Mục lục

1 Giới thiệu

- Nhân dịp tuyển sinh sắp tới chúng mình đã sưu tầm đề tuyển sinh toàn quốc bao gồm tầm 30 đề từ các tỉnh để viết lời giải cũng như làm trình chấm cho các bạn nộp. Vì có rất nhiều bài nên sẽ được tách thành 2 link chấm, link đầu sẽ gồm 15 tỉnh và link sau sẽ gồm phần còn lại.

- Link contest 15 tỉnh đầu: `https://codeforces.com/contestInvitation/1b52f4fa6162108be545ef4`

- Link contest các tỉnh còn lại: `https://codeforces.com/contestInvitation/2d5226bb7cd13fa3a217`

- Nếu có thắc mắc hay cần hỗ trợ các bạn có thể trao đổi trên group facebook :https://www.facebook.com/groups/7038269849574622
- Nếu bạn nào quan tâm tới việc học thuật toán để thi học sinh giỏi, đội tuyển quốc gia các cấp cho hè này thì có thể liên hệ mình qua facebook: https://www.facebook.com/Ton2808/ hoặc Zalo: 0762310005

2 Đề Sở Hồ Chí Minh

2.1 Bài 1: Lũy Thừa

2.1.1 Subtask 1 + 2

- Theo giả thiết của đề, là X_i là một số nguyên không âm có một chữ số, vì thế ta chỉ tách P_i bằng phép chia làm tròn xuống và sử dụng phép mod. Hay ta có thể viết $A_i = P_i/10$, $X_i = P_i\%10$. Tới đây, ta có thể sử dụng cách tính mũ nhị phân hoặc phép lũy thừa bình thường.

2.1.2 Code mẫu tham khảo:

```
#include <bits/stdc++.h>

using namespace std;

int bpow(int a, int b)
{
        if (b == 0)
                return 1;
        if (b == 1)
                return a;
        int temp = bpow(a,b / 2);
        if (b % 2 == 0)
                return temp * temp;
        return temp * temp * a;
}

int main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(0);

    int n;
    int ans = 0;
    cin >> n;
    for(int i = 1; i <= n; i++)
    {
        int x;
```

```
        cin >> x;
        ans += bpow(x / 10, x % 10);
    }

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

2.2 Bài 2: Giao Hàng

2.2.1 Subtask 1: $N = 2, 2 < M \leq 10^9$

Ở subtask này ta chỉ cần thử 6 trường hợp có thể xảy ra và lấy đáp án tối ưu nhất, đó là:
- Lấy 1 , Giao 1, Lấy 2, Giao 2.
- Lấy 1 , Lấy 2, Giao 1, Giao 2.
- Lấy 1 , Lấy 2, Giao 2, Giao 1.
- Lấy 2 , Giao 2, Lấy 1, Giao 1.
- Lấy 2 , Lấy 1, Giao 2, Giao 1.
- Lấy 2 , Lấy 1, Giao 1, Giao 2.

2.3 Subtask 2 + 3

Gọi S_i, T_i lần lượt là điểm bắt đầu và điểm kết thúc của lần giao hàng thứ i.
Nhận xét:
- Trên đường đi từ 0 tới M :
+ Với $S_i \leq T_i$: Ta có thể trực tiếp đi tới S_i và giao tới T_i nên ta sẽ bỏ qua trường hợp này.
+ Với $S_i > T_i$: Trường hợp này ta cần phải đi ngược lại từ S_i, quay về T_i để giao hàng và quay lại S_i và ta lại tốn thêm $2 \times (S_i - T_i)$. Nên việc của ta là tối ưu trường hợp này.
- Gọi $[x, y]$ là những đoạn đường ta sẽ đi ngược từ y về x rồi quay lại y.
- Xử lý các trường hợp $S_i > T_i$, giả sử ta có hai lần giao hàng $[T_i, S_i]$ và $[T_j, S_j]$, nếu chúng không **giao nhau**, thì ta có thể thực hiện 2 lần độc lập, nhưng nếu chúng giao nhau, thì ta có thể tối ưu **gộp** chúng lại thành 1 lần đi giao hàng.
Chứng minh:
- Giả sử ta có hai đoạn $[a, b]$ và $[c, d]$ giao nhau (Không mất tính tổng quát, giả thuyết $a \leq c \leq b \leq d$), chi phí để đi 2 lần là $2 \times (b - a + d - c)$. Chi phí để ta đi 1 lần là $2 \times (d - a)$
Ta có :
$2 \times (b - a + d - c) \geq 2 \times (d - a)$
$\Leftrightarrow\ b - a + d - c \geq d - a$
$\Leftrightarrow\ b - a + d - c \geq d - a$
$\Leftrightarrow\ b - c \geq 0$ (Đúng do $c \leq b$)
- Vậy ta đã chứng minh được khi **gộp các đoạn giao nhau** thành 1 đoạn, chi phí sẽ ít hơn, nên thuật toán của ta sẽ là lấy tất cả các đoạn giao nhau, gộp thành 1 đoạn rồi tính tổng

từng phần như vậy rồi cộng thêm M.

Thuật toán:

- Với những đoạn $S_i > T_i$, ta sẽ lưu chúng vô một mảng $pair < int, int >$ theo dạng (S_i, T_i), rồi sắp xếp chúng lại theo thứ tự $i < j \Rightarrow S_i < S$ hoặc ($S_i = S_j$ và $T_i < T_j$).

= Ta duyệt lần lượt các phần tử trong mảng sau khi sắp xếp :

- Khởi tạo hai biến $l = S_0, r = T_0$ mang ý nghĩa là **đoạn hiện tại** sau khi ghép hoặc giữ nguyên và biến đáp án ban đầu có giá trị là M (nghĩa là đi từ 0 đến M).

- Nếu đoạn $[T_i, S_i]$ tiếp theo giao nhau với đoạn $[l, r]$ thì ta sẽ mở rộng đoạn $[l, r]$ của ta, ngược lại thì ta đã gộp xong 1 phần riêng biệt, nên ta sẽ cộng vào đáp án của ta chi phí của đoạn $[l, r]$ và gán $l = T_i, r = S_i$ - Sau khi duyệt qua tất cả các đoạn, ta cộng thêm phần cuối cùng của các đoạn rồi trả về đáp án.

2.3.1 Code mẫu tham khảo:

```
#include <bits/stdc++.h>

using namespace std;

int n;
long long m;
vector < pair<int,int> > s;

int main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(0);

    cin >> n >> m;
    for(int i = 1; i <= n; i++)
    {
        int l,r;
        cin >> l >> r;
        if (l > r)
                s.push_back(make_pair(r, l));
    }

    if (s.size() == 0) // truong hop khong co don hang
    {
        cout << m;
        return 0;
    }

    sort(s.begin(),s.end());

    int l = s[0].first;
    int r = s[0].second;
```

```
    long long ans = 0;

    for(int i = 0; i < (int)s.size(); i++)

    {
        if (s[i].first > r)
        {
                ans += (r - l) * 2;
                l = s[i].first;
                r = s[i].second;
        }
        else
                r = max(r, s[i].second);
    }

    ans += (r - l) * 2;
    cout << ans + m;

    return 0;
}
```

2.4 Bài 3: Đào Vàng

2.4.1 Subtask 1: $K = 1, N \leq 1000$

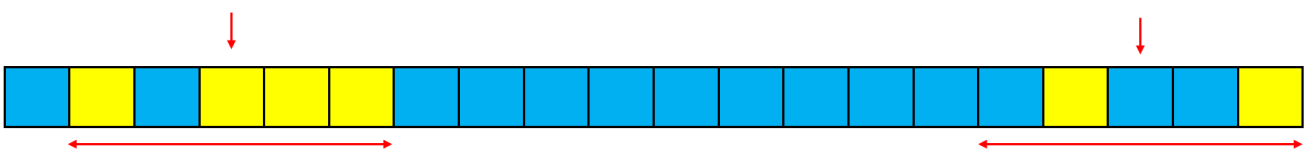
- Ở subtask này vì chỉ có 1 lần đào, vì thế ta chỉ cần tìm lực đập bé nhất thỏa: $X_1 + 2 * R \geq X_n$, trong đó X_1 là vị trí X bé nhất còn X_n là vị trí X lớn nhất.

2.4.2 Subtask 2: $K = 2, N \leq 10000$

- Ở subtask này, ta có thể sử dụng 2 vòng for lồng nhau để tìm 2 vị trí đập thích hợp.

2.4.3 Subtask 3: $K \leq 20, N \leq 50000$

- Đầu tiên ta có nhận xét rằng lực đập R_i thỏa yêu cầu đề bài thì mọi lực đập $R_j > R_i$ đều thỏa yêu cầu đề bài.
- Từ đây, ta sử dụng chặt nhị phân đáp án để tìm R bé nhất thỏa yêu cầu đề bài, với mỗi lực đập R, ta kiểm tra xem mình có thể dùng lực đập này để đào hết vàng mà số lần dùng vẫn không vượt quá K hay không.
- Để kiểm tra xem liệu lực đập R có thỏa hay không, ta xét ví dụ sau:



- Ta thấy rằng khi đập tại một vị trí X, ta có thể lấy được tất cả vàng trong đoạn $[X-R, X+R]$, vì thế, muốn lấy được vàng ở vị trí X với lực đập R sao cho tối ưu nhất, ta sẽ đập tại vị trí $X+R$, khi đập tại đây, vị trí xa X nhất mà ta có thể lấy được vàng là $X+2\times R$. Từ đó, ta có thể kiểm tra xem với lực đập R, ta phải đập bao nhiêu lần mới lấy được hết tất cả vàng, lực đập R thỏa yêu cầu khi số lần đập không vượt quá k.

2.4.4 Code mẫu tham khảo:

```
#include <bits/stdc++.h>

using namespace std;

int n, k;
int a[50000 + 5];

bool check(int x)
{
        int cnt = 1;
        int i = 1;
        int need = a[1] + 2 * x;
        while(i <= n)
        {
                if (a[i] <= need)
                        i++;
                else
                {
                        need = a[i] + 2 * x;
                        cnt++;
                }
        }

        return (cnt <= k);
}

int main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(0);

    cin >> n >> k;
    for(int i = 1; i <= n; i++)
        cin >> a[i];

    sort(a + 1,a + 1 + n);

    int l = 0;
    int r = 1000000000;
```

```
    int ans = 0;

    while(l <= r)
    {
        int mid = (l + r) / 2;
        if (check(mid) == true)
        {
                ans = mid;
                r = mid - 1;
        }
        else
                l = mid + 1;
    }

    cout << ans;

    return 0;
}
```

3 Đề Hồ Chí Minh trường Phổ Thông Năng Khiếu

3.1 Câu 1:

3.1.1 Subtask 1:

Nhận thấy rằng ta cần độ dài mảng tiền tố của mảng a không giảm và độ dài mảng hậu tố của mảng b không giảm dài nhất, và chọn vị trí sao cho tại đó $a_i \leq b_j$, vì thế, ta có thể chuẩn bị mảng tiền tố và hậu tố trên với độ phức tạp lần lượt là $\mathcal{O}(n)$ và $\mathcal{O}(m)$. Ở bước này ta chỉ cần duyệt 2 vòng for để tìm i và j phù hợp, đáp án là $i+$ khoảng cách từ j tới m.. Vậy độ phức tạp mà ta có là $\mathcal{O}(n^2)$.

3.1.2 Code mẫu tham khảo:

```
#include <bits/stdc++.h>

using namespace std;

const int MAX = 1e5 + 5 ;
int n , m, a [MAX] , b [MAX] ;

int distance(int l, int r){
    return r - l + 1;
}

int main(){
    ios_base::sync_with_stdio(false);
```

```
    cin.tie(0);

    cin >> n;
    for(int i = 1; i <= n; ++i)
        cin >> a[i];
    cin >> m;
    for(int i = 1; i <= m; ++i)
        cin >> b[i];

    int __left, __right;
    __left = n;
    __right = 1;

    for(int i = 2; i <= n; ++i)
        if (a[i] < a[i - 1]){
            __left = i - 1;
            break;
        }
    for(int i = m - 1; i >= 1; --i)
        if (b[i] > b[i + 1]){
            __right = i + 1;
            break;
        }

    int res = 0;
    for(int i = 1; i <= __left; ++i){
        for(int j = __right; j <= m; ++j)
            if (a[i] <= b[j])
                res = max(res,i + distance(j, m));
    }

    cout << res << '\n';


    return 0;
}
```

3.1.3 Subtask 2:

Ta sẽ sử dụng lại ý tưởng của Subtask 1, tuy nhiên ở bước chọn (i, j) phù hợp, ta sẽ sử dụng 2 con trỏ. Ta có nhận xét sau đây: với mỗi phần tử a_i mà ở đó $a_i \le b_j$ thì mọi $a_k \le a_i$ đều thỏa mãn $a_k \le b_j$. Từ đây, để tìm (i, j) phù hợp, ta duyệt qua mỗi i từ 1 tới n, sau đó duy trì j sao cho tại đó $a_i \le b_j$, đáp án là $i+$ khoảng cách từ j tới m.

3.1.4 Code mẫu tham khảo:

```
#include <bits/stdc++.h>
```

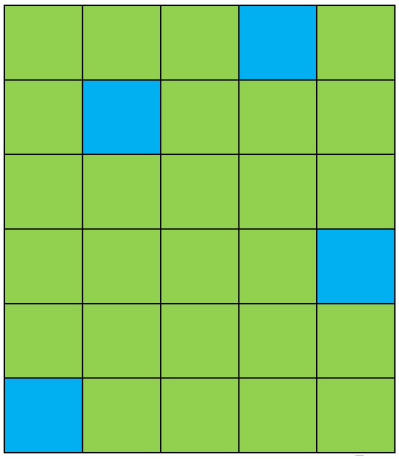
```
using namespace std;

const int MAX = 1e5 + 5;
int n, m, a[MAX], b[MAX];

int distance(int l, int r){
    return r - l + 1;
}

int main(){
    ios_base::sync_with_stdio(false);
    cin.tie(0);

    cin >> n;
    for(int i = 1; i <= n; ++i)
        cin >> a[i];
    cin >> m;
    for(int i = 1; i <= m; ++i)
        cin >> b[i];

    int __left, __right;
    __left = n;
    __right = 1;

    for(int i = 2; i <= n; ++i)
        if (a[i] < a[i - 1]){
            __left = i - 1;
            break;
        }
    for(int i = m - 1; i >= 1; --i)
        if (b[i] > b[i + 1]){
            __right = i + 1;
            break;
        }

    int res = 0;
    bool flag = true;
    for(int i = 1; i <= __left; ++i){
        while(a[i] > b[__right]){
            ++__right;
            if (__right > m){
                flag = false;
                break;
            }
        }

        if (!flag)
```

```
            break;
        res = max(res, i + distance(__right, m));
    }

    cout << res << '\n';


    return 0;
}
```

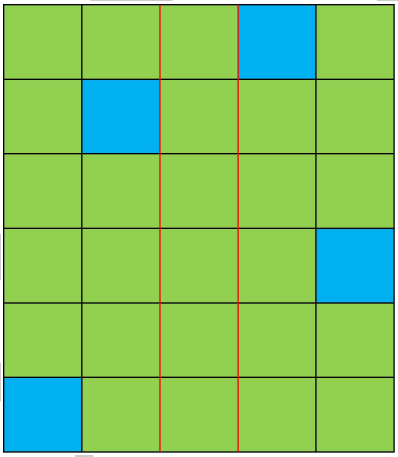
3.2 Câu 2:

3.2.1 Subtask 1:

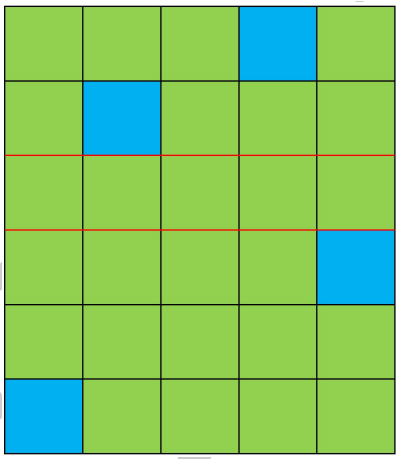
Ta sẽ xét ví dụ sau:



N = 6, M = 5
Các vị trí đặt cherry
- (1,4)
- (2,2)
- (4,5)
- (6,1)



Các cột dọc có thể kẻ được sao cho 2 bên đường kẻ có số lượng cherry bằng nhau: 2 đường



Các hàng ngang có thể kẻ được sao cho 2 bên đường kẻ có số lượng cherry bằng nhau: 2 đường

Từ đây ta thấy rằng: ta sẽ chọn những cột dọc sao cho 2 bên cột đều bằng nhau và chọn những hàng ngang sao cho 2 phần trên dưới của hàng ngang đều bằng nhau.
Gọi số cột dọc thỏa là x, số hàng ngang thỏa là y, vậy đáp án của chúng ta sẽ là $x \times y$.

3.3 Code mẫu tham khảo:

```cpp
#include <bits/stdc++.h>

using namespace std;

int n, m, k;
pair <int,int> a[5];

bool x_cmp(pair <int,int> x, pair <int,int> y){
    return x.first < y.first;
}
bool y_cmp(pair <int,int> x, pair <int,int> y){
    return x.second < y.second;
}

int main(){
    ios_base::sync_with_stdio(false);
    cin.tie(0);

    cin >> n >> m >> k;
    for(int i = 1; i <= 4 * k; ++i)
        cin >> a[i].first >> a[i].second;

    sort(a + 1,a + 1 + 4,x_cmp);

    long long x = (a[3].first - a[2].first);

    sort(a + 1,a + 1 + 4,y_cmp);

    long long y = (a[3].second - a[2].second);

    cout << 1ll * x * y << '\n';


    return 0;
}
```

3.3.1 Subtask 2:

Ở subtask này, ta thấy rằng việc chọn 1 cột và 1 hàng thỏa đồng nghĩa với việc chia bàng $n \times m$ thành 4 phần, trong đó giao điểm của cột dọc và hàng ngang là tại $a_{i,j}$.
Vậy, ta chỉ cần xây dựng một bảng cộng dồn 2 chiều, sau đó duyệt qua để đếm những cặp (i, j) thỏa yêu cầu đề bài.

3.3.2 Code mẫu tham khảo:

```cpp
#include <bits/stdc++.h>
```

```cpp
using namespace std;

const int MAX = 4e3 + 1;
int a[MAX][MAX];
long long pref[MAX][MAX];
int n, m, k;

int main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(0);

    cin >> n >> m >> k;
    for(int i = 1; i <= 4 * k; ++i){
        int x, y;
        cin >> x >> y;
        a[x][y] = 1;
    }

    for(int i = 1; i <= n; ++i){
        for(int j = 1; j <= m; ++j)
            pref[i][j] = pref[i - 1][j]
            + pref[i][j - 1] - pref[i - 1][j - 1] + a[i][j];
    }

    long long ans = 0;

    for(int i = 1; i <= n; ++i){
        for(int j = 1; j <= m; ++j){
            long long upper_left_corner = pref[i][j];
            long long upper_right_corner = pref[i][m] - upper_left_corner;
            long long lower_left_corner = pref[n][j] - upper_left_corner;
            long long lower_right_corner = pref[n][m] - upper_left_corner
            - upper_right_corner - lower_left_corner;
            if (upper_left_corner == upper_right_corner
                && upper_left_corner == lower_left_corner
                    && upper_left_corner == lower_right_corner)
                        ++ans;
        }
    }

    cout << ans << '\n';


    return 0;
}
```

3.3.3 Subtask 3:

Ở Subtask này, chúng ta thấy rằng tư tưởng sẽ giống với Subtask 1. Để chia $4 \times k$ quả cherry thành 4 phần bằng nhau đồng nghĩa với việc mỗi phần có đúng k quả cherry.
Vậy trước hết, ta sẽ đếm những đường chia bảng thành 2 phần có đúng $2 \times k$ quả cherry. Đây là $A_{2\times k} - A_{2\times k-1}$ với mảng A bắt đầu từ 0 và được sắp xếp theo x tăng dần.
Sau khi chia thành 2 phần bằng nhau, ta tiếp tục đếm cách chia mỗi phần đó thành 2 phần, thực hiện tương tự như ở trên những lần này ta sắp xếp theo y.
Gọi những đường thỏa ở bảng thứ nhất thuộc $[a,b]$, ở bảng thứ hai thuộc $[c,d]$, giờ đây ta cần tìm $[a,b] \cap [c,d]$.
Vậy đáp án của ta là tích của mỗi lần chia bảng.

3.3.4 Code mẫu tham khảo:

```
#include<bits/stdc++.h>

using namespace std;

bool cmp (pair<int,int> x, pair<int,int> y){
    return x.second < y.second;
}

int main(){
    ios_base::sync_with_stdio(0);
    cin.tie(0);

    int n, m, k;
    cin >> n >> m >> k;

    vector<pair<int, int>> points(k * 4);
    for(int i = 0; i < k * 4; ++i) cin >> points[i].first >> points[i].second;

    sort(points.begin(), points.end());
    int diff_x = points[k * 2].first - points[k * 2 - 1].first;

    sort(points.begin(), points.begin() + k * 2, cmp);
    sort(points.begin() + k * 2, points.end(), cmp);

    int diff_y = max(0, min(points[k].second, points[k * 3].second)
    - max(points[k - 1].second, points[k * 3 - 1].second));

    cout << 1ll * diff_x * diff_y << '\n';


    return 0;
}
```

3.4 Câu 3:

3.5 Subtask 1:

Ở Subtask này, ta chỉ cần làm theo như đề hỏi, xây dựng 1 mảng cộng dồn 2 chiều và kiểm tra lần lượt từng hình chữ nhật xem liệu nó có bằng k hay không.

3.5.1 Code mẫu tham khảo:

```
#include <bits/stdc++.h>

using namespace std;

const int MAX = 85;

int n, k;
int a[MAX], c[MAX][MAX], pref[MAX][MAX];

int sum(int x1, int y1, int x2, int y2){
    return pref[x2][y2] - pref[x1 - 1][y2] - pref[x2][y1 - 1] + pref[x1 - 1][y1 - 1];
}

int main(){
    ios_base::sync_with_stdio(false);
    cin.tie(0);

    cin >> n;
    for(int i = 1; i <= n; ++i)
        cin >> a[i];
    cin >> k;

    for(int i = 1; i <= n; ++i){
        for(int j = 1; j <= n; ++j)
            c[i][j] = a[i] * a[j];
    }

    for(int i = 1; i <= n; ++i){
        for(int j = 1; j <= n; ++j)
            pref[i][j] = pref[i - 1][j]
            + pref[i][j - 1] - pref[i - 1][j - 1] + c[i][j];
    }

    long long ans = 0;
    for(int x1 = 1; x1 <= n; ++x1){
        for(int y1 = 1; y1 <= n; ++y1){
            for(int x2 = x1; x2 <= n; ++x2){
                for(int y2 = y1; y2 <= n; ++y2){
                    int tmp = sum(x1, y1, x2, y2);
```

```
                    if (tmp == k)
                        ++ans;
                }
            }
        }
    }

    cout << ans << '\n';
    return 0;
}
```

3.5.2 Subtask 2 + 3:

Ta gọi F là tổng từ ô trái trên $C_{x1,y1}$ và ô phải dưới $C_{x2,y2}$.
Ta có:

$$F = C_{x1,y1} + C_{x1,y1+1} + ... + C_{x1,y2} + C_{x1+1,y1} + C_{x1+1,y1+1} + ... + C_{x1+1,y2} + ... + C_{x2,y1} + C_{x2,y1+1} + ... + C_{x2,y2}$$

$$\Leftrightarrow F = A_{x1} \times A_{y1} + A_{x1} \times A_{y1+1} + ... + A_{x1} \times A_{y2} + A_{x1+1} \times A_{y1} + A_{x1+1} \times A_{y1+1} + ... + A_{x1+1} \times A_{y2}$$
$$+ ... + A_{x2} \times A_{y1} + A_{x2} \times A_{y1+1} + ... + A_{x2} \times A_{y2}$$

$$\Leftrightarrow F = A_{x1} \times (A_{y1} + A_{y1+1} + ... + A_{y2}) + A_{x1+1} \times (A_{y1} + A_{y1+1} + ... + A_{y2})$$
$$+ ... + A_{x2} \times (A_{y1} + A_{y1+1} + ... + A_{y2})$$

$$\Leftrightarrow F = (A_{x1} + A_{x1+1} + ... + A_{x2}) \times (A_{y1} + A_{y1+1} + ... + A_{y2})$$

Ta đặt:

$$x = A_{x1} + A_{x1+1} + ... + A_{x2}$$
$$y = A_{y1} + A_{y1+1} + ... + A_{y2}$$

Mà ta cần:

$$F = k \Leftrightarrow k \vdots x, k \vdots y$$

Vậy, ta tính trước mọi đoạn (i, j) sao cho tổng của (i, j) là ước của k và duyệt qua lại để đếm đoạn thỏa mãn.

3.5.3 Code mẫu tham khảo:

```
#include<bits/stdc++.h>

using namespace std;

const int MAX = 1e6 + 5;
int a[MAX], cnt[MAX];
int n, k;
```

```cpp
int main(){
    ios_base::sync_with_stdio(0);
    cin.tie(0);

    cin >> n;
    for(int i = 1; i <= n; ++i)
        cin >> a[i];
    cin >> k;
    for(int i = 1; i <= n; ++i){
        int sum = 0;
        for(int j = i; j <= n; ++j){
            sum += a[j];
            if(k % sum == 0) ++cnt[sum];
        }
    }

    int ans = 0;
    for(int i = 1; i <= k; ++i){
        if(k % i == 0){
            ans += cnt[i] * cnt[k / i];
        }
    }
    cout << ans << '\n';

    return 0;
}
```

4 Đề An Giang

4.1 Bài 1 (2 điểm)

Do hiệu của A, B không âm nên kết quả là $|A - B|$.

Code mẫu.

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

int main(){
    int a, b;
    cin >> a >> b;
    cout << abs(a - b);
    return 0;
}
```

4.2 Bài 2 (2 điểm)

Ta duyệt i từ 2 đến $N-1$ và kiểm tra xem i có là ước chẵn của N hay không.
*Lưu ý: Khi N lẻ $\Rightarrow$ ước của N chỉ có số lẻ nên kết quả là 0.
Code mẫu.

