《编程能力综合测试》试题

> 学期：2023-2024学年秋季学期

说明

- 数据量极小，可以使用暴力算法求解。
- 允许使用STL（可以直接`#include <bits/stdc++.h>`）。
- 允许定义全局变量。
- 基本考查内容包括：排序、深度优先搜索、动态规划等。
- 考场提供dev、VS2012、sublime，但在命令行无法使用g++进行编译。
- 整数均在`int`范围内。
- 输入输出中，同一行内不同数字均按空格分开。

第一题（30分）

题干

给定一个长度为N的整数数组，对每个数按以下规则执行操作，共执行K次：

- 若该数小于2，则加上50；
- 若该数大于50，则取相反数；
- 其余情况，则加上不大于它的最小质数。

请给出执行完以后的数组。

输入

- 第一行：两个数，分别为N和K。
- 第二行：N个数，代表数组内容。

输出

- 仅一行，N个数，代表每个数执行K次操作以后的结果。

样例

- 输入：

```
5 2
19 53 1 -23 100
```

- 输出：

```
75 -3 -51 50 -50
```

- 解释：

第一轮操作结果：

- 第三轮操作结果：`75 -3 -51 50 -50`。

源码

```cpp
#include <bits/stdc++.h>
using namespace std;

int findPN(int a) {
    for (int i = a; i >= 2; i--) {
        for (int j = i - 1; j >= 2; j--) {
            if (i % j == 0) {
                break;
            }
            else if (j == 2) {
                return i;
            }
        }
    }
}

int main() {
    int n, k;
    cin >> n >> k;

    int num[n];
    for (int i = 0; i < n; i++) {
        cin >> num[i];
    }

    for (int i = 0; i < k; i++) {
        for (int j = 0; j < n; j++) {
            if (num[j] < 2) {
                num[j] += 50;
            }
            else if (num[j] > 50) {
                num[j] = -num[j];
            }
            else {
                num[j] += findPN(num[j]);
            }
        }

        // for (int i = 0; i < n; i++) {
        //  cout << num[i] << " ";
        // }
    }

    for (int i = 0; i < n; i++) {
        cout << num[i] << " ";
    }

    return 0;
}
```

第二题（30分）

题干

给定一个长度为N的整数数组，其中数字范围为$[0,100]$。对数组进行以下规则的排序：

- 以数组第一位为界，数组内所有大于它的，按降序排在数组开头；
- 数组第一位；
- 除数组第一位外，数组内所有小于等于它的，按升序排在后面。

请给出排序完以后的数组。

输入

- 第一行：一个数，代表数组长度N。
- 第二行：N个数，代表数组内容。

输出

- 仅一行，N个数，代表完成规定排序以后的结果。

样例

- 输入：

  ```
  7
  ```

  ```
  51 39 92 26 28 80 51
  ```

- 输出：

  ```
  92 80 51 26 28 39 51
  ```

源码

- 不使用STL的最暴力算法：

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    int n = 100000;
    cin >> n;

    int num[n];
    for (int i = 0; i < n; i++) {
        cin >> num[i];
    }

    int below[100000] = {0};
    int above[100000] = {0};
    int belowSize = 0;
    int aboveSize = 0;

    for (int i = 0; i <= num[0]; i++) {
        for (int j = 1; j < n; j++) { // skip num[0]
            if (num[j] == i) {
                below[belowSize] = num[j];
                belowSize++;
            }
```

```
        }
    }

    for (int i = 100; i > num[0]; i--) {
        for (int j = 1; j < n; j++) { // skip num[0]
            if (num[j] == i) {
                above[aboveSize] = num[j];
                aboveSize++;
            }
        }
    }

    for (int i = 0; i < aboveSize; i++) {
        cout << above[i] << " ";
    }

    cout << num[0] << " ";

    for (int i = 0; i < belowSize; i++) {
        cout << below[i] << " ";
    }

    return 0;
}
```

- 使用STL：

```
#include <bits/stdc++.h>
using namespace std;

int main() {
    int n = 100000;
    int numMetric;
    cin >> n >> numMetric;

    vector<int> below;
    vector<int> above;
    int temp;
    for (int i = 1; i < n; i++) {
        cin >> temp;
        if (temp > numMetric) {
            above.push_back(temp);
        }
        else {
            below.push_back(temp);
        }
    }

    sort(above.begin(), above.end()); // try using operator overloading!
    reverse(above.begin(),above.end());
    sort(below.begin(), below.end());

    for (int i : above) {
        cout << i << " ";
    }

    cout << numMetric << " ";

