ĐẠI HỌC BÁCH KHOA HÀ NỘI

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

——————— * ———————

ĐỒ ÁN 1

LẬP TRÌNH

BỘ MÔN KỸ THUẬT MÁY TÍNH

Sinh viên thực hiện: Vương Chí Sơn
MSSV: 20156407
Lớp CN CNTT 3 - k60
Giảng viên hướng dẫn: ThS Nguyễn Đình Thuận

HÀ NỘI 3-2017

Mục lục

Level 1

1. add

A function that adds two numbers.

Example

For `param1 = 1` and `param2 = 2`, the output should be
`add(param1, param2) = 3`.

Input/Output

- **[time limit] 500ms (cpp)**
- **[input] integer param1**

  Constraints:
  `-100 ≤ param1 ≤ 1000`.
- **[input] integer param2**

  Constraints:
  `-100 ≤ param2 ≤ 1000`.
- **[output] integer**

  The sum of the two inputs.

```
int add(int param1, int param2) {

return param1+param2;

}
```

2. centuryFromYear

Given a year, return the century it is in. The first century spans from the year 1 up to and including the year 100, the second - from the year 101 up to and including the year 200, etc.

Example

- For `year = 1905`, the output should be
  `centuryFromYear(year) = 20`;

- For `year = 1700`, the output should be `centuryFromYear(year) = 17`.

Input/Output

- **[time limit] 500ms (cpp)**
- **[input] integer year**

  A positive integer, designating the year.

  Constraints:
  `1 ≤ year ≤ 2005`.
- **[output] integer**

  The number of the century the year is in.

```
int centuryFromYear(int year) {

    return (year-1)/100+1;

}
```

3. checkPalindrome

Given the string, check if it is a palindrome.

Example

- For `inputString = "aabaa"`, the output should be `checkPalindrome(inputString) = true`;
- For `inputString = "abac"`, the output should be `checkPalindrome(inputString) = false`.

Input/Output

- **[time limit] 500ms (cpp)**
- **[input] string inputString**

  A non-empty string consisting of lowercase characters.

  Constraints:
  `1 ≤ inputString.length ≤ 10`.

- **[output] boolean**

  `true` if `inputString` is a palindrome, `false`otherwise.

```
bool checkPalindrome(std::string inputString) {
    int l=inputString.length();
    for(int i=0;i<l/2;i++)
    {
        if (inputString[i] != inputString[l-1-i]) return false;
    }
    return true;
}
```

Level 2

1. adjacentElementsProduct

Given an array of integers, find the pair of adjacent elements that has the largest product and return that product.

Example

For `inputArray = [3, 6, -2, -5, 7, 3]`, the output should be
`adjacentElementsProduct(inputArray) = 21`.
`7` and `3` produce the largest product.
Input/Output

- **[time limit] 500ms (cpp)**
- **[input] array.integer inputArray**

  An array of integers containing at least two elements.

  Constraints:
  `3 ≤ inputArray.length ≤ 10`,
  `-50 ≤ inputArray[i] ≤ 1000`.

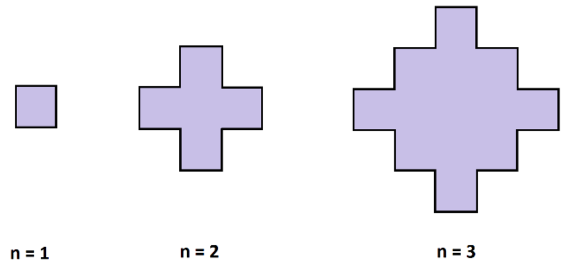
- **[output] integer**

  The largest product of adjacent elements.

```
int adjacentElementsProduct(std::vector<int> inputArray) {

    int n=inputArray.size();

    int max=inputArray[0]*inputArray[1];
    printf("%d",n);

    for(int i=1; i<n-1;i++)

    {

        max=(inputArray[i]*inputArray[i+1]<max) ? max :
inputArray[i]*inputArray[i+1];

    }

    return max;

}
```

2. shapeArea

Below we will define what and `n`-interesting polygon is and your task is to find its area for a given `n`.
A `1`-interesting polygon is just a square with a side of length `1`. An `n`-interesting polygon is obtained by taking the `n - 1`-interesting polygon and appending `1`-interesting polygons to its rim side by side. You can see the `1`-, `2`- and `3`-interesting polygons in the picture below.



Example

- For `n = 2`, the output should be `shapeArea(n) = 5`;
- For `n = 3`, the output should be `shapeArea(n) = 13`.

Input/Output

- **[time limit] 500ms (cpp)**
- **[input] integer n**

Constraints:
`1 ≤ n < 10^4.`

- **[output] integer**

  The area of the `n`-interesting polygon.

```
int shapeArea(int n) {

      return 2*n*(n-1)+1;

}
```

3. Make Array Consecuive 2

Ratiorg got `statues` of *different* sizes as a present from CodeMaster for his birthday, each statue having an non-negative integer size. Since he likes to make things perfect, he wants to arrange them from smallest to largest so that each statue will be bigger than the previous one exactly by `1`. He may need some additional statues to be able to accomplish that. Help him figure out the minimum number of additional statues needed.
Example

For `statues = [6, 2, 3, 8]`, the output should be
`makeArrayConsecutive2(statues) = 3`.
Ratiorg needs statues of sizes `4`, `5` and `7`.
Input/Output

- **[time limit] 500ms (cpp)**

- **[input] array.integer statues**

  An array of *distinct* non-negative integers.

  Constraints:
  `1 ≤ statues.length ≤ 10`,
  `0 ≤ statues[i] ≤ 20`.

- **[output] integer**

  The minimal number of statues that need to be added to existing `statues` such that it contains every integer size from an interval `[L, R]` (for some `L, R`) and no other sizes.

```
int makeArrayConsecutive2(std::vector<int> statues) {

      int all=statues.size();

      int max=0, min=20;
```

```
        for(int i=0; i<all; i++)
        {
              max = max > statues[i] ? max : statues[i];
              min = min < statues[i] ? min : statues[i];
        }
        return max-min+1-all;
}
```

4. almostIncreasingSequence

Given a sequence of integers, check whether it is possible to obtain a strictly increasing sequence by erasing no more than one element from it.

Example

- For `sequence = [1, 3, 2, 1]`, the output should be `almostIncreasingSequence(sequence) = false`;
- For `sequence = [1, 3, 2]`, the output should be `almostIncreasingSequence(sequence) = true`.

Input/Output

- **[time limit] 500ms (cpp)**
- **[input] array.integer sequence**

  Constraints:
  `2 ≤ sequence.length ≤ 10`^{5},
  `-10`^{5} `≤ sequence[i] ≤ 10`^{5}.
- **[output] boolean**
- `true` if it is possible, `false` otherwise.

```
 bool almostIncreasingSequence(std::vector<int> sequence) {
    int n=sequence.size(),CountDec=0;
    for (int i=1; i< n;i++)
    {
   if (sequence[i]<=sequence[i-1])
```

```
        {
                CountDec++;
                if (CountDec>1) return false;
                if (i==1) continue;
         }
      }
      return true;
   }
```

5. matrixElementSum

After becoming famous, CodeBots decided to move to a new building and live together. The building is represented by a rectangular `matrix` of rooms, each cell containing an integer - the price of the room. Some rooms are *free* (their cost is `0`), but that's probably because they are haunted, so all the bots are afraid of them. That is why any room that is *free* or is located anywhere below a *free* room in the same column is not considered suitable for the bots.
Help the bots calculate the total price of all the rooms that are suitable for them.

Example

For

```
matrix = [[0, 1, 1, 2],
          [0, 5, 0, 0],
          [2, 0, 3, 3]]
```

the output should be
`matrixElementsSum(matrix) = 9.`

Here's the rooms matrix with unsuitable rooms marked with `'x'`:

```
[[x, 1, 1, 2],
 [x, 5, x, x],
 [x, x, x, x]]
```

Thus, the answer is `1 + 5 + 1 + 2 = 9`.
Input/Output

- **[time limit] 500ms (cpp)**
- **[input] array.array.integer matrix**

  2-dimensional array of integers representing a rectangular matrix of the building.

Constraints:
`1 ≤ matrix.length ≤ 5`,
`1 ≤ matrix[0].length ≤ 5`,
`0 ≤ matrix[i][j] ≤ 10`.

- **[output] integer**

```
int matrixElementsSum(std::vector<std::vector<int>> matrix) {
    int sum=0;
    int check[5];
    for(int i=0; i<5; i++) check[i]=1;
    for(int i=0; i<matrix.size(); i++)
        for(int j=0; j<matrix[0].size();j++ )
        {
            if (matrix[i][j]==0) check[j]=0;
            if (check[j]==1) sum+=matrix[i][j];
        }
    return sum;
}
```

Level 3

1. All Longest Strings

Given an array of strings, return another array containing all of its longest strings.

Example

For `inputArray = ["aba", "aa", "ad", "vcd", "aba"]`, the output should be `allLongestStrings(inputArray) = ["aba", "vcd", "aba"]`.

Input/Output

- **[time limit] 500ms (cpp)**
- **[input] array.string inputArray**

  A non-empty array.

Constraints:
`1 ≤ inputArray.length ≤ 10`,
`1 ≤ inputArray[i].length ≤ 10`.

