Міністерство освіти і науки України
Чернівецький національний університет
імені Юрія Федьковича

Тетяна КАРАВАНОВА

Олімпіадна інформатика

Навчальний посібник

Чернівці
Чернівецький національний університет
імені Юрія Федьковича
2024

УДК 004.021(075.8)
К 210

Друкується за ухвалою вченої ради
Чернівецького національного університету імені Юрія Федьковича
(протокол №8 від 27.05.2024 р.).

Рецензенти:

Міца О.В., доктор технічних наук, доцент, зав. кафедри інформаційних управляючих систем та технологій ДВНЗ «Ужгородський національний університет»;

Чернікова Л.А., кандидат педагогічних наук, доцент, проректор з навчально-методичної роботи комунального закладу «Запорізький обласний інститут післядипломної педагогічної освіти» Запорізької обласної ради.

Караванова Т.П.

К 210 Олімпіадна інформатика : навч. посіб. / Т.П. Караванова. Чернівці: Чернівец. нац. ун-т. ім. Ю. Федьковича. 2024.232 с.

ISBN 978-966-423-864-6

У навчальному посібнику розглядаються питання методики розв'язування олімпіадних задач з інформатики. Усі теми супроводжуються прикладами типових задач, розбір яких дозволить закріпити новий матеріал.

Для студентів вищих навчальних закладів спеціальностей «Середня освіта (Інформатика)», «Середня освіта (Математика)» та інших педагогічних спеціальностей, учнів та викладачів загальноосвітніх навчальних закладів.

УДК 004.021(075.8)

ISBN 978-966-423-864-6

Передмова

Існує достатня кількість фахової літератури, в якій розглядаються алгоритми, що є базовими для розв'язування олімпіадних задач з інформатики. До переліку такої літератури входять також і збірники задач, що пропонувалися на різних олімпіадних змаганнях досить високого рівня, які містять не тільки умови задач, але й їх розбір та алгоритми розв'язків. Кожний тренер та олімпіадник знає високу цінність такої літератури.

Однак, шлях олімпіадника, як і його тренера, починається з простих задач, які повинні зацікавити, привабити своєю неординарністю, показати особливість олімпіадних задач на простих прикладах, для яких на перших кроках ще не потрібні знання складних алгоритмів.

Саме таке завдання намагається вирішити даний навчальний посібник, який складається з 9 розділів.

Перші 8 розділів містять теоретичний матеріал, який базується на розгляді конкретних прикладів задач з повним їх розбором: побудовою математичної моделі, описом оптимального алгоритму, покрокового його виконання, розгляду основних фрагментів кодів програм з відповідними коментарями, визначенням оцінки складності побудованого алгоритму.

Для розв'язування алгоритмічних задач в якості інструментарію запропоновані мови програмування C та C++.

Останній 9 розділ збірника містить 50 задач олімпіадного характеру. Ці задачі представлені у трьох підрозділах в такому порядку: умови задач, рекомендації щодо їх розв'язання, тексти програм мовами C та C++. Розбір і розв'язання цих задач та подібних до них дасть можливість користувачам посібника застосувати всі набуті знання і значно підняти свій професійний рівень. Однак, для розв'язання деяких задач з цього розділу тем, розглянутих у посібнику, замало. Варто звернутися до більш серйозної літератури, про яку йшлося на початку і де розглядаються питання методів складання алгоритмів.

Тетяна Караванова

Тематичні та організаційні особливості олімпіад з інформатики

Методика розробки та реалізації алгоритмів

Розв'язання будь-якої проблеми передбачає певні етапи своєї реалізації. Це стосується і розв'язування алгоритмічних задач. Умовно цю проблему можна поділити на два етапи: розробка алгоритму сформульованої задачі та його реалізація засобами програмування.

Перший етап передбачає теоретичне розв'язання задачі, а саме визначення базових методів та розробку власних алгоритмів, необхідних для отримання коректного розв'язку.

Другий етап не менш відповідальний і полягає у представленні розробленого алгоритму у вигляді коду програми, скориставшись обраним середовищем програмування.

Саме на другому етапі розв'язання алгоритмічної задачі є змога переконатися, що розроблений алгоритм є коректним і дає правильний результат виконання задачі.

