的位置。如果仅有操作一，那么我们对字符串排序，即可得到字典序最大的字符串，形如 "z...zy...y...b...ba...a"。
操作二类似于进位操作，把每个字符的数量视作当前位上的数字。在一的前提的下，进位后可以再排序，可以使字典序增大。
那么做法就是先不断进位，然后输出排序后的字符串就行了。

```cpp
#include <bits/stdc++.h>
using namespace std;
int cnt[26];
int main()
{
    string s;
    cin >> s;
    for (int i = 0; i < s.size(); i++)
        cnt[s[i] - 'a']++;
    for (int i = 0; i < 25; i++)
    {
        cnt[i + 1] += cnt[i] / 2;
        cnt[i] %= 2;
    }
    for (int i = 25; i >= 0; i--)
        for (int j = 0; j < cnt[i]; j++)
            cout << (char)('a' + i);
    return 0;
}
```

结果填空：煎牛排

答案：499122180

分情况讨论。
k 个平底锅，一次（5分钟）能煎 $2k$ 块牛排的一面。

- $n \leq 2k$，煎两次就能全部煎熟，需时 10 分钟。
- $n = t \times (2k) + m$
  - $m = 0$，总共需时 $10t$ 分钟。
  - $0 < m <= k$，前 $t-1$ 份 $2k$ 块牛排，需时 $10(t-1)$ 分钟。剩余 $2k+m$ 份，第一次煎 $2k$ 的一面，第二次煎 m 块的第一面和 $2k$ 块里中的的 $2k-m$ 块的第二面，第三次煎 $2m$ 块的第二面，这部分时间为 15 分钟，总需时 $10t+5$ 分钟。
  - $m > k$，剩余 m 块无法像上述处理，总会有剩余。总需时为 $10t+10$ 分钟。

```cpp
#include <bits/stdc++.h>
using namespace std;

int main()
{
    int n, k;
    cin >> n >> k;
    k *= 2;
    if (n <= k)
        cout << 10ll << endl;
    else
    {
        long long ans = n / k * 10ll;
        n %= k;
        if (n)
        {
            if (n <= k / 2)
                ans += 5;
            else
                ans += 10;
        }
        cout << ans << endl;
    }
    return 0;
}
```

结果填空：数列求值

答案：959741112

递推式求值。

```cpp
#include <bits/stdc++.h>
using namespace std;
const int mod = 998244353;
int main()
{
    int n;
    cin >> n;
    int a = 1, b = 1, c = 1, d;
    for (int i = 4; i <= n; i++)
    {
        d = (7ll * c + 10ll * b + 6ll * a) % mod;
        a = b;
        b = c;
        c = d;
    }
    cout << d << endl;
    return 0;
}
```

结果填空：卡片游戏

答案：8895

区间 dp，用递归去转移更直观一些。

DP(l, r, p)： [l, r] 表示剩余卡片区间，p 表示此时取卡篇的玩家，p = 0 表示蒜头君，p = 1 表示花椰妹。 返回值为蒜头君得分和花椰妹的差值。

p = 0 采用最佳策略，需要考虑子局面的分数：$DP(l,r,0)=max(DP(l,r-1,1)+a[r],DP(l+1,r,1)+a[l]);$
p = 1 采取贪心策略：

- $a[l]<a[r],DP(l,r,1)=DP(l,r-1,0)-a[r];$
- $a[l]\geq a[r],DP(l,r,1)=DP(l-1,r,0)-a[l];$

再记忆化处理一下，防止重复计算。

```
#include <bits/stdc++.h>
using namespace std;
const int maxn = 1000 + 5;
int dp[maxn][maxn];
bool vis[maxn][maxn];
int a[maxn], n;
int DP(int l, int r, int p)
{
    if (vis[l][r])
        return dp[l][r];
    if (l > r)
        return 0;
    vis[l][r] = 1;
    int res = 0;
    if (p == 0)
        res = max(DP(l, r - 1, 1) + a[r], DP(l + 1, r, 1) + a[l]);
    else
    {
        if (a[l] < a[r])
            res = DP(l, r - 1, 0) - a[r];
        else
            res = DP(l + 1, r, 0) - a[l];
    }
    return dp[l][r] = res;
}
int main()
{
    scanf("%d", &n);
    for (int i = 1; i <= n; i++)
        scanf("%d", &a[i]);
    printf("%d\n", DP(1, n, 0));
}
```

程序设计：打字

一个数组记录输入的文本，一个变量记录当前光标所在的位置，一个变量记录当前大小写状态，然后模拟操作就可以了。

标程

```
#include <bits/stdc++.h>
using namespace std;
const int N = 100005;
char s[N], t[N];
int p, sta;
int main()
{
    p = sta = 0;
    int m;
    cin >> m;
    while (m--)
    {
        char key[15];
        cin >> key;
        if (key[0] == 'C')
        {
            sta ^= 1;
        }
        else if (key[0] == 'B')
        {
            if (p)
                p--;
        }
        else if (key[0] == 'S')
        {
            t[p++] = ' ';
        }
        else
        {
            t[p++] = key[0] + (sta ? 'A' - 'a' : 0);
        }
    }
    t[p] = 0;
    cout << t << endl;
    return 0;
}
```