- **[output] array.string**

  Array of the longest strings, stored in the same order as in the `inputArray`.

```
std::vector<std::string>
allLongestStrings(std::vector<std::string> inputArray) {

    vector <string> output;

    int MaxLeng=1;

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

    {

        if (inputArray[i].size()<MaxLeng) continue;

        else if (inputArray[i].size()==MaxLeng)
output.push_back(inputArray[i]);

        else {


            MaxLeng=inputArray[i].size();

            output.clear();

            output.push_back(inputArray[i]);

        }

    }


    return output;

}
```

2. commonCharacterCount

Given two strings, find the number of common characters between them.

Example

For `s1 = "aabcc"` and `s2 = "adcaa"`, the output should be
`commonCharacterCount(s1, s2) = 3`.
Strings have `3` common characters - `2`"a"s and `1` "c".
Input/Output

- **[time limit] 500ms (cpp)**
- **[input] string s1**

  A string consisting of lowercase latin letters `a-z`.
  Constraints:
  `3 ≤ s1.length ≤ 15`.
- **[input] string s2**

  A string consisting of lowercase latin letters `a-z`.
  Constraints:
  `4 ≤ s2.length ≤ 15`.
- **[output] integer**

```
int commonCharacterCount(std::string s1, std::string s2) {
    int count=0;
    for(int i=0; i<s1.size();i++)
        for(int j=0; j<s2.size();j++)
            if (s1[i]==s2[j]) {
                count++;
                s1[i]='0';
                s2[j]='1';
            }
   return count;
}
```

3. isLucky

Ticket numbers usually consist of an even number of digits. A ticket number is considered *lucky* if the sum of the first half of the digits is equal to the sum of the second half.

Given a ticket number `n`, determine if it's *lucky* or not.

Example

- For `n = 1230`, the output should be `isLucky(n) = true`;
- For `n = 239017`, the output should be `isLucky(n) = false`.

Input/Output

- **[time limit] 500ms (cpp)**
- **[input] integer n**

  A ticket number represented as a positive integer with an even number of digits.

  Constraints:
  $10 \le n < 10^6$.

- **[output] boolean**

  `true` if `n` is a lucky ticket number, `false`otherwise.

```
bool isLucky(int n) {
    int sum1=0, sum2=0,x;
    int k= int(log10(n)) + 1;
    for(int i=0;i<k/2;i++)
        {
            sum1+=n/(int(pow(10,k-1-i)));
            n=n%int(pow(10,k-1-i));
        }
    for(int i=k/2;i<k;i++)
        {
            sum2+=n/(int(pow(10,k-1-i))) ;
            n=n%int(pow(10,k-1-i));
        }
    if (sum1==sum2) return true;
    return false;
 }
```

4. Sort by Height

Some people are standing in a row in a park. There are trees between them which cannot be moved. Your task is to rearrange the people by their heights in a non-descending order without moving the trees.

Example

For `a = [-1, 150, 190, 170, -1, -1, 160, 180]`, the output should be `sortByHeight(a) = [-1, 150, 160, 170, -1, -1, 180, 190]`.

Input/Output

- **[time limit] 500ms (cpp)**
- **[input] array.integer a**

  If `a[i] = -1`, then the `i`th position is occupied by a tree. Otherwise `a[i]` is the height of a person standing in the `i`thposition.
  Constraints:
  `5 ≤ a.length ≤ 15`,
  `-1 ≤ a[i] ≤ 200`.

- **[output] array.integer**

  Sorted array `a` with all the trees untouched.

```
std::vector<int> sortByHeight(std::vector<int> a) {
    int temp;
    for(int i=0; i< a.size();i++)
        if(a[i]!=-1)
            for(int j=i+1;j<a.size();j++)
                if ((a[j]!=-1)&&(a[i]>a[j]))
                {
                   temp=a[i];
                    a[i]=a[j];
                    a[j]=temp;
                }
    return a;
}
```

5. reverseParentheses

You have a string `s` that consists of English letters, punctuation marks, whitespace characters, and brackets. It is guaranteed that the parentheses in `s` form a regular bracket sequence.
Your task is to reverse the strings contained in each pair of matching parenthesis, starting from the innermost pair. The results string should not contain any parentheses.

Example

For string `s = "a(bc)de"`, the output should be
`reverseParentheses(s) = "acbde"`.
Input/Output

- **[time limit] 500ms (cpp)**
- **[input] string s**

  A string consisting of English letters, punctuation marks, whitespace characters and brackets. It is guaranteed that parenthesis form a *regular bracket sequence*.

  Constraints:

  `5 ≤ x.length ≤ 55.`
- **[output] string**

```
std::string reverseParentheses(std::string s) {
    char outPut[56];
    int index=0,i=0;
    for(i=0;i<s.length();i++)
    {
       if(s[i]==')')
             {
                 int j=i,k=i;
                 while(s[j]!='(') j--;
                 s[i]='0'; s[j]='0';
                 while (k>j+1)
```

```
                  {
                      j++;k--;
                       int temp= s[k];
                       s[k]=s[j];
                       s[j]=temp;
                   }
            }
      }
     for(int i=0;i<s.length();i++ )
          if (s[i]!='0')
          {
              outPut[index]=s[i];
              index++;
          }
      outPut[index]='\0';
      return outPut;
}
```

Level 4

1. alternatingSum

Several people are standing in a row and need to be divided into two teams. The first person goes into *team 1*, the second goes into *team 2*, the third goes into *team 1* again, the fourth into *team 2*, and so on.

You are given an array of positive integers - the weights of the people. Return an array of two integers, where the first element is the total weight of *team 1*, and the second element is the total weight of *team 2* after the division is complete.

Example

For `a = [50, 60, 60, 45, 70]`, the output should be `alternatingSums(a) = [180, 105]`.

Input/Output

- **[time limit] 500ms (cpp)**
- **[input] array.integer a**

  Constraints:
  `1 ≤ a.length ≤ 10,`
  `45 ≤ a[i] ≤ 100.`

- **[output] array.integer**

```
std::vector<int> alternatingSums(std::vector<int> a) {
    vector<int> outPut(2,0);
    for(int i=0;i<a.size();i++)
    { if (i%2) outPut[1]+=a[i];
        else outPut[0]+=a[i];
    }
    return outPut;
}
```

2. Add Border

Given a rectangular matrix of characters, add a border of asterisks(*) to it.
Example

For

```
picture = ["abc",
           "ded"]
```

the output should be

```
addBorder(picture) = ["*****",
                      "*abc*",
                      "*ded*",
                      "*****"]
```

Input/Output

- **[time limit] 500ms (cpp)**
- **[input] array.string picture**

  A non-empty array of non-empty equal-length strings.

  Constraints:
  `1 ≤ picture.length ≤ 5,`
  `1 ≤ picture[i].length ≤ 5.`

- **[output] array.string**

  The same matrix of characters, framed with a border of asterisks of width `1`.

```
std::vector<std::string> addBorder(std::vector<std::string> picture)
{
    int sleng=picture[0].size()+2, pleng=picture.size()+2;
    picture.push_back("");
    picture.push_back("");
    for(int i = 0; i < sleng; i++)
      picture[pleng-1].push_back('*');
    for(int i = pleng - 3; i >= 0; i--)
    {
        picture[i].push_back('*');  picture[i].push_back('*');
        for(int j=sleng-2;j>0;j--)
            picture[i][j]=picture[i][j-1];
        picture[i][0]='*';
        picture[i+1]=picture[i];
   }
    picture[0]=picture[pleng-1];
    return picture;
}
```

3. Are Similar?

Two arrays are called *similar* if one can be obtained from another by swapping at most one pair of elements.

Given two arrays, check whether they are *similar*.

Example

- For `A = [1, 2, 3]` and `B = [1, 2, 3]`, the output should be `areSimilar(A, B) = true`;
- For `A = [1, 2, 3]` and `B = [2, 1, 3]`, the output should be `areSimilar(A, B) = true`;

- For `A = [1, 2, 2]` and `B = [2, 1, 1]`, the output should be `areSimilar(A, B) = false`.

Input/Output

- **[time limit] 500ms (cpp)**
- **[input] array.integer A**

  Array of integers.

  Constraints:
  `3 ≤ A.length ≤` 10^5,
  `1 ≤ A[i] ≤ 1000`.

- **[input] array.integer B**

  Array of integers of the same length as `A`.
  Constraints:
  `B.length = A.length`,
  `1 ≤ B[i] ≤ 1000`.

- **[output] boolean**

  `true` if `A` and `B` are similar, `false`otherwise.

```
bool areSimilar(std::vector<int> A, std::vector<int> B) {

    int count[2], index=0;

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

        if (A[i]!=B[i])

        {

            index++;

            count[index-1]=i;

        }

    if (index==0) return true;

    if
((index==2)&&(A[count[0]]==B[count[1]])&&(A[count[1]]==B[count[0]]))
return true;

    return false;

}
```

4. arrayChange

You are given an array of integers. On each move you are allowed to increase exactly one of its element by one. Find the minimal number of moves required to obtain a strictly increasing sequence from the input.

Example

For `inputArray = [1, 1, 1]`, the output should be
`arrayChange(inputArray) = 3`.
Input/Output

- **[time limit] 500ms (cpp)**
- **[input] array.integer inputArray**

  Constraints:
  `3 ≤ inputArray.length ≤ 10`^{5},
  `-10`^{5} `≤ inputArray[i] ≤ 10`^{5}.