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

int main(){
    int n;
    cin >> n;
    long long result = 1;
    for(int i=2; i<n; i+=2){
        if (n % i == 0){
            result = result * i;
        }
    }
    if(result==1) result=0;
    cout << result;
    return 0;
}
```

4.3 Bài 3 (2 điểm)

- Nhận xét: Vì giới hạn $R \le 10^{12}$ do đó ta chỉ cần tạo mảng giai thừa từ 1!...14!.
Ta nhận thấy số nguyên tố mà chính nó là giai thừa chỉ có số $2 = 2! = 1! + 1$.
Do đó với mỗi số i $(1 \le i \le 14)$ ta chỉ cần xét xem $i! - 1$ và $i! + 1$ có phải là số nguyên tố và thuộc khoảng $[L,R]$ hay không.
Code mẫu.

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

bool isprime(long long x){
    for(int i=2; 1LL * i * i <= x; ++i)
        if (x % i  == 0) return false;
    return (x > 1);
}

int main(){
    long long l, r, fact[20];
    cin >> l >> r;
    fact[0] = 1;
    for(int i=1; i<=14; ++i) fact[i] = fact[i-1] * i;
    for(int i=1; i<=14; ++i){
        long long x = fact[i] - 1;
        if (l <= x && x <= r && isprime(x)) cout << x << '\n';
```

```
        x = fact[i] + 1;
        if (l <= x && x <= r && isprime(x)) cout << x << '\n';
    }
    return 0;
}
```

4.4 Bài 4 (2 điểm)

Với mỗi i ($L \le i \le R$) ta sẽ đảo ngược số i và dùng hàm $\_\_gcd$ để tìm ước chung lớn nhất. Code mẫu.

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

int rev(int x){
    int num = 0;
    while(x > 0){
        num = num * 10 + x % 10;
        x /= 10;
    }
    return num;
}

int main(){
    long long l, r;
    cin >> l >> r;
    int result = 0;
    for(int i=l; i<=r; ++i)
        if (__gcd(i, rev(i)) == 1) result++;
    cout << result;
    return 0;
}
```

4.5 Bài 5 (2 điểm)

Ta sẽ tạo struct lưu 3 biến $day, month, year$ tương ứng với ngày mà xâu kí tự đã nhập vào. Tạo mảng $months$ lưu các ngày trong tháng thứ i ($1 \le i \le 12$). Với ngày được nhập vào, ta sẽ xét:

+ Ngày đã nhập nằm trong khoảng $[L,R]$ hay không ?

+ Tìm ngày sau đó bằng cách:

- Nếu $day < months[month] : day++$.
- Nếu $day = months[month]$ và $month < 12 : month++$, day = 1.
- Nếu $day = months[month]$ và $month = 12$: day = month = 1, year++.

+ Kiểm tra ngày vừa tìm được có $\le R$ và có hợp lệ hay không.

* Để so sánh 2 ngày, ta so sánh năm rồi đến tháng sau đó mới đến ngày.

* Lưu ý: ngày in ra phải có dạng $dd/mm/yyyy$.
Code mẫu.

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

int months[] = {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

struct Day{
    int day, month, year;

    void convert(const string &s){
        for(int i=0; i<2; ++i) day = day * 10 + (s[i] - '0');
        for(int i=2; i<4; ++i) month = month * 10 + (s[i] - '0');
        for(int i=4; i<8; ++i) year = year * 10 + (s[i] - '0');
    }

    void print(){
        if (day < 10) cout << 0;
        cout << day;
        if (month < 10) cout << 0;
        cout << month;
        cout << year;
    }
};

bool cmp(const Day &A, const Day &B){
    if (A.year != B.year) return (A.year < B.year);
    if (A.month != B.month) return (A.month < B.month);
    return (A.day < B.day);
}

int main(){
    string s;
    cin >> s;
    Day L = {1, 1, 2000}, R = {4, 6, 2023}, now = {0, 0, 0};
    now.convert(s);
    if ((now.year % 4 == 0 && now.year % 100 != 0) || (now.year % 400 == 0))
        months[2] = 29;
    if (cmp(now, L) || now.day > months[now.month] || now.month > 12
    || now.day == 0 || now.month == 0)
        cout << "Khong tim duoc", exit(0);
    if (now.day < months[now.month]) ++now.day;
    else{
        if (now.month == 12) now = {1, 1, now.year + 1};
        else now = {1, now.month + 1, now.year};
    }
    if (cmp(R, now)) cout << "Khong tim duoc", exit(0);
    now.print();
```

```
    return 0;
}
```

5 Đề Vũng Tàu

5.1 Bài 1: Lật sách

- Phân tích đề bài, ta thấy việc lật sách từ trang 1 đến trang thứ P, đơn giản là đếm số lần lật sách tới P nếu P là số lẻ hoặc $P+1$ nếu P là số chẵn. Ta có thể tính số lần lật tới trang thứ P từ trang 1 như sau. Gọi T_1 là giá trị như sau, nếu P là số lẻ thì $T_1 = P$, ngược lại $T_1 = P+1$. Số lần lật từ trang 1 đến trang P là $\frac{T_1}{2}$.
- Tương tự như vậy, việc lật sách từ trang n đến trang P. Gọi T_2 là giá trị như sau nếu P là số chẵn thì $T_1 = P$, ngược lại $T_1 = P-1$. Số lần lật từ trang n đến trang P là $\frac{n-T_2+1}{2}$. Code mẫu như sau:

```
#include <iostream>

using namespace std;

int main() {
    long long n , P;
    cin >> n >> P;

    long long T1 = P;
    long long T2 = P;

    if(P % 2 == 0)
        T1++;
    else T2++;

    cout << min(T1 / 2 , (n - T2 + 1) / 2) << endl;
}
```

- **Lưu ý:** Sự khác biệt giữa subtask 1 và subtask 2 là int và long long như trong giới hạn của n và P trong input.

5.2 Bài 2: Đếm số chính phương từ hệ nhị phân

- Phân tích đề bài, ta thấy bài của ta được tách từ 2 bài toán như sau:

  - Biến đổi dãy nhị phân thành số thập phân.
  - Đếm số lượng số chính phương nằm trong khoảng $A \to B$ sau khi biến đổi.

- Vậy để giải bài toán này ta kết hợp 2 bài toán cùng nhau.

5.2.1 Biến đổi dãy nhị phân thành số thập phân

- Giả sử, ta có dãy bit S đánh dấu là $S_nS_{n-1}S_{n-2}...S_1S_0$ thì ta sẽ có giá trị của $S = S_n * 2^n + S_{n-1} * 2^{n-1} + ...S_1 * 2^1 + S_0 * 2^0$, tương ứng bit thứ i sẽ đóng góp vào tổng giá trị của S giá trị là $S_i * 2^i$. Vậy nên ta có thể duy trì giá trị của 2^i với từng bit sau đó cộng vào với từng vị trí i. Code mẫu như sau:

```
long long BinToNum(string s){
    long long cur = 0, sum = 0;
    for(int i = (int)s.size() - 1 ; i >= 0 ; i--){
        cur = cur * 2;
        if(s[i] == 1)
            sum += cur;
    }
    return sum;
}
```

5.2.2 Đếm số chính phương trong đoạn $A \to B$

- Sau khi biến đổi dãy bit thành 2 số A và B, ta sẽ thực hiện việc đếm số chính phương trong đoạn từ A đến B.
- Đầu tiên, ta hãy tiếp cận việc đếm với cách duyệt trâu từng số từ A đến B. Cách duyệt này có độ phức tạp là $O(B-A)$, đơn giản là vì ta duyệt từ A đến B nến số lần lặp là $B-A+1$ lần. Cách duyệt này chưa vượt qua được subtask đầu tiên. Nên ta cần 1 bước cải tiến.
- Đến đây ta sẽ có một bước cải tiến như sau. Nếu ta gọi các số chính phương cần đếm là X^2 (vì căn bậc hai của các số cần đếm là số nguyên) thì những số nguyên ta cần đếm là $A \le X^2 \le B \leftrightarrow \sqrt{A} \le X \le \sqrt{B}$. Vậy, ta cần đếm những số nguyên nằm trong đoạn $\sqrt{A} \to \sqrt{B}$ bằng một công thức là $\lfloor\sqrt{B}\rfloor - \lceil\sqrt{A}\rceil + 1$. Code mẫu của bài như sau:

```
#include <bits/stdc++.h>

using namespace std;

long long BinToNum(string s){
    long long cur = 1, sum = 0;
    for(int i = (int)s.size() - 1 ; i >= 0 ; i--){
        if(s[i] == '1')
            sum += cur;
        cur = cur * 2;
    }
    return sum;
}

int main() {
    string S1 , S2;
    cin >> S1 >> S2;
```

```
    long long A = BinToNum(S1), B = BinToNum(S2);

    long long L = sqrt(A);
    long long R = sqrt(B);

    if(L * L != A)
        L++;

    cout << R - L + 1;
}
```

5.3 Bài 3: Giao lưu

5.3.1 Subtask 1: $1 \le N \le 20$

- Với subtask này, ta có thể sử dụng **quay lui** để vét cạn, tức là liệt kê tất cả các trạng thái như một dãy nhị phân, với 0 là bạn nữ và 1 là bạn nam. Code mẫu như sau:

```
#include <bits/stdc++.h>

using namespace std;

int n , result = 0;
int a[21];

void Try(int x){
    if(x > n){
        for(int i = 3 ; i <= n ; i++)
            if(a[i] == a[i - 1] && a[i - 1] == a[i - 2] && a[i] == 0)
                return;

        result++;
        return;
    }

    a[x] = 0;
    Try(x + 1);
    a[x] = 1;
    Try(x + 1);
    a[x] = 0;
}

int main() {
    cin >> n;
    Try(1);
    cout << result << endl;
}
```

5.3.2 Subtask 1: $20 < N \leq 64$

- Với subtask này, giới hạn quá lớn để ta sử dụng quay lui. Lúc này ta cần suy nghĩ một thuật toán khác để có thể **AC** được bài toán. Đến đây, ta sẽ sử dụng **Quy Hoạch Động**.
- Ta sẽ có hàm Quy hoạch động như sau. Gọi $F_{i,j}$ là số cách để xếp i bạn vào hàng sao cho bạn cuối cùng là nam nếu $j = 0$ và là nữ nếu $j = 1$ không vi phạm điều kiện của đề bài (tức là không có 3 bạn nam liên tục). Ta có công thức truy hồi như sau:

$$F_{i,0} = F_{i-2,1} + F_{i-1,1} \forall i \leq n$$

$$F_{i,1} = F_{i-1,0} + F_{i-1,1} \forall i \leq n$$

- Giải thích cho công thức trên:

- Với $F_{i,0} = F_{i-2,1} + F_{i-1,1}$, vì không được có 3 bạn nam đứng liền nhau nên ta sẽ cộng thêm vào $F_{i,0}$ một lượng là $F_{i-2,1}$ (tượng trưng cho 2 bạn nam đứng liền nhau và trước đó là 1 bạn nữ) và $F_{i-1,1}$ (tượng trưng cho 1 bạn nam và trước đó là 1 bạn nữ). Ở đây, 1 bạn nữ là bạn nữ kết thúc tại $F_{i-2,1}$ và $F_{i-1,1}$.
- Với $F_{i,1} = F_{i-1,0} + F_{i-1,1}$, ta đang muốn thêm 1 bạn nữ vào trạng thái $F_{i-1,0}$ và $F_{i-1,1}$ để trở thành trạng thái $F_{i,1}$. Tức là hàng đang có $i-1$ bạn (không phân biệt nam nữ đứng cuối), ta thêm vào 1 bạn nữ đứng cuối hàng và hàng trở thành hàng có i bạn và có bạn nữ đứng cuối.

- Cuối cùng, kết quả của ta sẽ là $F_{n,0} + F_{n,1}$. Code mẫu như sau:

```
#include <iostream>

using namespace std;

int n;
long long F[65][2];

int main() {
    cin >> n;
    F[0][0] = F[0][1] = F[1][0] = F[1][1] = 1;

    for(int i = 2 ; i <= n ; i++){
        F[i][0] = F[i - 1][1] + F[i - 2][1];
        F[i][1] = F[i - 1][1] + F[i - 1][0];
    }

    cout << F[n][1] + F[n][0] << endl;
}
```

5.4 Bài 4: Lưu niệm

- Đối với bài toán này, hướng tiếp cận **Quy hoạch động** là một hướng tiếp cận mà đa số người làm sẽ hướng tới. Cụ thể là Quy hoạch động **cái túi**.
- Gọi $F_{i,j}$ là độ thẩm mỹ lớn nhất có thể đạt được khi xét i tấm ảnh đầu và số ảnh đã lấy chiếm j **MB**. Ta sẽ có công thức truy hồi như sau:

$$F_{i,j} = max(F_{i-1,j}, F_{i,j-a_i} + b_i)$$

- Giải thích cho công thức sau:

- Thêm vào 1 bức ảnh loại i, giả sử j là dung lượng sau khi thêm nên ta có $j - a_i$ là dung lượng trước khi thêm ảnh$\Rightarrow F_{i,j} = F_{i,j-a_i} + b_i$
- Không thêm vào bết kì ảnh loại i nào $\Rightarrow F_{i,j} = F_{i-1,j}$

- Code mẫu như sau:

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

int n , k;
long long dp[1005][5000];
pair<long long , long long> a[1005];

int32_t main(){
    ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
    cin >> n >> k;
    k *= 1024;
    for(int i = 1 ; i <= n ; i++)
        cin >> a[i].first >> a[i].second;

    for(int i = 1 ; i <= n ; i++){
        for(int j = 0 ; j <= k ; j++)
            if(j - a[i].first >= 0){
                dp[i][j] = max(dp[i - 1][j] , dp[i][j - a[i].first] + a[i].second);
            }else dp[i][j] = dp[i - 1][j];
    }

    cout << dp[n][k] << endl;
    return 0;
}
```

6 Đề Bến Tre

6.1 Câu 1:

Theo đề bài, mỗi lần số vi khuẩn sinh ra sẽ gấp đôi số vi khuẩn trước đó, hay ta có thể nói: tại thời điểm thử n thì số vi khuẩn bằng thời điểm $n-1 \times 2$. Vậy tại thời điểm thử n thì số vi khuẩn là 2^n.

6.1.1 Code mẫu tham khảo:

```cpp
#include <bits/stdc++.h>

using namespace std;

int main(){
    ios_base::sync_with_stdio(false);
    cin.tie(0);

    long long n;
    cin >> n;
    cout << (1ll << n);

    return 0;
}
```

6.2 Câu 2:

Theo đề yêu cầu, ta chỉ cần duyệt qua lại những số đã lặp qua và chọn ra duy nhất 1 số xuất hiện đúng 1 lần.

6.2.1 Code mẫu tham khảo:

```cpp
#include <bits/stdc++.h>

using namespace std;

const int MAX = 1e5 + 5;
int a[MAX], cnt[MAX], n;

int main(){
    ios_base::sync_with_stdio(false);
    cin.tie(0);

    cin >> n;
    for(int i = 1; i <= n; ++i){
        cin >> a[i];
```

```
        ++cnt[a[i]];
    }

    for(int i = 1; i <= n; ++i){
        if (cnt[a[i]] == 1){
            cout << a[i] << '\n';
            break;
        }
    }
    return 0;
}
```

6.3 Câu 3:

Theo giả thiết của đề, ta có $\frac{a_i+a_j+a_k}{3} = m \Leftrightarrow a_i + a_j + a_k = 3 \times m$.
Cách đơn giản là 3 for lồng nhau, để tìm ra cặp $a_i, a_j, a_k (i < j < k)$ thỏa mãn yêu cầu bài toán.
Độ phức tạp: $O(C(3,1000))$
Với $C(k,n)$ là tổ hợp chập k của n.
Nhưng để cải tiến chúng trở nên nhanh hơn, thì chúng ta sẽ sử dụng kĩ thuật "Lùa bò vào chuồng", sử dụng map để đếm, tương tự ý tưởng của bài toán Three Sums cơ bản trên $CSES$. Từ đề bài $a_i + a_j + a_k = 3 * m$, nên là mình sẽ cho chạy 2 for lồng với công thức toán là $3 * m - a_i - a_j$ với $(i < j, 1 \le i \le n-1, i+1 \le j \le n)$, xong sau đó là mình sẽ tăng giá trị a_i xuất hiện trong dãy số.
Độ phức tạp: $O(n^2 * log(n))$

6.3.1 Code mẫu tham khảo:

```
#include <bits/stdc++.h>
using namespace std;
#define fast ios_base::sync_with_stdio(false); cin.tie(NULL); cout.tie(NULL);
#define int long long
#define FOR(i,l,r) for(int i=l;i<=r;i++)
#define mii map<int,int>
const int N = (int)1e6+10;
int a[N];

signed main()
{
    fast;
    int n,m;
    cin >> n >> m;
    FOR(i,1,n) cin >> a[i];
    mii mp;
    int ans = 0;
```

```
    FOR(i,1,n-1)
    {
        FOR(j,i+1,n) ans += mp[3*m-a[i]-a[j]];
        mp[a[i]]++;
    }
    cout << ans;
    return 0;
}
```

6.4 Câu 4:

Nhận xét:
Yêu cầu của bài toán là chọn các các đơn đặt phòng sao cho thời gian không giao nhau và tiền thuê nhận được là nhiều nhất.
Định nghĩa DP:
Gọi DP_i là số tiền thuê nhiều nhất lấy được sao cho thời gian kết thúc của tất cả các đơn đặt phòng được chọn đều $\leq i$
Suy ra công thức DP Xét 1 yêu cầu có dạng $[L;R]$, giả sử đơn đặt phòng này có thời gian kết thúc lớn nhất so với các yêu cầu đã chọn thì những yêu cầu trước buộc phải có thời gian kết thúc $\leq L$.
Từ nhận xét trên, ta thấy với 1 đơn đặt phòng (L,R,C), DP_R sẽ được cập nhật là $DP_R = max(DP_L + C)$.
Vậy ta có công thức QHĐ như sau: $DP_R = max(DP_L + C) \forall R \leq 105$ và $\forall L$ tồn tại đơn đặt phòng (L,R,C)
Khởi gán:
$DP_0 = 0$
Đáp án:
Đáp án của ta là DP_{105} với ý nghĩa là số tiền thuê kiếm được là nhiều nhất có thể với các yêu cầu được chọn có thời gian kết thúc ≤ 105.

6.4.1 Code mẫu tham khảo:

```
#include<bits/stdc++.h>
using namespace std;
struct hoitruong
{
        long long x,y,z;
};
bool cmp (hoitruong a , hoitruong b)
{
        return (a.x<b.x||a.x==b.x&&a.y<b.y);
}
long long n,ans=-1e18,f[1000004];
hoitruong a[1000004];
```

```
int main ()
{
        cin>>n;
        for(int i =1 ;i<=n;i++) cin>>a[i].x>>a[i].y>>a[i].z;
        sort(a+1,a+1+n,cmp);
        for(int i = 1;i<=n;i++)
        {
                f[i]=a[i].z;
                for(int j = 1 ;j<i;j++)
                {
                        if(a[i].x>=a[j].y) f[i]=max(f[i],f[j]+a[i].z);
                }
        }
        for(int i =1 ;i<=n;i++)
         ans=max(ans,f[i]);
         cout<<ans;
}
```

7 Đề Bình Dương

7.1 Câu 1 (4,0 điểm)

7.1.1 Hướng dẫn giải

Thuật toán: Hai con trỏ + Cấu trúc dữ liệu
Lý thuyết Hai con trỏ: Kĩ thuật hai con trỏ
Lý thuyết về multiset trong C++: std::multiset
Với chuỗi con liên tiếp thứ i ($1 \le i \le m-n$) có độ dài n trong chuỗi B, ta sẽ kiểm tra xem i có phải là một hoán vị của chuỗi A không. Ở đây có nhiều cách để tối ưu, cách được trình bày ở đây là sử dụng cấu trúc dữ liệu multiset trong C++ để kiểm tra chuỗi i có phải là một hoán vị của chuỗi A không. (Tạm gọi multiset của chuỗi A và i lần lượt là s, t)
Ở đây, ta sẽ cố định một hoán vị ban đầu của chuỗi A, bằng cách cho mọi kí tự của A vào s. Khi đó, ta sử dụng kĩ thuật hai con trỏ để kiểm tra chuỗi i có phải là hoán vị của A không:
- Với n kí tự đầu của B, ta cho các kí tự đó vào t. Sau đó, với i đầu tiên, ta kiểm tra s có bằng t không.
- Với các kí tự còn lại, ta tìm và loại bỏ kí tự cuối cùng của chuỗi $i-1$ và thêm kí tự kế tiếp của chuỗi i, và lặp lại cho đến m.
Với độ phức tạp tuyến tính từ việc duyệt kí tự và độ phức tạp logarithm từ cấu trúc dữ liệu, độ phức tạp của cách làm này là $O(mlog_2(n)$.

7.1.2 Code mẫu

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

```
int n, m, ans;
string s, t;
multiset<char> a, b;

int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);
    cin >> n >> m;
    cin >> s >> t;
    for (int i = 0; i < n; ++i)
    {
        a.insert(s[i]);
        b.insert(t[i]);
    }
    ans = (a == b);
    for (int i = n; i < m; ++i)
    {
        b.erase(b.find(t[i - n]));
        b.insert(t[i]);
        ans += (a == b);
    }
    cout << ans;
    return 0;
}
```

7.2 Câu 2 (4,0 điểm)

7.2.1 Hướng dẫn giải

Thuật toán: Đệ quy có nhớ + Quy tắc đếm

Lý thuyết Đệ quy: Đệ quy và quay lui

Vì dữ liệu đề bài $n, k \leq 50$ nên ta sử dụng thuật toán đệ quy cho bài toán này. Mỗi lần gọi đệ quy, ta lưu giá trị n, k hiện tại vào một mảng hai chiều $dp_{n,k}$.

Ta sẽ chia bài toán thành 3 trường hợp như sau:

- Trường hợp cơ sở: Nếu $n = 1$, chúng ta luôn có 1 và chỉ 1 cách để hoàn thành bức tranh.

- Trường hợp đặc biệt: Nếu $n \leq k$, ta sẽ dùng những mảnh ghép đơn lẻ để hoàn thành bức tranh. Vì vậy số cách để hoàn thành bức tranh là 2^{n-1}.

- Các trường hợp còn lại: Đối với số mảnh ghép lớn hơn số mảnh tối đa để ghép, chúng ta thử tất cả các cách ghép từ 1 đến k. Để làm điều này, chúng ta sử dụng một vòng lặp từ 1 đến k và thực hiện gọi đệ quy với số mảnh ghép còn lại $(n - i)$ và số mảnh tối đa để ghép k. Kết quả của mỗi lần đệ quy được cộng dồn vào biến ans.

Với việc gọi k lần đệ quy, mỗi lần đệ quy chạy vòng lặp k số, ta có độ phức tạp cho hàm đệ quy này là $O(k^2 log_2 k)$, và đây cũng là độ phức tạp cho bài giải này.

7.2.2 Code mẫu

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

int n, k;
long long dp[55][55];

long long backtrack(const int &n, const int &k)
{
    if (n == 1) return 1;
    if (n <= k) return pow(2, n - 1);
    if (dp[n][k] > 0) return dp[n][k];
    long long ans = 0;
    for (int i = 1; i <= k; ++i) ans += backtrack(n - i, k);
    return dp[n][k] = ans;
}

int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);
    cin >> n >> k;
    cout << backtrack(n, k);
    return 0;
}
```

7.3 Câu 3 (6,0 điểm)

7.3.1 Hướng dẫn giải

Thuật toán: Tham lam

Lý thuyết Tham lam: Thuật toán Tham lam

Ngay khi nhìn vào đề, chúng ta sẽ nghĩ ngay được một cách làm đó là sắp xếp dãy tăng dần, khi đó các biểu thức max đã được phá, việc còn lại ta sẽ cộng hiệu của hai phần tử liên tiếp nhau. Sau khi tính xong ta cộng thêm tổng dãy vào đáp án.

Nhưng thuật toán trên chưa đúng, tại sao? Chúng ta đi vào test ví dụ.

Dãy sau khi sắp xếp tăng dần là 1, 4, 5, 7.

Với cách làm trên, nếu gọi biến đáp án là ans ta sẽ có

$ans = 1+4+5+7+(4-1)+(5-4)+(7-5) = 23$, khác với kết quả bài toán.

Vậy chúng ta sai vì sao? Ta có thể thấy, ans lớn nhất khi các biểu thức $a_{i+1}-a_i$ lớn nhất. $a_{i+1}-a_i$ lớn nhất khi a_{i+1} lớn nhất và a_i nhỏ nhất. Khi thực hiện sắp xếp theo giá trị tăng dần, với mỗi vị trí i, ta đã có a_i nhỏ nhất có thể, việc chúng ta cần làm là xác định a_{i+1} lớn nhất. Dễ dàng thấy được rằng ta lấy a_{i+1} là các phần tử cuối cùng thì kết quả biểu thức $a_{i+1}-a_i$ càng tối ưu.

Với độ phức tạp tuyển tính từ việc duyệt từng phần tử của dãy và độ phức tạp tuyển tính logarithm từ thuật toán sắp xếp có sẵn của C++, độ phức tạp tổng quát của cách làm này là $O(nlog_2n)$.

