# E. Коровы в стойла

The code reads input from the user.

It reads the number of fences (fences) and the number of cows (cows).

It reads the positions of the fences as a string and splits it into a vector of strings.

The vector of strings is converted to a vector of integers using the turn function.

The code uses binary search to find the maximum possible minimum distance between cows (right).

It sets the left boundary (left) to 0 and the right boundary to the distance between the last fence and the leftmost position.

In each iteration, it calculates the midpoint (mid) and checks if it is possible to place cows number of cows with a minimum distance of mid using the check function.

If it is possible, it updates the left boundary to mid + 1 since larger distances can still be valid.

If it is not possible, it updates the right boundary to mid - 1 since smaller distances cannot be valid.

Finally, it prints the maximum possible minimum distance between cows (right), which represents the optimal solution.

Code：

#include <iostream>

#include <vector>

using namespace std;

vector<int> positions;

vector<string> split(string s, string pattern){

vector<string> result;

while(s.find(" ")!=string::npos){

int p = s.find(" ");

result.emplace\_back(s.substr(0, p));

s = s.substr(p+1, s.size());

}

result.emplace\_back(s);

return result;

}

bool check(int mid, int fences, int cows){

int c = 1;

int d = 0;

for(int i = 1; i < fences; i++){

if(positions[i] - positions[d] >= mid){

d = i;

c++;

}

}

if(c < cows){

return false;

}else{

return true;

}

}

vector<int> turn(vector<string> v){

vector<int> result;

for(string i: v){

result.emplace\_back(stoi(i));

}

return result;

}

int main(){

string input;

getline(cin,input);

int cows;

int fences;

size\_t position = input.find(" ",0);

string sub1 = input.substr(0,position);

string sub2 = input.substr(position+1,input.size()-position-1);

fences = stoi(sub1);

cows = stoi(sub2);

getline(cin, input);

positions = turn(split(input," "));

int left = 0;

int right = positions.back() - left;

int mid;

while(left <= right){

mid = (left+right)/2;

if(check(mid, fences, cows)){

left = mid + 1;

}else{

right = mid - 1;

}

}

cout<<right<<endl;

return 0;

}

# Число

The code reads numbers from the user until there is no more input.

It stores each input number in the nums array.

If the number of elements in nums reaches the current capacity (n == capacity), the array is full and needs to be resized.

The capacity is doubled, and a new array new\_nums is created with the new capacity.

The elements from the old array nums are copied to the new array.

The old array nums is deleted to free the memory.

The nums pointer is updated to point to the new array new\_nums.

The sort function is used to sort the nums array.

A lambda function is provided as the comparison function to determine the order of elements during sorting.

The lambda function compares two strings a and b by concatenating them in two different ways (a + b and b + a).

The comparison is based on which concatenation results in a larger string. This ensures that the larger numbers appear earlier in the sorted array.

Code：

#include <iostream>

#include <string>

#include <algorithm>

using namespace std;

int main() {

int capacity = 10; // 数组初始容量为10

string \*nums = new string[capacity]; // 动态分配空间

int n = 0; // 当前已读入数字的数量

string num;

while (cin >> num) {

nums[n++] = num;

if (n == capacity) {

// 数组已满，需要扩容

capacity \*= 2;

string \*new\_nums = new string[capacity];

for (int i = 0; i < n; i++) {

new\_nums[i] = nums[i];

}

delete[] nums;

nums = new\_nums;

}

}

sort(nums, nums + n, [](string a, string b) {

return a + b > b + a;

});

string ans = "";

for (int i = 0; i < n; i++) {

ans += nums[i];

}

cout << ans << endl;

}

# G. Кошмар в замке

Define several global variables, including str to store the input string, freq to store the character frequencies, idx to store the character indices, values to store the sorted character frequencies, and some strings to store the parts of the result.

Implement a quicksort function, quick\_sort\_arr, to sort the values array representing the character frequencies. Additionally, adjust the corresponding elements in the idx and freq arrays accordingly.

Create a function, build\_result, to construct the result string based on the character frequencies stored in the freq array. According to the defined rules, characters with frequencies greater than 1 are placed on both sides of the result string, while characters with frequencies greater than 2 are placed in the middle.

In the main function, read the input string and the character frequency array. Then, apply the quick\_sort\_arr function to sort the character frequency array. Finally, call the build\_result function to construct the result string and output the result.