Розглянемо таке поняття як ***техніка програмування*** або ще можна сказати про ***технологію розв'язування*** алгоритмічних задач. Для цього перш за все слід визначити цілий комплекс умов їх реалізації. Термін «техніка» означає процес, сукупність засобів, шлях до отримання очікуваного результату. Часто до поняття техніки включається також поняття «технологія», що трактується як сукупність способів розв'язання різноманітних задач.

Таким чином, почнемо з головного питання: що треба знати та вміти, якими навичками треба володіти, щоб можна було стверджувати готовність учня щодо розв'язування олімпіадних задач з інформатики.

Почнемо з переліку тих складових, які забезпечують успішне розв'язання алгоритмічних задач, детальніше визначимо сутність кожної з них та обґрунтуємо їх значення.

Особливості олімпіадних задач

Умова задачі

Розв'язання будь-якої задачі починається з ознайомлення з її умовою. Умови олімпіадних задач з інформатики мають свої особливості та відрізняються від тих, які учням пропонуються на уроках інформатики при ознайомленні з розділом алгоритмізації та програмування.

Перше, що необхідно зазначити, так це сюжетність умов задач, в яких переважно прихована математична модель, тобто явним чином не вказується на використання того чи іншого математичного апарату.

На відміну від звичайних задач, які розв'язують учні на уроках інформатики за програмою шкільного курсу, в умовах олімпіадних задач з інформатики використовується різноманітна сценарна тематика – наукова, соціальна, літературна тощо. Вони практично не містять у явному вигляді математичних формул, у них не зазначається алгоритмічна тематика, тобто належність до того чи іншого розділу алгоритмізації.

Саме тому на першому етапі розв'язування алгоритмічної задачі необхідно зняти «обгортку» умови, тобто прибрати сюжетну лінію, визначити інформаційну та математичну модель задачі, що у свою чергу надасть доступ до наступного етапу її розв'язання, а саме до визначення алгоритмічних методів, які дозволять побудувати коректний алгоритм сформульованої задачі.

Переважно текст умови олімпіадної алгоритмічної задачі буває досить громіздким, що дещо ускладнює його читання і розбір, однак при цьому містить досить багато корисної інформації, якою у жодному разі не можна знехтувати.

Наведемо приклад однієї з таких задач, що пропонувалася учасникам ІІ етапу Всеукраїнської олімпіади з інформатики 2022/2023 рр. (мал.1).

II етап Всеукраїнської олімпіади з інформатики 2022-2023
Україна, 4-те грудня 2022

Задача F. До чого ЗНО доводить...

Назва вхідного файлу:	`standard input`
Назва вихідного файлу:	`standard output`
Ліміт часу:	1 second
Ліміт використання пам'яті:	256 megabytes

Поки Петрик ходив розважався у кіно — інші студенти активно вчились. Так, після хвилі ностальгії по старим часам, Даня з Діаною вирішили позмагатись у вирішенні двох останніх та улюблених ними задач із ЗНО з математики: стереометрії та параметра. Для обох студентів випадковим чином були згенеровані набори завдань. Таким чином Даня отримав n наборів задач по a_i задач в кожному, а Діана отримала m наборів задач по b_i в кожному. За умовами змагання, учасники мають послідовно розв'язувати свої набори завдань починаючи від першого.

Усе було б чудово, якби люди не мали фізичних обмежень, тож, на жаль, кожен з учасників має фіксовану максимальну ефективність e, яка дорівнює кількості задач, яку той може зробити за одну годину. Крім того, втомлюючись люди починають думати повільніше, тому щогодини ефективність учасника зменшується рівно на 1, поки не дійде до нуля.

Щоб компенсувати цю несправедливість життя, кожен раз як учасник повністю вирішує набір задач, то він одразу наливає собі святковий келих "хербатки"(від пол. herbata, чай) та завдяки ньому відновлює свою ефективність до максимуму. Зверніть увагу, що якщо в цей момент мало відбутись зниження ефективності, то воно не відбувається. Переможцем змагання стає учасник, що розв'язав найбільшу кількість задач.

Петрик послухав це все і вирішив зробити ставку на переможця. Допоможіть, будь ласка, Петрику визначитись, хто стане переможцем змагання, та, який буде фінальний рахунок вирішених задач, якщо обидва учасники хочуть перемогти.