7.3.2 Code mẫu

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

const int N = 1e5 + 5;
int arr[N];
int n, ans;

int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);
    cin >> n;
    for (int i = 1; i <= n; ++i)
    {
        cin >> arr[i];
        ans += arr[i];
    }
    sort(arr + 1, arr + n + 1);
    for (int i = 1; i * 2 <= n; ++i) ans += arr[n - i + 1] - arr[i];
    cout << ans;
    return 0;
}
```

7.4 Câu 4 (6,0 điểm)

7.4.1 Hướng dẫn giải

Thuật toán: Tham lam

Lý thuyết Tham lam: Thuật toán Tham lam

Mẹo cho bài này như sau: Trong n phần thưởng, cô giáo sẽ tặng cho tổ 1 $n/2$ phần thưởng mà giá trị yêu thích của tổ 1 lớn hơn giá trị yêu thích của tổ 2, còn $n/2$ phần thưởng còn lại cô giáo sẽ tặng cho tổ 2. Cụ thể, ta tính bằng cách sắp xếp các giá trị theo $a_i - b_i$ giảm dần, sau đó $n/2$ giá trị đầu tiên cho tổ 2, $n/2$ còn lại cho tổ 1.

Chứng minh giải thuật tham lam trên là đúng đắn: Giả sử tồn tại một giá trị yêu thích (i, j) mà thay vì tặng i cho tổ 1, j cho tổ 2 (gọi là cách 1), thì đổi ngược lại, tức i cho tổ 1 và j cho tổ 2 (gọi là cách 2) thì lời giải tối ưu hơn. Ta sẽ chỉ ra cách này không tối ưu.

Cụ thể theo giải thuật tham lam trên, ta có $a_i - b_i > a_j - b_j$. Gọi S là tổng giá trị của $n-2$ phần thưởng còn lại đã tối ưu. Khi đó cách 1 có $S_1 = S + a_i + b_j$ và cách 2 có $S_2 = S + a_j + b_i$. Dễ thấy $S_1 > S_2$, do đó cách 2 không tối ưu, nên giải thuật tham lam đúng đắn.

Với độ phức tạp tuyến tính từ duyệt các phần tử, độ phức tạp tuyến tính logarithm từ thuật toán sắp xếp có sẵn của C++, độ phức tạp tổng quát của cách làm này là $O(nlog_2n)$.

7.4.2 Code mẫu

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

const int N = 5e5 + 5;
pair<int, int> arr[N];
int n, ans;

bool cmp(const pair<int, int> &a, const pair<int, int> &b)
{
    return a.second - a.first > b.second - b.first;
}

int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);
    cin >> n;
    for (int i = 1; i <= n; ++i) cin >> arr[i].first >> arr[i].second;
    sort(arr + 1, arr + n + 1, cmp);
    for (int i = 1; 2 * i <= n; ++i) ans += arr[i].second;
    for (int i = n / 2 + 1; i <= n; ++i) ans += arr[i].first;
    cout << ans;
    return 0;
}
```

8 Đề Cần Thơ

8.1 Bài 1: Đếm số có k chữ số

Ý tưởng: Nhận thấy $k \leq 10$ và k là số nguyên dương nên có thể tìm số bé nhất có k chữ số và số lớn nhất có k chữ số (Tìm tuỳ ý, ưu tiên cách tìm có độ phức tạp bé nhất). Ta có thể tìm số bé nhất có k chữ số và số lớn nhất có k chữ số bằng công thức sau: Minvalue = 10^{k-1}, Maxvalue = 10^k-1 .Sau đó kiểm tra từng giá trị của mảng a có nằm trong khoảng [Minvalue,Maxvalue] hay không. Nếu có thì cập nhật đáp án lên 1 đơn vị.
Độ phức tạp: $O(n)$

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

long long n,k,ans;
```

```
long long pow(int a,int b){
    if (b == 0) return 0;
    long long res = 1;
    for (int i = 1; i <= b; i++)
        res *= 10;
    return res;
}

main(){
    ios::sync_with_stdio(0);
    cin.tie(0); cout.tie(0);
    cin >> n >> k;
    long long minval = pow(10,k-1);
    long long maxval = pow(10,k)-1;
    while (n--){
        int x; cin >> x;
        if (minval <= x && x <= maxval) ans++;
    }
    cout << ans;
}
```

8.2 Bài 2: Mã hoá văn bản

Ý tưởng: Ta đã có câu lệnh tìm kí tự đứng sau một kí tự cho trước (đề cho). Nhưng nếu thí sinh sử dụng ngôn ngữ lập trình $C++$ thì ta cần lập thêm trường hợp nếu kí tự ta nhận là $'Z'$ thì trả về $'A'$ (Ta có thể viết hàm cho trường hợp này) và những ngôn ngữ khác cũng tìm cách xử lý phù hợp. Tiếp đó ta duyệt từng kí tự trong xâu S và kiểm tra xem kí tự S_i có phải kí tự tiếng Anh in hoa không?. Nếu có thì áp dụng cách xử lý đã nêu trên.
Độ phức tạp: $O(n(S))$ với $n(S)$ là độ dài xâu S.

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

string s;

char get(char c){
    if (c == 'Z')
        return 'A';
    else
        return (char) (c+1);
}

main(){
    ios::sync_with_stdio(0);
    cin.tie(0); cout.tie(0);
    getline(cin,s);
    for (auto c : s){
```

```
        if ('A' <= c && c <= 'Z')
            cout << get(c);
        else
            cout << c;
    }
}
```

8.3 Bài 3: Tìm đơn giá

Ý tưởng: Ta chia bài toán trên thành các trường hợp sau

$\begin{cases} pa = x \\ 9a + (q-9)b = y \end{cases}$ p<10,$10 \le q$.

$\begin{cases} 9a + (p-9)b = x \\ qa = y \end{cases}$ $10 \le p, q < 10$.

$\begin{cases} pa = x \\ qa = y \end{cases}$ p<10,q< 10.

$\begin{cases} 9a + (p-9)b = x \\ 9a + (q-9)b = y \end{cases}$ $10 \le p, 10 \le q$.

Và rút ra công thức tính a, b từ những trường hợp trên.

Độ phức tạp: $O(1)$

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

long long p,q,x,y;

main(){
    ios::sync_with_stdio(0);
    cin.tie(0); cout.tie(0);
    cin >> p >> q >> x >> y;
    long long a,b;
    if (p < 10 && q >= 10){
        a = x / p;
        b = (y - 9*a) / (q - 9);
    }
    else if (p >= 10 && q < 10){
        a = y / q;
        b = (x - 9*a) / (p - 9);
    }
    else if (p < 10 && q < 10){
        a = x / p;
        b = 0;
    }
    else{
        b = (y - x) / (q - p);
        a = (x - (p-9)*b)/9;
```

```
    }
    cout << a << " " << b;
}
```

8.4 Bài 4: Vận chuyển hàng hoá

8.4.1 Subtask 1,2: $m,n \leq 10^5$.

Vì mảng a tăng dần theo giá trị và có các giá trị đôi một khác nhau nên ta duyệt từng hàng hoá và tìm thiết bị có khả năng vận chuyển từ b_i (b_i là khối lượng của hàng hoá thứ i) kilogam trở lên.

Độ phức tạp: $O(nlog2_m)$

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

const int maxM = 1e5+5;
int m,n,x,a[maxM];

main(){
    ios::sync_with_stdio(0);
    cin.tie(0); cout.tie(0);
    cin >> m >> n;
    for (int i = 1; i <= m; i++){
        cin >> a[i];
    }
    for (int i = 1; i <= n; i++){
        int x; cin >> x;
        int *it = lower_bound(a+1,a+m+1,x);
        cout << it - a << " ";
    }
}
```

8.5 Bài 5: Bảng số

8.5.1 Subtask 1: $m,n \leq 10^2$.

Vì m,n nhỏ nên ta xây dựng mảng hai chiều A với quy luật như đề mô tả. Bằng cách duyệt hàng duyệt cột và tạo một biến giá trị, lúc nào qua ô mới thì tăng giá trị lên 1. Kết quả là $A[x][y]$.

Độ phức tạp: $O(m*n)$

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

const int maxM = 1e2+5;
int m,n,x,y,a[maxM][maxM];
```

```
main(){
    ios::sync_with_stdio(0);
    cin.tie(0); cout.tie(0);
    cin >> m >> n >> x >> y;
    int value = 1;
    for (int i = 1; i <= m; i++){
        for (int j = 1; j <= n; j++){
            a[i][j] = value;
            value++;
        }
    }
    cout << a[x][y];
}
```

8.5.2 Subtask 2: $m, n \leq 10^6$.

Ý tưởng: Để ý thấy lúc hàng là **số lẻ** thì những giá trị được sắp từ bé đến lớn và hàng là **số chẵn** thì được sắp từ lớn đến bé. Ta chia hai trường hợp, nếu hàng là **số lẻ** thì đáp án là $(x-1)*n+y$. Nhưng nếu hàng là **số chẵn** ta cho tất cả giá trị của hàng đó vào một vector (hoặc mảng hay thứ gì khác) và sắp xếp từ bé đến lớn, đáp án là giá trị ở chỉ số $y-1$ của vector.

Độ phức tạp: $O(nlog_n)$

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

long long n,m,x,y;
vector < long long > v;

main(){
    ios::sync_with_stdio(0);
    cin.tie(0); cout.tie(0);
    cin >> m >> n >> x >> y;
    x -= 1;
    if (x % 2 == 0){
        cout << x*n + y;
    }
    else{
        for (long long i = x*n + 1; i <= x*n + n; i++) v.push_back(i);
        sort(v.begin(),v.end(),greater<long long>());
        cout << v[y-1];
    }
}
```

9 Đề Đà Nẵng

9.1 Bài 1

Ta chuẩn bị một mảng $isprime_i$, để kiểm tra tính nguyên tố của i ($1 \le i \le 1500$), vì i nhỏ nên bạn có thể duyệt mọi i và kiểm tra tính nguyên tố của từng số!

9.1.1 Subtask 1: $r \le 10^6$

Ở subtask này, ta thực hiện duyệt trâu từ L đến R, với mỗi số ta tính tổng các chữ số và tổng chữ số bình phương, sau đó kiểm tra tính nguyên tố nếu 2 tổng đều là số nguyên tố vậy số đang xét là một số anh em.

9.1.2 Subtask 2,3: $r \le 10^{18}$

Lúc này số lượng N tương đối lớn, ta gọi $recur(i)$ là số lượng số có tổng chữ số và tổng chữ số bình phương là số nguyên tố đồng thời các số $\le i$, dễ thấy đáp án của bài là $recur(r) - recur(l-1)$.

Để tính, $recur_x$ ta cần sử dụng kỹ thuật **dpdigit**, ta định nghĩa **dp**[**i**][**j**][**sum**][**sumsquare**] là số lượng số sau khi xây dựng xong i chữ số đầu tiên (tính từ hàng đơn vị lên cao) , $j = 0$ thì chữ số của số ta đang xây dựng phải bé hơn hoặc bằng chữ số x, $j = 1$ thì không có ràng buộc thế (tức chữ số có thể chạy từ $0 -> 9$), sum là tổng các chữ số của x chữ số đầu tiên, **sumsquare** là tổng chữ số bình phương của x chữ số đầu tiên. lúc này ta thấy nếu ta để chữ số hàng i ta nhét được từ 0 đến 9 nếu j=1 và 0 đến x_i nếu j=0. Ta có thể công thức truy hồi như code mẫu bên dưới

Code mẫu

```
#include <bits/stdc++.h>
using namespace std;
#define int long long
#define all(a) a.begin(), a.end()
string s;
int dp[20][2][165][1460],l,r;
bool prime[1460];
int recur(int pos, bool sm=0, int dsum=0, int dsqsum=0){
    if(pos==-1) return (prime[dsum] and prime[dsqsum]);

    if(dp[pos][sm][dsum][dsqsum] != -1) return dp[pos][sm][dsum][dsqsum];

    int lim = sm?9:s[pos]-'0';
    int ans=0;
    for(int d = 0; d <= lim; d++) ans+=recur(pos-1,sm or d!=lim, dsum+d, dsqsum+d*d);
    dp[pos][sm][dsum][dsqsum]=ans;
    return ans;
}
```

```
bool isprime(int x){
    for(int i = 2; i*i<=x; i++)if(x%i==0) return false;
    return true;
}

signed main(){
    ios::sync_with_stdio(0);cin.tie(0);
    for(int i = 2; i < 1460; i++) prime[i]=isprime(i);
    memset(dp, -1, sizeof(dp));
    cin >> l >> r;
    s = to_string(r); reverse(all(s));
    r = recur((int)s.size()-1);

    memset(dp, -1, sizeof(dp));
    s = to_string(l-1); reverse(all(s));
    l= recur((int)s.size()-1);

    cout << r-l;
    return 0;
}
```

9.2 Bài 2

9.2.1 sub 1: $|s| \leq 1000$

Đầu tiên ta gọi mảng $C_{i,j} = true$ nếu xâu i đến j là xâu đối xứng, ta duyệt i, j theo độ tăng dần của $j-i$, gợi ý có thể $forstep = 0-> n-1, for(inti = 0, j = i+step; j < n; i++, j++)$, lúc này nhận thấy $C_{i,j} = true$ nếu $j-i == 0$ hoặc ($s_i = s_j$ và $C_{i+1,j-1} = true$. phần còn lại ta đếm số lượng C có giá trị true là được.

9.2.2 sub 2: $|s| \leq 10000$

Ta không cần duy trì mảng C làm gì cả, lúc này ta đưa bài toán thành 2 phần đếm số lượng xâu đối xứng chẵn và xâu đối xứng lẻ. Với xâu đối xứng lẻ, ta luôn có vị trí ở giữa (được gọi là ký tự trung tâm), ta duyệt từ i từ 0 đến $n-1$, coi i như vị trí trung tâm, ta có $l = i-1$ và $r = i+1$, nếu $s_l = s_r$ thì xâu $l-> r$ là xâu đối xứng sau đó ta sẽ tăng r lên 1 và giảm l xuống 1 còn nếu $s_l != s_r$ thì lúc này ta dừng do không còn xâu đối xứng nào nhận i là ký tự trung tâm nữa. Đối với xâu đối xứng chẵn, thì lúc này vị trí trung tâm lúc này là 2 ký tự liền kề nhau thay vì 1 vì thế ta duyệt các vị trí i có ký tự i=ký tự $i+1$, sau đó gán $l = i-1$ và $r = i+2$ rồi lặp lại thuật toán như xâu lẻ!

9.2.3 sub 3: $|s| \leq 100000$

Đây là ý tưởng của thuật toán manacher, bạn có thể đọc tài liệu ở đây https://vnoi.info/wiki/algo/strin sau khi ta xây dựng mảng d_i việc của ta bây giờ là tính tổng các $(d_i+1)/2$ $(0 \leq i \leq |s|)$

Code mẫu

```
#include <bits/stdc++.h>
#define F first
#define S second
#define pb push_back
#define all(x) (x).begin(), (x).end()
using namespace std;

int p[1000005];
string s;

void manacher() {
    string str(2 * s.size() + 1, '#');
    for (int i = 1; i < str.size(); i += 2) str[i] = s[i / 2];
    int best = 0;
    for (int i = 0; i < str.size(); i++) {
        if (i > best + p[best]) p[i] = 0;
        else p[i] = min(p[best * 2 - i], best + p[best] - i);
        while (i - p[i] >= 0  and i + p[i] < str.size() and str[i + p[i]] == str[i - p[i]]
        if (i + p[i] > best + p[best]) best = i;
    }
    for (int i = 0; i < str.size(); i++) {
        p[i] = p[i] * 2 - 1;
    }
}

signed main() {
    ios_base::sync_with_stdio(0), cin.tie(0);
    cin >> s;
    manacher();
    int ans=0;
    for (int i = 1; i < s.size() * 2; i++) {
        ans+=(p[i]/2+1)/2;
    }
    cout<<ans;
    return 0;
}
```

9.3 Bài 3

9.3.1 $N \leq 500$

Ở bài toán này ta duyệt mọi i, với mỗi i ta đếm ước của A_i, để đếm bạn chỉ cần for $j = 1 -> A_i$ xong kiểm tra j có là ước của i không.

9.3.2 $N \leq 1000$

Ta cải tiến việc đếm bằng thủ thuật đếm ước trong $\sqrt{n}$, nhận thấy a là ước của b, nếu a lớn hơn $\sqrt{b}$ thì nhận thấy b/a sẽ bé hơn hoặc bằng $\sqrt{n}$, vậy duyệt từ 1 đến $\sqrt{n}$ nếu i là ước thì n/i cũng là ước và n/i nhỏ hơn $\sqrt{n}$. Từ đó ta không chỉ đếm được các ước nhỏ hơn hoặc bằng $\sqrt{n}$ mà còn đếm được cả các ước lớn hơn $\sqrt{n}$.

9.3.3 $N \leq 10^5$

Chúng ta tiếp tục cải tiến việc đếm, thay vì giờ mỗi x ta tiến hành đếm ước của x thì giờ ta đảo ngược quá trình, với mỗi x có bao nhiêu số nhận x là ước, rõ ràng các số đó chính là bội của x, và vì vậy ta duy trì F_x là số lượng ước của x, ta tiến hành duyệt x từ 1 đến 10^6 sau đó ta duyệt mọi bội của x, và tăng F_j lên 1. về độ phức tạp, mỗi x số lượng bội là $10^6/x$, ta có độ phức tạp $10^6/1 + 10^6/2 + 10^6/3...$ đặt 10^6 làm chung ta có $10^6 * (1/1 + 1/2 + 1/3 + ... + 1/10^6)$, tổng bên trong kia nếu đưa về tích phân thì tính ra được độ phức tạp là $log(10^6)$, vậy độ phức tạp tổng thể là $10^6 * log(10^6)$, và ý tưởng duyệt cũng là cảm hứng cho thuật toán sàng nguyên tố Eratosthenes.

Code mẫu

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

const int N=1e6+10;


long long x,F[N],ans,n;
int main(){
    ios::sync_with_stdio(0);cin.tie(0);
    for(int i=1; i<=1000000; i++){
      for(int j=i; j<=1000000; j+=i){
        F[j]++;
      }
    }
    cin>>n;
    for(int i=1; i<=n; i++){
        cin>>x;
        ans+=x*F[x];
    }
    cout<<ans;



    return 0;
}
```

9.4 Bài 4

9.4.1 Subtask 1: $N \leq 100$

Ý tưởng chính là dùng dp knapsack, gọi $dp_{i,j}$ là số lượng dãy con trong i số đầu tiên có tổng bằng j. Trường hợp cơ sở là, $dp_{i,A_i} = 1$. Xây dựng $dp_{i,j}$ ta có 2 trường hợp 1 là dãy con có phần tử i lúc này $dp_{i,j} = dp_{i-1,j-A_i}$ và trường hợp 2 là dãy con không có phần từ i lúc này $dp_{i,j} = dp_{i-1,j}$ ta lấy tổng hai trường hợp là coi như xây dựng xong $dp_{i,j}$.

9.4.2 Subtask 2: $N \leq 1000$

Vẫn với ý tưởng trên, tuy nhiên nếu dùng mảng $dp_{i,j}$ thì có thể sẽ bị vượt quá giới hạn bộ nhớ. Lúc này, ta sử dụng kỹ thuật con lắc, ta dùng $dp_{i\%2,j}$ để lưu dãy con trong i số đầu tiên có tổng bằng j, ý tưởng ở đây là tận dụng lại cái ta không sử dụng, nhận thấy ta dùng $dp_{1,j}$ để lưu dãy con trong 1 số đầu tiên và $dp_{0,j}$ là lưu dãy con trong 2 số đầu tiên, lúc này khi muốn tính dãy con trong 3 số đầu tiên ta cũng chỉ cần dựa trên dãy con trong 2 số đầu tiên (tức là $dp_{0,j}$) ta không sử dụng đến $dp_{1,j}$ nên ta sẽ tận dụng $dp_{1,j}$ như chỗ để chứa cho dãy con trong 3 số đầu tiên. Với cách này ta chỉ cần khai mảng $dp_{2,10^5}$

9.4.3 subtask 3: $N \leq 10^5$

Subtask này rất rất khó không chỉ so với đề tuyển sinh mà còn so với đề thi học sinh giỏi quốc gia (sử dụng kiến thức mà cho dù có biết vẫn rất ngại cài trong các cuộc thi offline). Có thể tham khảo sol thông qua blog của Demen: https://hackmd.io/@DeMen100ms/rJK-Wipd2

Code mẫu cho sub 2

```
        #include<bits/stdc++.h>
using namespace std;
const int mod=1e9+7;
const int N=105;
int n,S,dp[2][10005],A[N];

int main(){
    ios::sync_with_stdio(0);cin.tie(0);
    cin>>n>>S;
    for(int i=1; i<=n; i++){
        cin>>A[i];
    }
    dp[1][A[1]]=1;
    dp[1][0]=1;
    for(int i=2; i<=n; i++){
        for(int j=0; j<=S; j++){
            dp[i%2][j]=dp[(i-1)%2][j];
            if(j>=A[i]) dp[i%2][j]=(dp[i%2][j]+dp[(i-1)%2][j-A[i]])%mod;
        }
    }
```

```
    cout<<dp[n%2][S];
    return 0;
}
```

10 Đề Đắk Lắk

10.1 Bài 1

Với $k \le 10^{12}$ là rất lớn để ta có thể tính S bằng vòng lặp. Vì vậy, ta cần thực hiện rút gọn biểu thức S.

Nhận xét: $\frac{1}{(2k+1)(2k+3)} = \frac{1}{2}\cdot\frac{2}{(2k+1)(2k+3)} = \frac{1}{2}\cdot\frac{(2k+3)-(2k+1)}{(2k+1)(2k+3)} = \frac{1}{2}\cdot\left(\frac{1}{2k+1}-\frac{1}{2k+3}\right)$.

Khi đó:

$S = \frac{1}{1.3}+\frac{1}{3.5}+\cdots+\frac{1}{(2k+1)(2k+3)} = \frac{1}{2}\cdot\left(1-\frac{1}{3}\right)+\frac{1}{2}\cdot\left(\frac{1}{3}-\frac{1}{5}\right)+\cdots+\frac{1}{2}\cdot\left(\frac{1}{2k+1}-\frac{1}{2k+3}\right) = \frac{1}{2}\cdot\left(1-\frac{1}{2k+3}\right)$.

Code mẫu

```
#include <bits/stdc++.h>
using namespace std;
#define int long long
long long k;

signed main() {
    ios_base::sync_with_stdio(0); cin.tie(0);
    cin >> k;
    cout << fixed << setprecision(5) << 0.5 * (1 - 1.0 / (2 * k + 3));
    return 0;
}
```

10.2 Bài 2

Vì số chính phương là số có giá trị bằng bình phương của một số tự nhiên nên ta chỉ cần đếm số lượng số tự nhiên mà bình phương của nó thuộc đoạn $[M,N]$.

Một số tự nhiên x thỏa mãn điều kiện trên tức là $M \le x^2 \le N \Rightarrow \sqrt{M} \le x \le \sqrt{N}$.

Như vậy ta chỉ cần đếm số lượng số tự nhiên thuộc đoạn $[\sqrt{M},\sqrt{N}]$.

Lưu ý: Số lượng số tự nhiên thuộc đoạn $[L,R]$ (L,R nguyên) là $R-L+1$. Hàm $ceil(x)$ có công dụng làm tròn số thực x lên đến giá trị nguyên gần nhất.

Code mẫu

```
#include <bits/stdc++.h>
using namespace std;
#define int long long
long long M, N;
```

```
signed main() {
    ios_base::sync_with_stdio(0); cin.tie(0);
    cin >> M >> N;
    int l = ceil(sqrt(M));
    int r = sqrt(N);
    cout << r - l + 1;
    return 0;
}
```

10.3 Bài 3

Ta sẽ xử lý từng cặp kí tự liên tiếp trong xâu đã cho. Kí tự đầu tiên (gọi là c) là kí tự chữ cái và kí tự thứ hai (gọi là d) là số lượng vị trí cần dịch chuyển.

Vì $d \leq 9$ là rất nhỏ nên ta có thể sử dụng vòng lặp for để dịch chuyển từng vị trí một của kí tự c. Lưu ý ở đây là vị trí tiếp theo của chữ 'Z' là chữ 'A'.

Code mẫu

```
#include <bits/stdc++.h>
using namespace std;
#define int long long
string s;

signed main() {
    ios_base::sync_with_stdio(0); cin.tie(0);
    cin >> s;
    for(int i = 0; i < s.size(); i += 2) {
        char c = s[i];
        int d = s[i + 1] - '0';
        for(int j = 1; j <= d; j++) {
                if (c == 'Z') c = 'A';
                else c++;
        }
        cout << c;
    }
    return 0;
}
```

10.4 Bài 4

Để ý rằng hai số nguyên dương x, y chia cho k có cùng số dư thì hiệu của chúng $y - x$ sẽ chia hết cho k.

Để cho dễ dàng, ta sẽ tính hiệu của tất cả các phần tử trong dãy số cho phần tử nhỏ nhất trong đó. Như vậy, các phần tử trong dãy số khi chia cho k có cùng số dư khi và chỉ khi tất cả các giá trị hiệu trên đều chia hết cho k. Hay nói cách khác, k sẽ là ước của ước chung lớn

nhất của tất cả các giá trị hiệu này.

Vì vậy, ta cần tìm ước chung lớn nhất G của các giá trị hiệu trên. Khi đó, tất cả các số K cần tìm sẽ là ước của G.

Code mẫu

```
#include <bits/stdc++.h>
using namespace std;
#define int long long
int n, a[101];

signed main() {
    ios_base::sync_with_stdio(0); cin.tie(0);
    cin >> n;
    for(int i = 1; i <= n; i++) cin >> a[i];
    sort(a + 1, a + n + 1);
    int G = a[2] - a[1];
    for(int i = 3; i <= n; i++) G = __gcd(G, a[i] - a[1]);

    set<int> div;
    for(int i = 1; i <= sqrt(G); i++) {
        if (G % i == 0) {
                if (i > 1) div.insert(i);
                if (G / i > 1) div.insert(G / i);
        }
    }
    if (div.empty()) cout << -1;
    else for(auto u : div) cout << u << " ";
    return 0;
}
```

10.5 Bài 5

Số lượng chữ số của P là tổng số lượng các chữ số của $x^2,(x+1)^2,...,y^2$.

Xét ví dụ sau:
Ta có: $\sqrt{99} \approx 9.95$ và $\sqrt{999} \approx 31.61 \Rightarrow$ Các số tự nhiên thuộc đoạn $[10,31]$ sẽ có giá trị bình phương thuộc $(99,999]$, hiển nhiên chúng sẽ có 3 chữ số !

Như vậy, ta sẽ tính trước các giá trị $\sqrt{9},\sqrt{99},\sqrt{999},...,\sqrt{10^{24}-1}$ và lưu vào mảng pre. Dễ thấy như ở ví dụ trên, các số tự nhiên từ pre_i+1 đến pre_{i+1} khi bình phương sẽ có cùng số lượng các chữ số.

Chia đoạn $[x,y]$ thành các đoạn $[x,pre_i],[pre_i+1,pre_{i+1}],...,[pre_{j-1}+1,pre_j],[pre_j+1,y]$ (nếu không chia được tức là $[x,y]$ thuộc một đoạn $[pre_i+1,pre_{i+1}]$ nào đó). Các số tự nhiên trong từng đoạn như vậy khi bình phương sẽ có cùng số lượng các chữ số. Từ đó, việc tính toán tổng số lượng các chữ số của $x^2,(x+1)^2,...,y^2$ có thể được thực hiện dễ dàng bằng cách duyệt qua các đoạn trên.

Trước hết, ta cần tìm chỉ số i nhỏ nhất sao cho $pre_i \geq x$ và chỉ số j lớn nhất sao cho $pre_j < y$. Lưu ý, nếu $j < i$ thì tức là đoạn $[x, y]$ không chia được như trên.

Code mẫu

```
#include <bits/stdc++.h>
using namespace std;
#define int long long
long long x, y;
long long pre[] = {0, 3, 9, 31, 99, 316, 999, 3162, 9999, 31622, 99999, 316227,
999999, 3162277, 9999999, 31622776, 99999999, 316227766, 999999999, 3162277660,
9999999999, 31622776601, 99999999999, 316227766016, 999999999999};

signed main() {
    ios_base::sync_with_stdio(0); cin.tie(0);
    cin >> x >> y;
    int i = 0, j = 24;
    while(pre[i] < x) i++;
    while(pre[j] >= y) j--;
    if (j < i) cout << (y - x + 1) * i;
    else {
        int ans = 0;
        ans += (pre[i] - x + 1) * i;
        for(int z = i + 1; z <= j; z++) ans += (pre[z] - pre[z - 1]) * z;
        ans += (y - pre[j]) * (j + 1);
        cout << ans;
    }
    return 0;
}
```

11 Đề Đắk Nông

11.1 Bài 1: Mã số

Code mẫu

```
#include <bits/stdc++.h>
using namespace std;
#define int long long
int n, k;

signed main() {
    ios_base::sync_with_stdio(0); cin.tie(0);
    cin >> n >> k;
    cout << n - k << ' ' << n << ' ' << n + k;
    return 0;
}
```

Độ phức tạp thuật toán: $O(1)$.

11.2 Bài 2: Số cặp

Hàm $abs(x)$ có công dụng tính giá trị tuyệt đối của x.

Code mẫu