- **[output] integer**

  The minimal number of moves needed to obtain a strictly increasing sequence from `inputArray`.
  It's guaranteed that for the given test cases the answer always fits signed `32`-bit integer type.

```
int arrayChange(std::vector<int> inputArray) {
    int s=0;
    for(int i=0; i<inputArray.size()-1;i++)
        if(inputArray[i]>=inputArray[i+1])
        {
            s+=inputArray[i]+1-inputArray[i+1];
            inputArray[i+1]=inputArray[i]+1;
        }
    return s;
}
```

5. palindromeRearranging

Given a string, find out if its characters can be rearranged to form a palindrome.

Example

For `inputString = "aabb"`, the output should be
`palindromeRearranging(inputString) = true`.
We can rearrange `"aabb"` to make `"abba"`, which is a palindrome.

Input/Output

- **[time limit] 500ms (cpp)**
- **[input] string inputString**

  A string consisting of lowercase English letters.

  Constraints:
  `4 ≤ inputString.length ≤ 50.`

- **[output] boolean**

  `true` if the characters of the `inputString` can be rearranged to form a palindrome, `false`otherwise.

```
bool palindromeRearranging(std::string inputString) {
    int count=0,n=inputString.size();
    for(int i=0; i< n-1; i++)
        if (inputString[i]!='0')
            for(int j=i+1; j<n;j++)
                if (inputString[i]==inputString[j])
                {
                    count+=2;
                    inputString[i]='0';
                    inputString[j]='0';
                    break;
                }
    if(n-count <2) return true;
    return false;
}
```

Level 5

1. areEquallyStrong

Call two arms *equally strong* if the heaviest weights they each are able to lift are equal.

Call two people *equally strong* if their strongest arms are equally strong (the strongest arm can be both the right and the left), and so are their weakest arms.

Given your and your friend's arms' lifting capabilities find out if you two are equally strong.

Example

- For `yourLeft = 10`, `yourRight = 15`, `friendsLeft = 15` and `friendsRight = 10`, the output should be `areEquallyStrong(yourLeft, yourRight, friendsLeft, friendsRight) = true`;
- For `yourLeft = 15`, `yourRight = 10`, `friendsLeft = 15` and `friendsRight = 10`, the output should be `areEquallyStrong(yourLeft, yourRight, friendsLeft, friendsRight) = true`;
- For `yourLeft = 15`, `yourRight = 10`, `friendsLeft = 15` and `friendsRight = 9`, the output should be `areEquallyStrong(yourLeft, yourRight, friendsLeft, friendsRight) = false`.

Input/Output

- **[time limit] 500ms (cpp)**

- **[input] integer yourLeft**

  A non-negative integer representing the heaviest weight you can lift with your left arm.

  Constraints:
  `0 ≤ yourLeft ≤ 15`.

- **[input] integer yourRight**

  A non-negative integer representing the heaviest weight you can lift with your right arm.

  Constraints:
  `0 ≤ yourRight ≤ 15`.

- **[input] integer friendsLeft**

A non-negative integer representing the heaviest weight your friend can lift with his or her left arm.

Constraints:
`0 ≤ friendsLeft ≤ 15.`

- **[input] integer friendsRight**

  A non-negative integer representing the heaviest weight your friend can lift with his or her right arm.

  Constraints:
  `0 ≤ friendsRight ≤ 15.`

- **[output] boolean**

  `true` if you and your friend are equally strong, `false` otherwise.

```
bool areEquallyStrong(int a1, int a2, int b1, int b2) {
    return ((abs(a1-a2)==abs(b1-b2))&&(a1+a2==b1+b2));
}
```

2. arrayMaximalAdjacentDifference

Given an array of integers, find the maximal absolute difference between any two of its adjacent elements.

Example

For `inputArray = [2, 4, 1, 0]`, the output should be
`arrayMaximalAdjacentDifference(inputArray) = 3.`
Input/Output

- **[time limit] 500ms (cpp)**

- **[input] array.integer inputArray**

  Constraints:
  `3 ≤ inputArray.length ≤ 10,`
  `-15 ≤ inputArray[i] ≤ 15.`

- **[output] integer**

  The maximal absolute difference.

```
int arrayMaximalAdjacentDifference(std::vector<int> inputArray) {
    int m=-1;
   for(int i=1; i<inputArray.size(); i++)
    {
        m= max(m,abs( (inputArray[i]) - (inputArray[i-1]) ) );
    }
    return m;
}
```

3. isIPv4Address

An IP address is a numerical label assigned to each device (e.g., computer, printer) participating in a computer network that uses the Internet Protocol for communication. There are two versions of the Internet protocol, and thus two versions of addresses. One of them is the *IPv4 address*.

IPv4 addresses are represented in dot-decimal notation, which consists of four decimal numbers, each ranging from `0` to `255`, separated by dots, e.g., `172.16.254.1`. Given a string, find out if it satisfies the IPv4 address naming rules.

Example

- For `inputString = "172.16.254.1"`, the output should be `isIPv4Address(inputString) = true`;
- For `inputString = "172.316.254.1"`, the output should be `isIPv4Address(inputString) = false`.
  `316` is not in range `[0, 255]`.
- For `inputString = ".254.255.0"`, the output should be `isIPv4Address(inputString) = false`.

  There is no first number.

Input/Output

- **[time limit] 500ms (cpp)**
- **[input] string inputString**

  A string consisting of digits, full stops and lowecase Latin letters.

  Constraints:
  `5 ≤ inputString.length ≤ 15`.

- **[output]**

  `true` if `inputString`satisfies the IPv4 address naming rules, `false`otherwise.

```
bool isIPv4Address(std::string inputString) {

    int index=0,num=0,count=0, leng=inputString.size();

    if((inputString[0]=='.')||(inputString[leng-
1]=='.')||(leng>15)) return false;

    for(int i=leng-1;i>=0;i--)

    {

        if (inputString[i]!='.')

        {

            if ((inputString[i]>'9')||(inputString[i]<'0'))
return false;

            num+= ((int)(inputString[i]) - 48) *
(int(pow(10,index)));

            index++;

        }

       else

        {

            if (num>255) return false;

            if (inputString[i-1]=='.') return false;

            count++;

            num=0;

            index=0;

        }

        if (i==0)

        {

             if (num>255) return false;
```

```
                count++;
            }
        }
        if (count!=4) return false;
        return true;
}
```

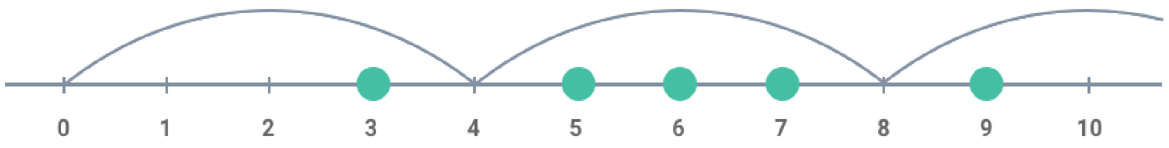
4. avoidObstacles

You are given an array of integers representing coordinates of obstacles situated on a straight line.

Assume that you are jumping from the point with coordinate `0` to the right. You are allowed only to make jumps of the same length represented by some integer.
Find the minimal length of the jump enough to avoid all the obstacles.

Example

For `inputArray = [5, 3, 6, 7, 9]`, the output should be
`avoidObstacles(inputArray) = 4`.
Check out the image below for better understanding:



Input/Output

- **[time limit] 500ms (cpp)**
- **[input] array.integer inputArray**
- Non-empty array of positive integers.

  Constraints:
  `2 ≤ inputArray.length ≤ 10`,
  `1 ≤ inputArray[i] ≤ 40`.

- **[output] integer**

  The desired length.

```
int avoidObstacles(std::vector<int> inputArray) {

    bool mark[41];

    int jump,max=-1,i;

    for(i=0 ; i<41; i++) mark[i]=false;

    for ( i=0; i<inputArray.size(); i++)

    {

        mark[ inputArray[i] ] = true;

        max = max > inputArray[i] ? max : inputArray[i] ;

    }

     for ( jump=1; jump<=max; jump++)

    {

        for(i=0; i<=max; i+=jump)

            if (mark[i]==true) break;

        if (i>max) return jump;

    }

    return jump;

 }
```

5. Box Blur

Last night you had to study, but decided to party instead. Now there is a black and white photo of you that is about to go viral. You cannot let this ruin your reputation, so you want to apply *box blur* algorithm to the photo to hide its content.

The algorithm works as follows: each pixel `x` in the resulting image has a value equal to the average value of the input image pixels' values from the `3 × 3` square with the center at `x`. All pixels at the edges are cropped.
As pixel's value is an integer, all fractions should be rounded down.

Example

For

```
image = [[1, 1, 1],
         [1, 7, 1],
         [1, 1, 1]]
```
the output should be `boxBlur(image) = [[1]]`.
In the given example all boundary pixels were cropped, and the value of the pixel in the middle was obtained as `(1 + 1 + 1 + 1 + 7 + 1 + 1 + 1 + 1) / 9 = 15 / 9 =~rounded down~ = 1`.

Input/Output

- **[time limit] 500ms (cpp)**
- **[input] array.array.integer image**
- An image is stored as a rectangular matrix of non-negative integers.

