ПРИКЛАДНІ ПРОБЛЕМИ КОМП'ЮТЕРНИХ НАУК,
БЕЗПЕКИ ТА МАТЕМАТИКИ, 2023, № 2

DOI: (*додає редакція при опублікуванні*)

УДК 004.4'236

АНАЛІЗ ТА ПРОГРАМУВАННЯ МОВОЮ С++ ОЛІМПІАДНИХ ЗАДАЧ НА ВИКОРИСТАННЯ РЕШЕТА ЕРАТОСФЕНА

Глинчук Л.Я. (ORCID: 0000-0002-8943-9604)
Волинський національний університет імені Лесі Українки

ANALYSIS AND PROGRAMMING IN C++ LANGUAGE OF THE OLYMPIC PROBLEMS USING THE GRID OF ERATOSTHENES

Hlynchuk L. Ya.
Lesya Ukrainka Volyn National University

Abstract. The article illustrates the solution of Olympiad problems in programming according to a certain scheme, which has several stages. Namely: analysis of the condition of the problem, drawing up a plan for solving the problem, building a mathematical model and solution scheme, implementing the algorithm, testing and debugging, submitting the solution. For analysis, programming and demonstration, problems of varying complexity were selected for the use of prime number search algorithms, i.e. the Sieve of Eratosthenes. For the first task, all the described stages were completed and analyzed, for the second – all stages, except for the last one, because for this task it is not possible to send the task for evaluation. The third task is analyzed and programmed without stages. But especially the last stage is demonstrated, because the evaluation result is displayed differently than in problem 1. The result of the work shows the correct approach to solving Olympic programming problems. The execution of the last stage, the submission of the solution, was done using the evaluation tool available on the platform https://www.eolymp.com/uk/problems.

Keywords: Olympic task, programming, mathematical model, algorithm implementation, prime number.

Щорічне змагання школярів чи студентів, таке як олімпіада з програмування, полягає в написанні розв'язків певних алгоритмічних задач на дозволених мовах програмування. Олімпіади можуть проводитися на рівні школи, району, області, ЗВО, Всеукраїнські, а також міжнародні. Кожна олімпіада проводиться в декілька етапів, а тоді з числа переможців IV етапу Всеукраїнської олімпіади формується команда для участі у Міжнародній олімпіаді. [1]

Олімпіади з програмування суттєво відрізняються від інших, і як зрозуміло, в обов'язковому порядку завдання реалізовуються на комп'ютері. Розв'язок задачі являє собою реалізовану програму, яка вирішує поставлену задачу для різних вхідних даних, що відповідають певному формату та виводить отриманий результат. Рівень складності завдань виходить за межі традиційної програми, оскільки розв'язання задачі потребує глибоких знань не тільки з конкретної мови програмування, а й знань спеціальних розділів алгебри і геометрії, теорії чисел, дискретної математики (теорії графів, операції з багаторозрядними числами, комбінаторики), теорії ігор та ін. [1]

Та чи потрібно таке, бо ж на це витрачається багато часу? Участь в олімпіадному програмуванні тренує швидко знаходити ідеї, алгоритми та розв'язки задач, які, на перший погляд, здаються нерозв'язними або дуже складними. Для цього потрібно вміти аналізувати умову задачі, підбирати методи розв'язання задач серед відомих, вести пошук невідомих розв'язків, обрати оптимальний шлях розв'язання, аналізувати складність роботи того чи іншого алгоритму тощо. [1]

Для того аби потренуватися розв'язувати олімпіадні задачі з програмування можна використовувати Інтернет-портали з інтерактивними архівами задач та онлайн змаганнями (наприклад, www.olymp.vinnica.ua або https://new.netoi.org.ua/, https://www.eolymp.com/uk/ та

інші). Тут можна не тільки прочитати умову задачі, але й перевірити розв'язки і відразу ж отримати результат перевірки. На цих ресурсах ведуться рейтинги учасників, проводяться онлайн змагання. [1] Наприклад, ресурс https://www.eolymp.com/uk/ дає можливість створити групу, запросити туди користувачів, яких обирає викладач, після створення групи, можна з бази вибрати задачі та організувати змагання. Кожен користувач групи може відправити розв'язок: вказати назву задачі (номер), вибрати зі списку мову програмування, вписати код програми для цієї задачі, якщо в задачі були використані файли, то поставити галочку біля «розв'язок використовує файли для читання та запису» і натиснути кнопку «відправити». Таким способом система прийме розв'язок, автоматично перевірить на правильність та на дотримання вимог по ліміту часу та ліміту використання пам'яті. Кожен користувач буде бачити чи зараховано завдання чи ні, має можливість приймати участь в обговореннях до задачі та інше.