Code：

#include <iostream>

#include <vector>

#include <algorithm>

#include <map>

using namespace std;

string str; // 存储输入的字符串

int freq[26] = {0}; // 存储字符频率的数组

char idx[26] = {'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z'}; // 存储字符索引的数组

vector<int> values; // 存储排序后字符频率的数组

string left\_part = ""; // 左半部分

string mid\_part = ""; // 中间部分

string right\_part = ""; // 右半部分

string result\_part = ""; // 结果字符串

// 根据指定分隔符分割字符串

vector<string> split\_string(string s\_1,const string pattern){

s\_1 = s\_1 + pattern;

size\_t pos = s\_1.find(pattern, 0);

vector<string> result;

string sub1;

do {

sub1 = s\_1.substr(0,pos);

string temp;

temp = s\_1.substr(pos+1,s\_1.size()-pos-1);

s\_1 = temp;

pos = s\_1.find(pattern, 0);

result.emplace\_back(sub1);

} while(pos != s\_1.npos);

return result;

}

// 快速排序

void quick\_sort\_arr(int left, int right){

if(left >= right) return;

int pivot = values[left]; // pivot值是当前数组中的第一个值

int save\_started = left; // 保存数组开始的位置

int save\_end = right; // 保存数组结束的位置

char pivot\_char = idx[left]; // 当前字符的索引值

int pivot\_freq = freq[left]; // 当前字符的出现频率

do {

// 找到第一个比pivot大的值

while(left < right && values[right] < pivot){

right--;

}

if(left < right){

values[left] = values[right];

idx[left] = idx[right];

freq[left] = freq[right];

}

// 找到第一个比pivot小的值

while(left < right && values[left] >= pivot){

left++;

}

if(left < right){

values[right] = values[left];

idx[right] = idx[left];

freq[right] = freq[left];

}

} while(left < right);

// 把pivot放到中间

values[left] = pivot;

idx[left] = pivot\_char;

freq[left] = pivot\_freq;

// 递归排序左右两边的数组

quick\_sort\_arr(save\_started, left - 1);

quick\_sort\_arr(left+1, save\_end);

}

// 构建结果字符串

void build\_result(){

for(int i = 0 ; i < 26 ; i++){

// 当字符出现次数大于1时

if(freq[i] > 1){

right\_part = string(1, idx[i]) + right\_part;

left\_part = left\_part + string(1, idx[i]);

if(freq[i] > 2){

mid\_part = mid\_part + string(freq[i] - 2, idx[i]);

}

}else if( freq[i] == 1){

mid\_part = mid\_part + string(1, idx[i]);

}

}

result\_part = left\_part + mid\_part + right\_part;

}

int main(){

cin >> str;

for(int i = 0 ; i < 26;i++){

int element;

cin >> element;

values.push\_back(element);

}

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

freq[str[i] - 'a']++;

}

quick\_sort\_arr(0, 25);

build\_result();

cout << result\_part << endl;

return 0;

}

# H. Магазин

Read item quantity and discount interval from stdin.

Create an integer array prices to store the price of each item.

Using a loop, read the price of each item from standard input and store it in the prices array.

Use the sort function to sort the prices array so that the items are ordered by price from largest to smallest.

Initialize a variable total to 0 to store the calculated total price.

Use a loop to iterate over the price of each item:

If the current item is not a discounted item (i.e. not a multiple of discountEvery minus one), add its price to the total variable.

Print the total price total on standard output.

Returns 0, indicating that the program ended successfully.

To sum up, the idea of this algorithm is to sort the commodity prices first, then calculate the total price according to the discount rules, and output the result to the standard output.

Code：

#include <iostream>

#include <algorithm>

using namespace std;

int main() {

int n, discountEvery;

cin >> n >> discountEvery; // 输入商品数量和打折间隔

int prices[n];

for (int i = 0; i < n; i++) {

cin >> prices[i]; // 输入每个商品的价格

}

// 将商品价格从大到小排序

sort(prices, prices + n, greater<int>());

int total = 0;

for (int i = 0; i < n; i++) {

if (i % discountEvery != discountEvery - 1) { // 如果不是打折的商品，加上价格

total += prices[i];

}

}

cout << total << endl; // 输出总价格

return 0;

}