中山大学本科生期中考试

考试科目：《程序设计II实验》

学年学期：2024-2025 学年第 2 学期

开课单位：计算机学院

考试方式：上机闭卷

考试时间：100 分钟

【A7】平均分配窗口*

（因存在争议，这题后来改为总分 80，编译分为 20）

题目描述

现在需要从多个队伍中选择编号是 **素数** 的成员，一次性返回 `N` 个窗口，每个窗口含有 `K` 个成员（即窗口大小为 `K`）。参考样例如下（注意队伍数量和窗口数量不一定相等）：

输入

N:4 K: 7

队伍1	2	47	97	37	19	5	7	127	13	107
队伍2	59	73	101	61	103	29	3	23	79	31
队伍3	113	67	83	89	71	11	6	43	53	17
队伍4	41	109	4							

处理

窗口1	窗口2	窗口3	窗口4
2	47	97	37
59	73	101	61
113	67	83	89
41	109	19	5
7	127	103	29
3	23	71	11
43	53	13	107

2 59 113 41 7 3 43 47 73 67 109 127 23 53 97 101 83 19 103 71 13 37 61 89 5 29 11 107

选择策略：

- 每个成员都有一个编号，只有是素数的成员可以进入各个窗口。
- 各个队伍的成员需要做穿插处理。
    1. 首先，队伍 1 的成员是 `2 47 97 37 19 5 7 127 13 107`，分别选择 `2 47 97 37` 这四个素数成员到窗口 1、窗口 2、窗口 3、窗口 4。
    2. 然后，队伍 1 的下一个队伍，即队伍 2 的成员是 `59 73 101 61 103 29 3 23 79 31`，分别选择 `59 73 101 61` 这四个素数成员到窗口 1、窗口 2、窗口 3、窗口 4，此时窗口 1 的成员为 `2 59`，窗口 2 的成员为 `47 73`，窗口 3 的成员为 `97 101`，窗口 4 的成员为 `37 61`。
    3. 依次类推，直到 `N` 个窗口中，每个窗口都有 `K` 个成员。

- 成员尽量均分到 `N` 个窗口。
    1. 例如，队伍 4 只有 `41 109` 两个素数成员，则先分别选择 `41 109` 分配到窗口 1、窗口 2。
    2. 然后，队伍 4 的下一个队伍，即队伍1的剩余成员是 `19 5 7 127 13 107`。由于上次仅为窗口 1、窗口 2分配了成员，因此此次分配从窗口 3 开始。分别选择 `19 5 7 127` 这四个素数成员到窗口 3、窗口 4、窗口 1、窗口 2。
    3. 接着，队伍 1 的下一个队伍，即队伍 2 的剩余成员是 `103 29 3 23 79 31`，分别选择 `103 29 3 23` 这四个素数成员到窗口 3、窗口 4、窗口 1、窗口 2。

参考处理过程

1. **从队伍 1 中选择** `4` 个成员 `2 47 97 37`，分别加入到 `1 2 3 4` 号 4 个窗口中。
2. **从队伍 2 中选择** `4` 个成员 `59 73 101 61`，分别加入到 `1 2 3 4` 号 4 个窗口中。
3. **从队伍 3 中选择** `4` 个成员 `113 67 83 89` ，分别加入到`1 2 3 4` 号 4 个窗口中。
4. 由于队伍 4 没有 4 个可选择的成员，因此 **从队伍 4 中选择** `2` 个成员 `41 109`，分别加入到 `1 2` 号 2 个窗口中。
5. **从队伍 1 中选择** `4` 个成员 `19 5 7 127`，分别加入到 `3 4 1 2` 号 4 个窗口中。
6. **从队伍 2 中选择** `4` 个成员 `103 29 3 23`，分别加入到 `3 4 1 2` 号 4 个窗口中。
7. **从队伍 3 中选择** `4` 个成员 `71 11 43 53`，分别加入到 `3 4 1 2` 号 4 个窗口中。
8. 队伍 4 没有可选择的成员，跳过。
9. **从队伍 1 中选择** `2` 个成员 `13 107`，分别加入到 `3 4` 号 2 个窗口中。此时所有窗口已满，分配结束。

输入描述

第一行输入为 `N`，表示需要输出的窗口数量，取值范围为 `[1，10]`
第二行输入为 `K`，表示每个窗口需要的成员数量，取值范围为 `[1,100]`
之后的行数不定（行数取值范围 `[1,10]` **）**，每行表示一个队伍的成员列表，成员之间以空格分隔，每个队伍的成员数量取值范围为 `[1,100]`

