突击

1. Dev-C++设置

1. 打开调试;
2. 设置为：`-std=c++11`
3. 基础模版

```
#include<bits/stdc++.h>
#define endl '\n'
#define ll long long
#define INF 0x3f3f3f3f
#define INL -0x3f3f3f3f
#define PII pair<int, int>
using namespace std;

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

    cout << "hello" << endl;
    return 0;
}
```

2. 二分

- 升序序列查找下标最小 `>= x` 的元素

```
int bin_search(int x, int l, int r) {
    while(l < r) {
        int mid = (l + r) / 2;
        if(x <= a[mid])
            r = mid;
        else
            l = mid + 1;
    }
    return l;
}
```

- 升序可重序列查找下标 `> x` 的元素

```
int bin_search(int x, int l, int r) {
    while(l < r) {
        int mid = (l + r) / 2;
        if(x < a[mid])
            r = mid;
        else
            l = mid + 1;
    }
    return l;
}
```

- 升序可重序列查找下标最大 `< x` 的元素

```
int bin_search(int x, int l, int r) {
    while(l < r) {
        int mid = (l + r) / 2;
        if(a[mid + 1] < x)
            l = mid + 1;
        else
            r = mid;
    }
    return l;
}
```

- 升序可重序列查找下标最大 `<= x` 的元素

```
int find(int x, int l, int r) {
    while(l < r) {
        int mid = (l + r) / 2;
        if(a[mid + 1] <= x)
            l = mid + 1;
        else
            r = mid;
    }
    return l;
}
```

- 浮点二分

```
double bin_search(double x, double l, double r) {
    while(r - l > 1e-5) {
        double mid  = (l + r) / 2;
        if(check(mid))
            l = mid;
        else
            r = mid;
    }
    return l;
}
```

STL的使用

- 两个函数默认在非降序条件下查找 **下标最小** 的 $\geq x$ 的元素下标 和 $> x$ 的元素下标。
- `lower_bound()`：查找第一个 `>= x` 的元素下标
- `upper_bound()`：查找第一个 `> x` 的元素下标

```
vector<int> a(n);
int lo = lower_bound(a + 1, a + n + 1, x) - a;
int hi = upper_bound(a + 1, a + n + 1, x) - a;

vector<int> b(n);
int lo = lower_bound(b.begin(), b.end(), x) - b.begin();
int hi = upper_bound(b.begin(), b.end(), x) - b.begin();
```

- a中元素 x 出现的次数 $O(logn)$

```
int cnt = upper_bound(b.begin(), b.end(), x) - lower_bound(b.begin(), b.end(), x);
```

3. 前缀和

```
// 预处理
s[1] = a[1];
for(int i = 2; i <= n; ++i)
    s[i] = s[i - 1] + a[i];

// 利用前缀和查询（询问）区间和 O(1)
long long calc(int l, int r) {
    return l == 1 ? s[r] : s[r] - s[r] - s[l - 1];
}
```

4. 差分

```
c[1] = a[1];
for(int i = 1; i <= n; ++i)
    c[i] = a[i] - a[i - 1];
```

- 原序列上区间 $[l, r]$ 修改（区间加/减）相当于差分序列上两个单点修改 $O(1)$

```
c[l] += v;
c[r + 1] -= v;
```

> 区间修改思路：

```
最少操作次数：max{ 正数绝对值之和 , 负数绝对值之和 }
最终数列种数：abs{ 正数绝对值之和 - 负数绝对值之和 } + 1
```

5. 最大公约数

```
int gcb(int a, int b) {
   return b == 0 ? a : gcb(b, a % b);
}
```

6. 最小公倍数

```
int gcb(int a, int b) {
   return b == 0 ? a : gcb(b, a % b);
}
int lcm = a * b / gcb(a, b);
```

7. 判断n是否为素数

```
bool is_prime(int n) {
    if(n <= 1)
        return false;
    for(int i = 2; i <= sqrt(n); i++) {
        if(n % i == 0)
            return false;
    }
    return true;
}
```

8. 快速幂

```
int qmi(int m, int k, int p) {
    int res = 1 % p, t = m;
    while (k) {
        if (k&1)
            res = res * t % p;
        t = t * t % p;
        k >>= 1;
    }
    return res;
}
```

9.字符串查找

```
int strfind(char str[],char key[]) {
    int l1, l2, i, j, flag;
    l1 = strlen(str);
    l2 = strlen(key);
    for(i = 0; i <= l1 - l2; i++) {
        flag = 1;
        for(j = 0; j < l2; j++)
            if(str[i + j] != key[j]) {
                flag = 0;
                break;
            }
        if(flag)
            return i;
    }
    return -1;
}
```

10. 任意进制转换

```
void conversion(char s[],char s2[],long d1,long d2) {
    long i, j, t, num;
    char c;
    num = 0;
    for(i = 0; s[i] != '\0'; i++) {
        if(s[i] <= '9' && s[i] >= '0')
            t = s[i] - '0';
        else
            t = s[i] - 'A' + 10;
        num = num * d1 + t;
    }
    i = 0;

    while(1) {
```

```
        t = num % d2;
        if(t <= 9)
            s2[i] = t + '0';
        else
            s2[i] = t + 'A' - 10;
        num /= d2;
        if(num == 0)
            break;
        i++;
    }
    for(j = 0; j < i / 2; j++) {
        c = s2[j];
        s2[j] = s[i - j];
        s2[i - j] = c;
    }
    s2[i + 1] = '\0';
}
```