Згідно [2] вважається, що розв'язання олімпіадної задачі з програмування містить дещо незвичайну роботу, а саме, аналітичну та розумову. Пошук відповіді на розв'язання таких задач не лежить на поверхні, а зрозуміло, що процес творчий і, попередньо, кожен учасник повинен добре володіти інструментами, які доведеться використати. Особливістю є ще і те, що єдиного підходу до таких задач знайти не можливо, але з досвіду практики, можна сказати, що є прийоми та методи, які можна використати для того аби навчитися. І, що цікаво, інколи в таких випадках не обійтися без інтуїції.

Для початку потрібно розібратися в умові задачі і уважно подивитися на вхідні дані та їх обмеження, які вказує задача, іншими словами, детально проаналізувати. Автор у [2] вказує, що кожна олімпіадна задача має розв'язуватися в декілька етапів. Отже, виділяють: 1-ий етап – аналіз умови задачі, 2-й етап – складання плану розв'язку задачі, 3-й етап – побудова математичної моделі та схеми розв'язку, 4-й етап – реалізація алгоритму, 5-й етап – тестування та відлагодження, 6-й етап – здача розв'язку. Детальніше про кожен з цих етапів можна прочитати у [2]. Оскільки тепер, найчастіше олімпіади з програмування проводяться онлайн, то можна здати на перевірку розв'язок і система все перевірить сама на своїх підібраних тестах. Таким чином є можливість уникнути помилок, пов'язаних із неправильним форматом виведення результату. Вважається, що відправлення розв'язку на перевірку є одним із самих важливих та відповідальних моментів роботи на олімпіаді, тому потрібно ще раз перечитати умову задачі і все перевірити.

Враховуючи усе вищеописане, при підготовці до олімпіади, слід відразу користуватися наведеними етапами, щоб показати якнайкращий результат.

Метою роботи є вдосконалення алгоритмів розв'язування олімпіадних задач згідно етапів на використання простих чисел (решета Ератосфена); програмування та тестування результатів.

Розглянемо, проаналізуємо та запрограмуємо декілька задач, які використовують решето Ератосфена.

Задача 1. Решето Ератосфена. За заданими числами a та b вивести усі прості числа з інтервалу від a до b включно. Вхідні дані: два числа a та b ($1 \le a \le b \le 100000$). Вихідні дані: ввести в одному рядку усі прості числа з інтервалу від a до b включно. Для перевірки можна використати подані вхідні дані # 1: 2 2, вихідні дані # 1: 2. [3]

Для розв'язування задачі пройдемо всі етапи, які описані вище, та скористаємося системою перевірки, яка є на https://www.eolymp.com/uk/ для визначення правильності та зарахування відповіді.

І так, згідно першого етапу, потрібно проаналізувати умову задачі. Задача полягає у знаходженні просто числа за алгоритмом решета Ератосфена. Алгоритм відбору простих чисел за решетом Ератосфена описаний у багатьох джерелах, наприклад, у [4], [5] та ін., тому

детально розписувати немає змісту. Бачимо, що вхідні дані, то два числа a та b, які задовольняють вказану умову $1 \le a \le b \le 100000$. Тому від'ємні числа на вхід поступати не можуть, оскільки, є верхня та нижня межа чисел і вони додатні. Звертаємо увагу, що вхідні числа можуть бути рівні між собою. Вихідні дані то рядок усіх простих чисел з вказаного інтервалу, зрозуміло, що виводити потрібно через пропуск. Задача полягає у виведенні не усіх простих чисел, а простих чисел з інтервалу, який може починатися не з 1, а з будь якого *a*, ось це головна суть виконання.

2-й етап, складаємо план розв'язку задачі, в цьому випадку робимо більший нахил до короткого опису алгоритму:

- побудуємо масив булевих значень для усіх чисел з проміжку від 1 до *b*;
- відберемо усі складені числа з проміжку від 1 до *b*, а у масиві булевих значень на їх позиції поставимо значення "хибно";
- виведемо усі позиції істинно масиву булевих значень з проміжку від *a* до *b*.

