作业

1. Longest Balanced Substring (EEEnd)

1.1 Modeling

首先，对一个区间来说，如果有个字母没有对应的大写或者小写字母，那么可以以这些字母为切割点分成若干段。

例如：(aAzBbzCcDdzUu 中字母z不合法。将其分割成aA Bb CcDd Uu)。

那么可以这样一直分割的做。

1.2 Algorithm description

```
#include <bits/stdc++.h>
const int N = 1e5 + 10;
char c[N];
std::optional<std::vector<std::pair<int, int>>> split(int l, int r)
{
    std::vector<std::pair<int, int>> ans;
    if (l >= r)
        return ans;

    bool dis[110] = {false};
    for (int i = l; i <= r; i++)
        dis[c[i]] = true;
    bool wa[26] = {false};
    for (int i = 0; i < 26; i++)
        if (dis['a' + i] ^ dis['A' + i])
            wa[i] = true;

    bool f = false;
    for (int i = l; i <= r; i++)
    {
        int p = c[i] - (islower(c[i]) ? 'a' : 'A');
        if (!wa[p])
            continue;
        if (i - 1 - l + 1 > 1)
            ans.push_back({l, i - 1});
        f = true;
        l = i + 1;
    }
    if (l < r)
        ans.push_back({l, r});
    if (f)
        return ans;
    return std::nullopt;
}
int main()
{
    scanf("%s", c + 1);
    std::queue<std::pair<int, int>> qu;
    qu.push({1, strlen(c + 1)});
    int L = 0, R = 0;
    while (qu.size())
    {
        auto [l, r] = qu.front();
        qu.pop();
        auto list = split(l, r);
        if (list.has_value())
        {
            for (auto [l, r] : list.value())
                qu.push({l, r});
        }
        else if (R - L < r - l)
            R = r, L = l;
    }
```

```
    c[R + 1] = '\0';
    std::cout << " --- " << L << " " << R << " --- " << "\n";
    std::cout << c + L;
    return 0;
}
```

1.3 Time complexity

由于每分割1次，其字母集的个数 -2, 最多进行26次分割，其中每次最大复杂度为$O(n)$。
$\because T(n,m)=\sum T(len_i,m_i)$。
$\because T(n+1,m)>T(n,m) \quad \sum len_i<n \quad m_i<m \quad T(n,1)\leqslant n \quad T(n,2)=n$
$\therefore T(n,m)\leqslant \sum T(len_i,m-2)+n\leqslant mn=26n \quad \sum len_i=n$
$\therefore$ 时间复杂度：$O(n)$

1.4 Space complexity

程序中可以最多可以分成n段

空间复杂度：$O(n)$

2. Cutting Bamboo Poles (EEEnd)

2.1 Modeling Modeling

答案具有单调性，二分答案。

2.2 Algorithm description

```
const int N = 1e5 + 10;
int main()
{
    int a[N];
    int l = 0, r = 1e9;
    int n, m;
    while (l < r)
    {
        int mid = l + r >> 1;
        long long sum = 0;
        for (int i = 0; i < n; i++)
            sum += a[i] / mid;
        sum < m ? r = mid : l = mid + 1;
    }
    int ans = l - 1;
    return 0;
}
```

2.3 Time complexity

$O(n\log n)$ 最多进行 $\log n$次遍历，每次遍历耗时$O(n)$

2.4 Space complexity

$O(n)$ 只使用了存储原始数据的空间

3. Multiple Calculations

3.1 Modeling

可以采用对于一个区间枚举分割点来求解。

3.2 Algorithm description

```
// 预处理 字符串列表, 将其变成 integer 和 操作符序列 c
list<int> dp[n][n]
list<integer or char> c

def dfs(l, r):
    if l == r:
        dp[l][r] = c[i]
    if(dp[l][r] is calc):
        return dp[l][r]
    for i in range(l, r):
        if c[i] is op
            dp[l][r] += dfs(l, i-1) op dfs(i+1, r)
    return dp[l][r]
```

3.3 Time complexity

由于其只根据加括号的操作顺序进行重复判断.
所以计算dp值的时间和空间复杂度为 $dp[i] = \sum_{j=1}^{i-1} dp[j] * dp[i-j] = \frac{1}{n+1}\binom{2n}{n} \approx \frac{4^n}{n^{3/2}\sqrt{\pi}}$

$T(n) = \sum_{i=1}^{n}(n-i+1) * dp[i] \sim \frac{4^n}{n^{3/2}\sqrt{\pi}}$

$O(n) = \frac{4^n}{n^{3/2}}$

3.4 Space complexity

其空间复杂度和时间复杂度等价。

$O(n) = \frac{4^n}{n^{3/2}}$

4. N-sum (EEEnd)

4.1 Modeling

将B数组构造 $F(x,1) = \sum x^{b[i]}$
$F(x,n) = F(x,1)^n$
$ans = F(x,n)$中x^m的系数。
可以用NTT优化 + NTT后将x前的系数变为$bool$值