  Constraints:
  `3 ≤ image.length ≤ 10`,
  `3 ≤ image[0].length ≤ 10`,
  `0 ≤ image[i][j] ≤ 255`.
- **[output] array.array.integer**

  A blurred image.

```
std::vector<std::vector<int>> boxBlur(std::vector<std::vector<int>>
image) {

    int m=image.size(), n=image[0].size(),sum;

    vector<vector<int>> box;

    vector<int> temp;

    for(int i=0; i< n-2; i++) temp.push_back(0);

    for(int i=0; i< m-2; i++) box.push_back(temp);


    for(int I=0; I<m-2; I++)

    {

           for(int J=0; J<n-2; J++)

           {

               sum=0;

               for(int i=I; i<=I+2; i++)

                   for(int j=J; j<=J+2; j++)
```

```
                              sum+=image[i][j];

                      box[I][J]=sum/9;

                  }

         }

          return box;

    }
```

6. Minesweeper

In the popular **Minesweeper** game you have a board with some mines and those cells that don't contain a mine have a number in it that indicates the total number of mines in the neighboring cells. Starting off with some arrangement of mines we want to create a **Minesweeper** game setup.

Example

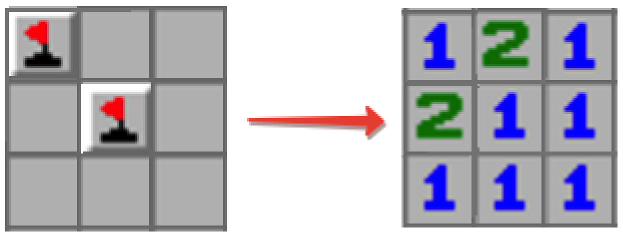
For

```
matrix = [[true, false, false],
          [false, true, false],
          [false, false, false]]
```

the output should be

```
minesweeper(matrix) = [[1, 2, 1],
                       [2, 1, 1],
                       [1, 1, 1]]
```

Check out the image below for better understanding:



Input/Output

- **[time limit] 500ms (cpp)**
- **[input] array.array.boolean matrix**

A non-empty rectangular matrix consisting of boolean values - `true` if the corresponding cell contains a mine, `false` otherwise.
Constraints:
`2 ≤ matrix.length ≤ 5,`
`2 ≤ matrix[0].length ≤ 5.`

- **[output] array.array.integer**

  Rectangular matrix of the same size as `matrix` each cell of which contains an integer equal to the number of mines in the neighboring cells. Two cells are called neighboring if they share at least one corner.

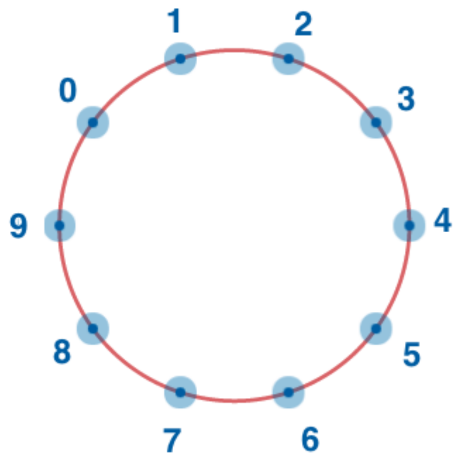
```
std::vector<std::vector<int>>
minesweeper(std::vector<std::vector<bool>> matrix) {
    vector<vector<int>> temp={{0}}, result={{0}};
    int m=matrix.size(), n=matrix[0].size();

    //khởi tạo vecto 0
    for(int i=0; i<=n; i++) temp[0].push_back(0);
    for(int i=0; i<=m; i++) temp.push_back(temp[0]);
    for(int i=0; i<=n-2; i++) result[0].push_back(0);
    for(int i=0; i<=m-2; i++) result.push_back(result[0]);

    //chuyển matrix thành vecto temp, true thành 1
    for(int i=0; i<m; i++)
        for(int j=0; j<n; j++)
            if (matrix[i][j]) temp[i+1][j+1]+=1;

    //tính result từ temp bằng cách cộng các ô xung quanh
     for(int i=0; i<m; i++)
        for(int j=0; j<n; j++)
           result[i][j]=temp[i][j] + temp[i][j+1] +
temp[i][j+2]                         + temp[i+1][j]+
       temp[i+1][j+2]

+temp[i+2][j]+temp[i+2][j+1]+temp[i+2][j+2];
    return result;
}
```

Level 7

1. Circle of Numbers

Consider integer numbers from `0` to `n - 1` written down along the circle in such a way that the distance between any two neighbouring numbers is equal (note that `0`and `n - 1` are neighbouring, too).
Given `n` and `firstNumber`, find the number which is written in the radially opposite position to `firstNumber`.
Example

For `n = 10` and `firstNumber = 2`, the output should be
`circleOfNumbers(n, firstNumber) = 7`.



Input/Output

- **[time limit] 500ms (cpp)**
- **[input] integer n**

  A positive **even** integer.

  Constraints:
  `4 ≤ n ≤ 20.`
- **[input] integer firstNumber**

  Constraints:
  `0 ≤ firstNumber ≤ n - 1.`
- **[output] integer**

```
int circleOfNumbers(int n, int firstNumber) {

    return (firstNumber + n/2) % n;

}
```

2. depositProfit

You have deposited a specific amount of dollars into your bank account. Each year your balance increases at the same growth `rate`. Find out how long it would take for your balance to pass a specific `threshold`with the assumption that you don't make any additional deposits.

Example

For `deposit = 100`, `rate = 20` and `threshold = 170`, the output should be
`depositProfit(deposit, rate, threshold) = 3`.
Each year the amount of money on your account increases by `20%`. It means that throughout the years your balance would be:

- year 0: `100`;
- year 1: `120`;
- year 2: `144`;
- year 3: `172,8`.

Thus, it will take `3`years for your balance to pass the `threshold`, which is the answer.
Input/Output

- **[time limit] 500ms (cpp)**
- **[input] integer deposit**

  The initial deposit as a positive integer.

  Constraints:
  `1 ≤ deposit ≤ 100`.
- **[input] integer rate**

  The rate of increase. Each year the balance increases by the `rate` *percent* of the current sum.
  Constraints:
  `1 ≤ rate ≤ 100`.
- **[input] integer threshold**

  The target balance.

  Constraints:
  `deposit < threshold ≤ 200`.
- **[output] integer**

  The number of years it would take to hit the `threshold`.

```
int depositProfit(int deposit, int rate, int threshold) {

    int years=0;

    while (deposit* pow(1+rate*0.01 , years ) < threshold) years++;

    return years;

}
```

3. absoluteValuesSumMinimization

Given a sorted array of integers `a`, find such an integer `x` that the value of
`abs(a[0] - x) + abs(a[1] - x) + ... + abs(a[a.length - 1] - x)`
is the *smallest possible* (here `abs` denotes the absolute value).
If there are several possible answers, output the *smallest* one.
Example

For `a = [2, 4, 7]`, the output should be
`absoluteValuesSumMinimization(a) = 4`.
Input/Output

- **[time limit] 500ms (cpp)**
- **[input] array.integer a**

  A non-empty array of integers, sorted in ascending order.

  Constraints:
  `1 ≤ a.length ≤ 200`,
  $-10^6 \le a[i] \le 10^6$.
- **[output] integer**

```
int absoluteValuesSumMinimization(std::vector<int> a) {

    return a[(a.size()-1)/2];

}
```

4. stringsRearrangement

Given an array of equal-length strings, check if it is possible to rearrange the strings in such a way that after the rearrangement the strings at consecutive positions would differ by exactly one character.

Example

- For `inputArray = ["aba", "bbb", "bab"]`, the output should be
  `stringsRearrangement(inputArray) = false`;
- For `inputArray = ["ab", "bb", "aa"]`, the output should be
  `stringsRearrangement(inputArray) = true`.

Input/Output

- **[time limit] 500ms (cpp)**
- **[input] array.string inputArray**

  A non-empty array of strings of lowercase letters.

  Constraints:
  `2 ≤ inputArray.length ≤ 10`,
  `1 ≤ inputArray[i].length ≤ 15.`
- **[output] boolean**

```
bool stringsRearrangement(std::vector<std::string> inputArray) {
     vector<string> lstString;
     std::vector<string>::iterator it;
     lstString.push_back(inputArray[0]);
     it = lstString.begin();
     vector<string> lstInputString;
     lstInputString=inputArray;
     lstInputString.erase(lstInputString.begin());
     int ii=0;
     for(ii=0; ii<inputArray.size(); ii++)
     {
          bool bResult = false;
          for(int i=0; i<lstInputString.size(); i++)
          {
               int nLength = lstString.size();

               if (isDifferentOneChar(lstString[nLength - 1],
lstInputString[i]))
               {
                    // lstString.Insert(nLength, lstInputString[i]);
                     it = lstString.insert ( it , lstInputString[i] );
                     bResult = true;
                     lstInputString.erase(lstInputString.begin()+i);
                    break;
                }
                else if (isDifferentOneChar(lstString[0],
lstInputString[i]))
                {
                    it = lstString.insert ( it , lstInputString[i] );
                    bResult = true;
                    lstInputString.erase(lstInputString.begin()+i);
                    break;
                }
            }
             if (bResult == false||lstInputString.size()==0)
                    break;
        }
        if (lstInputString.size() == 0)
                return true;
             else return false;

}
```

```
bool isDifferentOneChar(std::string str1, std::string str2)
{
    int nCount = 0;
            for (int i = 0; i < str1.size(); i++)
                if (str1[i] != str2[i])
                {
                    nCount++;
                    if (nCount > 1) break;
                }
            return nCount == 1;     }
```

Level 8

1. extractEachKth

Given array of integers, remove each `k`th element from it.
Example

For `inputArray = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]` and `k = 3`, the output should be `extractEachKth(inputArray, k) = [1, 2, 4, 5, 7, 8, 10]`.
Input/Output

- **[time limit] 500ms (cpp)**
- **[input] array.integer inputArray**

  Constraints:
  `5 ≤ inputArray.length ≤ 15`,
  `-20 ≤ inputArray[i] ≤ 20`.
- **[input] integer k**

  Constraints:
  `1 ≤ k ≤ 10`.
- **[output] array.integer**

  `inputArray` without elements `k - 1`, `2k - 1`, `3k - 1` etc.

```
std::vector<int> extractEachKth(std::vector<int> inputArray, int k) {
     for(int i=k-1; i<inputArray.size(); i+=k-1)
          inputArray.erase(inputArray.begin()+i);
     return inputArray;
}
```

2. firstDigit

Find the leftmost digit that occurs in a given string.