Згідно 3-го етапу, потрібно побудувати математичну модель та схему розв'язку. Тому за планом з етапу 2 опишемо детальніше алгоритм розв'язування. Математичною моделлю в даному випадку виступає сам алгоритм відбору простих чисел за решетом Ератосфена. Але уточнимо деякі моменти. Як уже було сказано в першому кроці плану, масив булевих значень будуємо для усіх чисел з проміжку від 1 до *b*, але на першому місці буде значення хибно, бо число 1 не відноситься ні до простих, ні до складених, решту заповнюємо масив значенням істинно. Другий крок – відбір складених чисел: згідно джерела [5] спочатку відбираються числа які кратні 2, далі 3 і т.д., але для проміжку від $i = 2$ до *b* встигають перевіритися лише числа *i*i* і не більші, цей момент слід врахувати при програмуванні завдання. У [5] сказано, що на практиці можна поліпшити алгоритм відбору таким чином, щоб починати викреслювати складені числа відразу з *i*i*, бо всі складені числа менші за число *i* вже будуть викреслені до цього часу. Останній крок важливий, бо умова задачі каже, що не усі прості числа покажи, а лише з вказаного інтервалу, тому цикл буде починатися з *a* і закінчуватися *b*. Виводити потрібно не значення булевого масиву, а позиції де у масиві ще лишилися значення істинно.

Переходимо до 4-го етапу, реалізовувати розв'язування задачі будемо на мові програмування C++ з врахуванням усіх описаних деталей. Код програми буде мати вигляд, рисунок 1.

5-й етап тестування та відлагодження. При тестуванні програми потрібно пройтися по граничних умовах, а потім уже тестувати із звичайними числами. Отже:

- при $a = 0$, $b = 10$ нічого виводитися не повинно, бо *a* виходить за вказану умову;
- при $a = 1$, $b = 1$, теж нічого виводитися не повинно, бо число 1 не відноситься ні до простих, ні до складених;
- при $a = 2$, $b = 2$, виводиться число 2, то є вірний результат, бо в умові сказано, що числа можуть бути рівні, і результатом дійсно є перше просте число 2;
- при $a = 6$, $b = 6$, нічого виводитися не повинно, бо число 6 складене;
- при $a = 1$, $b = 100001$ нічого виводитися не повинно, бо *b* виходить за вказану умову;
- при $a = 8$, $b = 10$, нічого виводитися не повинно, бо на цьому інтервалі не має простих чисел;
- при $a = 1$, $b = 10$, виводиться результат 2, 3, 5, 7, це є правильний результат, бо викреслилися числа кратні 2 – 4, 6, 8, 10, кратне 3 – 9 і все, кратне 5 вже і так було викреслено, а кратні 7 виходять за межі вказаного інтервалу;
- при $a = 201$, $b = 300$, виводяться прості 211 223 227 229 233 239 241 251 257 263 269 271 277 281 283 293, результат можна звірити з вказаним у джерелі [4] і т. д.

На етапі тестування дописали в код умову, яка перевіряє, чи задовольняють введені інтервали вказані у задачі умови, якщо так, тоді шукай прості числа за алгоритмом.

```
#include <iostream>
using namespace std;
int main ()
{ int a, b, i;
  cin >> a >> b;
  if (a>=1 and b<=100000)
 {bool *tf = new bool[b+1] ;
  for (i = 2; i <= b ; i++)
      tf[i] = true ;
  tf[1] = false ;

  for (i = 2; i*i <= b ; i++)
      if (tf[i])
         for (int j = i*i ; j <= b ; j+=i)
             tf[j]= false ;

  for (i = a ; i <= b ; i++)
      if (tf[i]) cout << i << " " ;
 }
}
```

Рисунок 1 – Програма до задачі 1

Переходимо до 6-го етапу, спробуємо завантажити наш код у вибрану систему і перевірити виконання. Заходимо у https://www.eolymp.com/uk/ вибираємо вкладку «Задачі», у пошуку вводимо запит «решето Ератосфена». Бачимо наступне вікно (рисунок 2.).

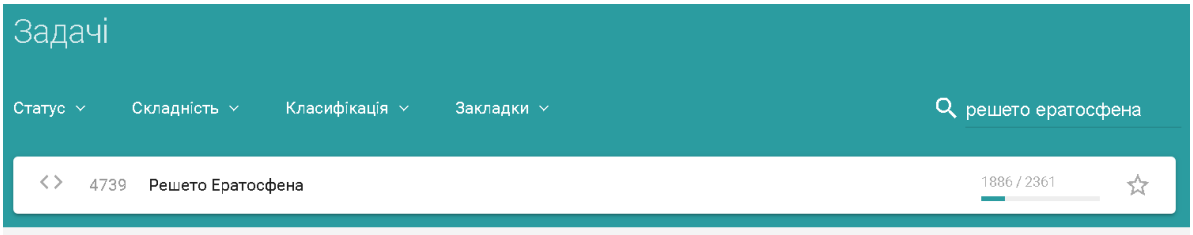

Рисунок 2 – Знайдена задача

Вибираємо дану задачу, заходимо у вкладку «Умова» і уважно перечитуємо умову задачі, вхідні та вихідні дані, дивимося приклад тестування. Якщо все так як у нас, то переходимо на вкладку «Розв'язки», з'являється вікно як на рисунку 3.