```
#include <bits/stdc++.h>
using namespace std;
#define int long long
int a, b;

signed main() {
    ios_base::sync_with_stdio(0); cin.tie(0);
    cin >> a >> b;
    if (a == b) cout << a;
    else cout << abs(a - b);
    return 0;
}
```

Độ phức tạp thuật toán: $O(1)$.

11.3 Bài 3: Game

Yêu cầu của bài thực chất là đi tìm 2 chữ số tận cùng của tổng $s = 1^2 + 2^2 + ... + n^2$.

11.3.1 Subtask 1 : $n \leq 10^5$

Sử dụng vòng lặp for để tính tổng s. Với $n \leq 10^5$, tổng s sẽ không vượt quá giới hạn của kiểu dữ liệu *long long* (trong C++).

Lưu ý: Nếu chữ số hàng chục của s là 0. Việc xuất kết quả là $s\%100$ sẽ bị sai.

Độ phức tạp thuật toán: $O(N)$.

11.3.2 Subtask 2 : $n \leq 10^8$

Sử dụng công thức $s = \dfrac{n(n+1)(2n+1)}{6}$.

Với $n \leq 10^8$, tổng s đã vượt quá giới hạn của kiểu dữ liệu *long long* (trong C++). Vì vậy ta cần thực hiện phép toán % 100 ngay sau mỗi phép tính.

Một vấn đề ở đây là trong công thức tính s, ta cần phải thực hiện việc chia 6. Điều này rất khó thực hiện khi cần phải % 100. Vì vậy, ta cần tính toán việc chia cho 6 trước tiên.

Trong 3 biểu thức $a = n, b = n+1, c = 2n+1$, ta sẽ chọn một biểu thức chia hết cho 2 để chia đi cho 2 và chọn một biểu thức chia hết cho 3 để chia đi cho 3. Như vậy, ta đã thực hiện việc chia cho 6 cho biểu thức $n(n+1)(2n+1)$.

Tại sao không chọn ra một biểu thức chia hết cho 6 ? Bởi lẽ có thể cả 3 biểu thức trên đều không chia hết cho 6 !

Lưu ý: Cẩn thận trường hợp khi chữ số hàng chục của s là 0.

Code mẫu

```
#include <bits/stdc++.h>
using namespace std;
#define int long long
int n;

signed main() {
    ios_base::sync_with_stdio(0); cin.tie(0);
    cin >> n;
    int a = n, b = n + 1, c = 2 * n + 1;

    if (a % 2 == 0) a /= 2;
    else b /= 2;

    if (a % 3 == 0) a /= 3;
    else if (b % 3 == 0) b /= 3;
    else c /= 3;

    int ans = a;
    (ans *= b) %= 100;
    (ans *= c) %= 100;
    if (ans < 10) cout << 0 << ans;
    else cout << ans;
    return 0;
}
```

Độ phức tạp thuật toán: $O(1)$.

11.4 Bài 4: Truy vấn

11.4.1 Subtask 1 + 2

Đây là bài sử dụng *Mảng hiệu* cơ bản ! Với mỗi truy vấn, ta chỉ cần cập nhật $a[x]$ += Val và $a[y+1]$ -= Val.

Sau tất cả các truy vấn, thực hiện tính toán cộng dồn từ đầu đến cuối mảng, ta sẽ thu được mảng a sau cùng.

Code mẫu

```
#include <bits/stdc++.h>
using namespace std;
#define int long long
const int N = 1e6 + 10;
int n, m, val, a[N] = {0};

signed main() {
```

```
    ios_base::sync_with_stdio(0); cin.tie(0);
    cin >> n >> m >> val;
    for(int i = 1; i <= m; i++) {
        int x, y;
        cin >> x >> y;
        a[x] += val;
        a[y + 1] -= val;
    }
    for(int i = 1; i <= n; i++) a[i] += a[i - 1];
    sort(a + 1, a + n + 1, greater<int>());
    cout << a[1] + a[2] + a[3];
    return 0;
}
```

Độ phức tạp thuật toán: $O(M + NlogN)$.

11.5 Bài 5: Candy

11.5.1 Subtask 1 + 2

Nhận xét:

- Nếu với giá trị k mà Tài có thể ăn được ít nhất một nửa số kẹo thì với giá trị $k+1, k+2, ...$ thì anh ấy cũng có thể ăn được ít nhất một nửa số kẹo.
- Nếu với giá trị k mà Tài không thể ăn được ít nhất một nửa số kẹo thì với giá trị $1, 2, ..., k-1$ thì anh ấy cũng không thể ăn được ít nhất một nửa số kẹo.

Vì vậy, ta có thể sử dụng chặt nhị phân đáp án để tìm giá trị k nhỏ nhất mà Tài có thể ăn được ít nhất một nửa số kẹo.

Code mẫu

```
#include <bits/stdc++.h>
using namespace std;
#define int long long
long long n, half;

bool check(int u) {
    int T = 0, tn = n;
    while(tn) {
        T += min(u, tn);
        tn -= min(u, tn);
        tn -= tn / 10;
    }
    return T >= half;
}

signed main() {
```

```
    ios_base::sync_with_stdio(0); cin.tie(0);
    cin >> n;
    half = (n + 1) / 2;
    int l = 1, r = half, ans;
    while(l <= r) {
        int mid = (l + r) / 2;
        if (check(mid)) {
                r = mid - 1;
                ans = mid;
        }
        else l = mid + 1;
    }
    cout << ans;
    return 0;
}
```

12 Đề Đồng Tháp

12.1 Bài 1: Các hộp kẹo (2.5 điểm)

12.1.1 Subtask 1: $1 \le n \le 10^9$

- Ta có thể duyệt i từ 1 đến n và kiểm tra xem i có phải là số chính phương hay không. Nếu i là số chính phương, ta trừ n cho i, vòng lặp dừng khi $n < i$.
Độ phức tạp bé hơn $O(n)$ vì trong lúc duyệt ta đã trừ n đi cho i nên số lần duyệt giảm đi nhiều.

12.1.2 Subtask 2: $10^9 < n \le 10^{18}$

Cũng giống với subtask 1 nhưng thay vì kiểm tra i có là số chính phương hay không thì ta trừ n cho $i * i$.
Code mẫu:

```
#include <bits/stdc++.h>
using namespace std;
bool scp(int x){
    int root = sqrt(x);
    return (root * root == x);
}

int main(){
    long long n;
    int ans = 0;
    cin >> n;
    if (n <= (int)1e9){
        for (int i=1; i<=n; ++i)
            if (scp(i)) n -= i, ++ans;
```

```
        cout << ans;
    }
    else{
        for (int i=1; 1LL * i * i <= n; ++i)
            n -= 1LL*i*i, ++ans;
        cout << ans;
    }
    return 0;
}
```

Độ phức tạp bé hơn $O(\sqrt{n})$

12.2 Bài 2: Xếp hàng mua vé (2.5 điểm)

- Vì bạn Nam đứng thứ k trong hàng nên phải chờ người thứ $1,2,...,k$. Vậy kết quả là $sum(a_1,...,a_k)$.
Code mẫu:

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

int main(){
    cin.tie(0) -> sync_with_stdio(0);
    int n, k;
    cin >> n >> k;
    vector<int> a(n + 1);
    for(int i=1; i<=n; ++i) cin >> a[i];
    long long result = 0;
    for(int i=1; i<=k; ++i) result += a[i];
    cout << result;
    return 0;
}
```

12.3 Bài 3: Câu lạc bộ (2.5 điểm)

12.3.1 Subtask 1: $1 \le n \le 10^3$

- Ở câu a, trong lúc nhập a_i, ta kiểm tra xem a_i có $\le k$ hay không. Kết quả là số lượng $a_i \le k$.
- Ở câu b, với mỗi i mà $a_i \le k$, ta duyệt j kiểm tra xem có bao nhiêu chỉ số j mà $a_j = a_i$.
Độ phức tạp: $O(n^2)$

12.3.2 Subtask 2: $10^3 < n \le 10^5$

- Vì $a_i \le 10^5$ nên ta sử dụng kĩ thuật mảng đánh dấu. Ta gọi cnt_i là số bạn có thời gian giải đề là i.
- Ta làm tương tự câu a ở subtask 1, ở câu b kết quả sẽ là $max(cnt_{1..k})$.
Code mẫu:

```cpp
#include <bits/stdc++.h>
using namespace std;
const int N = 1e5 + 5;
int a[N], cnt[N];

int main(){
    cin.tie(0) -> sync_with_stdio(0);
    int n, k;
    cin >> n >> k;
    int count = 0, res = 0;
    for(int i=1; i<=n; ++i){
        cin >> a[i];
        if (a[i] > k) continue;
        count++;
        cnt[a[i]]++;
    }

    if (n <= (int)1e3){
        int dem;
        for(int i=1; i<=n; ++i){
            dem = 0;
            if (a[i] <= k)
                for(int j=1; j<=n; ++j) if (a[i] == a[j]) dem++;
            res = max(res, dem);
        }
        cout << count << '\n' << res;
    } else{
        for(int i=1; i<=k; ++i) res = max(res, cnt[i]);
        cout << count << '\n' << res;
    }
    return 0;
}
```

Độ phức tạp: $O(n)$

12.4 Bài 4: Chỉ số bạn bè (2.5 điểm)

12.4.1 Subtask 1: $1 \le n \le 10^3$

- Với mỗi i ($1 \le i \le n$), ta duyệt j từ 1 tới m và kiểm tra xem $a_i + b_j = S$ hay không.
Độ phức tạp: $O(n * m)$.

12.4.2 Subtask 2: $10^3 < n, m \le 10^5$

- Ta sắp xếp lại mảng b, với mỗi i, ta tìm kiếm nhị phân:
+ L là chỉ số j đầu tiên sao cho $b_j = S - a_i$.
+ R là chỉ số j đầu tiên sao cho $b_j > S - a_i$.

+ Kết quả cho vị trí i là $R-L$.

Code mẫu:

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

int main(){
    cin.tie(0) -> sync_with_stdio(0);
    int n, m, S;
    cin >> n >> m >> S;
    vector<int> a(n), b(m);
    for(int i=0; i<n; ++i) cin >> a[i];
    for(int i=0; i<m; ++i) cin >> b[i];

    if (n <= (int)1e3 && m <= (int)1e3){
        int ans = 0;
        for(int i=0; i<n; ++i)
            for(int j=0; j<m; ++j)
                if (a[i] + b[j] == S) ++ans;
        cout << ans;
    } else{
        sort(b.begin(), b.end());
        int ans = 0;
        for(int i=0; i<n; ++i){
            int L = lower_bound(b.begin(), b.end(), S - a[i]) - b.begin();
            int R = upper_bound(b.begin(), b.end(), S - a[i]) - b.begin();
            ans += R - L;
        }
        cout << ans;
    }
    return 0;
}
```

Độ phức tạp: $O(n * \log_2 m)$

13 Đề Hà Tỉnh

13.1 Bài 1: Tổng dãy số

13.1.1 Sub 1 : $n \leq 10^6$

Ta dễ dàng rút gọn được:

$T_k = (k+1)^2 - k^2$

$T_k = (k+1-k)(k+1+k)$

$T_k = 2k+1$

=> $S = (2.1+1)+(2.2+1)+...(2n+1)$

Với $n \leq 10^6$ ta dễ dàng duyệt for để tính được S trong độ phức tạp thời gian $O(n)$.

13.1.2 Sub 2 : $n \leq 10^9$

Với $n \leq 10^9$ ta lại tiếp tục rút gọn S:

$S = (2.1+1)+(2.2+1)+...(2n+1)$

$S = 2(1+2+..n)+n$

$S = n(n+1)+n$

13.1.3 Code mẫu tham khảo

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

int n;
long long Ans;

main(){
    cin >> n;
    cout << n * 1LL * (n - 1) + n;
}
```

13.2 Bài 2: Tổng dãy số

13.3 Sub 1, 2 : $n \leq 10^{18}$

Ta có thể tính được ở trưởng hợp xấu nhất thì tổng bình phương các chữ số chỉ là 1458 vậy nên ta chỉ cần check xem liệu tổng bình phương của các chữ số có phải là số nguyên tố hay không?

13.4 Code mẫu tham khảo

```
#include<iostream>
#include<cmath>
using namespace std;
bool check(int n) {
        if (n < 2) {
                return false;
        }
        else {
                for (int i = 2; i <= sqrt(n); i++) {
                        if (n % i == 0) {
                                return false;
                        }
                }
                return true;
        }
}
```

```
int main() {
        long long n;
        cin >> n;
        int sum = 0;
        while (n != 0) {
                sum = sum + ((n%10)*(n%10));
                n = n/10;
        }
        if (check(sum) == true) {
                cout << "1" << endl;
                cout << sum;
        }
        else {
                cout << "-1" << endl;
                cout << sum;
        }
}
```

13.5 Bài 3: Dãy đặc trưng

13.5.1 Subtask 1

- Ta duyệt tới lần lượt các vị trí từ 1 -> n, với mỗi vị trí, ta duyệt tìm phần tử j từ $i+1$ -> n. Với mỗi vị trí j, ta sẽ kiểm tra xem các phần tử từ $i->j$ có đồng dấu hay không, nếu có thì cập nhật độ dài đoạn con dài nhất thỏa mãn.

13.5.2 Subtask 2

- Sử dụng một biến d = 1, lưu độ dài đoạn dài nhất thỏa mãn kết thúc tại vị trí i và cùng dấu với a_i. Nếu a_i * a_{i-1} <= 0. Ta cập nhật độ dài và gán d = 1.

13.5.3 Code mẫu tham khảo

```
#include<bits/stdc++.h>
#define tag "SPEC"
#define ll long long
const long long N = 1e6+9;
using namespace std;
ll n, a[N];
void sub1(){
    ll gmax = 0;
    for(int i = 1; i <= n; i++){
        ll res = 0;
        if(a[i] != 0) res = 1;
        for(int j = i+1; j <= n+1; j++){
            if(a[j]*a[j-1] > 0)
                res++;
```

```
            else
                break;
        }
        gmax = max(gmax, res);
    }
    cout << gmax;
}
void sub2(){
    ll d = 0, gmax = 0;
    if(a[1]!=0) d = 1, gmax = 1;
    for(int i = 2; i <= n;i++){
        if((a[i]*a[i-1]) <= 0)
            d = 1;
        else
            d++;
            gmax = max(gmax, d);
        //cout << d << " " ;
    }
    cout << gmax;
}
int main()
{
    cin.tie(NULL); cout.tie(NULL);
    ios_base::sync_with_stdio(false);
    if(fopen(tag".inp","r")){
        freopen(tag".inp","r",stdin);
        freopen(tag".out","w",stdout);
    }
    cin >> n;
    for(int i = 1; i <= n; i++)
        cin >> a[i];
    if(n <= 1e3)
        sub1();
    else
        sub2();
}
```

13.6 Bài 4: Đoạn thẳng

13.6.1 Subtask 1 + 2

- Sắp xếp các tọa độ X theo thứ tự tăng đần, với mỗi tọa độ X, ta duyệt tìm dãy con ngắn nhất có chứa 3 màu.

13.6.2 Subtask 3

- Ta sử dụng kĩ thuật 2 con trỏ, đặt 2 biến i = 1 và j = 1. Ta chạy j -> n, nếu kiểm tra thấy tại một vị trí j bất kì có xuất hiện đủ 3 màu. Ta tăng i và cập nhật dãy con ngắn nhất có đủ cả 3 màu đến khi dãy i -> j không xuất hiện đủ 3 màu nữa thì tiếp tục tăng j.

13.6.3 Code mẫu tham khảo

```
#include<bits/stdc++.h>
#define int long long
using namespace std;
const int maxn = 1e6+5;
int n, mini = 1e18, d[maxn];
struct pt{
int x,m;
};
pt a[maxn];
bool ss(pt a, pt b) {
return a.x<b.x;
}
main() {
ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
cin >> n;
for (int i = 1; i <= n; i++) cin >> a[i].x >> a[i].m;
sort(a+1,a+n+1,ss);
int i = 1, j = 2, cnt = 1;
d[a[1].m]++;
while(j <= n) {
    if (!d[a[j].m]) {
        cnt++;
    }
    d[a[j].m]++;
    while(cnt == 3) {
        mini = min(mini, a[j].x-a[i].x);
        d[a[i].m]--;
        if (!d[a[i].m]) {
            cnt--;
        }
        i++;
    }
    j++;
}
if (mini == 1e18) cout << "-1";
else cout << mini;
}
```

14 Đề Khánh Hoà

14.1 Bài 1: Đồng hồ (2 điểm)

14.1.1 Hướng dẫn giải

- Phân tích đề: Đề bài đã nhắc đến yếu tố "đồng hồ được chia thành 60 vạch và được đánh số từ 0 đến 59". Chính vì thế, kết quả của bài chỉ giới hạn từ 0 đến 59.
- Phương pháp giải: Ở bài này, ta sẽ sử dụng phép chia dư cho 60 nhằm đưa ra kết quả đúng với giới hạn mà ta nhận thấy được. Công thức: (m + n) mod 60.

14.1.2 Code mẫu

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

signed main()
{
    ios_base::sync_with_stdio(false); cin.tie(NULL). cout.tie(NULL);
    long long n,m;
    cin >> n >> m;
    cout << (n+m)%60;
    return 0;
}
```

Độ phức tạp $O(1)$

14.2 Bài 2: Mua bi (3 điểm)

14.2.1 Hướng dẫn giải

- Phân tích đề:
+ Đề đã nhắc đến hai yếu tố chính: Các hộp chứa 2 viên bi và các hộp chứa 1 viên bi.
+ Đối với hộp có 2 viên bi, tổng số tiền mà ta thu được ở các hộp đó đều là một giá trị không đổi (tổng giá trị của viên bi một và viên bi hai).
+ Riêng với hộp có 1 viên bi, ta nhận thấy rằng việc lựa chọn ra viên bi có giá trị nhỏ hơn trong số hai viên bi và bỏ chúng vào tất cả các hộp chứa 1 viên bi sẽ giúp giá trị thu được là nhỏ nhất.
- Phương pháp giải:
+ Duyệt qua tất cả các hộp từ hộp 1 đến hộp thứ n. Khi xét qua từng hộp, ta có được hai trường hợp riêng biệt.
+ Trường hợp đầu tiên, như đã phân tích bên trên, với hộp đang xét chứa 2 viên bi, số tiền cần bỏ ra sẽ là a + b.
+ Trường hợp còn lại, với hộp đang xét chứa 1 viên bi, số tiền cần bỏ ra sẽ là giá trị nhỏ hơn giữa a và b - min(a,b).

+ Sau cùng, lấy kết quả là tổng số tiền sau khi đã tính từ hộp 1 đến hộp thứ n.

14.2.2 Code mẫu

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

signed main()
{
    ios_base::sync_with_stdio(false); cin.tie(NULL). cout.tie(NULL);
    int n,a,b;
    cin >> n >> a >> b;
    int s = a+b, m = min(a,b), ans = 0;
    for (int i = 1; i <= n; i++)
    {
        int x;
        cin >> x;
        if (x == 1) ans += m;
        else ans += s;
    }
    cout << ans;
    return 0;
}
```

Độ phức tạp $O(n)$

14.3 Bài 3: Phần thưởng (2,5 điểm)

14.3.1 Subtask 1: $n, m \leq 10^3$

- Duyệt từ điểm tích lũy của người 1 đến người thứ m, với điểm tích lũy của người thứ i (p_i), ta duyệt lại toàn bộ từ điểm tích lũy của người 1 đến người thứ m và đếm xem có tất cả bao nhiêu người có điểm tích lũy ≥ với người thứ i. Dĩ nhiên, vì số lượng phần thưởng tối đa là n nên khi đã đếm xong số lượng người, chỉ lấy tối đa số lượng người là n.
- Sau đó, chúng ta sử dụng phép tính (p_i*số lượng người) và lấy giá trị lớn nhất của tất cả các phép tính trên với $1 \leq i \leq m$.

Độ phức tạp thuật toán: $O(m * n)$

14.3.2 Subtask 2: $n, m \leq 10^5$

- Từ cách giải của subtask 1, ta có thể nâng cấp thuật toán bằng cách sắp xếp điểm tích lũy của n người theo thứ tự giảm dần.
- Tiếp theo đó, khi xét đến người thứ i, ta thấy rằng người thứ i chính là người có điểm tích lũy thấp nhất trong số i người đầu tiên. Vậy nên, ta chỉ cần tìm giá trị lớn nhất của tất cả

các phép tính $p_i * i$ với $1 \le i \le min(m,n)$ - chỉ có thể phát thưởng cho tối đa n người nên chỉ duyệt i đến $min(n,m)$.

Độ phức tạp thuật toán: $O(m * log(m))$

14.3.3 Code mẫu

```
#include <bits/stdc++.h>
using namespace std;
#define int long long
const int N = (int)1e6+10;
int p[N];

signed main()
{
    ios_base::sync_with_stdio(false); cin.tie(NULL); cout.tie(NULL);
    int n,m;
    cin >> n >> m;
    for (int i = 1; i <= n; i++) cin >> p[i];
    sort(p+1,p+n+1,greater<int>());
    int ans = -(int)1e9;
    for (int i = 1; i <= min(n,m); i++) ans = max(ans,p[i]*i);
    cout << ans;
    return 0;
}
```

Độ phức tạp $O(m * log(m))$

14.4 Bài 4: Thừa số nguyên tố nhỏ nhất (2,5 điểm)

14.4.1 Subtask 1: $n,k \le 10^3$

- Để giảm bớt các phép tính cần thực hiện, ta có thể duyệt các số trong đoạn $[2;k]$ nhưng thực hiện một chút cải tiến.

- Ở đây, với mỗi số x trong đoạn $[2;k]$, thay vì phân tích x ra thành thừa số nguyên tố tìm số nguyên tố nhỏ nhất, ta chỉ cần dùng kỹ thuật kiểm tra tính nguyên tố thông thường bằng cách duyệt trong đoạn $[2;sqrt(x)]$ và đi tìm số đầu tiên mà x chia hết. Số ấy chính là thừa số nguyên tố nhỏ nhất của x, nếu trong đoạn $[2;sqrt(x)]$ không có số thỏa yêu cầu thì thừa số nguyên tố nhỏ nhất của x là x (khi đó x là số nguyên tố).

- Việc tiếp theo cần làm chỉ đơn giản là duyệt từ 2 đến k và sau đó đếm thừa số nguyên tố nhỏ nhất đã được tìm trước đó bằng phương pháp đếm phân phối.

- Cuối cùng, với mỗi số nguyên tố a_i, ta chỉ cần xuất ra kết quả từ mảng đếm phân phối đã tính.

Độ phức tạp thuật toán: $O(k * sqrt(k) + n)$

14.4.2 Subtask 2: $n,k \le 10^6$

Với việc sử dụng sàng Eratosthenes để tính trước các số nguyên tố, chúng ta có thể nâng cấp thuật toán từ subtask 1 đã nêu với việc chỉ cần kiểm tra các số nguyên tố mà ta đã tính sẵn trước đó trong đoạn $[2;sqrt(x)]$ với x là một số cần xét trong đoạn $[2;k]$.
Độ phức tạp thuật toán: $O(k*log(log(k))+n)$

14.4.3 Code mẫu

```
#include <bits/stdc++.h>
using namespace std;
#define int long long
const int N = (int)1e6+10;
bool b[N];
int d[N],t[N];

void sang()
{
    memset(b,true,sizeof(b));
    b[0] = b[1] = false;
    for (int i = 2; i <= sqrt(N-10); i++)
        if (b[i])
            for (int j = i*i; j <= N-10; j += i)
            {
                b[j] = false;
                if (!d[j]) d[j] = i;
            }
    for (int i = 2; i <= N-10; i++)
        if (b[i]) d[i] = i;
}

signed main()
{
    ios_base::sync_with_stdio(false); cin.tie(NULL); cout.tie(NULL);
    int n,k,x;
    cin >> n >> k;
    sang();
    for (int i = 2; i <= k; i++) t[d[i]]++;
    for (int i = 1; i <= n; i++)
    {
        cin >> x;
        cout << t[x] << '\n';
    }
    return 0;
}
```

Độ phức tạp $O(k*log(log(k))+n)$

15 Đề Kiên Giang

15.1 Câu 1

15.1.1 Hướng dẫn giải:

Ta gán hai số nhập vào là hai xâu, sau đó thực hiện đảo ngược hai xâu và chuyển về thành kiểu số để tìm số có giá trị lớn nhất.

15.1.2 Code mẫu:

```
#include<bits/stdc++.h>
using namespace std;
string s,t;
int main ()
{
  cin>>s>>t;
  reverse(s.begin(),s.end());
   reverse(t.begin(),t.end());
   cout<<max(stoi(s),stoi(t));
}
```

15.2 Câu 2

15.2.1 Subtask 1: $1 \le n \le 10^3$

Ta duyệt qua mọi cặp (a_i, a_j) có thể, sau đó kiểm tra tính "Cặp đôi may mắn" của cặp số đó.

15.2.2 Subtask 2: $10^3 < n \le 2 * 10^5$

Từ biểu thức $a_i + a_j = k$, ta biến đổi tương đương $a_i = k - a_j$. Do đó, ta có thể đếm tần số xuất hiện của a_i là có thể đếm được số "Cặp đôi may mắn" mà không cần thử mọi cặp số nguyên dương.

15.2.3 Code mẫu:

```
#include<bits/stdc++.h>
using namespace std;
int  n , k , a[1000004],ans;
map<int,int> d;
void sub1()
{
  for(int i =1 ;i<n;i++)
    for(int j = i+1;j<=n;j++)
     if(a[i]+a[j]==k) ans++;
```

```
}
void sub2()
{
  for(int i =1 ;i<=n;i++)
  {
    d[a[i]]--;
    ans+=d[k-a[i]];
  }
}
int main ()
{
 cin>>n>>k;
 for(int i =1 ;i<=n;i++)
 {
   cin>>a[i];
   d[a[i]]++;
 }
 if(n<=1000) sub1();
 else  sub2();
 cout<<ans;
}
```

15.3 Câu 3:

15.3.1 Subtask 1: $1 \le k \le 10^3$

Ta duyệt qua mọi số thuộc đoạn $[1,n]$, sau đó đếm số ước của số đó. Nếu số đó có ba ước, đó là một số đặc biệt.

15.3.2 Subtask 2: $10^4 < k \le 10^9$

Ta có nhận xét rằng, số có đúng 3 ước nguyên dương chỉ có thể là số chính phương mà căn bậc hai số học của nó là một số nguyên tố. Do đó, ta có thể sàng nguyên tố trên đoạn $[1,\sqrt{k}]$, sau đó đếm số số là số nguyên tố trên đoạn đó là có kết quả thoả mãn yêu cầu đề bài.