Example

- For `inputString = "var_1__Int"`, the output should be `firstDigit(inputString) = '1'`;
- For `inputString = "q2q-q"`, the output should be `firstDigit(inputString) = '2'`;

- For `inputString = "0ss"`, the output should be `firstDigit(inputString) = '0'`.

Input/Output

- **[time limit] 500ms (cpp)**
- **[input] string inputString**

  A string containing at least one digit.

  Constraints:
  `3 ≤ inputString.length ≤ 10`.
- **[output] char**

```
char firstDigit(std::string inputString) {
    for(int i=0; i<inputString.size(); i++)
        if (inputString[i]>='0' && inputString[i]<='9') return
inputString[i];
}
```

3. differentSymbolsNaive

Given a string, find the number of different characters in it.

Example

For `s = "cabca"`, the output should be
`differentSymbolsNaive(s) = 3`.
There are `3` different characters `a`, `b` and `c`.
Input/Output

- **[time limit] 500ms (cpp)**
- **[input] string s**

  A string of lowercase latin letters.

  Constraints:
  `3 ≤ s.length ≤ 15`.
- **[output] integer**

```
int differentSymbolsNaive(std::string s) {
    bool check[26];
    int count=0;

    for(int i=0; i<26; i++)
        check[i]=0;

    for(int i=0; i<s.size(); i++)
        if ((check[s[i]-97])==0)
            {
                check[s[i]-97] = 1;
                count++;
            }

    return count;
}
```

4. arrayMaxConsecutiveSum

Given array of integers, find the maximal possible sum of some of its `k` consecutive elements.

Example

For `inputArray = [2, 3, 5, 1, 6]` and `k = 2`, the output should be `arrayMaxConsecutiveSum(inputArray, k) = 8`.
All possible sums of `2`consecutive elements are:

- `2 + 3 = 5`;
- `3 + 5 = 8`;
- `5 + 1 = 6`;
- `1 + 6 = 7`.
  Thus, the answer is `8`.

Input/Output

- **[time limit] 500ms (cpp)**
- **[input] array.integer inputArray**

Array of positive integers.

Constraints:
`3 ≤ inputArray.length ≤ 10`^{5},
`1 ≤ inputArray[i] ≤ 1000`.

- **[input] integer k**

  An integer (not greater than the length of `inputArray`).
  Constraints:
  `1 ≤ k ≤ inputArray.length`.

- **[output] integer**

  The maximal possible sum.

```
int arrayMaxConsecutiveSum(std::vector<int> inputArray, int k) {
    int s=0, max;
    for(int i=0; i<k; i++) s+=inputArray[i];
    max=s;
    for(int i=k; i<inputArray.size(); i++)
    {
        s = s + inputArray[i] - inputArray[i-k];
        if (s > max) max = s;
    }
    return max;
}
```

Level 9

1. growingPlant

Each day a plant is growing by `upSpeed` meters. Each night that plant's height decreases by `downSpeed`meters due to the lack of sun heat. Initially, plant is 0 meters tall. We plant the seed at the beginning of a day. We want to know when the height of the plant will reach a certain level.
Example

For `upSpeed = 100`, `downSpeed = 10` and `desiredHeight = 910`, the output should be `growingPlant(upSpeed, downSpeed, desiredHeight) = 10`.
Input/Output

- **[time limit] 500ms (cpp)**
- **[input] integer upSpeed**

  A positive integer representing the daily growth.

  Constraints:
  `5 ≤ upSpeed ≤ 100.`
- **[input] integer downSpeed**

  A positive integer representing the nightly decline.

  Constraints:
  `2 ≤ downSpeed < upSpeed.`
- **[input] integer desiredHeight**

  A positive integer representing the threshold.

  Constraints:
  `4 ≤ desiredHeight ≤ 1000.`
- **[output] integer**

  The number of days that it will take for the plant to reach/pass `desiredHeight`(including the last day in the total count).

```
int growingPlant(int upSpeed, int downSpeed, int desiredHeight) {
    if (desiredHeight<=upSpeed) return 1;
    return 1+growingPlant(upSpeed, downSpeed, desiredHeight-
upSpeed+downSpeed); }
```

2. Knapsack Light

You found two items in a treasure chest! The first item weighs `weight1` and is worth `value1`, and the second item weighs `weight2` and is worth `value2`. What is the total maximum value of the items you can take with you, assuming that your max weight capacity is `maxW` and you can't come back for the items later?
Example

- For `value1 = 10`, `weight1 = 5`, `value2 = 6`, `weight2 = 4` and `maxW = 8`, the output should be
  `knapsackLight(value1, weight1, value2, weight2, maxW) = 10.`

  You can only carry the first item.

- For `value1 = 10`, `weight1 = 5`, `value2 = 6`, `weight2 = 4` and `maxW = 9`, the output should be
  `knapsackLight(value1, weight1, value2, weight2, maxW) = 16`.

  You're strong enough to take both of the items with you.

Input/Output

- **[time limit] 500ms (cpp)**
- **[input] integer value1**

  Constraints:
  `2 ≤ value1 ≤ 20`.
- **[input] integer weight1**

  Constraints:
  `2 ≤ weight1 ≤ 10`.
- **[input] integer value2**

  Constraints:
  `2 ≤ value2 ≤ 20`.
- **[input] integer weight2**

  Constraints:
  `2 ≤ weight2 ≤ 10`.
- **[input] integer maxW**

  Constraints:
  `1 ≤ maxW ≤ 20`.
- **[output] integer**

```
int knapsackLight(int value1, int weight1, int value2, int weight2,
int maxW) {

    if (maxW < weight1 && maxW < weight2) return 0;

    else if (maxW >= weight1 && maxW < weight2) return value1;

    else if (maxW < weight1 && maxW >= weight2) return value2;

    else if (maxW >= (weight1+weight2) ) return value1 + value2;

    else return max(value1, value2);

}
```

3. longestDigitsPrefix

Given a string, output its longest prefix which contains only digits.

Example

For `inputString="123aa1"`, the output should be
`longestDigitsPrefix(inputString) = "123"`.

Input/Output

- **[time limit] 500ms (cpp)**
- **[input] string inputString**

  Constraints:
  `3 ≤ inputString.length ≤ 35.`
- **[output] string**

```
std::string longestDigitsPrefix(std::string inputString) {
    string s;
    for(int i = 0; i < inputString.size(); i++)
    {
        if (inputString[i] >= '0' && inputString[i] <= '9') s +=
inputString[i];
        else break;
    }
    return s;
}
```

4. digitDegree

Let's define *digit degree* of some positive integer as the number of times we need to replace this number with the sum of its digits until we get to a one digit number.

Given an integer, find its digit degree.

Example

- For `n = 5`, the output should be
  `digitDegree(n) = 0`;

- For `n = 100`, the output should be
  `digitDegree(n) = 1`.
  `1 + 0 + 0 = 1`.
- For `n = 91`, the output should be
  `digitDegree(n) = 2`.
  `9 + 1 = 10` -> `1 + 0 = 1`.

Input/Output

- **[time limit] 500ms (cpp)**
- **[input] integer n**

  Constraints:
  $5 \leq n \leq 10^9$.

