1184. 公交站间的距离（简单）

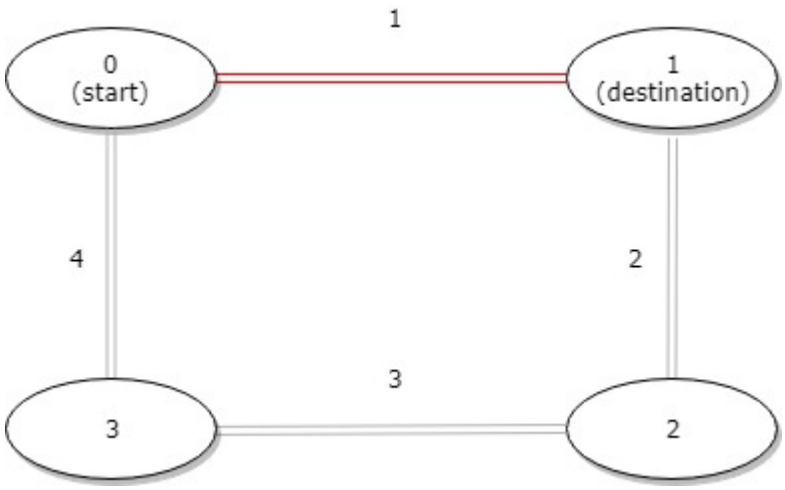
1. 题目描述

环形公交路线上有 `n` 个站，按次序从 `0` 到 `n - 1` 进行编号。我们已知每一对相邻公交站之间的距离，`distance[i]` 表示编号为 `i` 的车站和编号为 `(i + 1) % n` 的车站之间的距离。

环线上的公交车都可以按顺时针和逆时针的方向行驶。

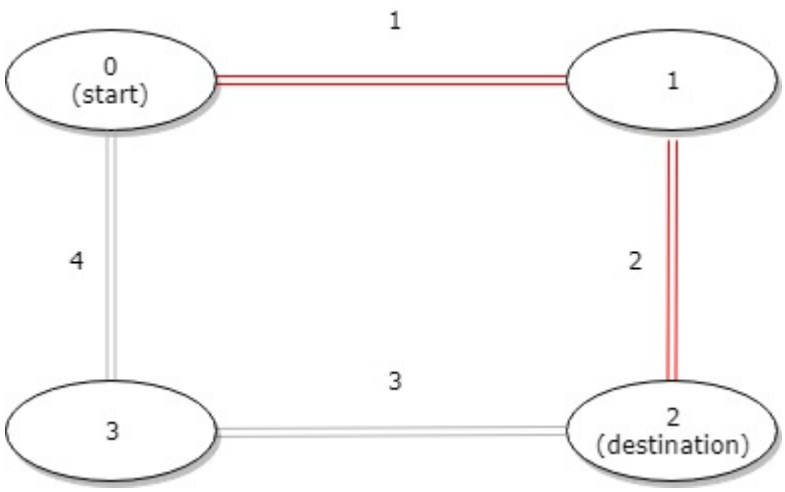
返回乘客从出发点 `start` 到目的地 `destination` 之间的最短距离。

示例 1：



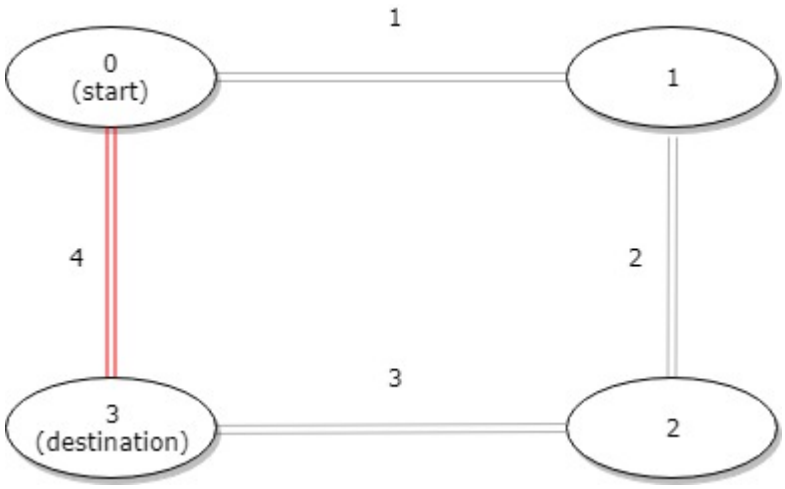

```
输入：distance = [1,2,3,4], start = 0, destination = 1
输出：1
解释：公交站 0 和 1 之间的距离是 1 或 9，最小值是 1。
```