15.3.3 Code mẫu:

```
#include<bits/stdc++.h>
using namespace std;
long long  n,res;
long long demuoc (long long n )
{
        long long dem  = 0 ;
        for (int i = 1 ;i<=sqrt(n);i++)
                if (n%i==0) dem=dem+2;
        long long k =sqrt (n);
        if (k*k==n)dem--;
```

```
        return dem ;
}
 bool NT[1000005];
void SangNt() {
        memset(NT,true,sizeof (NT)) ;
        NT[1]=NT[0]=false ;
        for (int i = 2 ;i*i<=1000000;i++) {
                if (NT[i])
                for (int j=i*i;j<=1000000;j+=i) {
                        NT[j]=false;
                }
        }

}
void sub1()
{
  for(int i =1 ;i<=n;i++)
    if(demuoc(i)==3) res++;

  cout<<res;
}
void sub2()
{
  for(int i =1 ;i<=sqrt(n);i++ )
  if(NT[i]) res++;
  cout<<res;
}
int main ()
{
  SangNt();
 cin>>n;
 if(n<=10000) sub1 ();
 else sub2();
}
```

15.4 Câu 4:

15.4.1 Hướng dẫn giải:

Ta duyệt qua mọi kí tự của chuỗi S, sau đó kiểm tra xem liệu chuỗi đó có phải là ước của S không. Điều này có thể được thực hiện bằng cách lặp lại câu X liên tục cho đến khi độ dài của X lớn hơn hoặc bằng S và kiểm tra sự bằng nhau của chúng.

15.4.2 Code mẫu:

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

```
string s;
int main ()
{
  ios_base::sync_with_stdio(0);
  cout.tie(0);cin.tie(0);
  cin>>s;
  int n =s.size();
  string tmp ="";
  for(int i =0 ;i<n;i++)
  {
   tmp=tmp+s[i];
   string tmp2=tmp;
   while(tmp2.size()<n)
       tmp2=tmp2+tmp;
       if(tmp2==s)
       {
         cout<<tmp;
         return 0;
       }

  }
}
```

16 Đề Kon Tum

16.1 Câu 1: SỐ CHÍNH PHƯƠNG NHỎ NHẤT

16.1.1 Subtask 1 + 2: $N \leq 10^3$

Ta sẽ lần lượt kiểm tra từng số chính phương từ số chính phương nhỏ nhất (0, 1, 4, 9, ...) liệu có xuất hiện trong dãy số đã cho hay không bằng cách dùng vòng lặp for để duyệt qua toàn bộ dãy số.

Số chính phương đầu tiên không xuất hiện trong dãy số chính là đáp án cần tìm.

Code mẫu

```
#include <bits/stdc++.h>
using namespace std;
#define int long long
const int N = 1e4 + 10;
int n, a[N];

bool isInArr(int k) {
    for(int i = 1; i <= n; i++) if (a[i] == k) return true;
    return false;
}

signed main() {
    ios_base::sync_with_stdio(0); cin.tie(0);
    cin >> n;
    for(int i = 1; i <= n; i++) cin >> a[i];
    int res = 0;
    while(isInArr(res * res)) res++;
    cout << res * res;
    return 0;
}
```

Độ phức tạp thuật toán: $O(N^2)$.

16.1.2 Subtask 3: $N \leq 10^4$

Sử dụng cấu trúc dữ liệu *map* để đánh dấu sự xuất hiện của các giá trị trong dãy số.

Ta sẽ lần lượt kiểm tra từng số chính phương từ số chính phương nhỏ nhất (0, 1, 4, 9, ...) liệu có xuất hiện trong dãy số đã cho hay không bằng cách dùng *map* ở trên.

Số chính phương đầu tiên không xuất hiện trong dãy số chính là đáp án cần tìm.

Code mẫu

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

```
const int N = 1e4 + 10;
int n, a[N];
map<int, int> mp;

signed main () {
        ios_base::sync_with_stdio(0); cin.tie(0);
        cin >> n;
        for(int i = 1; i <= n; i++) cin >> a[i], mp[a[i]] = 1;
        int res = 0;
        while(mp[res * res]) res++;
        cout << res * res;
        return 0;
}
```

Độ phức tạp thuật toán: $O(NlogN)$.

16.2 Câu 2: XÓA SỐ

16.2.1 Subtask 1 + 2 + 3

Để ý rằng chênh lệch lớn nhất max giữa hai số bất kỳ trong dãy số là chênh lệch giữa số lớn nhất và số nhỏ nhất trong dãy.

Để giá trị max nhỏ nhất thì ta cần phải thay đổi giá trị nhỏ nhất và giá trị lớn nhất trong dãy số đã cho.

Để thay đổi giá trị nhỏ nhất, ta có thể lần lượt xóa đi những giá trị nhỏ bắt đầu từ số nhỏ nhất trong dãy số.

Tương tự, để thay đổi giá trị lớn nhất, ta có thể lần lượt xóa đi những giá trị lớn bắt đầu từ số lớn nhất trong dãy số.

Như vậy, ta cần phải sắp xếp lại dãy số đã cho.
Xem xét K khả năng xóa số như sau:

- Xóa 0 số đầu tiên và K số cuối cùng trong dãy số sau khi đã sắp xếp.
- Xóa 1 số đầu tiên và $K-1$ số cuối cùng trong dãy số sau khi đã sắp xếp.
- ...
- Xóa $K-1$ số đầu tiên và 1 số cuối cùng trong dãy số sau khi đã sắp xếp.
- Xóa K số đầu tiên và 0 số cuối cùng trong dãy số sau khi đã sắp xếp.

Duyệt qua các khả năng trên, ta sẽ lấy được giá trị max nhỏ nhất có thể khi xóa K số từ dãy số đã cho.

Code mẫu

```cpp
#include <bits/stdc++.h>
using namespace std;
#define int long long
const int N = 1e6 + 10;
int n, k, a[N];

signed main() {
    ios_base::sync_with_stdio(0); cin.tie(0);
    cin >> n >> k;
    for(int i = 1; i <= n; i++) cin >> a[i];
    sort(a + 1, a + n + 1);
    int ans = INT_MAX;
    for(int i = 0; i <= k; i++) {
        int j = k - i;
        ans = min(ans, a[n - j] - a[i + 1]);
    }
    cout << ans;
    return 0;
}
```

Độ phức tạp thuật toán: $O(NlogN)$.

16.3 Câu 3: ĐẾM SỐ LƯỢNG DÃY CON LIÊN TIẾP CÙNG TỔNG

16.3.1 Subtask 1 + 2: $N \leq 10^3$

Sử dụng 2 vòng lặp for lồng nhau để duyệt qua mọi dãy con liên tiếp để tính tổng và dùng cấu trúc dữ liệu *map* để lưu lại số lần xuất hiện của tổng đó.

Code mẫu

```cpp
#include <bits/stdc++.h>
using namespace std;
#define int long long
const int N = 1e6 + 10;
int n, a[N];
map<int, int> mp;

signed main() {
    ios_base::sync_with_stdio(0); cin.tie(0);
    cin >> n;
    for(int i = 1; i <= n; i++) cin >> a[i];
    int ans = 0;
    for(int i = 1; i <= n; i++) {
        int s = 0;
        for(int j = i; j <= n; j++) {
                s += a[j];
                mp[s]++;
                ans = max(ans, mp[s]);
```

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

Độ phức tạp thuật toán: $O(N^2 logN^2)$.

16.3.2 Subtask 3: $N \leq 10^6$

Tương truyền rằng người nào giải được câu này sẽ trở thành VUA CODE TẶC.

16.4 Câu 4: CHỌN SỐ

16.4.1 Subtask 1 + 2 + 3

Gọi dp[i][j] là tổng các phần tử đã chọn lớn nhất cho đến phần tử thứ i và số phần tử liên tiếp đang chọn là j. Cụ thể hơn về chiều j:

- dp[i][0]: Không chọn phần tử thứ i (số phần tử liên tiếp đang chọn là 0).
- dp[i][1]: Chọn phần tử thứ i, trước đó không chọn phần tử thứ $i-1$ (số phần tử liên tiếp đang chọn là 1).
- dp[i][2]: Chọn phần tử thứ i và $i-1$, trước đó không chọn phần tử thứ $i-2$ (số phần tử liên tiếp đang chọn là 2).
- ...

Vì không được chọn quá 3 phần tử liên tiếp nên giới hạn của chiều j là 3.

Chuyển trạng thái quy hoạch động:

- Nếu chọn phần tử thứ i thì dp[i][j] = dp[$i-1$][$j-1$]+ a[i] (với $1 \leq j \leq 3$) (số phần tử liên tiếp đang chọn sẽ tăng thêm 1).
- Nếu không chọn phần tử thứ i thì dp[i][0] = max(dp[$i-1$][0], dp[$i-1$][1], dp[$i-1$][2], dp[$i-1$][3]).

Trường hợp cơ sở: dp[1][1] = a[1] và dp[1][0] = dp[1][2] = dp[1][3] = 0.

Như vậy, đáp án cần tìm là max(dp[n][0], dp[n][1], dp[n][2], dp[n][3]).

Code mẫu

```
#include <bits/stdc++.h>
using namespace std;
#define int long long
const int N = 1e6 + 10;
int n, a[N], dp[N][4];
```

```
signed main() {
    ios_base::sync_with_stdio(0); cin.tie(0);
    cin >> n;
    for(int i = 1; i <= n; i++) cin >> a[i];
    dp[1][1] = a[1];
    dp[1][0] = dp[1][2] = dp[1][3] = 0;
    for(int i = 2; i <= n; i++) {
        dp[i][0] = max({dp[i - 1][0], dp[i - 1][1], dp[i - 1][2], dp[i - 1][3]});
        dp[i][1] = dp[i - 1][0] + a[i];
        dp[i][2] = dp[i - 1][1] + a[i];
        if (i >= 3) dp[i][3] = dp[i - 1][2] + a[i];
    }
    cout << max({dp[n][0], dp[n][1], dp[n][2], dp[n][3]});
    return 0;
}
```

Độ phức tạp thuật toán: $O(N)$.

17 Đề Lâm Đồng

17.1 Câu 1: Tính tổng các ước

17.1.1 Hướng dẫn chấm:

Để ý $n \leq 10^{10}$, ta không thể duyệt tuần tự từ 1 đến n để tính tổng các ước của n. Ta có nhận xét: Với mỗi i là ước của n, ta luôn có n/i cũng là ước của n. Do đó, ta chỉ cần kiểm tra liệu i có bằng n/i không, đồng nghĩa n có là số chính phương hay không, khi đó ta sẽ có thể tính tổng các ước của n mà không cần duyệt đến n.

17.1.2 Code mẫu:

```
#include<bits/stdc++.h>
using namespace std;
long long n;
long long tonguoc (long long n )
{
        long long s = 0 ;
        for (int i = 1 ;i<=sqrt(n);i++)
        {
                if (n%i==0) s =s+i +n/i;
        }
        long long k =sqrt (n);
        if (k*k==n) s= s -k;
        return s-n ;
}
int main ()
```

```
{
  cin>>n;
  cout<<tonguoc(n);
}
```

17.2 Câu 2: Xếp hình chữ nhật

17.2.1 Hướng dẫn giải:

- Để tạo n hình chữ nhật có tổng diện tích lớn nhất, ghép bốn cạnh có độ dài bằng nhau ở mỗi loại cạnh thứ i. Điều này là vì theo bất đẳng thức Cauchy, với hai loại cạnh a và b, ta luôn có: $a^2 + b^2 > 2ab$, với $a \neq b$.
- Để tạo n hình chữ nhật có tổng diện tích bé nhất, ta sẽ ghép giữa cạnh 1 với cạnh n, cạnh 2 với cạnh $n-1$,... rồi cộng tổng diện tích của các hình chữ nhật lại. Điều này là vì với mỗi cạnh i cố định, ta có diện tích của hình chữ nhật chứa cạnh i tăng đơn điệu theo độ dài của cạnh còn lại. Do đó, để có diện tích nhỏ nhất của hình chữ nhật, ta gán độ dài cạnh còn lại là cạnh nhỏ nhất có thể tính được.

17.2.2 Code mẫu:

```
#include<bits/stdc++.h>
using namespace std;
long long n,smax,smin;
int main ()
{
  cin>>n;
  for(int i =1 ;i<=n;i++) smax+=i*i;
  for(int i = 1 ;i<=n;i++)
    smin+=(i*(n-i+1));
  cout<<smax<<" "<<smin;
}
```

17.3 Câu 3: Dãy số

17.3.1 Hướng dẫn giải:

Với mỗi phần tử a_i của dãy a, ta kiểm tra nếu phần tử đó có dạng $3k+5$, đồng nghĩa $a_i - 5$ chia hết cho 3, thì ta đưa phần tử ấy vào danh sách đáp án. Sau đó sắp xếp lại danh sách đáp án theo thứ tự tăng dần và đưa ra lần lượt các phần tử thoả mãn yêu cầu đề bài.

17.3.2 Code mẫu:

```
#include<bits/stdc++.h>
using namespace std;
long long n,x , d=0 , a[1000004];
```

```
int main ()
{
  cin>>n;
  for(int i =1 ;i<=n;i++)
  {
    cin>>x;
    if((x-5)%3==0)
    {
      d++;
      a[d]=x;
    }
  }
  if (d==0)
  {
  cout<<0;
  return 0;
  }
  sort(a+1,a+1+d);
  for(int i =1 ;i<=d;i++) cout<<a[i]<<'\n';
}
```

17.4 Câu 4: Dây chuyền sản xuất

17.4.1 Hướng dẫn giải:

Lý thuyết Quy hoạch động: Quy hoạch động
Gọi dp_i là thời gian hoàn thành công việc ít nhất tính đến cánh tay thứ i.
Từ đề bài, với cách nhìn ngược lại, ta có các cánh chuyển trạng thái như sau:
- Cánh tay thứ i làm việc độc lập: $dp_i = dp_{i-1} + t_i$.
- Cánh tay thứ i phối hợp với cánh tay thứ $i-1$: $dp_i = dp_{i-2} + p_{i-1}$.
Từ đó ta có **Công thức quy hoạch động:** $dp_i = min(dp_{i-1} + t_i, dp_{i-2} + p_{i-1}$.
Với việc khởi gán $dp_1 = t_1$, $dp_2 = min(t_1 + t_2, p_1$, đáp án của bài toán là dp_n.

17.4.2 Code mẫu:

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

const int N = 1e6 + 5;
int t[N], p[N], dp[N];
int n;

signed main() {
    ios::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);
```

```
    cin >> n;
    for (int i = 1; i <= n; i++) cin >> t[i];
    for (int i = 1; i < n; i++) cin >> p[i];
    dp[1] = t[1];
    for (int i = 2; i <= n; i++) dp[i] = min(dp[i - 1] + t[i], dp[i - 2] + p[i - 1]);
    cout << dp[n];
    return 0;
}
```

18 Đề Lào Cai

18.1 Câu 1a

Thuật toán: Xử lý số học / xâu

18.1.1 Subtask 1: $1 \le a, b \le 10^9$

Với $a, b \le 10^9$, ta sử dụng kiểu dữ liệu int để đảo ngược hai số: Lấy kết quả đang xử lý nhân 10 với số đứng trước và cộng với chữ số hàng đơn vị của số đang xét, sau đó chia số đang xét cho 10; lặp lại cho đến khi số đang xử lý bằng 0. Sau đó ta lấy giá trị lớn nhất của hai số hậu xử lý.
Với việc chia cho 10 liên tục, ta có độ phức tạp cho thuật toán này là $O(log(max(a, b)))$.

18.1.2 Subtask 2: $10^{10} \le a, b \le 10^{18}$

Với $a, b \le 10^{18}$, ta có thể sử dụng kiểu dữ liệu $string$ từ thư viện $< string >$ hoặc kiểu dữ liệu $longlong$ để xử lý đều được, với cách làm giống như subtask 1.
Với việc chia cho 10 liên tục, ta có độ phức tạp cho thuật toán này là $O(log(max(a, b)))$.

18.1.3 Code mẫu:

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

long long a, b;

void subtask1(const long long &a, const long long &b)
{
    int n = a, m = b;
    int ans1 = 0, ans2 = 0;
    while (n > 0)
    {
        ans1 = ans1 * 10 + n % 10;
        n /= 10;
    }
```

```
    while (m > 0)
    {
        ans2 = ans2 * 10 + m % 10;
        m /= 10;
    }
    cout << max(ans1, ans2);
}

void subtask2(const long long &a, const long long &b)
{
    long long n = a, m = b;
    long long ans1 = 0, ans2 = 0;
    while (n > 0)
    {
        ans1 = ans1 * 10 + n % 10;
        n /= 10;
    }
    while (m > 0)
    {
        ans2 = ans2 * 10 + m % 10;
        m /= 10;
    }
    cout << max(ans1, ans2);
}

int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);
    cin >> a >> b;
    if (a <= 1e9 && b <= 1e9) subtask1(a, b);
    else subtask2(a, b);
    return 0;
}
```

18.2 Câu 1b

18.2.1 Subtask 1: $Q \le 100$, $3 \le N \le 10^6$

Thuật toán: Duyệt cơ bản + Số học

Với $N \le 10^6$, ta có thể tính trước mọi số số là bội của 3 hoặc 7 trong đoạn $[1, max(N)]$, sau đó đưa ra kết quả với mỗi truy vấn.

Với việc tính trước kết quả trong $O(max(N))$, và trả lời q truy vấn trong $O(Q)$, ta có độ phức tạp tổng quát cho thuật toán này là $O(max(N))$.

18.2.2 Subtask 2: $Q \leq 100$, $10^7 < N \leq 10^{12}$

Thuật toán: Bao hàm - Loại trừ

Lý thuyết Bao hàm - Loại trừ: Bao hàm - Loại trừ

Gọi A là tập hợp các số chia hết cho 3 trên đoạn $[1,N]$, B là tập hợp các số chia hết cho 7 trên đoạn $[1,N]$. Khi đó, theo lý thuyết Bao hàm - Loại trừ, ta có công thức:

$|A \cup B| = |A| + |B| - |A \cap B|$

Với tập hợp A và B, ta có thể dễ dàng tính được $|A|$ và $|B|$ bằng cách lấy N chia lần lượt cho 3 và 7. Với tập hợp $A \cap B$, ta có thể tính được $|A \cap B|$ bằng cách lấy N chia cho bội chung nhỏ nhất của 3 và 7, tức 21.

Khi đó, ta có thể trả lời mỗi truy vấn trong $O(1)$. Vì có Q truy vấn, độ phức tạp của thuật toán này là $O(Q)$.

18.2.3 Code mẫu:

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

const int N = 1e2 + 5;
int arr[N];
int q;
bool subtask = 0;

void subtask1()
{
    int cnt = 0;
    for (int i = 1; i <= q; ++i)
    {
        cnt = 0;
        for (int j = 1; j <= arr[i]; ++j)
        {
            if (j % 3 == 0 || j % 7 == 0) cnt++;
        }
        cout << cnt << '\n';
    }
}

void subtask2()
{
    for (int i = 1; i <= q; ++i) cout << arr[i] / 3 + arr[i] / 7 - arr[i] / 21 << '\n';
}

int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);
    cin >> q;
```

```
    for (int i = 1; i <= q; ++i)
    {
        cin >> arr[i];
        if (!subtask)
        {
            if (arr[i] > 1e7) subtask = 1;
        }
    }
    if (!subtask) subtask1();
    else subtask2();
    return 0;
}
```

18.3 Câu 2a:

18.3.1 Hướng dẫn giải:

Thuật toán: Duyệt cơ bản

Với $|S| \leq 100$, ta có thể duyệt qua mọi kí tự của chuỗi S, sau đó kiểm tra xem liệu chuỗi đó có phải là ước của S không.

Với việc duyệt qua mọi phần tử của chuỗi S, mỗi lần duyệt sẽ kiểm tra tính ước, ta có độ phức tạp trong trường hợp xấu nhất của thuật toán này là $O(|S|^2)$.

18.3.2 Code mẫu:

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

string s, ans;

bool check(const string &s, const string &t)
{
    string res;
    while (res.size() < t.size())
    {
        for (char i: s) res.push_back(i);
    }
    return res == t;
}

int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);
    cin >> s;
    for (char i: s)
    {
```

```
        ans.push_back(i);
        if (check(ans, s)) break;
    }
    cout << ans;
    return 0;
}
```

18.4 Câu 2b:

Thuật toán: Sắp xếp
Lý thuyết Sắp xếp: Thuật toán sắp xếp

18.4.1 Subtask 1: $|S| \leq 1000$

Với mỗi kí tự của chuỗi S, ta sẽ kiểm tra liệu kí tự đó in hoa hay in thường, sau đó tách các kí tự đó thành hai chuỗi con in hoa và in thường. Sau đó, với mỗi chuỗi, ta sử dụng thuật toán sắp xếp với độ phức tạp tổng quát $O(n^2)$ để sắp xếp lại các chuỗi theo thứ tự giảm dần, và in ra theo yêu cầu của đề bài. Với việc duyệt qua các phần tử của S, sau đó thực hiện sắp xếp các chuỗi con, ta có độ phức tạp trong trường hợp xấu nhất của thuật toán này là $O(|S|^2)$.

18.4.2 Subtask 2: $|S| \leq 10^5$

Cũng áp dụng tư tưởng như subtask 1, nhưng ta sẽ sử dụng những thuật toán sắp xếp khác nhanh hơn để tối ưu. Ở đoạn code mẫu, người viết sử dụng thuật toán QuickSort có sẵn của C++ để sắp xếp. Do đó độ phức tạp của thuật toán này là $O(|S|log_2|S|)$.

18.4.3 Code mẫu:

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

string s, t;
char x;
int n;

void subtask1()
{
    for (int i = 0; i < s.size(); ++i)
    {
        for (int j = 0; j < i; ++j)
        {
            if (s[i] > s[j])
            {
                char tmp = s[i];
```

```
                s[i] = s[j];
                s[j] = tmp;
            }
        }
    }
    for (int i = 0; i < t.size(); ++i)
    {
        for (int j = 0; j < i; ++j)
        {
            if (t[i] > t[j])
            {
                char tmp = t[i];
                t[i] = t[j];
                t[j] = tmp;
            }
        }
    }
    cout << t << s;
}

void subtask2()
{
    sort(s.begin(), s.end(), greater<char>());
    sort(t.begin(), t.end(), greater<char>());
    cout << t << s;
}

int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);
    while (cin >> x)
    {
        n++;
        if (x >= 'a' && x <= 'z') s.push_back(x);
        else t.push_back(x);
    }
    if (n <= 1000) subtask2();
    else subtask2();
    return 0;
}
```

18.5 Câu 3:

18.5.1 Subtask 1: $N \le 10^3$; $a_i \le 10^3$

Thuật toán: Kiểm tra số nguyên tố
Lý thuyết số nguyên tố: Kiểm tra số nguyên tố

Với mỗi phần tử a_i, ta sẽ kiểm tra tính nguyên tố của a_i và tổng các chữ số của a_i. Với việc sử dụng thuật toán kiểm tra số nguyên tố cơ bản trong $O(a_i)$, và duyệt qua từng phần tử trong mảng trong $O(n)$, ta có độ phức tạp của thuật toán này là $O(n * max(a_i)$.

18.5.2 Subtask 2: $N \leq 10^4$; $a_i \leq 10^6$

Thuật toán: Kiểm tra số nguyên tố
Lý thuyết về số nguyên tố: Kiểm tra số nguyên tố
Với tư tưởng của subtask 1, ta cải tiến thuật toán kiểm tra tính nguyên tố trong $O(\sqrt{n}$, từ đó cải tiến độ phức tạp của thuật toán là $O(n * \sqrt{max(a_i)}$.

18.5.3 Subtask 3: $N \leq 10^6$; $a_i \leq 10^6$

Thuật toán: Sàng nguyên tố Eratosthenes
Lý thuyết sàng nguyên tố Eratosthenes: Sàng nguyên tố
Ở subtask này, ta sẽ thực hiện sàng nguyên tố để kiểm tra tính nguyên tố của mọi số từ 1 đến 10^6, sau đó với mỗi phần tử của dãy a, ta sẽ kiểm tra tính nguyên tố của số đó và tổng các chữ số trong $O(1)$.
Với việc thực hiện sàng nguyên tố trong $O(nlog_2 n)$, duyệt các phần tử trong $O(n)$, ta có độ phức tạp tổng quát cho thuật toán này là $O(nlog_2 n)$.

18.5.4 Code mẫu:

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

const int N = 1e6 + 5;
int arr[N];
int n, maxi, ans;

int sum(int n)
{
    int res = 0;
    while (n > 0)
    {
        res += n % 10;
        n /= 10;
    }
    return res;
}

void subtask1()
{
    for (int i = 1; i <= n; ++i)
    {
        bool prime = true;
```

```
        for (int j = 2; j < arr[i]; ++j)
        {
            if (arr[i] % j == 0)
            {
                prime = false;
                break;
            }
        }
        if (prime == true)
        {
            int pos = sum(arr[i]);
            for (int j = 2; j < pos; ++j)
            {
                if (pos % j == 0)
                {
                    prime = false;
                    break;
                }
            }
        }
        if (prime == true) ans++;
    }
}

void subtask2()
{
    for (int i = 1; i <= n; ++i)
    {
        bool prime = true;
        for (int j = 2; j * j <= arr[i]; ++j)
        {
            if (arr[i] % j == 0)
            {
                prime = false;
                break;
            }
        }
        if (prime = true)
        {
            int pos = sum(arr[i]);
            for (int j = 2; j * j <= pos; ++j)
            {
                if (pos % j == 0)
                {
                    prime = false;
                    break;
                }
            }
```

```
        }
        if (prime == true) ans++;
    }
}

void subtask3()
{
    bool prime[N];
    for (int i = 1; i <= 1000000; ++i) prime[i] = true;
    prime[0] = prime[1] = 0;
    for (int i = 2; i * i <= 1000000; ++i)
    {
        if (prime[i] == true)
        {
            for (int j = i * i; j <= 1000000; j += i)
            {
                prime[j] = false;
            }
        }
    }
    for (int i = 1; i <= n; ++i)
    {
        if (prime[arr[i]] == true && prime[sum(arr[i])] == true) ans++;
    }
}

int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);
    cin >> n;
    for (int i = 1; i <= n; ++i)
    {
        cin >> arr[i];
        maxi = max(maxi, arr[i]);
    }
    if (n <= 1000 && maxi <= 1000) subtask1();
    else if (n <= 10000 && maxi <= 1000000) subtask2();
    else subtask3();
    cout << ans;
    return 0;
}
```

18.6 Câu 4:

18.6.1 Subtask 1: $N, M \leq 10^3$; $1 \leq A_i \leq 10^9$; $1 \leq k_j \leq 10^9$

Thuật toán: Duyệt cơ bản

Với mỗi độ dài k_j của thanh sắt, ta duyệt qua mọi thanh sắt A_i hiện có để tính tổng độ dài cần cắt đi và nối thêm. Độ phức tạp tổng quát của thuật toán này là $O(N*M)$.