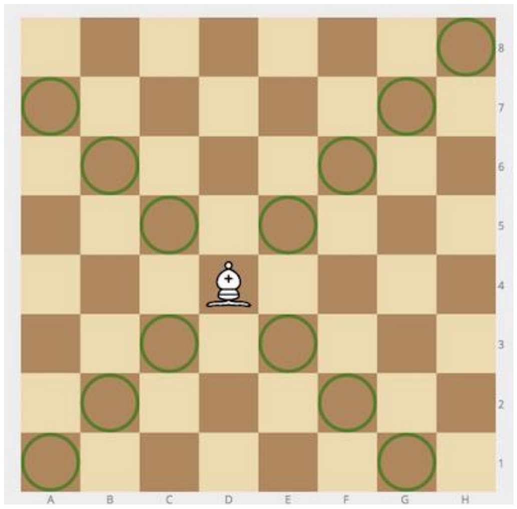
- **[output] integer**

```
int digitDegree(int n) {
    int degree = 0;
    while(n > 9)
    {
        int numberDigits = floor(log10( n )) +1 , m = n ;
        n=0;
        degree++;
        for(int i = numberDigits-1; i >= 0; i--)
        {
            n += m / ((int) pow(10,i)) ;
            m = m %  ((int) pow(10,i));
        }
    }
    return degree;
}
```

5. Bishop and Pawn

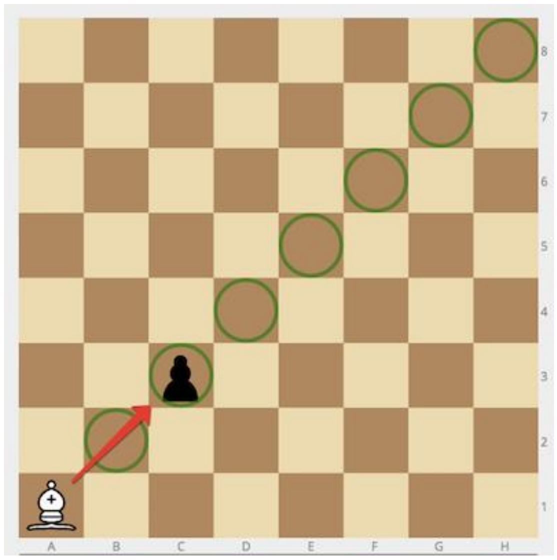
Given the positions of a white bishop and a black pawn on the standard chess board, determine whether the bishop can capture the pawn in one move.

The bishop has no restrictions in distance for each move, but is limited to diagonal movement. Check out the example below to see how it can move:



Example

- For `bishop = "a1"` and `pawn = "c3"`, the output should be
  `bishopAndPawn(bishop, pawn) = true`.



- For `bishop = "h1"` and `pawn = "h3"`, the output should be
  `bishopAndPawn(bishop, pawn) = false`.

Input/Output

- **[time limit] 500ms (cpp)**
- **[input] string bishop**

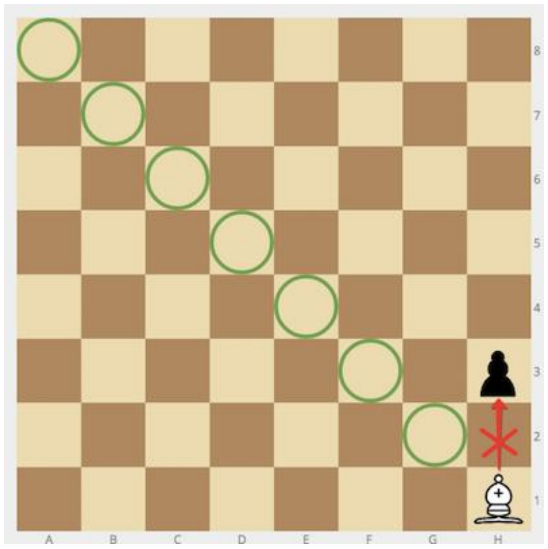
  Coordinates of the white bishop in the chess notation.

- **[input] string pawn**

  Coordinates of the black pawn in the same notation.

- **[output] boolean**

  `true` if the bishop can capture the pawn, `false` otherwise.



```
bool bishopAndPawn(std::string bishop, std::string pawn) {
    return (bishop[0]-bishop[1] == pawn[0]-pawn[1]) ||
(bishop[0]+bishop[1] == pawn[0]+pawn[1]);
}
```

Level 10

1. isBeautifylString

A string is said to be *beautiful* if `b` occurs in it no more times than `a`; `c` occurs in it no more times than `b`; etc.
Given a string, check whether it is *beautiful*.

Example

- For `inputString = "bbbaacdafe"`, the output should be `isBeautifulString(inputString) = true`;
- For `inputString = "aabbb"`, the output should be `isBeautifulString(inputString) = false`;
- For `inputString = "bbc"`, the output should be `isBeautifulString(inputString) = false`.

Input/Output

- **[time limit] 500ms (cpp)**

- **[input] string inputString**

  A string of lowercase letters.

  Constraints:
  `3 ≤ inputString.length ≤ 50.`

- **[output] boolean**

```
bool isBeautifulString(std::string inputString) {
    int count[26]={0};
    for(int i=0; i<inputString.size(); i++)
    {
        count[inputString[i]-97] ++;
    }
    for(int i=1; i<26; i++) if ( count[i] > count[i-1] ) return 0;
    return 1;
}
```

2. Find Email Domain

An email address such as `"John.Smith@example.com"` is made up of a local part (`"John.Smith"`), an `"@"` symbol, then a domain part (`"example.com"`).
The domain name part of an email address may only consist of letters, digits, hyphens and dots. The local part, however, also allows a lot of different special characters. Here you can look at several examples of correct and incorrect email addresses.

Given a valid email address, find its domain part.

Example

- For `address = "prettyandsimple@example.com"`, the output should be
  `findEmailDomain(address) = "example.com";`
- For `address = "<>[]:,;@\"!#$%&*+-/=?^_{}| ~.a\"@example.org"`, the output should be
  `findEmailDomain(address) = "example.org".`

Input/Output

- **[time limit] 500ms (cpp)**

- **[input] string address**

Constraints:
`10 ≤ address.length ≤ 50.`

- **[output] string**

```
std::string findEmailDomain(std::string address) {
    string domain;
    int i = address.size()-1;
    while (address[i] != '@')
    {
        domain = address[i] + domain;
        i--;
    }
    return domain;
}
```

3. buildPalindrome

Given a string, find the shortest possible string which can be achieved by adding characters to the end of initial string to make it a palindrome.

Example

For `st = "abcdc"`, the output should be
`buildPalindrome(st) = "abcdcba"`.

Input/Output

- **[time limit] 500ms (cpp)**
- **[input] string st**

  A string consisting of lowercase latin letters.

  Constraints:
  `3 ≤ st.length ≤ 10.`
- **[output] string**

```
std::string buildPalindrome(std::string st) {

    int n=st.size();

    for(int i=0; i<n/2+1; i++)

    {

        if (st[i] != st[st.size()-i-1])
st.insert(st.begin()+st.size()-i,st[i]);

    }

    return st;

}
```

4. Elections Winners

Elections are in progress!

Given an array of the numbers of votes given to each of the candidates so far, and an integer `k` equal to the number of voters who haven't cast their vote yet, find the number of candidates who still have a chance to win the election.
The winner of the election must secure strictly more votes than any other candidate. If two or more candidates receive the same (maximum) number of votes, assume there is no winner at all.

Example

For `votes = [2, 3, 5, 2]` and `k = 3`, the output should be `electionsWinners(votes, k) = 2`.

- The first candidate got `2` votes. Even if all of the remaining `3`candidates vote for him, he will still have only `5` votes, i.e. the same number as the third candidate, so there will be no winner.
- The second candidate can win if all the remaining candidates vote for him (`3 + 3 = 5 > 6`).
- The third candidate can win even if none of the remaining candidates vote for him. For example, if each of the remaining voters cast their votes for each of his opponents, he will still be the winner (the `votes` array will thus be `[3, 4, 5, 3]`).
- The last candidate can't win no matter what (for the same reason as the first candidate).

Thus, only 2candidates can win (the second and the third), which is the answer.
Input/Output

- **[time limit] 500ms (cpp)**
- **[input] array.integer votes**

  A non-empty array of non-negative integers. Its i^th^element denotes the number of votes cast for the i^th^ candidate.
  Constraints:
  `4 ≤ votes.length ≤ 10^5,`
  `0 ≤ votes[i] ≤ 10^4.`

- **[input] integer k**

  The number of voters who haven't cast their vote yet.

  Constraints:
  `0 ≤ k ≤ 10^5.`

- **[output] integer**

```
int electionsWinners(std::vector<int> votes, int k) {
    int max=votes[0], count=0, countMax=1;
    for(int i = 1; i < votes.size(); i++)
    {
        if (votes[i]>max)
        {
            max=votes[i];
            countMax=1;
        }
        else if (votes[i]==max)
            countMax++;
    }
    for(int i = 0; i < votes.size(); i++)
        if (votes[i] + k > max) count++;
    if (k==0 && countMax!=1) return 0;
    if (k==0) return 1;
    return count;
}
```

5. Is MAC48 Address?

A media access control address (MAC address) is a unique identifier assigned to network interfaces for communications on the physical network segment.

The standard (IEEE 802) format for printing MAC-48 addresses in human-friendly form is six groups of two hexadecimal digits (`0`to `9` or `A` to `F`), separated by hyphens (e.g. `01-23-45-67-89-AB`).
Your task is to check by given string `inputString` whether it corresponds to MAC-48 address or not.
Example

- For `inputString = "00-1B-63-84-45-E6"`, the output should be `isMAC48Address(inputString) = true`;
- For `inputString = "Z1-1B-63-84-45-E6"`, the output should be `isMAC48Address(inputString) = false`;
- For `inputString = "not a MAC-48 address"`, the output should be `isMAC48Address(inputString) = false`.

Input/Output

- **[time limit] 500ms (cpp)**
- **[input] string inputString**

  Constraints:
  `15 ≤ inputString.length ≤ 20`.
- **[output] boolean**

  `true` if `inputString`corresponds to MAC-48 address naming rules, `false` otherwise.

```
bool isMAC48Address(std::string inputString) {
    if (inputString.size() != 17 ) return 0;
    for(int i=0; i<17; i++)
    {
        if (( i % 3 ) != 2)
        {
            if ( ! (( inputString[i] >= 'A' && inputString[i] <= 'F'
) || ( inputString[i] >= '0' && inputString[i] <= '9' )))return 0;
            printf("OK");
        }
        else if (inputString[i] != '-') return 0;
    }
    return 1;
}
```

Level 11

1. isDigit

Determine if the given character is a digit or not.