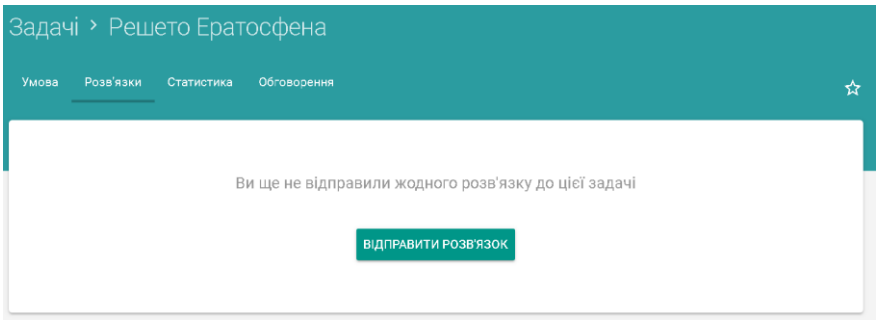

Рисунок 3 – Вкладка «Розв'язки»

Натискаємо кнопку «Відправити розв'язок» (рисунок 4.), заповнюємо усі поля і натискаємо знизу кнопку «Відправити».

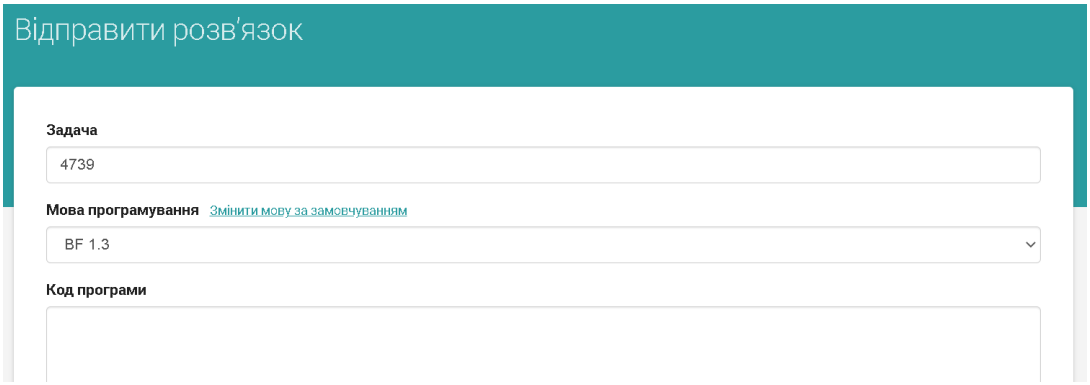

Рисунок 4 – Вікно для відправки розв'язку

Отримали результат (рисунок 5 а та 5 б.):