18.6.2 Subtask 2: $N, M \le 10^5$; $1 \le A_i \le 10^9$; $1 \le k_j \le 10^9$

Thuật toán: Mảng cộng dồn + Tìm kiếm nhị phân
Lý thuyết Mảng cộng dồn: Mảng cộng dồn và mảng hiệu
Lý thuyết Tìm kiếm nhị phân: Tìm kiếm nhị phân
Với subtask này, ta thực hiện sắp xếp lại dãy thanh sắt A tăng dần. Khi đó, gọi pre_i là tổng độ dài các thanh sắt, tính đến thanh thứ i. Với mỗi độ dài k_j, ta thực hiện tìm kiếm nhị phân vị trí lớn nhất của dãy A có A_i nhỏ hơn k_j (tạm gọi vị trí này là pos), từ đó ta có thể trả lời yêu cầu của đề bài bằng công thức: $(pos * query_j - pre_{pos}) + (pre_n - pre_{pos}) - (n - pos) * query_j$.
Với việc duyệt qua từng giá trị k_j, tìm kiếm nhị phân trên tập hợp các thanh sắt A, ta có độ phức tạp cho thuật toán này là $O(M * log_2 N)$.

18.6.3 Code mẫu:

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

const int N = 1e5 + 5;
int arr[N], query[N];
int n, m;
long long ans;

int binarysearch(int k)
{
    int res = 0, l = 1, r = n, mid;
    while (l <= r)
    {
        mid = (l + r) / 2;
        if (k >= arr[mid])
        {
            res = mid;
            l = mid + 1;
        }
        else r = mid - 1;
    }
    return res;
}

void subtask1()
{
    for (int j = 1; j <= m; ++j)
    {
        ans = 0;
```

```
        for (int i = 1; i <= n; ++i)
        {
            if (arr[i] > query[j]) ans += (long long)arr[i] - query[j];
            else if (arr[i] < query[j]) ans += (long long)query[j] - arr[i];
        }
        cout << ans << ' ';
    }
}

void subtask2()
{
    long long pre[N];
    pre[0] = 0;
    sort(arr + 1, arr + n + 1);
    for (int i = 1; i <= n; ++i) pre[i] = pre[i - 1] + arr[i];
    for (int j = 1; j <= m; ++j)
    {
        int pos = binarysearch(query[j]);
        cout << (pos * query[j] - pre[pos]) + (pre[n] - pre[pos])
        - (n - pos) * query[j] << ' ';
    }
}

int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);
    cin >> n >> m;
    for (int i = 1; i <= n; ++i) cin >> arr[i];
    for (int j = 1; j <= m; ++j) cin >> query[j];
    if (n <= 1000 && m <= 1000) subtask1();
    else subtask2();
    return 0;
}
```

18.7 Câu 5:

18.7.1 Subtask 1: $n, m \leq 1000$, $1 \leq k \leq 10$

Với mỗi S_k ta duyệt mọi cặp i, j có thể có để tìm ra chênh lệch tối thiểu.

18.7.2 Subtask 2: $n, m \leq 10^5, 1 \leq k \leq 10$

Ta sắp xếp lại mảng A tăng dần, bây giờ với mỗi S_k ta duyệt j từ 1 đến n, bây giờ với cặp j, k ta cần áp dụng tìm kiếm nhị phân trên A để tìm i lớn nhất sao cho $A_i + B_j \leq S_k$, sau đó ta cập nhật $ans_k = min(ans_k, A_i + B_j - S_k, S_k - A_{i+1} - B_j)$.

18.7.3 Subtask 3: $n, m \leq 10^5, 1 \leq k \leq 500$

Ta cải tiến thuật toán từ sub 2 sắp xếp mảng B và mảng A, lúc này ở chỗ tìm i ta hoàn toàn có thể áp dụng kỹ thuật two pointer, tuy nhiên ta tìm i nhỏ nhất sao cho $A_i + B_j >= S_k$, lúc này ta có nhận xét khi j tăng thì i sẽ giữ nguyên hoặc giảm nên ta có thể áp dụng two pointer không cần phải chặt nhị phân.

code mẫu

```
    #include<bits/stdc++.h>
using namespace std;
#define long long long
#define ll pair<long,long>
#define f first
#define s second

const int N=1e5+10;
long l,n,m,k,ans,A[N],B[N],S[N];
int main(){
    ios::sync_with_stdio(0);cin.tie(0);
    cin>>n>>m>>k;
    for(int i=1; i<=n; i++) cin>>A[i];
    for(int j=1; j<=m; j++) cin>>B[j];
    for(int i=1; i<=k; i++) cin>>S[i];
    sort(A+1,A+n+1);
    sort(B+1,B+m+1);
    for(int i=1; i<=k; i++){
        ans=2e9;
        l=n;
        for(int j=1; j<=m; j++){
            while(l>=1 && A[l]+B[j]>=S[i]) l--;
            if(l<n) l++;
            if(l){
                ans=min(ans,abs((A[l]+B[j])-S[i]));
            }
            if(l!=1){
                ans=min(ans,abs(S[i]-(A[l-1]+B[j])));
            }
        }
        cout<<ans<<' ';
    }
    return 0;
}
```

19 Đề Nghệ An Chuyên Đại học Vinh

19.1 Bài 1: Tin nhắn (6 điểm)

Ta sử dụng mảng a để lưu trữ, với a_i là số tin nhắn ngày thứ i.
Ta có $a_1 = 1$, số tin nhắn trong các ngày tiếp theo được tính bằng công thức:

$$\begin{cases} a_i = a_{i-1} \text{ nếu } i \text{ lẻ} \\ a_i = 2 * a_{i-1} \text{ nếu i chẵn} \end{cases}$$

Kết quả là sum($a_1, a_2, \ldots, a_n$).
Code mẫu

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

int main(){
    int n;
    cin >> n;
    vector<int> a(n + 1, 0);
    a[1] = 1;
    for(int i=2; i<=n; ++i){
        if (i % 2 == 1) a[i] = a[i-1];
        else a[i] = 2 * a[i-1];
    }
    int result = 0;
    for(int i=1; i<=n; ++i) result += a[i];
    cout << result;
    return 0;
}
```

uu

19.2 Bài 2: Chào hỏi (5 điểm)

Ta thấy một xâu có thể xóa thành xâu "hello" khi các chữ cái 'h', 'e', 'l', 'l', 'o' xuất hiện theo đúng thứ tự, tức 'h' xuất hiện đầu, rồi tới 'e', sau đó tới 2 chữ 'l' và cuối cùng là 'o'.
Ta sử dụng 5 biến boolean $h, e, l1, l2, o$ để lưu xem các chữ cái đã xuất hiện chưa.
Ta thấy 'h' xuất hiện trước 'e' nên biến e sẽ phụ thuộc vào biến h. Tương tự với các biến còn lại.
Kết quả là "YES" nếu biến o là $true$, ngược lại là "NO".
Code mẫu

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

```
int main(){
    string s;
    while(getline(cin, s)){
        bool h, e, l1, l2, o;
        h = e = l1 = l2 = o = false;
        for(char c : s){
            if (c == 'h') h = true;
            if (c == 'e') e = h;
            if (c == 'l'){
                if (l1 == false) l1 = e;
                else l2 = l1;
            }
            if (c == 'o') o = l2;
        }
        cout << (o ? "YES\n" : "NO\n");
    }
    return 0;
}
```

19.3 Bài 3: Trung vị (5 điểm)

Ta sẽ duyệt mọi cặp (i, j) với $(1 \le i \le j \le n)$, sau đó dùng kĩ thuật Merge Sort để gộp hai mảng đã được sắp xếp.

Code mẫu

```
#include <bits/stdc++.h>
using namespace std;
const int inf = INT_MAX;
const int N = 1e2 + 5;
const int M = 1e4 + 5;
int a[N][M], n, m;

int main(){
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    cin >> n >> m;
    for(int i=1; i<=n; ++i)
        for(int j=1; j<=m; ++j) cin >> a[i][j];
    int maxMiddle = -inf;
    int minMiddle = inf;
    for(int i=1; i<=n; ++i){
        for(int j=i+1; j<=n; ++j){
            vector<int> merge;
            int l=1, r=1;
            while(l <= m && r <= m){
                while(a[i][l] >= a[j][r]) merge.push_back(a[j][r]), ++r;
                while(a[i][l] <= a[j][r]) merge.push_back(a[i][l]), ++l;
            }
```

```
            while(l <= m) merge.push_back(a[i][l]), ++l;
            while(r <= m) merge.push_back(a[j][r]), ++r;
            maxMiddle = max(maxMiddle, merge[m-1]);
            minMiddle = min(minMiddle, merge[m-1]);
        }
    }
    cout << minMiddle << ' ' << maxMiddle;
    return 0;
}
```

Độ phức tạp: $O(n^2 * m)$

19.4 Bài 4: Giải mã (4 điểm)

Ta gọi End_i là khả năng để dãy $1..i$ là dãy có thể mã hóa. Với mọi i ($1 \le i \le n$), ta xét 2 TH:
- TH1: i là vị trí bắt đầu $\Rightarrow i-1$ phải là vị trí kết thúc cũa dãy trước đó $\Rightarrow End_{i+a_i} | = End_{i-1}$.
- TH2: i là vị trí kết thúc $\Rightarrow i - a[i] - 1$ phải là vị trí kết thúc của dãy trước đó $\Rightarrow End_i | = End_{i-a_i-1}$.

Code mẫu

```
#include <bits/stdc++.h>
using namespace std;
const int N = 1e5 + 5;
int n, a[N];
bool End[N];

int main(){
    ios_base::sync_with_stdio(0);
    cin.tie(0); cout.tie(0);
    int tests;
    cin >> tests;
    while(tests--){
        cin >> n;
        for(int i=1; i<=n; ++i) cin >> a[i];
        memset(End, false, sizeof End);
        End[0] = true;
        for(int i=1; i<=n; ++i){
            if (i + a[i] <= n) End[i + a[i]] |= End[i-1];
            if (i - a[i] - 1 >= 0) End[i] |= End[i-a[i]-1];
        }
        cout << (End[n] ? "YES\n" : "NO\n");
    }
    return 0;
}
```

Độ phức tạp: $O(n * k)$

20 Đề Nghệ An Chuyên Phan Bội Châu

20.1 Bài 1: Tổng nhỏ nhất

20.1.1 Subtask 1: $1 \le m \le n \le 10^6$

Lý thuyết: A * B = m * n. Từ đây ta có B = m*n/A.

Chạy vòng lặp A : 1 -> 10^6 và tính B theo công thức trên. Sau khi có A và B ta chỉ cần kiểm tra UCLN(A,B) = m và BCNN(A,B) = n có đúng không, nếu có thì cập nhật đáp án.

Lưu ý: Vì $n \le 10^6$ suy ra $A + B \le 2 * 10^6$. Vậy nếu đáp án đã cập nhật vượt quá $2 * 10^6$ thì in -1.

Độ phức tạp: $O(n)$

```
#include <bits/stdc++.h>
using namespace std;
const int maxA = 1e6;
long long n,m;

long long UCLN(long long a,long long b){
    if (b == 0){
        return a;
    }
    else return UCLN(b, a%b);
}

main(){
    ios::sync_with_stdio(0);
    cin.tie(0); cout.tie(0);
    cin >> m >> n;
    long long ans = 1e18;
    for (int a = 1; a <= maxA; a++){
        long long b = m*n / a;
        long long x = UCLN(a,b);
        if (x == m && (a*b)/x == n){
            ans = min(ans, a+b);
        }
    }
    if (ans <= 2*maxA)
        cout << ans;
    else
        cout << -1;
    return 0;
}
```

20.1.2 Subtask 2,3: $10^6 < m \le n \le 10^{12}$

Vì n là BCNN của A và B nên ta có A * B = n.Lúc này ta chỉ cần duyệt A : 1 -> $\sqrt{n}$. Bước này ta có được hai giá trị A thoả mãn là: A và n/A rồi tìm B theo công thức ở $Subtask1$ tương ứng với hai giá trị của A vừa đề cập. Và kiểm tra cùng với in đáp án làm như $Subtask1$.
Lưu ý: Vì $m,n \le 10^{12}$ nên lúc ta lắp công thức B = m*n/A thì chuyển thành B = (n/A) * m để tránh tràn số.
Độ phức tạp: $O(\sqrt{n})$

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

long long n,m;

long long UCLN(long long a,long long b){
    if (b == 0){
        return a;
    }
    else return UCLN(b, a%b);
}

main(){
    ios::sync_with_stdio(0);
    cin.tie(0); cout.tie(0);
    cin >> m >> n;
    long long ans = 1e18;
    for (long long a = 1; a <= sqrt(n)+1; a++){
        long long b1 = (n/a)*m;
        long long x1 = UCLN(a,b1);
        if (x1 == m && (a*b1)/x1 == n){
            ans = min(ans, a+b1);
        }
        long long b2 = (n/(n/a))*m;
        long long x2 = UCLN(n/a,b2);
        if (x2 == m && (n/a*b2)/x2 == n){
            ans = min(ans, (n/a)+b2);
        }
    }
    if (ans <= 2*1e12)
        cout << ans;
    else
        cout << -1;
    return 0;
}
```

20.2 Bài 2: Tách mã số

20.2.1 Subtask 1,2,3: $0 \le |S| \le 10^6$

Duyệt từng kí tự trong xâu chính, nếu gặp kí tự số ta lưu dần vào một xâu phụ đến khi nào gặp chữ cái thì đưa xâu phụ đó vào một vector <string> và làm mới xâu phụ. Sau khi duyệt hết xâu chính thì sort vector <string> từ bé đến lớn, đây cũng chính là đáp án.
Độ phức tạp: $O(|S|)$

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

vector< pair<int, string> > ans;
string st,s;
int value;

int main(){
    ios_base::sync_with_stdio(0);
    cin.tie(0); cout.tie(0);
    cin >> st;
    st += "#";
    bool ok = false;
    int cnt = 0;
    for (int i = 0; i < st.size(); i++){
        value = st[i]-'0';
        if (value >= 0 && value <= 9){
            if (value == 0 && !ok) ++cnt;
            else s += st[i], ok = true;
        }
        else{
            if (s != "") ans.push_back({cnt, s});
            s = "";
            cnt = 0;
            ok = false;
        }
    }
    sort(ans.begin(),ans.end(),[&](const pair<int, string> &a, const pair<int, string> &b)
        string A = a.second, B = b.second;
        return (A.size() < B.size() || (A.size() == B.size() && A < B));
    });
    for (auto x : ans){
        for(int i=0; i < x.first; ++i) cout << '0';
        cout << x.second << ' ';
    }
    return 0;
}
```

20.3 Bài 3: Thống kê sản phẩm

20.3.1 Subtask 1,2,3: $1 \le n \le 4 * 10^5$

Ta dùng mảng đếm lưu lại số lần xuất hiện của a_i. Áp dụng kĩ thuật hai con trỏ, với mỗi con trỏ i hãy "dãn" con trỏ j ra xa nhất có thể sao cho $j \le n$ và số lần xuất hiện của $a_j < k$. Sau khi j được "dãn" hết cỡ thì với mỗi (i,j) ta cập nhật đáp án thêm $n-j+1$ (Giải thích: vì đoạn (i,j) hợp lệ nên đoạn $(i,j+1),(i,j+2),...,(i,n)$ cũng hợp lệ) và giảm số lần xuất hiện của a_i đi 1 đơn vị để chuẩn bị cho lần "dãn" tiếp theo.
Độ phức tạp: $O(n)$.

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

const int maxN = 1e6+5;
long long n,k,cnt[maxN],a[maxN],ans;

main(){
    ios::sync_with_stdio(0);
    cin.tie(0); cout.tie(0);
    cin >> n >> k;
    for (int i = 1; i <= n; i++){
        cin >> a[i];
    }
    int j = 0;
    for (int i = 1; i <= n; i++){
        while (j <= n && cnt[a[j]] < k){
            j++;
            cnt[a[j]]++;
        }
        ans += (n-j+1);
        cnt[a[i]]--;
    }
    cout << ans;
    return 0;
}
```

20.4 Bài 4: Đèn chiếu sáng công cộng

20.4.1 Subtask 1: $1 \le N,M \le 10^4$

Dùng thuật trâu, với mỗi ngôi nhà ta tính khoảng cách từ ngôi nhà đấy tới các cột đèn điện và lấy MIN các khoảng cách lưu vào d_i (d_i là khoảng cách ngắn nhất từ ngôi nhà thứ i tới cột đèn bất kì). Lấy đáp án là MAX của $d_i (1 \le i \le n)$ vừa tính được.
Độ phức tạp: $O(n^2)$

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

```
const int maxN = 1e4+5;
long long n,m,a[maxN],b[maxN],d[maxN],ans;

main(){
    ios::sync_with_stdio(0);
    cin.tie(0); cout.tie(0);
    cin >> n >> m;
    for (int i = 1; i <= n; i++)
        cin >> a[i];
    for (int i = 1; i <= m; i++)
        cin >> b[i];
    for (int i = 1; i <= n; i++){
        d[i] = 1e18;
        for (int j = 1; j <= m; j++)
            d[i] = min(d[i],abs(a[i]-b[j]));
        ans = max(ans, d[i]);
    }
    cout << ans;
}
```

20.4.2 Subtask 2: $10^4 < N, M \leq 10^5$

Sort mảng A và mảng B theo thứ tự không giảm. Với mỗi ngôi nhà, tìm cột đèn điện gần nhất phía bên trái của ngôi nhà và từ đó suy ra cột đèn điện gần nhất phía bên phải của ngôi nhà là cột đèn kế bên cột đèn điện gần nhất phía bên trái (Ta có thể dùng kĩ thuật hai con trỏ để giải quyết vấn đề này). Khi tìm được hai cột đèn gần nhất với ngôi nhà, ta tìm MIN khoảng cách giữa hai cột đèn đó với ngôi nhà và lưu vào d_i. Đáp án là MAX của $d_i (1 \leq i \leq n)$.

Độ phức tạp: $O(nlogn)$

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

const int maxN = 1e5+5;
int n,m,a[maxN],b[maxN],ans;

main(){
    ios::sync_with_stdio(0);
    cin.tie(0); cout.tie(0);
    cin >> n >> m;
    for (int i = 1; i <= n; i++)
        cin >> a[i];
    for (int i = 1; i <= m; i++)
        cin >> b[i];
    sort(b+1, b+m+1);
    sort(a+1,a+n+1);
```

```
    int r = 0;
    for (int l = 1; l <= n; l++){
        while (r+1 <= m && b[r+1] <= a[l]){
            r++;
        }
        if (r != 0)
            ans = max(ans,min(abs(a[l]-b[r]),abs(a[l]-b[min(r+1,m)])));
        else
            ans = abs(abs(a[l]-b[min(r+1,m)]));
    }
    cout << ans;
}
```

21 Đề Ninh Bình

21.1 Bài 1: Chuỗi vỏ ốc

21.1.1 Subtask 1 : $0 \le m,n \le 10^3$

- Phân tích đề bài, ta có thể thấy số lượng vỏ sò màu trắng trong 1 xâu phải chia hết cho m và vỏ sò màu xám trong 1 xâu phải chia hết cho n. Điều này xảy ra bởi vì ta phải chia đều ra và không dư bất kì vỏ sò nào. Tóm lại số vỏ sò màu trắng phải là **ước** của m và số vỏ sò màu xám phải là **ước** của n.
- Với hướng tiếp cận trâu bò nhất, ta có thể duyệt 2 số x - số vỏ sò màu trắng và số y - số vỏ sò màu xám trong 1 xâu ốc sao cho x là ước của m và y là ước của n. Và kiểm tra $\frac{m}{x} = \frac{n}{y}$ hay không, nếu có thì đó là 1 phương án. Ta có thể code như sau với độ phức tạp là $O(n*m)$:

```
for(int y = 1 ; y <= n ; y++){
    if(n % y != 0) continue;
    for(int x = 1 ; x <= m ; x++){
        if(m % x != 0) continue;
        if(m / x == n / y)
            result = max(result , m / x);
    }
}
```

21.1.2 Subtask 2 : $10^3 \le m,n < 10^9$

- Ta có một bước cải tiến bằng việc duyệt ước trong $O(\sqrt{n}+\sqrt{m})$ để liệt kê tất cả ước của n và m. Và sau đó duyệt qua tất cả các cặp ước. Để tìm tất cả cặp $\frac{m}{x_i} = \frac{n}{y_i}$ (với x_i là ước thứ i của m và y_i là ước thứ i của n). Tóm lại độ phức tạp tổng quát của ta hiện tại là $O(\sqrt{n}*\sqrt{m})$. Ta có thể code như sau:

```
int result = 0;
vector<int> x , y;
```

```
for(int i = 1 ; i * i <= n ; i++){
    if(n % i == 0){
     y.push_back(i);
        if(n / i != i)
            y.push_back(n / i);
    }
}

for(int i = 1 ; i * i <= m ; i++){
        if(m % i == 0){
        x.push_back(i);
         if(m / i != i)
             x.push_back(m / i);
    }
}

for(auto a : x){
    for(auto b : y){
        if(m / a == n / b)
            result = max(result , m / a);
    }
}
```

21.1.3 Subtask 3 : $(10^9 \le m,n \le 10^{18})$

- Tới subtask cuối cùng, ta hãy lật lại góc nhìn của mình, cho tới thì bài toán của ta đang là tìm cặp x và y là ước của m và n sao cho $\frac{m}{x} = \frac{n}{y}$, ta hãy đặt một giá trị $k = \frac{m}{x} = \frac{n}{y}$, tới đây bài toán của ta thay đổi thành, tìm số k là ước chung lớn nhất của m và n. Và sau đó kết quả của ta là $\frac{m}{GCD(n,m)}$. Code mẫu:

```
#include <bits/stdc++.h>

using namespace std;

int main() {
    freopen("CHUOI.INP" , "r" , stdin);
    freopen("CHUOI.OUT" , "w" , stdout);

    long long n , m;
    cin >> m >> n;
    cout << m / __gcd(m , n);
    return 0;
}
```

21.2 Bài 2: Đếm kí tự

21.2.1 Subtask 1 + 2: $(1 \le k \le n < 10^4)$

- Ta có thể thực hiện việc duyệt qua xâu độ dài n 26 lần, vỡi mỗi lần check 1 kí tự trong khoảng $A \to Z$ để đếm kí tự ấy có xuất hiện không ít hơn k lần hay không. Độ phức tạp tổng quát là $O(26 * n)$. Ta có thể code như sau:

```
for(char x = 'A' ; x <= 'Z' ; x++){
    int cnt = 0;
    for(int i = 0 ; i < s.size() ; i++)
        if(s[i] == x)
            cnt++;
    if(cnt >= k)
        cout << x;
}
```

21.2.2 Subtask 3: $(10^4 \le k \le n \le 10^6)$

- Ta có bước cải tiến như sau, việc duyệt qua n chữ cái là **không cần thiết**. Thay vào đó, ta hãy duyệt duy nhất **1 lần** và lưu lại số lần xuất hiện của các kí tự từ $A \to Z$. Việc về bản chất chính là sử dụng **mảng đếm**. Gọi cnt_x là số lần kí tự x xuất hiện trong xâu S, kết quả của ta sẽ là các x có $cnt_x \ge k$ và được in ra theo thứ tự từ điển. Độ phức tạp của ta chỉ còn $O(n)$. Code mẫu như sau:

```
#include <bits/stdc++.h>

using namespace std;

int n , k;
string s;
int cnt[26];

int main() {
    freopen("KITU.INP" , "r" , stdin);
    freopen("KITU.OUT" , "w" , stdout);

    cin >> n >> k >> s;

    for(int i = 0 ; i < s.size() ; i++)
        cnt[s[i] - 'A']++;

    for(int i = 0 ; i < 26 ; i++)
        if(cnt[i] >= k)
            cout << (char)(i + 'A');
    return 0;
}
```

21.3 Bài 3: Khám phá vũ trụ

- Phân tích đề bài, ta thấy trong 1 giây, con tàu thử nghiệm đi được $k(m)/s$ và con tàu giám sát tại giây thứ i đi được $a_i(m)$. Gọi X_A là vị trí của tàu thử nghiệm, X_B là vị trí của tàu giám sát. Để kiểm tra ảnh rõ nét hay không, ta đơn giản chỉ cần kiểm tra $|X_A - X_B| \leq p$. Như vậy, bài toán này đơn giản chỉ là ta duy trì X_A và X_B trong mỗi giây, với giây thứ i ta kiểm tra $|X_A - X_B| \leq p$ hay không, nếu có thì đó là 1 tấm ảnh rõ. Code mẫu như sau:

```
#include <bits/stdc++.h>

using namespace std;

int n, k, p;
int a[1000005];

int main() {

    ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);

    freopen("SAOHOA.INP" , "r" , stdin);
    freopen("SAOHOA.OUT" , "w" , stdout);

    cin >> n >> k >> p;
    for(int i = 1 ; i <= n ; i++)
        cin >> a[i];

    int XA = 0 , XB = 0, res = 0;

    for(int i = 1; i <= n ; i++){
        XA += k;
        XB += a[i];

        if(abs(XA - XB) <= p)
            res++;
    }

    cout << res << endl;
    return 0;
}
```

21.4 Bài 4: Xếp hàng

21.4.1 Subtask $1+2$: $5 \leq n < 10^4, 1 \leq a_i < 10^9$

- Với subtask này, ta chỉ cần làm y hệt như đề bảo. Tức là ta sẽ duyệt mọi giá trị $value$ và sau đó thử nghiệm để kiểm tra n kiện hàng có chia đều sao cho mọi chiếc xe tải đều mang tổng cân nặng của hàng bằng $value$ hay không.

- Ta kiểm tra bằng cách cách duyệt mọi phần tử i và duy trì một biến cur (là tổng cân nặng của hàng trên chiếc xe tải hiện tại), nếu :

- $cur > value$ thì giá trị $value$ không thỏa mãn yêu cầu, ta sẽ kết thúc quả trình kiểm tra vì không thỏa mãn.
- $cur = value$ thì tới i mình có thể chia một cách thỏa mãn cho các chiếc xe tải rồi ta đặt lại $cur = 0$, tượng trưng cho việc tạo ra một chiếc xe tải mới rồi duyệt tiếp tục.
- $cur < value$ tức là xe tải hiện tại vẫn chưa có đủ hàng nên ta sẽ cộng thêm món hàng thứ i vào chiếc xe tải và duyệt tiếp tục.

- Code mẫu như sau:Link.

