进制转换

10 (0-9) ----> 26（A-Z）

```
string convertToTitle(int columnNumber) {
        string result = "";
        while (columnNumber > 0) {
            int remainder = (columnNumber - 1) % 26;
            result += ('A' + remainder);
            columnNumber = (columnNumber - 1) / 26;
        }
        reverse(result.begin(), result.end());
        return result;
    }
```

双指针

```
int removeDuplicates(vector<int>& nums) {
        int n = nums.size();
        if(n == 0) return 0;
        int j = 0;
        for(int i = 0; i < n; i++){
            if(nums[j] != nums[i]){
                nums[++j] = nums[i];
            }
        }
        return j + 1;
    }
```

std::vector

最大值：int maxitem = *max_element(vec.begin(), vec.end()); // 注意返回的是指针需要加 “ * ” 累加：int sumA = accumulate(vec.begin(), vec.end(), 0); // 从 0 开始累加

std::string

大小：string str; int length = str.size(); 翻转：string str; reverse(str.begin(), str.end()); 即为反转后的字符串 比较：== < > 看字典序 子串：string str = s.substr(size_t pos = 0, size_t count = npos); 查找： // 查找子串 size_t p = find(const string& str, size_t pos = 0); // 查找字符 size_t p = find(char ch, size_t pos = 0);

差分数组（区间优化）

位置: |0 1 2 3 4 5 6 7| 差分数组: |0 0 0 0 0 0 0 0| 操作后: |0 0 1 0 0 0 -1 0| 我们只需要做两次操作：

在起点 a=2 处 +1：diff[2] += 1 在终点后一位 b+1=6 处 -1：diff[6] -= 1 然后通过前缀和还原原数组：

前缀和计算过程: 位置: 0 1 2 3 4 5 6 7 差分: 0 0 1 0 0 0 -1 0 前缀和:0 0 1 1 1 1 0 0 ← 这就是我们想要的结果

哈希表优化 O(N) 遍历

两数和等于一个指定的目标值 target

```
unordered_map<int, int> hashtable;
    for (int i = 0; i < nums.size(); ++i) {
        auto it = hashtable.find(target - nums[i]);
        if (it != hashtable.end()) {
            return {it->second, i};
        }
        hashtable[nums[i]] = i;
    }
return {};
```

动态规划

最大子序列和 贪心：

```
int maxSubArray(vector<int>& nums) {
        int pre=0,maxs=nums[0];
        for(int i=0;i<nums.size();i++){
            pre = max(pre+nums[i],nums[i]);
            maxs = max(pre, maxs);

        }
        return maxs;
    }
```

动规：

```
int maxSubArray(vector<int>& nums) {
        int n = nums.size();
        for(int i=1;i<n;i++){
            if(nums[i-1]>0) nums[i]+=nums[i-1];
        }
        return *max_element(nums.begin(),nums.end());
    }
```