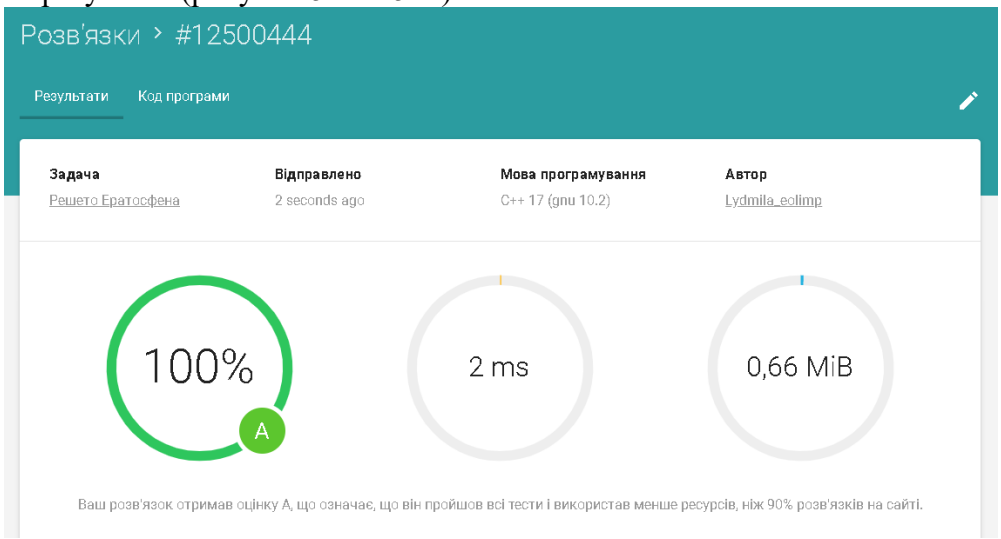

Рисунок 5 а – Результат до задачі перша частина

Test #	Status	Score	Duration	CPU	Memory
✓ Набір тестів #1	Зараховано	100 / 100	2 ms	1 ms	680 KiB
✓ Тест #1	Зараховано	12 / 12	1 ms	1 ms	584 KiB
✓ Тест #2	Зараховано	12 / 12	1 ms	1 ms	580 KiB
✓ Тест #3	Зараховано	12 / 12	1 ms	1 ms	580 KiB
✓ Тест #4	Зараховано	12 / 12	1 ms	1 ms	584 KiB
✓ Тест #5	Зараховано	13 / 13	2 ms	1 ms	680 KiB
✓ Тест #6	Зараховано	13 / 13	1 ms	1 ms	580 KiB
✓ Тест #7	Зараховано	13 / 13	1 ms	1 ms	584 KiB
✓ Тест #8	Зараховано	13 / 13	1 ms	1 ms	588 KiB
		100 / 100	2 ms	1 ms	680 KiB

Рисунок 5 б – Результат до задачі друга частина

У задачі на цьому ресурсі, був вказаний ліміт часу 1 секунда, ліміт використання пам'яті 128 МіВ. Як бачимо, код для задачі вписався в задані ліміти і пройшов 8 тестів. В результаті отримали найкращу оцінку.

На рисунку 5 а зверху справа є білий олівець, він дає можливість відправити на перевірку код програми ще раз, якщо вас не задовольнив результат.

Розглянемо ще одну задачу, яка теж олімпіадна і теж використовує решето Ератосфена. Але, нажаль, код до задачі 2 ми не зможемо перевірити на тому ресурсі, що і задачу 1, бо її там немає, пошуком не знаходить, а назва задачі невідома. Взята вона була з іншого джерела. Проте виконати її згідно етапів, проаналізувати та протестувати, звичайно, зможемо.

Задача 2. Решето Ератосфена – простий алгоритм знаходження всіх простих чисел до деякого цілого числа n, що був створений давньогрецьким математиком Ератосфеном. Алгоритм:

1. Запишемо усі цілі числа від 2 до n включно.
2. Знайдемо найменше число, яке ще не викреслено, і позначим його p; p є простим.
3. Викреслимо p та всі числа, кратні йому, які ще не викреслені.
4. Якщо не всі числа були викреслені, то перейдемо до кроку 2.

Напишіть програму, яка, за заданим n і k, знайде k-те ціле число, яке потрібно викреслити.

Формат вхідних даних: у єдиному рядку задано два числа n і k ($2 \le k < n \le 1000$). Формат вихідних даних: виведіть k-те число, яке потрібно викреслити. Приклади у таблиці 1. [с. 20, 6]

Таблиця 1

Приклади до задачі 2

Вхідні дані	Вихідні дані
7 3	6
15 12	7
10 7	9

У джерелі [5] вказано розбір задачі: «Будемо виконувати всі операції, які записані в умові. Для зручності візьмемо вектор, у який по черзі будемо вписувати числа, в порядку викреслення. Потім виводимо k-1 елемент у векторі, тому що 1-й елемент – 0-вий.». Ідея розв'язування є, зрозуміло, що потрібно виконувати вказаний алгоритм за кроками. Переходимо до етапів.

Етап 1 – аналіз умови задачі. Задача про порядок викреслювання складених чисел згідно алгоритму решета Ератосфена. У вхідних даних задано формат та межі двох чисел, тобто вони чітко повинні задовольняти дані умови. Вихідне дане одне число, тобто k-те число, яке буде викреслено. Уважно потрібно розібратися, що це за k у вхідних даних та що це за k-те число у вихідних даних. Отже, n – то число до якого виписуємо усі цілі числа і включно з ним, k у вхідних даних – то число, яке буде вказувати на порядок викреслення, тобто по іншому правильно назвати індекс по порядку, а вже саме число (його значення) k у вихідних даних. Розглянемо приклад з першими вхідними даними 7 та 3 з таблиці 1. Згідно вказаного в задачі алгоритму, виписуємо всі цілі числа від 2 до 7 включно: 2 3 4 5 6 7. Знаходимо найменше просте і викреслюємо кратні йому, в даному випадку це число 2, тому креслимо 2 4 6, далі найменше просте 3, але кратне йому 6 було викреслено, тому викреслюємо тільки 3. Утворений вектор буде: 2 4 6 3. Далі аналогічно викреслюємо 5 та 7, утворений вектор матиме вигляд: 2 4 6 3 5 7. Тоді, у вхідних даних сказано виведи 3-тє