示例 2：




```
输入：distance = [1,2,3,4], start = 0, destination = 2
输出：3
解释：公交站 0 和 2 之间的距离是 3 或 7，最小值是 3。
```

示例 3：




```
输入：distance = [1,2,3,4], start = 0, destination = 3
输出：4
解释：公交站 0 和 3 之间的距离是 6 或 4，最小值是 4。
```

提示：

- `1 <= n <= 10^4`
- `distance.length == n`
- `0 <= start, destination < n`
- `0 <= distance[i] <= 10^4`

2. 简单实现

```cpp
class Solution {
public:
    int distanceBetweenBusStops(vector<int>& distance, int start, int destination)
{
        int n = distance.size();
        int ans1 = 0;//顺时针start->destination
        int cur = start;
        while(cur != destination){
            ans1 += distance[cur];
            cur = (cur + 1) % n;
        }
        int ans2 = 0;//顺时针destination->start，相当于逆时针start->destinatio
        cur = destination;
        while(cur != start){
            ans2 += distance[cur];
            cur = (cur + 1) % n;
        }
        return min(ans1, ans2);
    }
};
```

1185. 一周中的第几天（简单）

1. 题目描述

给你一个日期，请你设计一个算法来判断它是对应一周中的哪一天。

输入为三个整数：`day`、`month` 和 `year`，分别表示日、月、年。

您返回的结果必须是这几个值中的一个 `{"Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"}`。

示例 1：

```
输入：day = 31, month = 8, year = 2019
输出："Saturday"
```

示例 2：

```
输入：day = 18, month = 7, year = 1999
输出："Sunday"
```

示例 3：

```
输入：day = 15, month = 8, year = 1993
输出："Sunday"
```

提示：

- 给出的日期一定是在 `1971` 到 `2100` 年之间的有效日期。

2. 简单实现

计1971.1.1为第1天，计算当前日期对应的天数再换算为周几即可

```
class Solution {
public:
    vector<int> month_cnt = {0, 31, 59, 90, 120, 151, 181, 212, 243, 273, 304, 334};//前i月的总天数
    vector<string> ansstr = {"Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"};
    bool isRunnian(int y){//判断是否闰年
        return (y%4 == 0 && y%100 != 0) || y%400 == 0;
    }
    string dayOfTheWeek(int day, int month, int year) {
        int runyear = 0;//闰年数量
        for(int y = 1971; y < year; y++)
            if(isRunnian(y)) runyear++;
        int days = (year-1971-runyear)*365 + runyear*366;//1971~year-1年的整年天数
        days += month_cnt[month-1] + day;//整月天数+剩余的月内天数
        if(month > 2 && isRunnian(year))//当前是闰年且超过二月，要加上2月多出来的那一天
            days++;
        int ans = (4 + days) % 7;
        return ansstr[ans];
    }
};
```

1186. 删除一次得到子数组最大和（中等）

1. 题目描述

   给你一个整数数组，返回它的某个 **非空** 子数组（连续元素）在执行一次可选的删除操作后，所能得到的最大元素总和。

   换句话说，你可以从原数组中选出一个子数组，并可以决定要不要从中删除一个元素（只能删一次哦），（删除后）子数组中至少应当有一个元素，然后该子数组（剩下）的元素总和是所有子数组之中最大的。

   注意，删除一个元素后，子数组 **不能为空**。

   请看示例：

   示例 1：

   ```
   输入：arr = [1,-2,0,3]
   输出：4
   解释：我们可以选出 [1, -2, 0, 3]，然后删掉 -2，这样得到 [1, 0, 3]，和最大。
   ```

   示例 2：

   ```
   输入：arr = [1,-2,-2,3]
   输出：3
   解释：我们直接选出 [3]，这就是最大和。
   ```

   示例 3：

   ```
   输入：arr = [-1,-1,-1,-1]
   输出：-1
   解释：最后得到的子数组不能为空，所以我们不能选择 [-1] 并从中删去 -1 来得到 0。
        我们应该直接选择 [-1]，或者选择 [-1, -1] 再从中删去一个 -1。
   ```

   提示：

   - `1 <= arr.length <= 10^5`
   - `-10^4 <= arr[i] <= 10^4`

2. 正确解法——动态规划

   `dp[i：以下标i结尾的序列][0或1：0表示序列没有进行删减，1表示序列进行过删减]` `dp[i][0] = max(arr[i], dp[i-1][0] + arr[i])` `dp[i][1] = max(dp[i-1][0], dp[i-1][1] + arr[i])` 可以发现每一个状态都只和上一轮更新的状态有关，因此可以直接用两个变量代替，空间复杂度从O(n)变为O(1)。