    for (int i : below) {
        cout << i << " ";
    }

    return 0;
```

```
}
```

第三题（30分）

题干

给定一个长度为 `sLength` 的字符串 `s` ，并规定 `s` 的“子串”为：任意删去 `s` 中 `0` 至 `sLength` 个字符得到的新字符串。现再给定一个长度为 `tLength` 的字符串 `t` ，求 `s` 有多少种方法能得到和 `t` 完全一样的子串。

字符串内容均为区分大小写的字母。

输入

- 第一行：两个数，分别为 `sLength` 和 `tLength` 。
- 第二行：一个字符串，代表 `s` 。
- 第三行：一个字符串，代表 `t` 。

输出

- 仅一个数，代表待求的方法数。

样例

- 输入：

  `12 3`

  `baggbagbabag`

  `bag`

- 输出：

  `15`

- 解释：

  `BAGgbagbabag` 、 `BAgGbagbabag` 、 `BAggbaGbabag` 、 `BAggbagbabaG` 、
  `BaggbAGbabag` 、 `BaggbAgbabaG` 、 `BaggbagbAbaG` 、 `BaggbagbabAG` 、
  `baggBAGbabag` 、 `baggBAgbabaG` 、 `baggBagbAbaG` 、 `baggBagbabAG` 、
  `baggbagBAbaG` 、 `baggbagBabAG` 、 `baggbagbaBAG` 。

源码

```cpp
#include <bits/stdc++.h>
using namespace std;

int solutions = 0;
int sLength = 100000;
int tLength = 100000;
string s;
string t;

void findT(int nextStart = 0, int solved = 0) {
    for (int i = nextStart; i < sLength; i++) {
        if (s[i] == t[solved]) {
            // cout << i << " " << solved << " " << t[solved] << endl;
```

```
            if (solved + 1 == tLength) {
                solutions ++; // no need to find more, but need to update
solutions
            }
            else {
                findT(i + 1, solved + 1); // must find more
            }
        }
    }
}

int main() {
    cin >> sLength >> tLength >> s >> t;
    findT();
    cout << solutions;
    return 0;
}
```

第四题（10分）

题干

给定一个维度为N、只包括1和0两个数的方阵，定义“块”为通过上、下、左、右任一维度包含1而连接起来的图形，且图形中1的个数称为块的面积。例如：$\begin{bmatrix}1 & 1 & 0\\0 & 1 & 0\\1 & 0 & 1\end{bmatrix}$的左下、右下角均有面积为1的块$[1]$，而上方有面积为3的块$\begin{bmatrix}1 & 1\\ & 1\end{bmatrix}$。

对于输入的方阵，允许进行最多1次操作，将一个0转为1。求在完成输入和操作后，方阵理论上可能达到的最大块的面积。

输入

- 第一行：一个数，代表维度N。$N \leq 500$。
- 其后N行，每行N个数：代表输入的方阵。

输出

- 仅一个数，代表最大块面积的值。

样例

- 样例1：
  - 输入：

    `3`

    `1 1 0`

    `0 1 0`

    `1 0 1`
  - 输出：

    `6`

    - 解释：选择第三行第二列做操作，可将三个分离的块连结起来，得到最大块，其面积为6。
- 样例2：
    - 输入2：

      `2`

      `1 1`

      `1 1`
    - 输出2：

      `4`
    - 解释：不做任何操作，就已经能找到理论上最大的块，其面积为4。

源码

```
#include <bits/stdc++.h>
using namespace std;

int n = 500;
int block[500][500];
int maxSizeReached = 0;
int currentSize = 0;
bool checked[500][500] = {false};

void findBlock(int i, int j) {
    checked[i][j] = true;
    currentSize ++;
    if (currentSize >= maxSizeReached) {
        maxSizeReached = currentSize;
    }

    // cout << i << " " << j << " " << currentSize << " " << maxSizeReached <<
endl;

    if (i >= 1 && block[i - 1][j] == 1 && checked[i - 1][j] == false) {
        findBlock(i - 1, j);
    }

    if (i <= n - 2 && block[i + 1][j] == 1 && checked[i + 1][j] == false) {
        findBlock(i + 1, j);
    }

    if (j >= 1 && block[i][j - 1] == 1 && checked[i][j - 1] == false) {
        findBlock(i, j - 1);
    }

    if (j <= n - 2 && block[i][j + 1] == 1 && checked[i][j + 1] == false) {
        findBlock(i, j + 1);
    }
}


int main() {
    cin >> n;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            cin >> block[i][j];
        }
    }
```

```
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            // cout << "starting from: " << i << " " << j << endl;
            // cout << "====================" << endl;
            currentSize = 0;
            for (int ii = 0; ii < n; ii++) {
                for (int jj = 0; jj < n; jj++) {
                    checked[ii][jj] = false;
                }
            }

            if (block[i][j] == 0) {
                block[i][j] = 1;
                findBlock(i, j);
                block[i][j] = 0;
            }

            else {
                findBlock(i, j);
            }
        }
    }

    cout << maxSizeReached;
}
```