викреслене число, дивимося у вектор і на 3-му місці стоїть число 6. Дивимося в таблицю 1, дійсно, при вхідних даних 7 та 3 вихідним даним є число 6

Етап 2 – план розв'язку задачі, короткий:

- будуємо масив цілих чисел від 2 до n включно (згідно кроку 1 алгоритму);
- будуємо масив викреслених цілих чисел (за кроками 2, 3, 4);
- виводимо з цього масиву k-те викреслене число.

Етап 3 – математична модель та схема розв'язку. Оскільки, у задачі подано алгоритм, як має відбуватися викреслення цілих чисел, то його можна вважати математичною моделлю за якою знайдеться розв'язок. В даному випадку ми не лишаємо простих чисел, а викреслюємо всі, важливий тільки порядок. Ідея взята з алгоритму решета Ератосфена, тобто вибираємо найменше просте і викреслюємо йому кратні, але в задачі просять знайти k-те викреслене число.

Етап 4 – реалізація алгоритму на мові програмування. Працюємо у мові програмування C++. Уточнимо деякі деталі:

- кількість елементів першого масиву буде n-1, бо нумерація в масиві починається з 0;
- для того, аби вхідні дані задовольняли умову подану в задачі, мусимо її чітко перевірити, тобто мають виконуватися умови (k >= 2 and n > k and n <= 1000);
- масив з викреслених елементів теж буде розміром n-1, логічно, що не можна викреслити більше елементів як є у першому масиві;
- про побудові масиву з викреслених чисел перевіряємо чергове число з першого масиву чи воно кратне (ділиться на просте) і чи воно не 0, якщо умови виконуються, то у перший масив на місце числа, що перевіряється ставимо 0 і так поки не пройдемо кроки 2, 3, 4 алгоритму задачі, тобто весь перший масив стане з нулів;
- як було сказано у джерелі [5], як результат виводимо останній k-1 елемент, бо почали з 0.

Враховуючи вище описані деталі та алгоритм задачі, отримаємо код програми (рисунок 6.).

Етап 5 – тестування та відлагодження. Протестуємо програму:

- при $n = 1$, $k = 1$, не виконуються 2 перші умови для вхідних даних, тому нічого не виведеться;
- при $n = 10$, $k = 10$, не виконується умова для вхідних даних n > k, тому знову нічого не виведеться;
- при $n = 1001$, $k = 10$, не виконується умова для вхідних даних n <= 1000, тому знову нічого не виведеться;
- при $n = 10$, $k = 9$, умови всі виконується, прослідкуємо яке 9-те число має вивестися: перший масив – 2 3 4 5 6 7 8 9 10, масив викреслюваних – 2 4 6 8 10 3 9 5 **7**, а результат 9-те число 7;
- перевіримо по таблиці 1 згідно 2-го рядка, $n = 15$, $k = 12$, програма виводить число 7, перевіримо: перший масив – 2 3 4 5 6 7 8 9 10 11 12 13 14 15, другий масив – 2 4 6 8 10 12 14 3 9 15 5 **7** 11 13, результат дійсно 7.

При граничних значеннях протестували, звичайні значення теж. При відлагодженні забрали код, який виводив проміжні результати (обидва масиви), тепер виводиться тільки те, про що просить задача. Можна вважати, що програмний код пройшов етап тестування та відлагодження.

Етап 6 – здача розв'язку: в даному випадку завантажити в систему не має можливості, тому цей етап опускаємо. Але враховуючи детальний аналіз та проведене тестування можна сміливо вважати, що задача виконана і результат виводиться вірний.

```
int main()
{   int n, k;
    cin >> n >> k;
    cout << endl;
    int ysi[n-1];
    if (k>=2 and n>k and n<=1000)
    { int poch = 2;
      for(int i = 0; i < n-1; i++){
          ysi[i] = poch;
          poch = poch+1;
    }
    int p=2;
    int vukr[n-1];
    int j=0;
    while (p<=n) {
    for (int i = 0; i < n-1; i++)
    {if (ysi[i] % p == 0 and ysi[i] != 0)
        {vukr[j]=ysi[i];
         ysi[i]=0;
         j=j+1;
        }
    }
    p=p+1;
    }
    cout << vukr[k-1] << endl;}
}
```

Рисунок 7 – Програма до задачі 2

Розглянемо ще одну олімпіадну задачу, але порівняно з 2-ма попередніми вона найпростіша, тому детальний аналіз згідно етапів опустимо, але цю задачу можна перевірити в системі, що і виконаємо.