Формат вхідних даних

Перший рядок містить два цілі числа n, m ($1 \leqslant n, m \leqslant 2 \cdot 10^5$).
Другий рядок містить n цілих чисел a_i ($1 \leqslant a_i \leqslant 10^9$).
Третій рядок містить m цілих чисел b_i ($1 \leqslant b_i \leqslant 10^9$).
Четвертий рядок містить одне ціле число e ($1 \leqslant e \leqslant 10^9$)

Формат вихідних даних

У першому рядку виведіть ім'я переможця («`Danya`» або «`Diana`») або «`Draw`» у випадку нічиї.
У другому рядку через двокрапку виведіть фінальний рахунок: кількість задач розв'язаних переможцем та кількість задач розв'язаних його опонентом.

Приклади

standard input	standard output
2 3 2 1 3 4 5 2	Diana 6:3
3 2 4 8 3 4 2 5	Danya 15:6
2 2 3 2 3 2 2	Draw 5:5

Мал.1.

Що можна помітити з першого погляду у цій умові задачі? Сценарність присутня, відсутні математичні формули, будь-які натяки на використання тих чи інших алгоритмів. Однак є інформація про вхідні та вихідні дані, якими оперуватиме розроблений алгоритм, тестові приклади цих даних, в умові задачі є підказки щодо необхідності розроблення ефективного

алгоритму. Початку тексту умови задачі передує інформація про назви вхідного та вихідного файлів, обмеження на час виконання розробленої програми та обмеження на використання нею пам'яті комп'ютера.

Ще однією особливістю олімпіадних задач є обов'язкове задання коректних вхідних даних у тестах, що перевіряють коректність роботи програми учасника і які відповідають умові задачі. А це означає, що немає потреби у додатковій перевірці коректності уведення вхідних даних. Прикладом вхідних та вихідних даних є тести, наведені в умові задачі.

Яка саме скористатися цією інформацією, а чим у жодному разі не можна знехтувати, розглянемо далі.

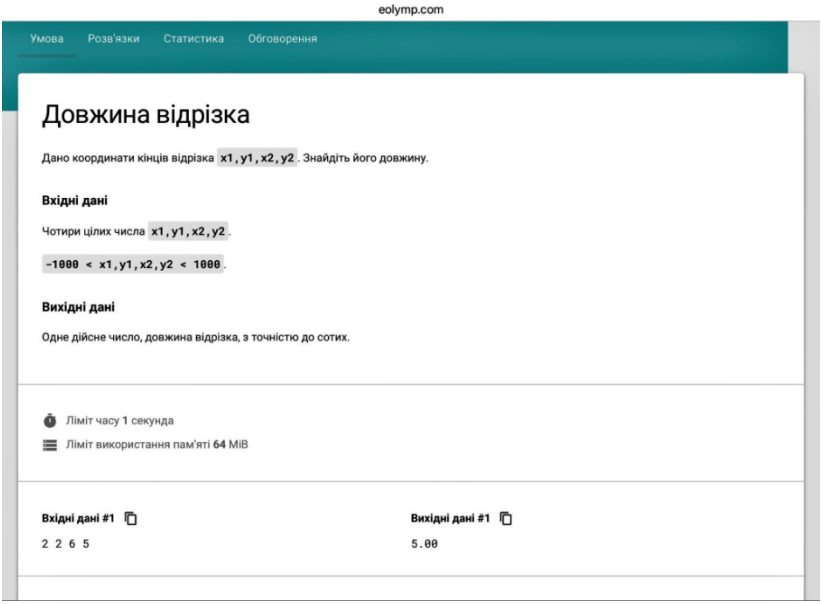
Типи даних

Як зазначалося вище, умови практично всіх олімпіадних задач з інформатики містять достатню кількість корисної інформації, яка може бути використана під час її розв'язання.

До такої інформації у першу чергу слід віднести обмеження на допустимі значення, які задаються для вхідних даних. За цими значеннями можна визначити типи змінних, принцип уведення вхідної інформації, подекуди навіть натяк на алгоритм, який найефективніше дозволить розв'язати задачу.

У якості приклада розглянемо одну з нескладних задач з сайту **groups.eolymp.com** (мал.2). У цій задачі математична модель прописана досить явно, тобто зрозуміло, що необхідно застосувати формулу обчислення відстані між двома точками на площині, які задані своїми координатами. Саме це визначає побудову математичної моделі задачі.