输出描述

依次输出每个窗口的成员，成员之间以空格分隔，**成员数量=窗口数量*窗口大小**，如上文参考样例所示。

实现要求

现在已经实现了部分的读入操作和输出结果的操作，将对 `int** result` 打印，请你实现：

- `void Implementation::constructNewLine();` 动态扩展存储结构并添加一个新的“行”。

1. 读入队伍时，由于初始容量较小，当前行数（`lineCount`）达到容量上限（`capacity`）时，令容量上限为原来的 2 倍。
2. 扩容涉及三个数组：
   - `lineCapacity`：存储每行的容量
   - `valCount`：存储每行的当前成员数量
   - `val`：二维数组，存储实际数据
3. 添加新行：
   - 为新行分配初始空间 `NormalSize`。
   - 初始化新行的容量 `lineCapacity` 为 `NormalSize`，将元素计数 `valCount` 置为 0。
   - 增加总行数计数 `lineCount`。

- `void Implementation::insert(int val)`：这个函数在当前的最末尾行的最末尾添加值 `val`。
  - 当最末尾行的成员数量 `valCount` 达到最末尾行的容量上限 `lineCapacity` 时，令容量上限为原来的 2 倍。
- `int Implementation::stringToInt(const std::string &input)` 的输入为 `std::string` 类型数据，将其转换为整型数字。
- `int **initialResult(int n, int k)` 的输入为窗口数量 `n` 和每个窗口含有的成员数量 `k`。输出为 `n` × `k` 的二维数组。
- `void process(Implementation &imp, int n, int k, int **result)`：
  1. 输入为 `Implementation` 对象，窗口数量 `n` 和每个窗口含有的成员数量 `k`。
  2. 执行操作，最终将结果存放到 `result` 这个二维数组中。

输入样例

所有的输入最终以 `exit` 为末尾结束。

```
4 7
2 47 97 37 19 5 7 127 13 107
59 73 101 61 103 29 3 23 79 31
113 67 83 89 71 11 6 43 53 17
41 109 4
exit
```

输出样例

```
2 59 113 41 7 3 43 47 73 67 109 127 23 53 97 101 83 19 103 71 13 37 61 89 5 29
11 107
```

提示

1. **每个列表会保证元素数量满足窗口要求，不需要考虑元素不足情况**
2. **每个列表的元素已去重，不需要考虑元素重复情况**
3. **每个列表的元素列表均不为空，不需要考虑列表为空情况**
4. **输出结果要保证不改变同一个列表的元素顺序**
5. **每个列表的元素数量可能是不同的**

getline

`getline()` 是一个用于从输入流中读取一行文本的函数。

```
std::istream& getline(std::istream& is, std::string& str, char delim);
```

参数：

- `is`：输入流对象（例如 `std::cin` 或文件流）。
- `str`：存储读取内容的目标字符串。
- `delim`（可选）：分隔符，默认为换行符（`\n`）。当读取到该字符时，停止读取。

返回值：

- 返回输入流对象 `is`，可以用于链式调用或检查流状态。

cin.ignore

`std::cin.ignore()` 是一个用于忽略输入流中字符的函数。它通常用于清理输入缓冲区中的多余字符，以避免影响后续输入操作。
在这里表示为默认忽略一个字符。

2. `delim`（可选，默认为 `EOF`），当读取到该字符时，忽略操作会提前结束。

`main.cpp`：

```
#include "implementation.h"
#include <iostream>
#include <string>
```

```cpp
int main() {
  int n, k;
  std::cin >> n >> k;
  std::cin.ignore();
  Implementation imp;

  std::string line;
  // preprocess
  while (std::getline(std::cin, line)) {
    if (line == "exit") {
      break;
    }
    std::string currentNumber;
    imp.constructNewLine();
    for (int i = 0; i < line.length(); i++) {
      if (line[i] == ' ') {
        if (!currentNumber.empty()) {
          imp.insert(imp.stringToInt(currentNumber));
          currentNumber.clear();
        }
      } else {
        currentNumber += line[i];
      }
    }
    if (!currentNumber.empty()) {
      imp.insert(imp.stringToInt(currentNumber));
      currentNumber.clear();
    }
  }



  int **result = initialResult(n, k);

  process(imp, n, k, result);
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < k; j++) {
      std::cout << result[i][j] << " ";
    }
  }
  std::cout << std::endl;

  for (int i = 0; i < n; i++) {
    for (int j = 0; j < k; j++) {
      if (result[i]) {
        delete[] result[i];
        result[i] = nullptr;
      }
    }
```

```
  }

  if (result) {
    delete[] result;
    result = nullptr;
  }
}
```