Задача 3. На вході програми маємо натуральне число n (n > 1). Потрібно перевірити чи задане число – просте, тобто ділиться тільки на 1 та n. Вхідні дані: натуральне число n ($1 < n < 2^{31}$). Вивести 1, якщо число n просте і 0 у протилежному випадку. [7]

Ідея виконання: для того аби визначити чи задане число n просте чи складене потрібно перевірити чи воно має більше дільників як 1 та n. Для цього потрібно починати перевірку з 2, бо 1 і так зрозуміло, якщо задане число ділиться хоча б на одне з проміжку від 2 до n (n – не включно, бо так само як і з 1) то воно має більше дільників як 2 і його можна вважати складеним. Перевіряти числа на які ділиться n теж не потрібно усі, аби зменшити кількість перевірок достатньо перевірити для i = 2 до i * i < n. Для позначення чи знайшовся ще дільник відмінний від 1 та n використаємо прапорець булевої змінної, якщо знайшовся то в прапорець ставимо значення хибно, бо задане число перестало бути простим.

Програмна реалізація буде виглядати як на рисунку 7.

```
#include <iostream>
using namespace std;
int main()
{   int n;
    cin >> n;
    bool f = true;
    if ( 1 < n )
    {    for (int i = 2; i*i < n; i++)
         if (n % i == 0)
         f = false;

    if (f == true)
        cout << "1";
    else
        cout << "0";
    }
}
```

Рисунок 7 – Програма до задачі 3

Дану задачу завантажили та перевірили на ресурсі https://www.eolymp.com/uk/ і отримали результати, рисунки 8 а і 8 б.

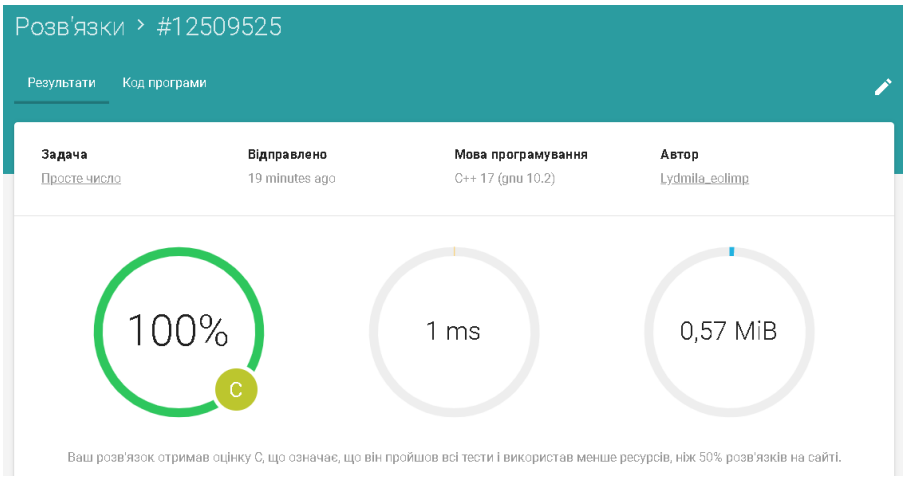

Рисунок 8 а – Результат до задачі 3 перша частина

Test #	Status	Score	Duration	CPU	Memory
✓ Набір тестів #1	Зараховано	100 / 100	1 ms	1 ms	584 KiB
✓ Тест #1	Зараховано	10 / 10	1 ms	1 ms	576 KiB
✓ Тест #2	Зараховано	10 / 10	1 ms	1 ms	584 KiB
✓ Тест #3	Зараховано	10 / 10	1 ms	1 ms	576 KiB
✓ Тест #4	Зараховано	10 / 10	1 ms	1 ms	580 KiB
✓ Тест #5	Зараховано	10 / 10	1 ms	1 ms	584 KiB
✓ Тест #6	Зараховано	10 / 10	1 ms	1 ms	584 KiB
✓ Тест #7	Зараховано	10 / 10	1 ms	1 ms	580 KiB
✓ Тест #8	Зараховано	10 / 10	1 ms	1 ms	584 KiB
✓ Тест #9	Зараховано	10 / 10	1 ms	1 ms	576 KiB
✓ Тест #10	Зараховано	10 / 10	1 ms	1 ms	584 KiB
		100 / 100	1 ms	1 ms	584 KiB