Наступним кроком є реалізація алгоритму задачі, що відповідає математичній моделі у вигляді програми обраною мовою програмування. Одним із перших кроків при написанні коду будь-якою мовою програмування є визначення кількості змінних, які братимуть участь в обчисленнях, та їх типів. Саме для цього необхідно звернутися до умови задачі і проаналізувати інформацію, щодо типа значень (числові або символьні) та діапазона зміни їх значень.



Мал.2.

У нашій здачі зазначено, що координати точок на площині є цілими числами, а значення задаватимуться в діапазоні [-1000..1000]. Отже, необхідно визначити мінімальний за обсягом пам’яті цілий тип, який дозволить задавати такі значення координат (табл.1).

Табл.1

Вид	**Назва**	**Діапазон**	**Обсяг байтів**
Символьний	**unsigned char**	0 .. 255	1
	char	-128 .. 127	1
Цілі числа	**unsigned int**	0 .. 65535	2
	short int	-32768 .. 32767	2
	Int	-32768 .. 32767	2
	unsigned long	0 .. 4 294 967295	4

	long	-2 147 483 648.. 2 147 483647	4
	long long або __int64	–9 223 372 036 854 775 808 … 9 223 372 036 854 775 807	8
	unsigned __int64	0 … 18446744073709551615	8
Дійсні числа	**float**	3.4 * (10e-38) .. 3.4 * (10e+38)	4
	double	1.7 * (10e-308) .. 1.7 * (10e+308)	8
	long double	3.4 * (10e-4932) .. 1.1 * (10e+4932)	10

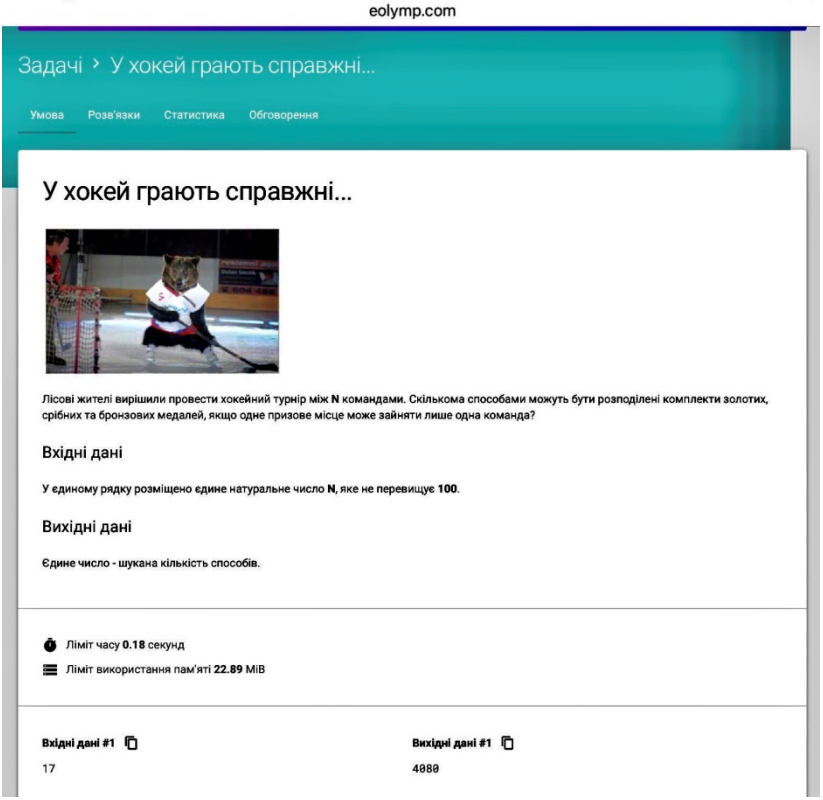
Як бачимо, найоптимальніший тип, що займає 2 байти та дозволяє працювати зі значеннями у діапазоні від -32768 до 32767, є тип **short int**. Звичайно, можна зауважити, що для даної задачі немає ніякого значення вибір оптимального за обсягом пам'яті типу змінних і це буде беззаперечно. Однак, якщо би ми в іншій задачі мали справу з великою послідовністю елементів, то це питання стало би дуже актуальним!

Проаналізуємо вихідні дані даної задачі. В умові зазначено, що результатом обчислення буде дійсне число і вказана точність, з якою необхідно вивести результат. Перший висновок – для результуючої змінної необхідно обирати тип серед дійсних. Другий – у табл.1 є три дійсних типи, однак аналіз вхідних значень підказує, що результат обчислень не виходитиме за діапазон зміни значень типа **float**.