21.4.2 Subtask 3: $n \leq 10^6, A_i \leq 10^9$

Gọi sum_i là tổng tiền tố $(A_1 + A_2 + A_3 ... A_i)$ - Ở subtask này, ta nhận xét đáp án sẽ là ước của tổng của dãy và đồng thời là 1 trong các giá trị sum_i. Từ đây ta duyệt từng sum_i, nếu sum_i là ước của tổng dãy số (sum_n) thì lúc này ta tiến hành kiểm tra coi có thể set các xe bán hàng có độ tải trọng là sum_i. Để kiểm tra nhanh ta cần áp dụng chặt nhị phân tìm các giá trị $sum_i, sum_i * 2, sum_i * 3, sum_i * 4, ...$ nếu như trong lúc chặt nhị phân, nếu như ta tìm không ra $sum_i * x$ mà vị trí có giá trị $sum_i * (x-1)$ không phải n thì lúc này cách chia sum_i là không thỏa mãn. về độ phức tạp, số lượng ước cần xét chắc chắn sẽ nhỏ hơn n, và làm kỹ hơn (chạy một số số lớn và đếm ước chúng) thì ta thấy số lượng ước rơi vào khoảng $<= 5 * 10^5$. Về chi phí hàm kiểm tra sum_i, ta tốn một lần chặt nhị phân để tìm ra các giá trị thỏa mãn, đồng thời số lần chặt nhị phân= số lượng bội của sum_i mà ta cần check. Chặt nhị phân tốn log, còn số lượng bội cần check nếu các bạn liên tưởng sang cách hoạt động sàng nguyên tố (duyệt mọi bội của mỗi số một số thì ta thấy độ phức tạp tổng thể là $nlog$) thế nên đút kết lại độ phức tạp bài này $5 * 10^5 * log(n)^2$

21.4.3 Subtask 4:

-Giải thuật như subtask 3, tuy nhiên vì đáp án vướt quá long long và không thể dùng bignum, giải pháp duy nhất có thể là chuyển code sang python hoặc nếu bạn dùng C++20 thì có thể tham khảo kiểu dữ liệu int128 để lưu trữ, vì vài lý do bất tiện nên subtask này mình không có uptest lên nhé ! và phần điểm subtask 4 chia đều cho 3 sub trên

22 Đề Phú Yên

22.1 Bài 1: (4,0 điểm) Tìm mã OTP

Ta gọi ans là đáp án của bài toán, lúc này ta duyệt qua tất cả kí tự và đánh dấu sự xuất hiện của các kí tự, ta tăng ans với mỗi kí tự chưa được đánh dấu.

22.1.1 Code mẫu tham khảo:

```
#include <bits/stdc++.h>

using namespace std;



int main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(0);

    string s;
    while(cin >> s)
    {
            bool a[311] = {};
            int ans = 0;
            for(char v : s)
            {
                    if (a[v] == 0)
                    {
                            a[v] = 1;
                            ++ans;
                    }
            }

            cout << ans;
    }


    return 0;
}
```

22.2 Bài 2: (4,0 điểm) Tìm khoảng cách ngày

Vì dữ kiện đề bài cho các ngày xuất hiện trong thế kỉ từ 20 tới 21, nên ta chỉ cần duyệt qua các ngày trong 2 thế kỉ này và đánh dấu nó là ngày thứ mấy, lưu ý kiểm tra năm nhuận, các ngày trong năm.

22.2.1 Code mẫu tham khảo:

```
#include <bits/stdc++.h>

using namespace std;

string Start, End;
```

```
int DMY[32][13][2101];

bool check_leap_year(int year)
{
        if (year % 400 == 0)
                return true;
        if (year % 4 == 0 && year % 100 != 0)
        return true;
    return false;
}

int day_of_month(int month, int year)
{
        if (month == 2)
        {
                if (year % 4 == check_leap_year(year) == true)
                        return 29;
                else
                        return 28;
        }
        if (month == 4 || month == 6 || month == 9 || month == 11)
                return 30;
        return 31;
}

void precompute()
{
        int cur = 0;
        for(int year = 1900; year <= 2100; ++year)
                for(int month = 1; month <= 12; ++month)
                        for(int day = 1; day <= day_of_month(month, year); ++day)
                                DMY[day][month][year] = ++cur;
}

int STOI(string s) // change string to int
{
        int res = 0;
        for(int i = 0; i < (int)s.size(); ++i)
                res = res * 10 + (s[i] - '0');
        return res;
}

int main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(0);
```

```
    precompute();
        cin >> Start >> End;
        string Day_Start, Month_Start, Year_Start, Day_End, Month_End, Year_End;

        // Start
        Day_Start += Start[0];
        Day_Start += Start[1];
        Month_Start += Start[3];
        Month_Start += Start[4];
        Year_Start += Start[6];
        Year_Start += Start[7];
        Year_Start += Start[8];
        Year_Start += Start[9];

        // End
        Day_End += End[0];
        Day_End += End[1];
        Month_End += End[3];
        Month_End += End[4];
        Year_End += End[6];
        Year_End += End[7];
        Year_End += End[8];
        Year_End += End[9];

        int x = DMY[STOI(Day_End)][STOI(Month_End)][STOI(Year_End)];
        int y = DMY[STOI(Day_Start)][STOI(Month_Start)][STOI(Year_Start)];
        cout <<  x - y ;
    return 0;
}
```

22.3 Bài 3: (4,0 điểm) Ai nhanh hơn

Ta có thể thấy rằng việc lật ma trận ngược chiều tương đương với việc lật ma trận thuận chiều $n - |k|$ lần, vì thế ta chỉ cần sử dụng 1 hàm để lật bảng thuận chiều.
Để lật bảng thuận chiều 90°, với $i_1 \to n$ và $j_1 \to n$, ta thay hàng thứ i, cột thứ j thành hàng thứ j, cột thứ $n - i + 1$. Để thuận tiện hơn trong việc lưu trữ, ta có thể dùng một mảng tạm.

22.3.1 Code mẫu tham khảo:

```
#include <bits/stdc++.h>

using namespace std;

int a[51][51], n, k;

void clockwiseTurn()
{
```

```
        int rotated[51][51] = {};
        for(int i = 1; i <= n; i++)
                for(int j = 1; j <= n; j++)
                        rotated[j][n - i + 1] = a[i][j];
        for(int i = 1; i <= n; i++)
                for(int j = 1; j <= n; j++)
                        a[i][j] = rotated[i][j];
}

int main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(0);

    cin >> n >> k;
    for(int i = 1; i <= n; ++i)
        for(int j = 1; j <= n; ++j)
                cin >> a[i][j];

    if (k > 0)
    {
        int turn = k;
        for(int i = 1; i <= turn; i++)
                clockwiseTurn();
    }
    else
    {
        int turn = n + k;
        for(int i = 1; i <= turn; i++)
                clockwiseTurn();
    }

    for(int i = 1; i <= n; i++)
    {
        for(int j = 1; j <= n; j++)
                cout << a[i][j] << " ";
        cout << "\n";
    }

    return 0;
}
```

22.4 Bài 4: (4,0 điểm) Chuẩn hóa văn bản

Ta sẽ chuẩn hóa chuỗi theo những gì đề bảo, đây là bài thuần cài đặt:

22.4.1 Code mẫu tham khảo:

```cpp
#include <bits/stdc++.h>

using namespace std;

int main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(0);

    string s;
    while (getline(cin,s))
    {
        if (s == "") break;
        while (s[0] == ' ') s.erase(s.begin());
        int i = 0;
        while (i < s.size())
        {
            if ((s[i] >= 65 && s[i] <= 90) || (s[i] >= 97 && s[i] <= 122)) ++i;
            else
            {
                if (s[i] != ' ' && s[i] != '-' && s[i - 1] == ' ' && i - 1 > -1)
                {
                    s.erase(s.begin() + i - 1);
                    --i;
                }
                while (s[i + 1] == ' ') s.erase(s.begin() + i + 1);
                if (s[i] != ' ') s.insert(s.begin() + i + 1, ' ');
                ++i;
            }
        }
        cout << s << "\n";
    }

    return 0;
}
```

22.5 Bài 5: (4,0 điểm) Mua quà

22.5.1 Subtask 1: $n \leq 10^3$

Theo giả thiết của đề bài, thì chất lượng mỗi loại đều như nhau, vì thế ta sẽ chọn những thùng có số lượng bì càng lớn càng tốt, đầu tiên ta sắp xếp lại mảng giảm dần theo số lượng bì, vậy với mỗi i từ $1 \rightarrow n$ thì số tiền chúng ta nhận được là giá tiền loại i*min(số thùng còn có thể mua, số thùng loại i có), sau mỗi lần như vậy ta giảm số thùng còn mua được. Ta sắp xếp với độ phức tạp là $O(n^2)$.

22.5.2 Subtask 2: $n \leq 10^5$

Ý tưởng tương tự như Subtask 1 nhưng ta sử dụng hàm sắp xếp của C++ cho độ phức tạp tối ưu hơn là $O(n \log n)$.

22.5.3 Code mẫu tham khảo:

```cpp
#include <bits/stdc++.h>

using namespace std;

const int maxN = 1e5 + 5;
pair <int,int> a[maxN];
int n, m;

bool cmp(pair <int,int> x, pair <int,int> y)
{
        return x.second > y.second;
}

int main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(0);

    cin >> n >> m;
    for(int i = 1; i <= n; i++)
        cin >> a[i].first >> a[i].second;

    sort(a + 1, a + 1 + n, cmp);

    long long ans = 0;

    for(int i = 1; i <= n; i++)
    {
        int x = min(m, a[i].first);
        ans += x * a[i].second;
        m -= x;
    }

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

23 Đề Quảng Bình

23.1 Bài 1: Tính tiền điện (3 điểm)

23.1.1 Thuật toán: Cấu trúc rẽ nhánh

- Đây là một bài toán tính tiền điện kinh điển. Gọi t là số kwh chúng ta cần phải tính ($t = m - n$). Chúng ta xét t với mỗi mức kwh nếu t lớn hơn hoặc bằng thì ta trừ t đi số kwh cho sử dụng ở mức đó và cộng với một lượng tiền là số tiền ở mức đó nhân cho t - (số kwh ở mức đang xét), còn không thì cộng với số tiền ở mức đó nhân cho t rồi kết thúc bài toán.

23.1.2 Code mẫu tham khảo

```
#include<bits/stdc++.h>
using namespace std;
const int maxn = 1e6 + 5;
long long m,n,cnt;
int main() {
    ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
    cin >> m >> n;
    long long t = m-n;
    if (t >= 100) {
        t-=100;
        cnt += 1700*100;
    }
    else {
        cnt += t*1700;
        t = 0;
    }
    if (t >= 50) {
        t -= 50;
        cnt += 1900*50;
    }
    else {
        cnt += 1900*t;
        t = 0;
    }
    if (t >= 50) {
        t -= 50;
        cnt += 2100*50;
    }
    else {
        cnt += 2100*t;
        t = 0;
    }
    if (t != 0) cnt += t*2500;
```

```
    cout << cnt;
}
```

23.2 Bài 2: Nguyên tố (3.5 điểm)

23.2.1 Thuật toán: sàng số nguyên tố + mảng đếm

- Gọi 2 số nguyên tố lần lượt là x và y, ta có $x + y = n$, suy ra min(x,y) <= n/2.
- Bây giờ chỉ cần dùng kĩ thuật "lùa bò vào chuồng" để tìm các cặp, duyệt các số từ 1 đến n/2, với mỗi i nếu số đó là số nguyên tố và số $n - i$ cũng là số nguyên tố thì ta tăng biến đếm lên.

23.2.2 Code mẫu tham khảo

```
#include<bits/stdc++.h>
#define endl '\n'
#define int long long
using namespace std;
const int maxn = 3e7 + 5;
int n;
bool prime[maxn];
void sieve() {
    for (int i = 2; i <= 30000000; i++) {
        prime[i] = true;
    }
    for (int i = 2; i <= sqrt(30000000); i++) {
        if (prime[i] == true) {
            for (int j = i*i; j <= 30000000; j+=i) {
                prime[j] = false;
            }
        }
    }
}
//ai+aj = n => a[i] = n - a[j]
main() {
    ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
    sieve();
    cin >> n;
    int cnt = 0;
    for(int i = 1 ; i <= n / 2 ; i++)
        if(prime[i] == true && prime[n - i] == true)
            cnt++;
    cout << cnt;
}
```

23.3 Bài 3: dãy lồi (3.5 điểm)

23.3.1 Thuật toán: quy hoạch động

- Ta cần tạo 2 mảng:
$dp1[i]$ là dãy con giảm dần dài nhất kết thúc tại vị trí i. Công thức giống bài LIS kinh điển, vì đề bài giới hạn $n \leq 5000$ nên ta có thể tính trong $O(n^2)$.

```
for (int i = 1; i <= n; i++) {
dp[i] = 1; cong thuc co so
for (int j = 1; j < i; j++) {
    if (a[j] > a[i]) {
           dp[i] = max(dp[i],dp[j] + 1); cong thuc truy hoi
    }
}
}
```

- Tương tự cách tính dãy $dp1[i]$, $dp2[i]$ là dãy con tăng dần dài nhất kết thúc tại vị trí i, bắt đầu từ n.
- Sau khi tính được 2 dãy, ta có nhận xét: 1 dãy là dãy lồi khi và chỉ khi thỏa mãn $dp1[i]$ khác 1 và $dp2[i]$ khác 1, dãy lồi với điểm giữa là i được tính với công thức $dp[i]+dp2[i]-1$, kết quả là max của các dãy lồi.

23.3.2 Code mẫu tham khảo

```
#include<bits/stdc++.h>
#define endl '\n'
#define int long long
using namespace std;
const int maxn = 1e6 + 5;
int n,a[maxn],dp[maxn],dp2[maxn];
main() {
    ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
    cin >> n;
    for (int i = 1; i <= n; i++) cin >> a[i];
    for (int i = 1; i <= n; i++) {
        dp[i] = 1;
        for (int j = 1; j < i; j++) {
                if (a[j] > a[i]) {
                        dp[i] = max(dp[i],dp[j] + 1);
                }
        }
    }
    for (int i = n; i >= 1; i--) {
        dp2[i] = 1;
        for (int j = i+1; j <= n; j++) {
                if (a[i] < a[j]) {
```

```
                        dp2[i] = max(dp2[i], dp2[j] + 1);
                }
        }
    }
    int maxi = 0;
        for (int i = 1; i <= n; i++) {
                if (dp[i] != 1 && dp2[i] != 1) maxi = max(maxi,dp[i]+dp2[i]-1);
        }
        cout << maxi;
}
```

24 Đề Quảng Nam

24.1 Bài 1: Số lớn thứ k (3 điểm)

24.1.1 Hướng dẫn giải

- Phân tích đề: Bo muốn tìm chữ số lớn thứ K trong chuỗi số S -> nghĩ đến sắp xếp
- Lý thuyết sắp xếp: Sắp xếp
- Phương pháp phải: Vì là chữ số lớn thứ K nên là chúng ta sẽ sắp xếp giảm dần, để tìm ra chữ số lớn thứ K thỏa mãn trong chuỗi S. Tuy nhiên như thế là chưa đủ, vì các chữ số có thể trùng nhau, và K với giới hạn $1 \leq K \leq 9$ nên chúng ta cần tìm phần tử lớn thứ K phân biệt trong dãy số. Chúng ta có thể sử dụng *set* hoặc phương pháp thông thường là *sort* như bình thường, xong những kí tự nào trùng nhau thì mình sẽ bỏ qua, cho tới thấy được kí tự phân biệt mới, mình sẽ tăng giá trị đếm lên, cho tới khi bằng K thì dừng.

24.1.2 Code mẫu

```
#include <bits/stdc++.h>
using namespace std;
#define int long long
signed main()
{
    ios_base::sync_with_stdio(false); cin.tie(NULL); cout.tie(NULL);
    string s; int k;
    cin >> s >> k;
    sort(s.begin(),s.end(),greater<char>());
    int cnt = 1;
    if (cnt == k) return cout << s[0], 0;
    for (int i = 1; i < s.size(); i++)
        if (s[i] == s[i-1]) continue;
        else
        {
            cnt++;
            if (cnt == k)
```

```
            {
                cout << s[i];
                return 0;
            }
        }
    return 0;
}
```

Độ phức tạp $O(n * log(n))$.

24.2 Bài 2: Tổng chính phương (2 điểm)

24.2.1 Subtask 1: $1 \le n \le 10^3, 1 \le a_i \le 10^3$

Khi nhìn vào giới hạn đề bài, chúng ta sẽ nghĩ tới thuật toán chạy 2 vòng for lồng nhau, để tìm ra tổng các số chính phương từ $i -> j (1 \le i \le n, i+1 \le j \le n)$ sao cho thỏa mãn không vượt quá Max của dãy số.
Độ phức tạp: $O(n^2)$

24.2.2 Subtask 2: $1 \le n \le 10^4, 1 \le a_i \le 10^6$

- Với n lớn hơn 10^3, chúng ta không thể 2 for như Subtask 1 được nữa, nhưng mà nhìn vào giới hạn của mảng A, chúng ta có thể nghĩ tới việc sài mảng để lưu. Đây sẽ là phiên bản cần được cải tiến để Subtask 3 có thể làm được.
- Chúng ta sẽ lưu trước các số chính phương cho nó luôn bé hơn hoặc bằng 10^6 vào một mảng hoặc vector, sau đó chúng ta sẽ kiểm tra xem trong dãy số đã có những số chính phương nào và loại bỏ nó đi. Và chạy một for nữa để tính tổng các số chính phương không vượt quá Max của dãy số.
- Nhưng vì độ giới hạn về mặt lưu trữ của mảng hay vector, nên với thuật toán như này sẽ không pass được Subtask 3.
Độ phức tạp: $O(n * log(n))$

24.2.3 Subtask 3: $1 \le n \le 10^6, 1 \le a_i \le 10^9$

- Đây là thuật toán cải tiến của Subtask 2, chúng ta sẽ thay vì sử dụng mảng hay vector một cách thụ động và không đủ "diện tích" lưu trữ, chúng ta sẽ chuyển qua *set*, một cấu trúc dữ liệu cơ bản trong C++.
- Lý thuyết CTDL: std::set
- Cũng như subtask 2, chúng ta sẽ thực hiện công việc là lưu lại các số chính phương sao cho nó bé hơn hoặc bằng 10^9, sau đó cũng thực hiện công việc xóa đi các phần tử là số chính phương trong dãy, và tính tổng sao cho không vượt quá Max của dãy số.
Độ phức tạp: $O(n * log(n))$

24.2.4 Code mẫu

```
#include <bits/stdc++.h>
using namespace std;
#define int long long
const int N = (int)1e6+10;
int a[N];

signed main()
{
    ios_base::sync_with_stdio(false); cin.tie(NULL); cout.tie(NULL);
    int n;
    cin >> n;
    set<int> s;
    for (int i = 1; i <= sqrt(1e9); i++) s.insert(i*i);
    for (int i = 1; i <= n; i++)
    {
        cin >> a[i];
        if (s.find(a[i]) != s.end()) s.find(s.erase(a[i]));
    }
    int m = *max_element(a+1,a+n+1);
    int ans = 0;
    for (int x : s)
        if (x <= m) ans += x;
        else break;
    cout << ans;
    return 0;
}
```

Độ phức tạp $O(n * log(n))$.

24.3 Bài 3: Biến đổi xâu (3 điểm)

24.3.1 Hướng dẫn giải

Đọc xong đề bài, chúng ta có thể dễ dàng nhận ra một bài xử lí xâu cơ bản, chúng ta cứ làm theo yêu cầu đề bài là in ngược xâu lại, tìm ra các từ có độ dài dài nhất bằng viết xét các chuỗi con cách nhau bằng dấu ' ', sau đó sử dụng hàm $find$ trong C++ sử dụng cho việc tìm một chuỗi con hay kí tự con trong chuỗi chính trở nên dễ dàng hơn.
Độ phức tạp $O(n * log(n))$

24.3.2 Code mẫu

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

signed main()
```

```
{
    ios_base::sync_with_stdio(false); cin.tie(NULL); cout.tie(NULL);
    string s;
    getline(cin,s);
    s += ' ';
    int n = s.size();
    s = '#'+s;
    string st="";
    vector<string> v;
    for (int i = 1; i <= n; i++)
        if (s[i] == ' ')
        {
                reverse(st.begin(),st.end());
                v.push_back(st);
                st = "";
        }
        else st += s[i];
    s = "";
    for (auto x : v) cout << x << " ", s += x+" ";
    cout << endl;
    int m = -(int)1e18;
    for (auto x : v) m = max(m,(int)x.size());
    s = '#'+s;
    for (auto x : v)
        if (m == x.size())
        {
            int vt = s.find(x);
            cout << x << " " << vt << endl;
        }
    return 0;
}
```

24.4 Bài 4: Thiền nguyện (2 điểm)

24.4.1 Hướng dẫn giải

- Đây là một bài tham lam điển hình, đề bài yêu cầu việc vận chuyển phải thuận lợi nhất và ít tốn kém nhất, vậy nên chúng ta sẽ nghĩ tới $Sort$, việc $Sort$ giúp cho tối ưu công việc và yêu cầu đề bài trở nên tối ưu hơn, và đó cũng là cách làm của bài.
- Lý thuyết sắp xếp: Sắp xếp

Độ phức tạp: $O(n * log(n))$

24.4.2 Code mẫu

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

```cpp
const int N = (int)1e6+10;
int a[N];
pair<int,int> b[N];

signed main()
{
    ios_base::sync_with_stdio(false); cin.tie(NULL); cout.tie(NULL);
    int n,m;
    cin >> n >> m;
    for (int i = 1; i <= n; i++) cin >> a[i];
    for (int i = 1; i <= m; i++) cin >> b[i].first, b[i].second = i;
    sort(a+1,a+n+1,greater<int>());
    sort(b+1,b+m+1);
    int s = 0;
    for (int i = 1; i <= n; i++) s += a[i]*b[i].first;
    cout << s << endl;
    for (int i = 1; i <= n; i++) cout << b[i].second << " ";
    return 0;
}
```

25 Đề Quảng Ninh

25.1 Câu 1:

25.1.1 Subtask 1 + 2 + 3:

Đây là một bài toán kêu gì làm đó, ta chỉ cần duy trì 2 biến x và y và mỗi ghi gặp kí tự, chỉ cần tăng như đề yêu cầu.

25.1.2 Code mẫu tham khảo:

```cpp
#include<bits/stdc++.h>

using namespace std;

int main(){
    ios_base::sync_with_stdio(0);
    cin.tie(0);

    string s; cin >> s;

    int x = 0, y = 0;
    for(int i = 0; i < (int)s.size(); ++i){
        if(s[i] == 'E') ++x;
        else if(s[i] == 'W') --x;
        else if(s[i] == 'N') ++y;
        else --y;
```

```
    }

    cout << x << ' ' << y << '\n';

    return 0;
}
```

25.2 Câu 2: Đoán số:

25.2.1 Subtask 1:

Để dễ dàng giải bài toán, ta sẽ định nghĩa $a \geq b \geq c$. Ở subtask đầu tiên, vì khoảng cách giữa a và b bằng khoảng cách giữa b và c, từ đây ta có chèn thêm 2 số vào đầu và cuối dãy, ý kiểm tra số đầu dãy có lớn hơn 0 hay không.

25.2.2 Subtask 2:

Ở subtask 2, vì giới hạn các số đều bé hơn 10^3, ta có thể duyệt qua tất cả các số trong đoạn $[1, 10^3]$ để kiểm tra xem có thỏa mãn yêu cầu đề bài hay không.

25.2.3 Subtask 3:

Ở subtask cuối cùng, ta sẽ phải kiểm tra 4 vị trí có thể chèn số vào được hay không, đó là trước a, giữa a và b, giữa b và c, sau c. Từ đây ta chỉ cần kiểm tra khoảng cách với các đoạn còn lại.

25.2.4 Code mẫu tham khảo:

```
#include<bits/stdc++.h>

using namespace std;

int main(){
    ios_base::sync_with_stdio(0);
    cin.tie(0);

    vector<int> a = {0, 0, 0};
    cin >> a[0] >> a[1] >> a[2];
    sort(a.begin(), a.end());

    int constant = min(a[1] - a[0], a[2] - a[1]);

    if(a[1] - a[0] == a[2] - a[1]){
        if(a[0] - constant > 0) cout << a[0] - constant << ' ';
        cout << a[2] + constant << '\n';
    }
```

```
    else{
        if(a[1] - a[0] == 2 * constant) cout << a[0] + constant << ' ';
        else cout << a[1] + constant << ' ';
    }

    return 0;
}
```

25.3 Câu 3: Ước số

25.3.1 Subtask 1 + 2 + 3:

Cả 3 subtask đầu đều cần duyệt trâu với cả tiến rất ít, nên chúng ta sẽ gom cả 3 để giải cùng 1 lượt. Đầu tiên ta sẽ chuẩn bị 1 mảng D với ý nghĩa D_i là số ước nguyên dương của i, ta có thể thực hiện nó bằng cách đếm số ước trong độ phức tạp $O(\sqrt{n})$. Sau đó ta duyệt trâu và đếm số cặp (x, y) thỏa yêu cầu đề bài.

25.3.2 Subtask 4:

Ở subtask này, ta sẽ cải thiện việc đếm ước lên trong độ phức tạp $O(n \log n)$. Từ đây ta vẫn sử dụng duyệt trâu để đếm số cặp (x, y) thỏa yêu cầu đề bài.

25.3.3 Subtask 5:

Đầu tiên, từ phương trình của đề bài: $k \times d(x) \times d(y) = x \times y \Leftrightarrow \frac{k \times d(x)}{x} = \frac{y}{d(y)}$.
Từ đây, ta có thể xây dựng trước vế phải của phương trình, cố định vế trái và tìmy phù hợp.
Đầu tiên, ta định nghĩa $ndiv_i$ là số lượng ước nguyên dương của i, được xây dựng từ sàng nguyên tố $Eratosthene$.

25.3.4 Code mẫu tham khảo đếm ước bằng sàng nguyên tố:

```
void eratosthene(int lim){ // n_div[i] : so luong uoc cua i
    for(int i = 1; i * i <= lim; ++i){
        n_div[i * i] += 1;
        for(int j = i * (i + 1); j <= lim; j += i){
            n_div[j] += 2;
        }
    }
}
```

Vì với phân số đáp án sẽ trả về số thực, nên ta sẽ rút gọn phân số bằng gcd.
Gọi $division$ là hàm rút gọn phân số

25.3.5 Code mẫu tham khảo division:

```
pair<long long, long long> division(long long a, long long b){
    long long  g = __gcd(a, b);
    a /= g, b /= g;
    return make_pair(a, b);
}
```

Đồng thời, vì với mỗi cặp $\frac{a}{b}$, ta sẽ có vô số cặp (c,d) có $\frac{c}{d} = \frac{a}{b}$, vì thế để dễ xử lí, ta sẽ xóa đi những cặp bằng nhau.
Từ đây ta có thể dễ dàng giải phương trình đề bài.

25.4 Code mẫu tham khảo:

```
#include<bits/stdc++.h>

using namespace std;

const int MAX = 3e5 + 5;

int n, k, n_div[MAX];

void eratosthene(int lim){
    for(int i = 1; i * i <= lim; ++i){
        n_div[i * i] += 1;
        for(int j = i * (i + 1); j <= lim; j += i){
            n_div[j] += 2;
        }
    }
}

pair<long long, long long> division(long long a, long long b){
    long long  g = __gcd(a, b);
    a /= g, b /= g;
    return make_pair(a, b);
}

template<class T> int find_position(vector<T>& a, const T& v){
    int l = 0, r = (int)a.size();
    while(l <= r){
        int mid = (l + r) / 2;
        if(a[mid] == v) return mid;
        if(a[mid] < v) l = mid + 1;
        else r = mid - 1;
    }
    return -1;
}

int main(){
    ios_base::sync_with_stdio(0);
```

```
    cin.tie(0);

    cin >> n >> k;
    eratosthene(n);

    vector<pair<long long, long long>> fractions;
    for(int x = 1; x <= n; ++x){
        fractions.push_back(division(x, n_div[x])); // lam tron
    }

    sort(fractions.begin(), fractions.end());
    fractions.erase(unique(fractions.begin(), fractions.end()), fractions.end());

    vector<int> cnt((int)fractions.size());

    long long ans = 0;
    for(int x = 1; x <= n; ++x){
        pair<long long, long long> cur = division(x, n_div[x]);
        int pos = lower_bound(fractions.begin(), fractions.end(), cur)
        - fractions.begin();
        ++cnt[pos];

        cur = division(1LL * k * n_div[x], 1LL * x);
        pos = find_position(fractions, cur);
        if(pos != -1) ans += cnt[pos];
    }
    cout << ans << '\n';

    return 0;
}
```

25.5 Câu 4: Mật mã kho báu

25.5.1 Subtask 1:

Ở subtask đầu, ta chỉ cần duyệt trâu qua tất cả các cặp $l1, r1, l2, r2$ và kiểm tra xem những cặp nào thỏa yêu cầu đề bài, từ đó tối ưu độ dài.