`implementation.h`：

```
#include <string.h>
#include <string>
#define NormalSize 2
class Implementation {
public:
  inline Implementation() : lineCount(0), capacity(NormalSize) {
    this->val = new int *[NormalSize];
    this->lineCapacity = new int[NormalSize];
    this->valCount = new int[NormalSize];
  }
  ~Implementation();
  void constructNewLine();
  int stringToInt(const std::string &input);
  void insert(int val);

  inline int getLineSize(int idx) { return valCount[idx]; }
  inline int getVal(int i, int j) { return val[i][j]; }
  inline int getLineCount() { return lineCount; }

private:
  int **val;
  int lineCount;
  int capacity;
  int *lineCapacity;
  int *valCount;
};

int **initialResult(int n, int k);

void process(Implementation &imp, int n, int k, int **result);

bool isPrime(int val);
```

`implementation.cpp`：（答案，仅供参考）

编译检查 20%，标准测试 20%，随机测试 40%，内存测试 20%

```cpp
#include "implementation.h"
#include <cmath>
#include <iostream>

// 析构函数释放动态内存
Implementation::~Implementation() {
    for (int i=0; i<lineCount; ++i) {
        delete[] val[i];
    }
    delete[] val;
    delete[] lineCapacity;
    delete[] valCount;
}

// 构造新的一行，如果容量不足就扩容
void Implementation::constructNewLine() {
    if (lineCount == capacity) {
        capacity *= 2;
        int **newVal = new int *[capacity];
        int *newLineCapacity = new int[capacity];
        int *newValCount = new int[capacity];

        for (size_t i=0; i<lineCount; ++i) {
            newVal[i] = val[i];
            newLineCapacity[i] = lineCapacity[i];
            newValCount[i] = valCount[i];
        }
        delete[] val;
        delete[] lineCapacity;
        delete[] valCount;
        val = newVal;
        lineCapacity = newLineCapacity;
        valCount = newValCount;
    }
    val[lineCount] = new int[NormalSize];
    lineCapacity[lineCount] = NormalSize;
    valCount[lineCount] = 0;
    lineCount++;
}

// 插入一个值到当前最后一行
void Implementation::insert(int value) {
    int last = lineCount - 1;
    if (valCount[last] == lineCapacity[last]) {
        lineCapacity[last] *= 2;
        int *newRow = new int[lineCapacity[last]];
        for (int i=0; i<valCount[last]; ++i) {
```

```
            newRow[i] = val[last][i];
        }
        delete[] val[last];
        val[last] = newRow;
    }
    val[last][valCount[last]++] = value;
}

// 字符串转整数
int Implementation::stringToInt(const std::string &input) {
    return std::stoi(input);
}

// 判断素数
bool isPrime(int val) {
    if (val<=1)
        return false;
    for (int i=2; i<=std::sqrt(val); ++i) {
        if (val%i == 0)
            return false;
    }
    return true;
}

// 初始化结果数组
int **initialResult(int n, int k) {
    int **result = new int *[n];
    for (int i=0; i<n; ++i) {
        result[i] = new int[k];
    }
    return result;
}

// 正确的轮转穿插分配逻辑
void process(Implementation &imp, int n, int k, int **result) {
    int *count = new int[n](); // 每个窗口当前人数
    int startWindow = 0;
    int filled = 0;
    int total = n*k;
    int lineCount = imp.getLineCount();
    int *pos = new int[lineCount](); // 每个队伍当前读取位置

    while (filled < total) {
      for (int l=0; l<lineCount && filled<total; ++l) {
            int added = 0;
            for (int j=pos[l]; j<imp.getLineSize(l); ++j) {
                int val = imp.getVal(l, j);
                if (!isPrime(val))
                    continue;
```

```
                while (count[startWindow] == k) {
                    startWindow = (startWindow+1) % n;
                }
                result[startWindow][count[startWindow]++] = val;
                filled++;
                startWindow = (startWindow+1) % n;
                added++;
                pos[l] = j+1;
                if (added == n || filled == total)
                    break;
            }
        }
    }
    delete[] count;
    delete[] pos;
}
```