Розглянемо ще одну задачу (мал.3). Вхідною інформацією є кількість команд, тобто значення, яке не може бути а ні від'ємним, а ні дісним. Окрім цього в умові задачі сказано, що його значення натуральне і обжене числом 100. Це означає, що

кількість команд не може бути 0, тобто $1 \le N \le 100$, і для цієї змінної оптимальніше підійде тип **char** або **unsigned char**.

Завданням задачі є обчислення кількості способів розподілення призових місць. І ось тут варто детальніше зупинитись на аналізі інформації, що прихована в умові даної задачі.



Мал.3.

Перше – це визначення типу результуючої змінної. Якщо у попередній задачі вихідна інформація описувалася досить явним чином, то з поточної умови відомо лише, що результатом є єдине число, що визначає кількість способів розподілення призових місць. Зрозуміло, що кількість може бути лише цілою і додатною! А от у якому діапазоні можуть знаходитись

обчислені результати? Мова йде про кількість варіантів, що наводить на думку відношення цієї задачі до комбінаторних задач, де у формулах присутні факторіали, що у свою чергу з невеликим зростанням значення *N* дають значне зростання результату! Те, що наші міркування спрямовані у правильному напрямі, підтверджують такі фактори з умови задачі:

— діапазон зміни значення *N* досить невеликий;

— у тестовому прикладі при вхідному значенні 17 результат сягає значення аж 4080!

Висновок може бути таким: для результуючої змінної необхідно обирати цілий беззнаковий тип з найбільшим обсягом байтів **unsigned long** або **unsigned __int64**. Однак, слід зазначити, що треба переконатися, чи передбачені вказані типи у компіляторі, яким ви користуєтесь. Про це детальніше мова йтиметься трохи пізніше.

Однак, може статися і так, що навіть обраний вами тип з найбільшим обсягом байтів не дає коректного результату. Таку проблему можна помітити під час тестування програми на великих значеннях вхідних даних: результати на якомусь етапі можуть давати від'ємні значення або явним чином не зростати. У розділі «Довга арифметика» детально розглядаються питання, чому це відбувається та як можна вийти з цієї ситуації. Якщо коротко, то у разі, коли під час використання арифметичних операцій результат виходить за межі визначених типів, необхідно застосовувати довгу арифметику, тобто розглядати цілі додатні числа як одновимірні масиви цифр і реалізовувати звичайні алгоритми додавання та множення чисел.

На початку розгляду питання про визначення типів даних зазначалося, що цей процес подекуди навіть дає можливість отримати певні підказки щодо алгоритму розв'язання задачі. І саме у задачі, яку ми розглядаємо, варто на це звернути увагу.

Віднесення задачі до комбінаторного типу, як вже зазначалося вище, вимагає обчислення значень факторіалів. Але відомо, що обмеження на діапазони зміни значень цілочислових типів навіть з найбільшим обсягом в байтах **unsigned __int64**(8б.) не дозволяють обчислити факторіал більше, ніж для *N*=20 (21!=51090942171709440000>18446744073709551615), а за

умовою задачі максимально можливе значення N=100. З цього можна зробити висновок, що використання формул комбінаторики не вирішує проблеми у побудові алгоритму даної задачі. Необхідно шукати інший шлях до побудови математичної моделі і відповідно алгоритму задачі, покроково знаходити залежність між вхідними даними і вихідними.

Подібні алгоритми розглядаються у наступному розділі посібника «Степеневі та факторіальні залежності. Рекурентні послідовності».

Введення та виведення інформації

Всі олімпіадні задачі з інформатики передбачають роботу з вхідними та вихідними даними, структура яких, послідовність значень та формат введення і виведення обов'язково зазначені в умові задачі.

Подекуди в умовах задач зазначаються назви вхідного та вихідного файлів (мал.1). Це пов'язано з особливостями перевірки роботи розробленого алгоритму, для чого застосовуються спеціально розроблені середовища перевірки олімпіадних задач.