程序设计：删除字符

可以用 vector 顺序存入每个字母对应的位置，然后把对应的前 k 个位置标记。
遍历输出字符串，跳过标记的位置。

写法多样，注意不要让复杂度退化成 $O(n^2)$ 就行。

标程

```cpp
#include <bits/stdc++.h>
using namespace std;
const int N = 1e5 + 5;
vector<int> p[32];
char s[N];
bool vis[N];
int n, k;

int main()
{
    cin >> n >> k;
    cin >> s + 1;
    for (int i = 1; i <= n; i++)
        p[s[i] - 'a'].push_back(i);
    for (int i = 0; i < 26; i++)
    {
        for (int j = 0; j < p[i].size(); j++)
        {
            vis[p[i][j]] = true;
            k--;
            if (k == 0)
                break;
        }
        if (k == 0)
            break;
    }
    for (int i = 1; i <= n; i++)
        if (!vis[i])
            cout << s[i];
    cout << endl;
    return 0;
}
```

程序设计：两个数组

先考虑第二个数组中只有一个数 x 的情况，对第一个数组中的元素 e 执行任意次操作就是 $e+ = ax$ ，其中 a 可以为任意整数。
这样只需要满足条件：所有数对 x 同余（即对 x 取模的结果相同），就可以通过操作将数变成相同的。
如果有两个数 x, y，则操作为 $e+ = ax + by$。
$ax + by = k \times gcd(x, y)$。$a$，$b$为任意整数，$k$ 也可以取到任意整数。
那么如果所有数对 $gcd(x, y)$ 同余，就可以通过操作将所有数变成一样的。
更多的数，可以同样类推过去。

因此最终只需要对第二个数组的所有数求 gcd，再判断第一个数组的所有数是否对 gcd 同余就可以了。

标程

```
#include <bits/stdc++.h>
using namespace std;
const int N = 1e6 + 5;
int a[N], n, m;
int main()
{
    int T;
    scanf("%d", &T);
    while (T--)
    {
        scanf("%d%d", &n, &m);
        for (int i = 1; i <= n; i++)
            scanf("%d", &a[i]);
        int gd;
        scanf("%d", &gd);
        for (int i = 2; i <= m; i++)
        {
            int x;
            scanf("%d", &x);
            gd = __gcd(gd, x);
        }
        int res = a[1] % gd;
        bool flag = true;
        for (int i = 2; i <= n; i++)
            if (a[i] % gd != res)
            {
                printf("No\n");
                flag = false;
                break;
            }
        if (flag)
            printf("Yes\n");
    }
    return 0;
}
```

程序设计：函数求和

$$S=\sum_{1\le l\le r\le n} f(l,r)=\sum_{1\le l\le r\le n}\sum_{i=l}^{r} a_i\cdot b_i=\sum_{i=1}^{n} i(n-i+1)a_ib_i$$

令 $c_i=i(n-i+1)a_i$，则 $S=\sum_{i=1}^{n} c_ib_i$。

问题就变成了最小化两个数组的乘积和。对两个数组排序，一个升序，一个降序，然后相乘求和，就可以计算出最小的 S 了。

标程

```cpp
#include <bits/stdc++.h>
using namespace std;
const int N = 1e5 + 5;
using ll = long long;
const int mod = 998244353;
int a[N], b[N], n;
ll c[N];
int main()
{
    scanf("%d", &n);
    for (int i = 1; i <= n; i++)
        scanf("%d", &a[i]);
    for (int i = 1; i <= n; i++)
        scanf("%d", &b[i]);
    sort(b + 1, b + n + 1, [](int x, int y) { return x > y; });
    for (int i = 1; i <= n; i++)
        c[i] = 1ll * i * (n - i + 1) * a[i];
    sort(c + 1, c + n + 1);
    ll ans = 0;
    for (int i = 1; i <= n; i++)
        ans = (ans + c[i] % mod * (b[i] % mod) % mod) % mod;
    printf("%lld\n", ans);
    return 0;
}
```

程序设计：序列划分

子序列长度为 1 或者 c 是最优选择。

- 若子序列长度 $< c$，则可以全拆成长度为 1 的子序列，结果等价。
- 若子序列长度 $c \leq k \leq 2c-1$，可以拆成一段长度为 c 的子序列，其余部分均拆成长度为 1 的子序列，结果等价。
- 若子序列长度为 $2c$，则拆成两段长度为 c 的子序列，结果不会更差。
- 其他情况可以类推。

那么可以得到递推式为:

$$dp[i] = dp[i-1] + a[i] \ (i < c)$$

$$dp[i] = min(dp[i-1] + a[i], dp[i-c] + \sum_{j=i-c+1}^{i} a[j] - MIN_{i+c-1 \leq j \leq i}(a[j]) \ (i >= c)$$

求和式可以用前缀和优化。
区间取 MIN 可以用 ST/线段树/单调队列 优化。

标程

```cpp
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
const int maxn = 1e5 + 5;
int a[maxn][33];
ll sum[maxn], dp[maxn];
int n, c;
int query(int i, int j)
{
    int log2_l = log2(j - i + 1);
    int l = 1 << log2_l;
    return min(a[i][log2_l], a[j - l + 1][log2_l]);
}
int main()
{
    ios::sync_with_stdio(false), cin.tie(0), cout.tie(0);
    cin >> n >> c;
    memset(a, 0x3f, sizeof a);
    for (int i = 1; i <= n; i++)
    {
        cin >> a[i][0];
        sum[i] = sum[i - 1] + a[i][0];
    }
    for (int l = 1, j = 1; (l << 1) <= n; l <<= 1, j++)
        for (int i = 1; i + (l << 1) - 1 <= n; i++)
            a[i][j] = min(a[i][j - 1], a[i + l][j - 1]);
    for (int i = 1; i <= n; i++)
    {
        if (i < c)
            dp[i] = dp[i - 1] + a[i][0];
        else
            dp[i] = min(dp[i - 1] + a[i][0], dp[i - c] + sum[i] - sum[i - c] - query(i - c + 1,
    }
    cout << dp[n] << endl;
}
```