4.2 Algorithm description

初始化$G(x) = 1$ 从高到低遍历n的二进制

为1时$G(x) = G(x) * G(x) * F(x,1)$;
为0时$G(x) = G(x) * G(x)$

```
G(x) = 1, F(x, 1) = init
for i in range (high_position, 0):
    G(x) = NTT(G(x), G(x))
    if(n>>i&1)
        G(x) = NTT(G(x), F(x, 1))
    for i in G(x): # 改变G(x)每项的系数为bool值。
        G(x)[i] = G(x)[i] > 0
```

4.3 Time complexity

n次多项式相乘NTT时间复杂度：$O(n\log n)$

$$
\begin{aligned}
T(n) &\leqslant \sum_{i=0}^{high\_position} 2(\texttt{n>>i}) * n\log(\texttt{n>>i}) \\
&\leqslant 2n\log(n) \sum_{i=0}^{high\_position} \texttt{n>>i} \\
&\leqslant 2n\log(n) * (2n)
\end{aligned}
$$

$\therefore O(n) = n^2\log(n)$

4.4 Space complexity

其中间暂存结果$O(n^2)$

5. Ex. Unary Cubic Equation (EEEnd)

5.1 Modeling

如果有三个解的话那么 可以通过导数求零点 找出驻点，驻点之间可以二分出一个解，两个驻点到两侧分别也能找到一个解。

5.2 Algorithm description

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

#define eps (1e-10)
double a, b, c, d;

double calc(double y) { return ((a * y + b) * y + c) * y + d; } // 获取 函数值
bool sig(double a, double b){ return !((a >= 0) ^ (b >= 0)); } // 判断 a 和 b 是否同号
std::pair<double, double> getss(double a, double b, double c) // 获取导数的根。
{
    double f = std::sqrt(b * b - 4 * a * c);
    return {(-b - f) / (2 * a), (-b + f) / (2 * a)};
}

double getvalue(double l, double r) // 在l-r之间二分找到根
{
    while (r - l > eps)
    {
        double mid = (l + r) / 2;
        double value = calc(mid);
        if (sig(calc(l), value)) l = mid;
        else if (sig(value, calc(r)))  r = mid;
        else std::cout << "error \n";
    }
    return l;
}
int main()
{
    while (true)
    {
        std::cin >> a >> b >> c >> d;
        double l = -1e9, r = 1e9;
        auto [x1, x2] = getss(3 * a, 2 * b, c);
        std::cout << (3 * a * x1 * x1 + 2 * b * x1 + c) << " " << (3 * a * x2 * x2 + 2 * b * x2 + c) << "\n"; // 验证 导数零点是否求解正确。
        auto y1 = getvalue(l, x1), y2 = getvalue(x1, x2), y3 = getvalue(x2, r);
        printf("x     : %.10lf   %.10lf   %.10lf\n", y1, y2, y3);                    // 输出根
        printf("value : %.10lf   %.10lf   %.10lf\n", calc(y1), calc(y2), calc(y3)); // 验证根是否是根
        printf(" ------- \n");
    }
    return 0;
}

/*
(x-1)(x-2)(x-3)
x^3 - 6x^2 + 11x - 6
(x - 10) * (x - 1) * (x - 2)
x**3 - 13*x**2 + 32*x - 20
(x - 100.113)*(x - 100)*(x - 1.111)
x**3 - 201.224*x**2 + 10233.625543*x - 11122.5543
(x - 3000.113)*(x - 100)*(x - 1.111)
x**3 - 3101.224*x**2 + 303455.525543*x - 333312.5543
输入:
1 -6 11 -6
1 -13 32 -20
```

```
1.00000000000000 -201.224000000000 10233.6255430000 -11122.5543000000
1.00000000000000 -3101.22400000000 303455.525543000 -333312.554300000
输出
1 2 3
1 2 10
1.111 100 100.113
1.111 100 3000.113
*/
```

5.3 Time complexity

时间复杂度 : $O(\log n)$

5.4 Space complexity

空间复杂度：$O(1)$

6. Ex. Distance

6.1 Modeling

首先对 $arr2$ 使用归并排序，再遍历 $arr1$ 的每个元素，寻找 $arr2$ 中第一个比其大的元素，然后判断条件。

6.2 Algorithm description

```
arr1, arr2 = [], []
# 归并排序
merge_sort(arr2)

ans = 0
for i in range(n):
    pos = arr2.lower_bound(arr1[i])
    if (pos == n or arr2[pos] - arr1[i] > d) and (pos == 0 or arr1[i] - arr2[pos-1] > d):
        ans ++
```

6.3 Time complexity

6.4 Space complexity