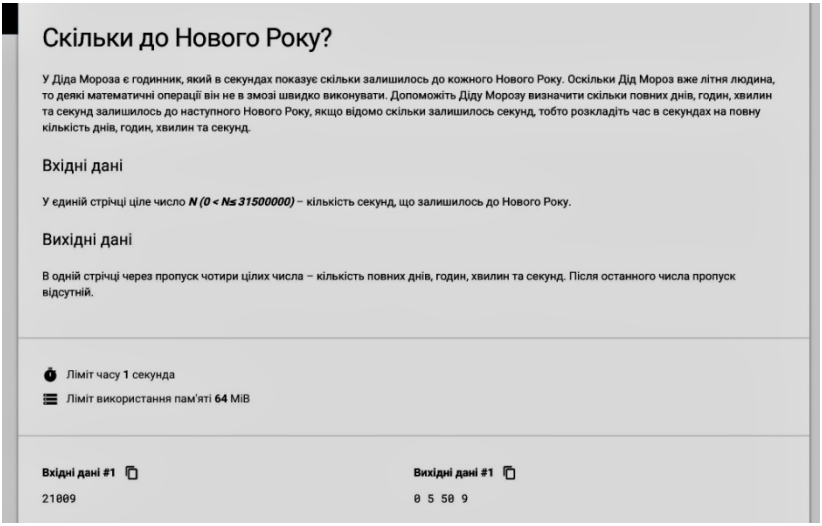
Призначення цих середовищ – це компіляція представлених учасниками кодів програм з використанням зазначеного ними компілятора та спрямовування його на читання відповідного вхідного файлу з розробленим авторським тестом та перевірки посимвольного збігу вихідного файлу учасника з еталонним авторським вихідним файлом.

У разі повного посимвольного збігу обох вихідних текстових файлів учаснику зараховуються бали за даний тест. Кожний тест визначається певною кількістю балів. Загалом до кожної задачі розробляється від десятка до декількох десятків тестів, сумарна кількість балів яких складає вартість всієї задачі. Детальніше про системи перевірки олімпіадних задач йтиметься нижче.

Деякі системи перевірки олімпіадних задач не ставлять таких вимог щодо розміщення вхідної та вихідної інформації. У них використовуються стандартні потоки для читання та виведення інформації.

У будь-якому разі перед початком олімпіади всім учасникам роздаються пам’ятки щодо умов виконання олімпіадних завдань, які відповідають системі перевірки, що використовуватиметься на даній олімпіаді.

Прикладом інформації про вигляд вхідної та вихідної інформації може бути наступний слайд (мал.4) для однієї з задач, розміщеної на сайті **eolymp.com**:



Мал.4.

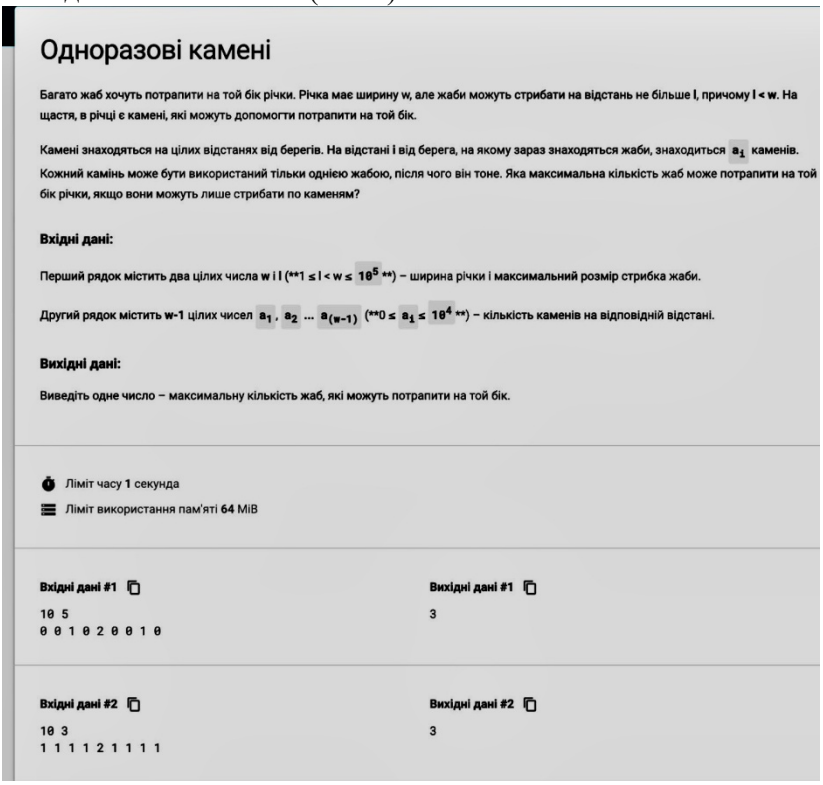
Оскільки мова йде про вхідні та вихідні дані, то звернемо увагу саме на ці фрагменти умови задачі.