25.5.2 Subtask 2:

Ở subtask 2, đề bài yêu cầu độ dài của mật mã phải lớn nhất có thể, từ đây ta có thể bắt đầu độ dài từ $|s|$, sau đó kiểm tra xem có cặp $l1, r1, l2, r2$ thỏa yêu cầu đề bài hay không.

25.5.3 Subtask 3 + 4:

Vì theo yêu cầu của đề bài nên 2 đoạn không được giao nhau, từ đó ta có thể chọn 1 cặp vị trí (i, j) bất kì sao cho $s_i = s_j$ và $i < j$, từ đó 2 cặp $l1, r1, l2, r2$ thỏa mãn là $i, j-1$ và $i+1, j$, mà

đề bài yêu cầu chọn đoạn sao cho độ dài là dài nhất có thể, từ đó ta thấy rằng ta cần tìm cặp (i,j) sao cho i là vị trí xuất hiện đầu tiên của s_i còn j là vị trí xuất hiện cuối cùng, từ đây ta kiểm tra xem độ dài của cặp (i,j) với $s_i = s_j = 0$ hay $s_i = s_j = 1$ có độ dài lớn hơn.

25.6 Code mẫu tham khảo:

```
#include <bits/stdc++.h>

using namespace std;

int first_position[2], last_position[2];

int main(){
    ios_base::sync_with_stdio(false);
    cin.tie(0);

    int T;
    cin >> T;

    for(int i = 1; i <= T; i++){
        string s;
        cin >> s;
        s = ' ' + s;
        first_position[0] = first_position[1] = INT_MAX;
        last_position[0] = last_position[1] = INT_MIN;

        for(int i = s.size() - 1; i >= 1; i--){
            first_position[s[i] - '0'] = min(first_position[s[i] - '0'], i);
            last_position[s[i] - '0'] = max(last_position[s[i] - '0'], i);
        }

        if ((int)s.size() == 2)
            cout << -1 << '\n';
        else if (first_position[0] == last_position[0]
                && first_position[1] == last_position[1]){
            cout << -1 << '\n';
        }
        else if (last_position[0] - first_position[0] >=
                    last_position[1] - first_position[1])
            cout << first_position[0] << ' ' << last_position[0] - 1 << ' '
                    << first_position[0] + 1 << ' ' << last_position[0] << '\n';
        else
            cout << first_position[1] << ' ' << last_position[1] - 1 << ' '
                    << first_position[1] + 1 << ' ' << last_position[1] << '\n';
    }

    return 0;
}
```

26 Đề Quảng Trị

26.1 Câu 1:

26.1.1 Hướng dẫn giải:

Ta chia bài toán thành 2 trường hợp:
- Nếu c_1 là màu đỏ (tức là $'R'$), ta có kết quả là giá trị nhỏ hơn giữa n và k.
- Nếu c_1 là màu xanh, ta có thể chia thành 2 trường hợp con:
+ Nếu $k \leq n$, tức là toàn bộ số thùng được xếp lên xe là màu xanh, kết quả là 0. + Nếu $k > n$, ta sẽ xem sau khi xếp n thùng xanh thì số lượng thùng đỏ bốc lên là giá trị nhỏ hơn giữa $k-n$ và m.

26.1.2 Code mẫu:

```
#include<bits/stdc++.h>
using namespace std;
long long n,m,k;
char c1,c2;
int main ()
{
    cin>>n>>c1;
    cin>>m>>c2;
    cin>>k;
    if(c1=='R') cout<<min(k,n);
    else
    {
        if(k<=n) cout<<0;
        else cout<<min(k-n,m);
    }
}
```

26.2 Câu 2:

26.2.1 Hướng dẫn giải:

Ta sẽ duyệt k lần, với mỗi lần nhảy ta lưu biến s để tính vị trí:
- Nếu $s+(d\%n)$ bằng n, thì $s=n$. Tức là tại vị trí s cũ, ta sẽ nhảy đến vị trí n ở bước nhảy hiện tại.
- Trường hợp còn lại, ta dễ dàng tính được $s=(s+d\%n)\%n$, vì khi ta chuyển động tròn thì tính được vị trí hiện tại bằng cách là xem thử đã đi được bao nhiêu vòng.
Sau khi tính s xong ta cộng lần lượt s vào biến res là kết quả của bài toán.

26.2.2 Code mẫu:

```
#include<bits/stdc++.h>
using namespace std;
long long n,d,k,s=1,res=1;
int main ()
{
    cin>>n>>d>>k;
    while(k--)
    {
        if(s+(d%n)==n) s=n;
        else s=(s+(d%n))%n;
        res+=s;
    }
    cout<<res;
}
```

26.3 Câu 3:

26.3.1 Subtask 1: $n \le 100$, $1 \le l \le r \le 100$

Ta duyệt lần lượt từ l đến r, tại mỗi giá trị i, ta cho chạy vòng lặp từ 2 đến i để tìm số ước chung của i và n. Nếu số ước chung không quá 2 và $i \neq n$ thì ta ghi nhận đó là một đáp án thoả mãn.

26.3.2 Subtask 2: $1 \le n \le 10^9$, $1 \le l \le r \le 10^9$, $r - l \le 1000$

Nhìn lại tính chất của số "không tương đồng": Hai số được gọi là không tương đồng là hai số có số ước chung không quá 2, mà trong đó có 1 ước là 1.

Ta có nhận xét: Tập hợp ước chung của 2 số chính là tập hợp ước của ước chung lớn nhất của 2 số đó. Do đó, 1 cặp số được gọi là không tương đồng khi và chỉ khi ước chung lớn nhất của chúng có không quá 2 ước và trong đó có 1 ước là 1. Vì vậy, ước chung lớn nhất của chúng chỉ có thể là 1 hoặc là số nguyên tố.

Từ nhận xét trên, ta duyệt lần lượt từ l lên đến r, số nào thoả mãn tính chất trên và khác n thì ta ghi nhận đó là một đáp án thoả mãn.

26.3.3 Code mẫu:

```
#include<bits/stdc++.h>
using namespace std;
int n,l,r,cs=0,a[1000004];
bool check(int n)
{
 for(int i =2  ;i<=sqrt(n);i++)
    if(n%i==0) return false;
 return true;
}
void sub1 ()
```

```
{

    for(int i = l;i<=r;i++)
     {
    int d=0;
    for(int j=2;j<=100;j++)
    {
         if(n%j==0&&i%j==0) d++;
    }
  if(d<=1&&i!=n)
  {
      cs++;
      a[cs]=i;
  }
    }
    cout<<cs<<endl;
for(int i=1;i<=cs;i++) cout<<a[i]<<" ";
}
void sub2 ()
{
for(int i =l;i<=r;i++)
    if(check(__gcd(i,n))&&i!=n)
    {
        cs++;
    a[cs]=i;
    }
    cout<<cs<<endl;
for(int i=1;i<=cs;i++) cout<<a[i]<<" ";

}
int main ()
{

cin>>n;
cin>>l>>r;
if(n<=100) sub1 ();
else sub2 ();

}
```

26.4 Câu 4:

26.4.1 Subtask 1: $2 \le n \le 1000$, chỉ gồm phép toán loại 1

Ta cập nhật giá trị $a_i = x$, sau đó sử dụng một vòng lặp để tính tổng các phần tử.

26.4.2 Subtask 2: $2 \le n \le 1000$

Với truy vấn loại 2: Ta thực hiện đổi chỗ theo đề bài, tức là ta sẽ gán lại dãy thành $a_{n-k+1}, a_{n-k+2}, \ldots, a_n, a_1, a_2, a_3, \ldots, a_k$.

Để làm điều này một cách đơn giản, ta sử dụng một mảng tmp để lưu các vị trí cũ của dãy a rồi thực hiện việc tráo đổi. Ta tráo đổi như sau: $a_1 = tmp_{n-k+1}$

$a_2 = tmp_{n-k+2}$

...

Sau khi hoán đổi xong dãy a, ta cập nhật lại mảng tmp để chuẩn bị cho các truy vấn loại 2 tiếp theo.

26.4.3 Subtask 3: $2 \le n \le 10^5$

Ta có những nhận xét sau:

- Ta không đi tính lại tổng ở mỗi truy vấn 1, mà ta sẽ tính tổng dãy ban đầu, rồi trừ đi hoặc công thêm phần chênh lệch của a_i cũ và a_i mới.
- Vì đây là dãy vòng tròn nên vị trí i tương đương vị trí $i\%n$.
- Với mỗi truy vấn loại 2, ta sẽ dùng một biến để lưu lại những thay đổi của phần tử đầu tiên khi dịch chuyển k lần: $ans = (ans - k + n)\%n$.

Ví dụ: a_1, a_2, a_3, a_4 khi đổi 2 lần ta có dãy a_3, a_4, a_1, a_2.

Ta thấy so với công thức trên thì vị trí của các giá trị a_i sẽ lệch 1 phần tử, thế nên ở mỗi truy vấn loại 1, thay vì cập nhật $i = (i + ans)\%n$, thì ta phải giảm đi 1, tức là $i = (i + ans - 1)\%n$. Khi đã có vị trí i, ta thực hiện nhận xét 1, sau đó cập nhật lại $a_i = x$ và đưa ra biến kết quả.

26.4.4 Code mẫu:

```
#include<bits/stdc++.h>
using namespace std;
long long n, q ,a[1000003],tmp[100003],sum,ans;
void sub12()
{
  while(q--)
  {
    long long type , i ,x ,k;
    cin>>type;
    if(type==1)
    {
      cin>>i>>x;
       a[i]=x;
   long long s=0;
    for(int   j=1 ;j<=n;j++) s+=a[j];
    cout<<s<<'\n';
    }
    else
    {
```

```
      cin>>k;
      for(int j =1 ;j<=n;j++) tmp[j]=a[j];
      int pos =0;
      for(int j = n-k+1;j<=n;j++)
      {
        pos++;
        a[pos]=tmp[j];
      }
      for(int j = 1;j<=n-k;j++)
      {
        pos++;
        a[pos]=tmp[j];
      }
       }
  }
}
void sub3 ()
{
  while(q--)
  {
    long long type,k,i,x;
    cin>>type;
    if(type==2)
    {
      cin>>k;
      ans=(ans-k+n)%n;
    }
    else
    {
      cin>>i>>x;
      i=(i+ans-1)%n+1;
      sum-=a[i]-x;
      a[i]=x;
      cout<<sum<<endl;
    }
  }
}
 int main ()
{

ios_base::sync_with_stdio(0);
cin.tie(0);cout.tie(0);
  cin>>n;
  for(int i =1 ;i<=n;i++)
  {
    cin>>a[i];
    sum+=a[i];
  }
```

```
  cin>>q;
  if(n<=1000) sub12();
  else sub3();

}
```

27 Đề Thanh Hoá

27.1 Bài 1: Phần thưởng (2 điểm)

27.1.1 Subtask1

- Ta duyệt qua toàn bộ n số, kiểm tra xem có số nào chia hết cho 90 không thì tăng biến kết quả lên 1:

27.1.2 Subtask2

- Do giới hạn đề bài cho vượt quá kiểu long long nên ta phải xử lí các số ở dạng xâu. Nhận xét rằng một số chia hết cho 90 thì phải chia hết cho 9 và 10 (do ước chung lớn nhất của 9 và 10 là 1), vì thế số chia hết cho 90 phải có tổng các chữ số chia hết cho 9 và chữ số tận cùng là 0.

27.1.3 Code mẫu tham khảo

```
 #include<bits/stdc++.h>
using namespace std;
//---variation---
int n,k,a[1000021],i,j;
int S,P;
string s;
//---solution---
int main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    cin>>n;
    S=0,P=0;
    for(int i=1;i<=n;i++)
    {
        cin>>s;
        for(int j=0;j<s.size();j++) P+=s[j]-'0';
            if(s[s.size()-1]-'0'==0)
            {
                if(P%9==0) S++;
            }
```

```
        P=0;
    }
    cout<<S;
}
```

27.2 Bài 2: Số đẹp (2 điểm)

27.2.1 Subtask 1 + Subtask 2

- Ta sẽ duyệt các số từ 1 cho đến khi tìm được số thỏa mãn yêu cầu. Vì cần phải kiểm tra nhiều số nguyên tố cho nên ta có thể áp dụng thuật toán sàng Eratosthenes để chương trình chạy nhanh hơn.

27.2.2 Code mẫu tham khảo

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

bool nt(int x)
{
if(x<=1) return false;
if(x==2) return true;
if(x>2 && x%2==0) return false;
for(int i=3;i<=sqrt(x);i+=2)
if(x%i==0)
return false;
return true;
}
//---variation---
long long n;
long long a[1000021],d;
long long S;
//---solution---
int main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    cin>>n;
    S=n;
    for(long long i=1;i<=1e5+7;i++)
        {
            a[i]=a[i/10]+((i%10)*(i%10));
            if(S>0)
            {
                if(nt(a[i])) d=i,S--;
            }
        }
```

```
        cout<<d;
}
```

27.3 Bài 3: Kế hoạch luyện tập (3 điểm)

27.3.1 Thuật toán: 2 con trỏ

- Ta sử dụng kĩ thuật 2 con tr, đặt 2 biến $i = 1$ và $j = 1$. Ta chạy j -> n, với mỗi j cộng a_j vào biến sum. Nếu kiểm tra thấy biến sum đã lớn hơn hoặc bằng S. Ta tăng i và cập nhật dãy con ngắn nhất có tổng lớn hơn hoặc bằng S từ i -> j. Nếu không xuất hiện tổng lớn hơn hoặc bằng S nữa thì tiếp tục tăng j.

27.3.2 Code mẫu tham khảo

```
#include<bits/stdc++.h>
#define int long long
using namespace std;
const int maxn = 1e7 + 5;
int n,s,sum,a[maxn],mini = 1e18;
int mod = 1e9+7;
main() {
    ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
    cin >> n >> s;
    for (int i = 1; i <= n; i++) cin >> a[i];
    int j = 1;
    for (int i = 1; i <= n; i++) {
        sum += a[i];
        while(sum - a[j] >= s) {
            sum -= a[j];
            j++;
        }
        if (sum >= s)
        mini = min(mini,i-j+1);
     //   cout << j << " " << i << endl;
    }
    if (mini == 1e18) cout << "-1";
    else
    cout << mini;
}
```

27.4 Bài 4: Biến đổi xâu (3 điểm)

27.4.1 Subtask 1

- Ta lưu hết tất cả các xâu có thể xảy ra của tập M, sau đó đếm số lượng xâu phân biệt. Độ phức tạp $O(n^2)$.

27.4.2 Subtask 2

- Ta có nhận xét như sau: với hai vị trí (i,j) (i < j) nếu s_i != s_j, ta thực hiện phép đảo (i,j) sẽ luôn nhận được xâu mới phân biệt. Ngược lại, khi $s_i = s_j$, ta thực hiện phép đảo xâu (i,j) sẽ tương tự với việc thực hiện phép đảo $(i+1, j-1)$. Khi đó, với mỗi vị trí i ta có $i-1$ kí tự trước đó và có k kí tự trong i–1 kí tự ấy giống với kí tự thứ i, ta sẽ có $i-1-k$ phép biến đổi xấu để có được một xâu phân biệt mới. Độ phức tap $O(n)$.

27.4.3 Code mẫu tham khảo

```
#include<bits/stdc++.h>

using namespace std;
bool nt(int x)
{
if(x<=1) return false;
if(x==2) return true;
if(x>2 && x%2==0) return false;
for(int i=3;i<=sqrt(x);i+=2)
if(x%i==0)
return false;
return true;
}
//---variation---
long long n,a[1000021];
long long S;
string s;
//---solution---
int main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);cout.tie(NULL);
    cin>>s;
     n=s.size();
     for(int i=0;i<n;i++) a[s[i]-'a']++;
        S=(n-1)*n/2 +1;
    for(int i=0;i<=25;i++) S-=a[i]*(a[i]-1)/2;
        cout<<S;
}
```

28 Đề Tiền Giang

28.1 Bài 1: Hàng rào (2 điểm)

Xét các hàng rào, ta thấy:
+ Có $n+1$ hàng rào nằm ngang, mỗi hàng rào có m ô $\Rightarrow$ có $(n+1)*m$ ô.

+ Có $m+1$ hàng rào nằm dọc, mỗi hàng rào có n ô $\Rightarrow$ có $(m+1)*n$ ô.
Code mẫu

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

int main(){
    cin.tie(0) -> sync_with_stdio(0);
    cout.tie(0);
    int n, m;
    cin >> n >> m;
    cout << 1LL * n * (m + 1) + 1LL * (n + 1) * m;
    return 0;
}
```

28.2 Bài 2: Lẻ chẵn (2 điểm)

- Phân tích đề: ta thấy được đề bài yêu cầu xây dựng mảng theo công thức là với i chẵn, ta sẽ có công thức là $A_i = (A_{i-1} + A_{i-2})\%k$ và với i lẻ, ta sẽ có công thức là $A_i = (A_{i-1} - A_{i-2})\%k$ -> đây là dạng DP Fibo cơ bản, chúng ta sẽ sử dụng mảng F để lưu các giá trị phần tử $1 -> n$.
- Phương pháp giải: giải y như cách DP Fibo thực hiện, là sẽ có trường hợp cơ sở là $F[1] = x$ và $F[2] = y$, tiếp tục như thế theo công thức như đề bài yêu cầu, chúng ta sẽ tìm ra giá trị của phần tử $F[n]$.
Độ phức tạp $O(n)$.
Code mẫu

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

int main(){
    cin.tie(0) -> sync_with_stdio(0);
    cout.tie(0);
    int n, x, y, k = (int)(1e9 + 7);
    cin >> x >> y >> n;
    vector<int> f(n + 1);
    f[1] = x;
    f[2] = y;
    for(int i=3; i<=n; ++i)
        if (i % 2 == 0) f[i] = (f[i-1] + f[i-2]) % k;
        else f[i] = abs(f[i-1] - f[i-2]) % k;
    cout << f[n];
    return 0;
}
```

28.3 Bài 3: Vòng tròn (2 điểm)

Nhận xét: Chiến thuật chơi tốt nhất của Bo sẽ là chạm vào vòng tròn có bán kính nhỏ nhất trước, rồi đến nhỏ nhì... cho đến khi hết vòng tròn hoặc vòng tròn tiếp theo có bán kính lớn hơn vòng tròn của Bo hiện tại.
Code mẫu

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

int main(){
    cin.tie(0) -> sync_with_stdio(0);
    cout.tie(0);
    int n, k;
    cin >> n >> k;
    vector<int> r(n);
    for(int i=0; i<n; ++i) cin >> r[i];
    sort(r.begin(), r.end());
    for(int i=0; i<n; ++i) if (k >= r[i]) k += r[i];
    cout << k;
    return 0;
}
```

28.4 Bài 4: Mật khẩu (2 điểm)

Ta sẽ sử dụng một biến j để chạy trong mảng f. Mỗi khi gặp dấu '*':
+ Nếu $f_j > 0$ thì trừ f_j đi 1 và thay dấu * thành kí tự thứ j kể từ sau kí tự 'a'.
+ Nếu $f_j = 0$ thì tăng biến j đến khi gặp $f_j > 0$.
Code mẫu

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

int main(){
    cin.tie(0) -> sync_with_stdio(0);
    cout.tie(0);
    string s;
    cin >> s;
    vector<int> f(26);
    for(int i=0; i<26; ++i) cin >> f[i];
    int j=0;
    for(int i=0; i<s.size(); ++i){
        if (s[i] != '*') continue;
        while(f[j] == 0) ++j;
        f[j] -= 1;
        s[i] = char(j + 'a');
    }
    cout << s;
```

```
    return 0;
}
```

28.5 Bài 5: Cắt hình (2 điểm)

Nhận xét: Vì $m \geq n$ nên cạnh hình vuông lớn nhất cắt được là chiều rộng của mảnh giấy. Do đó với mảnh giấy m x n, ta có thể cắt được là $\lfloor \frac{m}{n} \rfloor$.
Hình chữ nhật còn lại sau lần cắt là có độ dài cạnh là ($m\%n$, n). Ta hoán đổi hai cạnh để thỏa điều kiện $m \geq n$ và tiếp tục cắt theo nhận xét.
Code mẫu

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

int main(){
    cin.tie(0) -> sync_with_stdio(0);
    cout.tie(0);
    int m, n, result = 0;
    cin >> m >> n;
    while(m > 0 && n > 0){
        if (m < n) swap(m, n);
        result += m/n;
        m %= n;
    }
    cout << result;
    return 0;
}
```

29 Đề Vĩnh Long

29.1 Câu 1:

29.1.1 Hướng dẫn giải:

Ta tìm chiều rộng của khẩu hiệu theo dữ kiện chiều rộng bằng 2/3 chiều dài, sau đó tính diện tích khẩu hiệu bằng công thức tính diện tích của hình chữ nhật. Chú ý đến kiểu dữ liệu số thực của bài.

29.1.2 Code mẫu:

```
#include<bits/stdc++.h>

using namespace std;
long double n ,k;
 main ()
{
  cin>>n;
   k=2*n/3;
  cout<<setprecision(2)<<fixed<<n*k;
}
```

29.2 Câu 2:

29.2.1 Hướng dẫn giải:

Để ý rằng trong lịch các năm, chỉ có ngày 29 tháng 2 của năm nhuận là lặp lại sau bốn năm, còn lại đều xuất hiện hàng năm. Vì dữ liệu đảm bảo ngày tháng năm đúng, ta chỉ cần kiểm tra dữ liệu đầu vào có phải là ngày 29 tháng 2 hay không, nếu đúng thì in ra 4, nếu không thì in ra 1.

29.2.2 Code mẫu:

```
#include<bits/stdc++.h>
using namespace std;
int ngay ,thang ,nam;
int main ()
{
cin>>ngay>>thang>>nam;
if(ngay==29&&thang==2) cout<<4;
else cout<<1;
}
```

29.3 Câu 3:

29.3.1 Hướng dẫn giải:

Ta kiểm tra ba điều kiện: Số có hai chữ số trở lên, các chữ số tăng dần từ trái sang phải, tổng các chữ số chia hết cho 9. Nếu thoả mãn hai điều kiện này thì số n là số tiến đẹp, nếu không thì in ra -1.

29.3.2 Code mẫu:

```
#include<bits/stdc++.h>
using namespace std;
string s;
long long tong=0;
int d=0;
int main ()
{
  cin>>s;
  if (s.size()==1)
  {
      cout<<-1;
      return 0;
  }
  for(int i = 0 ;i<s.size();i++)
  {
    tong+=int(s[i]-48);
    if(int (s[i])>int(s[i-1]))
      d++;
  }

 if(d==s.size()&&tong%9==0) cout<<s;
   else cout<<-1;

}
```

29.4 Câu 4:

29.4.1 Hướng dẫn giải:

Ta dựng một mảng đếm, với cnt_i là số lượng số cây bút màu thứ i. Khi đó, ta chỉ cần đếm số lượng số $cnt_i \neq 0$ là có số màu khác nhau trong N cây bút. Ở mỗi cnt_i, ta kiểm tra liệu có bao nhiêu cây bút thứ i dư sau khi xếp vào các hộp có đúng K bút, điều này có thể dễ dàng được tính bằng phép chia lấy dư.

29.4.2 Code mẫu:

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

const int N = 1e3 + 5, M = 1e6 + 5;
int arr[N], cnt[M];
int n, k, ans, res, maxi;

int main()
{
    ios::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);
    cin >> n >> k;
    for (int i = 1; i <= n; ++i)
    {
        cin >> arr[i];
        if (cnt[arr[i]] == 0) ans++;
        cnt[arr[i]]++;
        maxi = max(maxi, arr[i]);
    }
    for (int i = 1; i <= maxi; ++i) res += cnt[i] % k;
    cout << ans << '\n' << res;
    return 0;
}
```

29.5 Câu 5:

29.5.1 Hướng dẫn giải:

Để đếm số lượng trụ gỗ cùng màu nhiều nhất, ta áp dụng cách dựng mảng đếm như bài 4, với bốn vị trí khác nhau tương trưng cho bốn màu xanh, đỏ, tím, vàng. Sau khi duyệt qua mọi kí tự, ta chỉ cần đưa ra giá trị lớn nhất của bốn giá trị đếm đó.

Để đếm số lượng trụ gỗ cùng màu liên tiếp nhiều nhất, ta duy trì hai biến *cnt* và *maxi*. Biến *cnt* dùng để đếm số lượng trụ gỗ cùng màu liên tiếp: tại vị trí mà màu trụ gỗ thay đổi, ta gán $cnt = 1$, nếu không thì tăng giá trị *cnt* lên 1. Với mỗi lần duyệt kí tự, ta gán *maxi* là giá trị lớn nhất giữa *cnt* và *maxi*. Sau khi duyệt mọi kí tự, biến *maxi* là kết quả cần tìm.

29.5.2 Code mẫu:

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

string s;
int dem[4];
int cnt = 1, ans = 1, maxi = 1;
```

```
int main()
{
    ios::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);
    cin >> s;
    for (int i = 0; i < s.size(); ++i)
    {
        dem[0] += (s[i] == 'X');
        dem[1] += (s[i] == 'D');
        dem[2] += (s[i] == 'T');
        dem[3] += (s[i] == 'V');
        ans = max({ans, dem[0], dem[1], dem[2], dem[3]});
        if (i == 0) continue;
        if (s[i] == s[i - 1])
        {
            cnt++;
            maxi = max(maxi, cnt);
        }
        else cnt = 1;
    }
    cout << ans << '\n' << maxi;
    return 0;
}
```

30 Đề Nam Định

30.1 Chưa kiếm được đề năm ngoái

31 Credit

31.1 Tổng biên tập

Nguyễn Nhâm Tấn

31.2 Ban tạo bài

Nguyễn Huy Phước

31.3 Ban viết đề + solution

Bùi Tuấn Anh
Võ Tấn Bảo
Lê Hồng Đăng
Nguyễn Thái Hưng
Nguyễn Trần Quang Lâm
Nguyễn Thanh Minh
Nguyễn Huy Phước
Phạm Vũ Phương
Vũ Minh Phương
Nguyễn Nhâm Tấn
Phạm Giao Thức
Nguyễn Ngọc Tuân