Example

- For `symbol = '9'`, the output should be `isDigit(symbol) = true`;
- For `symbol = '-'`, the output should be `isDigit(symbol) = false`.

Input/Output

- **[time limit] 500ms (cpp)**
- **[input] char symbol**

  A character which is either a digit or not.
- **[output] boolean**

  `true` if `symbol` is a digit, `false`otherwise.

```
bool isDigit(char symbol) {
    return (symbol >= '0') && (symbol <= '9');
}
```

2. lineEncoding

Given a string, return its encoding defined as follows:

- First, the string is divided into the least possible number of disjoint substrings consisting of identical characters
    - for example, `"aabbbc"` is divided into `["aa", "bbb", "c"]`
- Next, each *substring* with length greater than one is replaced with a concatenation of its length and the repeating character
    - for example, *substring*`"bbb"` is replaced by `"3b"`
- Finally, all the new strings are concatenated together in the same order and a new string is returned.

Example

For `s = "aabbbc"`, the output should be
`lineEncoding(s) = "2a3bc"`.

Input/Output

- **[time limit] 500ms (cpp)**
- **[input] string s**

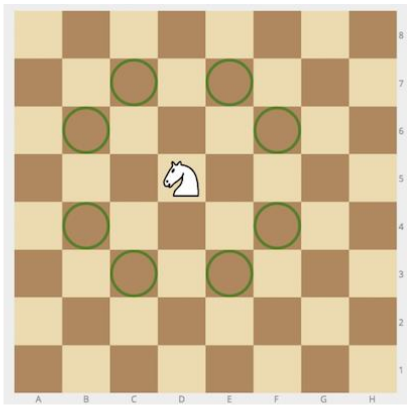
  String consisting of lowercase English letters.

  Constraints:
  `4 ≤ s.length ≤ 15.`
- **[output] string**

  Encoded version of `s`.

```
std::string lineEncoding(std::string s) {
    string encoding;


    int count=1, i;
    for (i = 1; i < s.size(); i++)
    {
        if (s[i] != s[i-1])


        {
            if (count != 1)
            {
                //convert count from int to string
                stringstream convert;
                convert << count;

                encoding += convert.str();
                count=1;
             }
            encoding += s[i-1];
        }
        else count++;
    }
    if (count != 1)
            {
                //convert count from int to string
                stringstream convert;
                convert << count;

                encoding += convert.str();
                count=1;
             }
    encoding += s[i-1];
    return encoding;
}
```

3. chessKnight



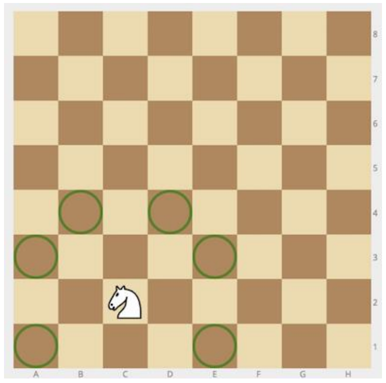
Given a position of a knight on the standard chessboard, find the number of different moves the knight can perform.

The knight can move to a square that is two squares horizontally and one square vertically, or two squares vertically and one square horizontally away from it. The complete move therefore looks like the letter L. Check out the image below to see all valid moves for a knight piece that is placed on one of the central squares.

Example

- For `cell = "a1"`, the output should be `chessKnight(cell) = 2`.

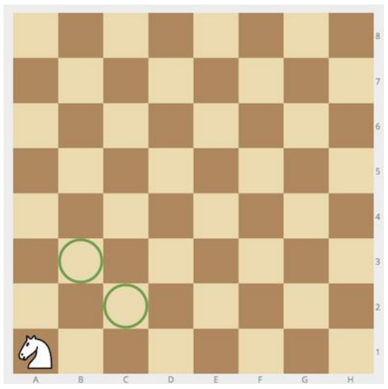
- For `cell = "c2"`, the output should be `chessKnight(cell) = 6`.



Input/Output

- **[time limit] 500ms (cpp)**
- **[input] string cell**

  String consisting of 2 letters - coordinates of the knight on an `8 × 8` chessboard in chess notation.
- **[output] integer**



```
int chessKnight(std::string cell) {
    int number[8][8]={  {2, 3, 4, 4, 4, 4, 3, 2},
                        {3, 4, 6, 6, 6, 6, 4, 3},
                        {4, 6, 8, 8, 8, 8, 6, 4},
                        {4, 6, 8, 8, 8, 8, 6, 4},
                        {4, 6, 8, 8, 8, 8, 6, 4},
                        {4, 6, 8, 8, 8, 8, 6, 4},
                        {3, 4, 6, 6, 6, 6, 4, 3},
                        {2, 3, 4, 4, 4, 4, 3, 2},
                     };
    return number[cell[1]-49][cell[0]-97];
}
```

4. deleteDigit

Given some integer, find the maximal number you can obtain by deleting exactly one digit of the given number.

Example

- For `n = 152`, the output should be `deleteDigit(n) = 52`;
- For `n = 1001`, the output should be `deleteDigit(n) = 101`.

Input/Output

- **[time limit] 500ms (cpp)**
- **[input] integer n**

  Constraints:
  $10 \le n \le 10^6$.
- **[output] integer**

```
int deleteDigit(int n) {
    int result = 0, temp;
    for(int i = 1; i <= n; i*=10)
    {
        temp = (n/(10*i))*i + n%i;
        result = max(result, temp);
    }
    return result;
 }
```

Level 12

1. longestWord

Define a *word* as a sequence of consecutive letters. Find the longest *word* from the given string.

Example

For `text = "Ready, steady, go!"`, the output should be
`longestWord(text) = "steady"`.

Input/Output

- **[time limit] 500ms (cpp)**
- **[input] string text**

  Constraints:
  `4 ≤ text.length ≤ 50.`
- **[output] string**

  The longest *word* from `text`. It's guaranteed that there is a unique output.

```
std::string longestWord(std::string text) {
    int count=0, max=0, index=0;
    string result = "";
    for(int i = 0; i < text.size(); i++)
    {
        if (isalpha(text[i]))
        {
            count++;
            if (count > max)
            {
                max = count;
                index = i;
            }
        }
        else
```

```
                count=0;
         }
        for (int i = index - max + 1; i <= index; i++)
        {
            result.push_back(text[i]);
        }
        return result;
}
```

2. Valid Time

Check if the given string is a correct time representation of the 24-hour clock.

Example

- For `time = "13:58"`, the output should be `validTime(time) = true`;
- For `time = "25:51"`, the output should be `validTime(time) = false`;
- For `time = "02:76"`, the output should be `validTime(time) = false`.

Input/Output

- **[time limit] 500ms (cpp)**
- **[input] string time**

  A string representing time in `HH:MM`format. It is guaranteed that the first two characters, as well as the last two characters, are digits.
- **[output] boolean**

  `true` if the given representation is correct, `false` otherwise.

```
bool validTime(std::string time) {
    return (((time[0]-48) * 10 + time[1] - 48) <= 23) && (((time[3]-48) * 10 + time[4]-48) <= 59)  ;
}
```

3. sumUpNumbers

CodeMaster has just returned from shopping. He scanned the check of the items he bought and gave the resulting string to Ratiorg to figure out the total number of purchased items. Since Ratiorg is a bot he is definitely going to automate it, so he needs a program that sums up all the numbers which appear in the given input.

Help Ratiorg by writing a function that returns the sum of numbers that appear in the given `inputString`.

Example

For `inputString = "2 apples, 12 oranges"`, the output should be `sumUpNumbers(inputString) = 14`.

Input/Output

- **[time limit] 500ms (cpp)**
- **[input] string inputString**

  Constraints:
  `6 ≤ inputString.length ≤ 60`.
- **[output] integer**

```
int sumUpNumbers(std::string inputString) {
   int sum = 0, temp = 0;
   inputString += "a";
   for (int i = 0; i < inputString.size(); i++)
   {
      if (isdigit(inputString[i]))
      {
         temp *= 10;
         temp += inputString[i] - 48;
         printf("%d ", temp);
      }
      else
      {
         sum += temp;
         temp = 0;
```

```
        }
    }
    return sum;
}
```

4. Different Squares

Given a rectangular matrix containing only digits, calculate the number of different `2 × 2` squares in it.