   ```cpp
   class Solution {
   public:

       int maximumSum(vector<int>& arr) {
           int n = arr.size();
           int s0, s1, ans;
           s0 = ans = arr[0], s1 = -10002;
           for (int i = 1; i < n; i++) {
               int ts0 = max(arr[i], s0 + arr[i]);
   ```

```
                int ts1 = max(s0, s1 + arr[i]);
                s0 = ts0;
                s1 = ts1;
                ans = max(ans, max(s0, s1));
            }
            return ans;
        }
    };
```

1187. 使数组严格递增（困难）

1. 题目描述

   给你两个整数数组 `arr1` 和 `arr2`，返回使 `arr1` 严格递增所需要的最小「操作」数（可能为 0）。

   每一步「操作」中，你可以分别从 `arr1` 和 `arr2` 中各选出一个索引，分别为 `i` 和 `j`，`0 <= i < arr1.length` 和 `0 <= j < arr2.length`，然后进行赋值运算 `arr1[i] = arr2[j]`。

   如果无法让 `arr1` 严格递增，请返回 `-1`。

   示例 1：

   ```
   输入：arr1 = [1,5,3,6,7], arr2 = [1,3,2,4]
   输出：1
   解释：用 2 来替换 5，之后 arr1 = [1, 2, 3, 6, 7]。
   ```

   示例 2：

   ```
   输入：arr1 = [1,5,3,6,7], arr2 = [4,3,1]
   输出：2
   解释：用 3 来替换 5，然后用 4 来替换 3，得到 arr1 = [1, 3, 4, 6, 7]。
   ```

   示例 3：

   ```
   输入：arr1 = [1,5,3,6,7], arr2 = [1,6,3,3]
   输出：-1
   解释：无法使 arr1 严格递增。
   ```

   提示：

   - `1 <= arr1.length, arr2.length <= 2000`
   - `0 <= arr1[i], arr2[i] <= 10^9`

2. 正确解法

国际版解法参考

1. `dp[i][j]` 表示，将数组 `arr1` 的前 `j` 个元素通过 `i` 次替换后变为严格递增序列时，序列中最后一个元素的最小值,第 `j` 个元素的最小值。
2. 求 `dp[i][j+1]` 时，递推如下：当 `arr[j+1] > dp[i][j]` 时，即前 `j` 个元素已经严格递增，这时 `arr[j+1]` 大于严格递增序列的最大值时， `arr[j+1]` ,直接加在序列末尾，即这时的序列应该严格递增，且此时的序列进行替换的次数仍然为 `i` 次。
3. 另一种选择，或者选择将 `arr[j+1]` 进行元素替换，此时，我们应当在数组 `arr2` 中找到第一个比 `dp[i-1][j]` 大的数， `dp[i-1][j]` 即前 `j` 个元素进行 `i-1` 替换后的序列的最大值，我们使用二分查找即可在 `O(lgn)` 时间复杂度内找到该值,此时我们仍然保证前 `j+1` 个元素进行了 `i` 次替换。
4. 递推公式如下:

$$dp[i][j+1] = min \begin{cases} arr1[j+1] & if\ arr1[j+1] > dp[i][j] \\ arr2.upperbound(dp[i-1][j]) & if\ arr2.upperbound(dp[i-1][j])\ exis \end{cases}$$

5. 我们只需要找到最小转换次数的 `i` ，能够使得满足前 `n` 个元素成为严格递增序列即可。

```
class Solution {
public:
    int makeArrayIncreasing(vector<int>& arr1, vector<int>& arr2) {
        int n = arr1.size();
        std::vector<vector<int>> dp(n+1, vector<int>(n+1, INT_MAX));

        dp[0][0] = -1;
        sort(arr2.begin(),arr2.end());

        for(int j = 1; j <= n; ++j){//为了处理第一个元素，j从1开始
            for(int i = 0; i <= j; ++i){//改0~j次
                if(arr1[j-1] > dp[i][j-1])
                    dp[i][j] = arr1[j-1];
                if(i > 0){
                    auto it = upper_bound(arr2.begin(),arr2.end(),dp[i-1][j-1]);
                    if(it != arr2.end())
                        dp[i][j] = min(dp[i][j], *it);
                }
                if( j == n && dp[i][j] != INT_MAX)
                    return i;
            }
        }
        return -1;
    }
};
```