Рисунок 8 б – Результат до задачі 3 друга частина

Вказані на ресурсі ліміти: ліміт часу 1 секунда, ліміт використання пам'яті 64 МіВ. Дана програмна реалізація в ресурсні ліміти помістилася, але результати гірші, як у задачі 2, система виставила оцінку С. Проте всі тести програмна реалізація пройшла. При бажанні можна перепрограмувати код до задачі і завантажити новий, натиснувши на олівець зверху справа. Якщо взяти код перевірки числа на простоту з джерела [8], для мови програмування С++ та завантажити його на ресурс https://www.eolymp.com/uk для перевірки, то результати отримаємо аналогічні рисункам 8 а та 8 б.

Підготовка здобувачів освіти до участі в олімпіадах та змагання із спортивного програмування із використанням сучасних платформ для проведення дистанційних олімпіад вимагає оновлення та вдосконалення алгоритмів розв'язування здавалося б класичних задач. У даній роботі авторами запропоновано вдосконалені алгоритми розв'язування задач, які використовують алгоритм пошуку простих чисел. По новому зроблено аналіз усіх вибраних задач (відповідно до вибраної тематики) та показано результати такого аналізу з завантаженням на ресурс https://www.eolymp.com/uk для перевірки. Схеми роз'язування таких задач адаптовано до випадків, коли тестування здійснюється із використанням автоматичних засобів та на розв'язок накладаються обмеження за часом розв'язання та об'ємом використаної пам'яті.

Бібліографія

1. Жуковський С. С. Педагогічні умови підготовки обдарованих школярів до олімпіад з інформатики : Київ. 2013. URL: http://eprints.zu.edu.ua/13961/1/08_dis_Жуковський_20_10_13__xx1.Pdf
2. Методичні рекомендації щодо розв`язання олімпіадних задач з програмування. URL: https://www.eolymp.com/en/blogs/posts/32#:~:text=Власний%20досвід%20автора%20дає%20можливість,здача%20розв'язку%20на%20перевірку
3. Задачі. Решето Ератосфена. URL: https://www.eolymp.com/uk/problems/4739
4. Прості числа. Решето Ератосфена. Дослідницька робота учнів 6 класу. *Освітній проект «На Урок» для вчителів*. URL: https://naurok.com.ua/prosti-chisla-resheto-eratosfena-doslidnicka-robota-uchniv-6-klasu-18502.html
5. Решето Ератосфена - алгоритм знаходження простих чисел. *www.mathros.net.ua - Сайт для студентів спеціальності інформатика*. URL: https://www.mathros.net.ua/resheto_erastofena.html
6. Горошко Ю., Мельник В., Міца О. ПРО ТУРНІРИ ЮНИХ ІНФОРМАТИКІВ. *Науково-методичний фаховий журнал «Комп'ютер у школі та сім'ї»*. 2017. № 8 (144). URL: https://vlapinsky.at.ua/CSF2017/CSF_8_2017i.pdf
7. Матвійчук С. Задачі. Просте число. URL: https://www.eolymp.com/uk/problems/8929

References

1. Zhukovsky S. S. Pedagogical conditions for preparing gifted students for computer science olympiads: Kyiv. 2013. URL: http://eprints.zu.edu.ua/13961/1/08_dis_Жуковський_20_10_13__xx1.Pdf
2. Methodical recommendations for solving Olympic programming problems. URL: https://www.eolymp.com/en/blogs/posts/32#:~:text=Власний%20досвід%20автора%20дає%20можливість,здача%20розв'язку%20на%20перевірку
3. Tasks. Sieve of Eratosthenes. URL: https://www.eolymp.com/uk/problems/4739
4. Simple numbers. Sieve of Eratosthenes. Research work of 6th grade students. *Educational project "On Lesson" for teachers.* URL: https://naurok.com.ua/prosti-chisla-resheto-eratosfena-doslidnicka-robota-uchniv-6-klasu-18502.html
5. Sieve of Eratosthenes - an algorithm for finding prime numbers. *www.mathros.net.ua - Site for students majoring in computer science.* URL: https://www.mathros.net.ua/resheto_erastofena.html
6. Horoshko Y., Melnyk V., Mitsa O. ON TOURNAMENTS FOR YOUNG COMPUTER SCIENTISTS. *Scientific and methodical professional magazine "Computer in school and family".* 2017. No. 8 (144). URL: https://vlapinsky.at.ua/CSF2017/CSF_8_2017i.pdf
7. Matviychuk S. Tasks. Prime number. URL: https://www.eolymp.com/uk/problems/8929