Example

For

```
matrix = [[1, 2, 1],
          [2, 2, 2],
          [2, 2, 2],
          [1, 2, 3],
          [2, 2, 1]]
```

the output should be
`differentSquares(matrix) = 6.`

Here are all `6` different `2 × 2` squares:

- 1 2
  2 2
- 2 1
  2 2
- 2 2
  2 2
- 2 2
  1 2
- 2 2
  2 3
- 2 3
  2 1

Input/Output

- **[time limit] 500ms (cpp)**
- **[input] array.array.integer matrix**

  Constraints:
  `1 ≤ matrix.length ≤ 100,`
  `1 ≤ matrix[i].length ≤ 100,`
  `0 ≤ matrix[i][j] ≤ 9.`

- **[output] integer**

  The number of different `2 × 2`squares in `matrix`.

```
int differentSquares(std::vector<std::vector<int>> matrix) {
    int count=0, temp=0;
    vector<int> squares;
    for(int i = 0; i < matrix.size() - 1; i++)
        for(int j = 0; j < matrix[0].size() - 1; j++ )
        {
            temp = matrix[i][j]*1000 + matrix[i][j+1]*100
                 + matrix[i+1][j]*10 + matrix[i+1][j+1];
            if (!check(temp, squares))
            {
                squares.push_back(temp);
                count++;
            }
        }
    return count;
}
```

```
bool check(int temp, std::vector<int>squares)
{
    for(int i = 0; i < squares.size(); i++)
        if (temp == squares[i]) return 1;
    return 0;
}
```

5. digitsProduct

Given an integer `product`, find the smallest **positive** integer the product of whose digits is equal to `product`. If there is no such integer, return `-1` instead.
Example

- For `product = 12`, the output should be `digitsProduct(product) = 26`;
- For `product = 19`, the output should be `digitsProduct(product) = -1`.

Input/Output

- **[time limit] 500ms (cpp)**
- **[input] integer product**

  Constraints:
  `0 ≤ product ≤ 600`.
- **[output] integer**

```
int digitsProduct(int product) {

      for (int i=1; i < 10000; i++)

      {

            if ( product == multiplication( i ) ) return i;

      }

      return -1;

}
```

```
int multiplication(int n)

{

      if (n == 0) return 0;

      int result = 1;

      while (n > 0)

      {

            result *= n % 10;

            n /= 10;

      }

      return result;

}
```

6. File Naming

You are given an array of desired filenames in the order of their creation. Since two files cannot have equal names, the one which comes later will have an addition to its name in a form of `(k)`, where `k` is the smallest positive integer such that the obtained name is not used yet.
Return an array of names that will be given to the files.

Example

For `names = ["doc", "doc", "image", "doc(1)", "doc"]`, the output should be
`fileNaming(names) = ["doc", "doc(1)", "image", "doc(1)(1)", "doc(2)"]`.
Input/Output

- **[time limit] 500ms (cpp)**
- **[input] array.string names**

  Constraints:
  `5 ≤ names.length ≤ 15`,
  `1 ≤ names[i].length ≤ 15`.
- **[output] array.string**

```
std::vector<std::string> fileNaming(std::vector<std::string>
names) {

    for(int i = 1; i < names.size(); i++)

        for(int j = 0; j < i; j++)

        {

           if (names[ i ] == names[ j ])

           {

               int count = 1;

               while ( notYet(i, count, names)) count++;

               names[ i ] += "(" + to_string(count) + ")";

           }

        }

    return names;     }
```

```
bool notYet(int i, int count, std::vector<std::string> names)

{

    string temp = names[ i ] + "(" + to_string(count) + ")";

    for(int j = 0; j < i; j++)

        if ( names[ j ] == temp ) return 1;

    return 0;

}
```

7. messageFromBinaryCode

You are taking part in an Escape Room challenge designed specifically for programmers. In your efforts to find a clue, you've found a binary code written on the wall behind a vase, and realized that it must be an encrypted message. After some thought, your first guess is that each consecutive `8` bits of the code stand for the character with the corresponding extended ASCII code.
Assuming that your hunch is correct, decode the message.

Example

For `code = "010010000110010101101100011011000110111100100001"`, the output should be `messageFromBinaryCode(code) = "Hello!"`.
The first `8` characters of the code are `01001000`, which is `72` in the binary numeral system. `72` stands for `H` in the *ASCII-table*, so the first letter is `H`.
Other letters can be obtained in the same manner.
Input/Output

- **[time limit] 500ms (cpp)**
- **[input] string code**

  A string, the encrypted message consisting of characters `'0'` and `'1'`.
  Constraints:
  `0 < code.length < 800`.
- **[output] string**

  The decrypted message.

```
std::string messageFromBinaryCode(std::string code) {
    string result="";
    char ch;
    for(int i = 0; i < code.size(); i++)
    {
        if (i%8==0) ch = 0;
        ch *= 2;
        ch += code[i]-48;
        if (i%8==7)
        {
        printf("%c ", ch);
        result.push_back(ch);
        }
    }
    return result;
}
```

8. spiralNumbers

Construct a square matrix with a size `N × N` containing integers from `1` to `N * N` in a spiral order, starting from top-left and in clockwise direction.

Example

For `n = 3`, the output should be

```
spiralNumbers(n) = [[1, 2, 3],
                    [8, 9, 4],
                    [7, 6, 5]]
```

Input/Output

- **[time limit] 500ms (cpp)**
- **[input] integer n**

  Matrix size, a positive integer.

  Constraints:
  `3 ≤ n ≤ 10.`
- **[output] array.array.integer**

```
std::vector<std::vector<int>> spiralNumbers(int n) {
    vector<vector<int>> s(n,vector<int>(n));
    int i=1, c=0, r=0, count=0;
    while (i <= n*n)
    {
        count++;
        for(c = count-1; c <= n-count ; c++)
        {
             s[r][c] = i;
             printf("%d ",i);
             i++;
        }
        c--;

        for(r = count; r <= n-count ; r++)
        {
            s[r][c]=i; printf("%d ",i);
            i++;
        }
        r--;

        for(c = n-count-1; c >= count-1; c--)
        {
            s[r][c]=i; printf("%d ",i);
            i++;
        }
        c++;

        for(r = n-count-1; r >= count ; r--)
        {
            s[r][c]=i; printf("%d ",i);
            i++;
        }
        r++;
    }
    return s;
}
```

9. Sudoku

Sudoku is a number-placement puzzle. The objective is to fill a `9 × 9` grid with digits so that each column, each row, and each of the nine `3 × 3` sub-grids that compose the grid contains all of the digits from `1` to `9`.
This algorithm should check if the given grid of numbers represents a correct solution to Sudoku.

Example

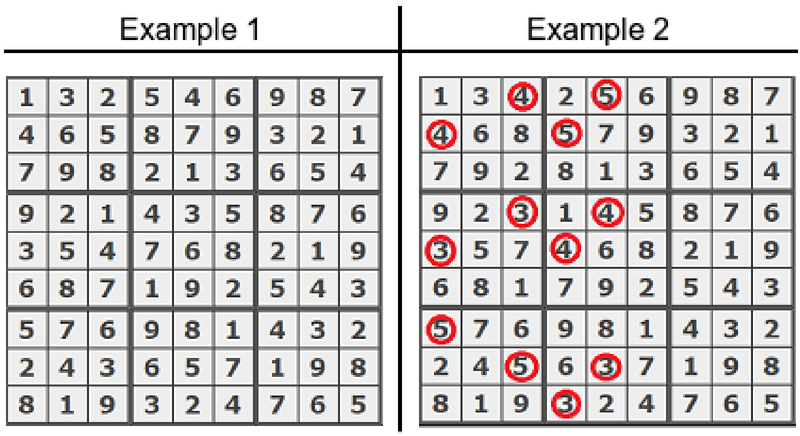
For the first example below, the output should be `true`. For the other grid, the output should be `false`: each of the nine `3 × 3` sub-grids should contain all of the digits from `1` to `9`.

Input/Output

- **[time limit] 500ms (cpp)**
- **[input] array.array.integer grid**

  A matrix representing `9 × 9` grid already filled with numbers.
- **[output] boolean**

  `true` if the given grid represents a correct solution to Sudoku, `false`otherwise.



Example 1

1	3	2	5	4	6	9	8	7
4	6	5	8	7	9	3	2	1
7	9	8	2	1	3	6	5	4
9	2	1	4	3	5	8	7	6
3	5	4	7	6	8	2	1	9
6	8	7	1	9	2	5	4	3
5	7	6	9	8	1	4	3	2
2	4	3	6	5	7	1	9	8
8	1	9	3	2	4	7	6	5

Example 2

1	3	4	2	5	6	9	8	7
4	6	8	5	7	9	3	2	1
7	9	2	8	1	3	6	5	4
9	2	3	1	4	5	8	7	6
3	5	7	4	6	8	2	1	9
6	8	1	7	9	2	5	4	3
5	7	6	9	8	1	4	3	2
2	4	5	6	3	7	1	9	8
8	1	9	3	2	4	7	6	5

```
bool markCol[ 9 ][ 10 ] = {{0}}, markRow[ 9 ][ 10 ] = {{0}},
markSquare[ 3 ][ 3 ][ 10 ] = {{{0}}};


bool sudoku(std::vector<std::vector<int>> grid) {
        for(int i = 0; i < 9; i++)
                for(int j = 0; j < 9; j++)
                {
                        if ( !check(i, j, grid[ i ][ j ]))  return 0;
                        mark(i, j, grid[ i ][ j ]);
                }
        return 1;
}
```

```
void mark(int r, int c, int v)
{
        markCol[ c ][ v ] = 1;
        markRow[ r ][ v ] = 1;
        markSquare[ r/3 ][ c/3 ][ v ] = 1;
}
```

```
bool check(int r, int c, int v)
{
        if (markCol[ c ][ v ] == 1) return 0;
        if (markRow[ r ][ v ] == 1) return 0;
        if (markSquare[ r/3 ][ c/3 ][ v ] == 1) return 0;
        return 1;
}
```