Про вхідні дані зазначено, що треба вводити лише одне ціле число визначеного типу.

А от щодо вихідної інформації вимоги чітко виписані: чотири цілих числа через ***пропуск***, але після останнього пропуск робити ***не треба***! Окрім цього чітко зазначено, в якій послідовності мають виводитися значення змінних: дні, години, хвилини, секунди.

Якщо цих вимог не дотриматися, то навіть при ідеально реалізованому алгоритмі задача не буде зарахована, оскільки не буде посимвольного збігу текстового результуючого файлу учасника і авторського тестового файлу.

Розглянемо ще один приклад задачі, запропонованої на сайті **eolymp.com**, у якій у якості вхідних даних задається послідовність елементів (мал.5).



Одноразові камені

Багато жаб хочуть потрапити на той бік річки. Річка має ширину w, але жаби можуть стрибати на відстань не більше l, причому l < w. На щастя, в річці є камені, які можуть допомогти потрапити на той бік.

Камені знаходяться на цілих відстанях від берегів. На відстані i від берега, на якому зараз знаходяться жаби, знаходиться a_i каменів. Кожний камінь може бути використаний тільки однією жабою, після чого він тоне. Яка максимальна кількість жаб може потрапити на той бік річки, якщо вони можуть лише стрибати по каменям?

Вхідні дані:

Перший рядок містить два цілих числа w і l ($1 \le l < w \le 10^5$**) – ширина річки і максимальний розмір стрибка жаби.**

Другий рядок містить w-1 цілих чисел a_1, a_2 ... $a_{(w-1)}$ ($0 \le a_i \le 10^4$**) – кількість каменів на відповідній відстані.**

Вихідні дані:

Виведіть одне число – максимальну кількість жаб, які можуть потрапити на той бік.

Ліміт часу 1 секунда

Ліміт використання пам'яті 64 MiB

Вхідні дані #1	Вихідні дані #1
10 5 0 0 1 0 2 0 0 1 0	3

Вхідні дані #2	Вихідні дані #2
10 3 1 1 1 1 2 1 1 1 1	3

Мал.5.

В описовій частині умови задачі чітко визначається вміст першого рядка вхідної інформації та другого рядка. Значення змінних першого рядка повинні відповідати при читанні інформації їх тематичному змісту, а у другому рядку через пробіл повинно бути стільки додатних цілих чисел, скільки зазначено в умові задачі.

Ще одна особливість даної задачі – це наявність двох тестів. Подекуди автори задач умисне пропонують декілька тестів в умові задачі, щоб продемонструвати певні її нюанси, однак при цьому не натякаючи на алгоритм розв’язання. Часом навіть

буває так, що підбір вхідних даних та відповідні їм результати не є натяками на алгоритм розв'язання задачі. Навпаки, треба глибоко проаналізувати, для чого були запропоновані саме такі тести, яке додаткове інформаційне рмвантаження вони несуть.

Визначення алгоритму

Частково вище ми вже торкнулися питання підказок в умові задачі щодо алгоритму її розв'язання.

Розглянемо ще декілька можливих ситуацій. Наприклад, у деяких задачах передбачається обробка досить великої кількості елементів послідовності вхідних даних, наприклад, 10^9 і більше. Ця обставина може натякати на обробку їх значень під час уведення без запам'ятовування вхідної інформації у масиві. Додатково про підтвердження ідеї покрокового введення і одночасної обробки великої послідовності елементів говоритимемо пізніше, коли обговорюватимемо питання обмеження на пам'ять комп'ютера під час виконання програми.

До задач, у яких можна обійтися без використання масивів, можна віднести задачі накопичення суми та добутку елементів послідовності, пошук мінімального, максимального та заданого значення серед усіх уведених значень послідовності. Причому, ці масиви за умовою задачі можуть бути як одновимірними так і двовимірними.

І, навпаки, невелика кількість вхідних даних, що складає пару десятків елементів, може бути ознакою наявності використання алгоритму повного перебору всіх можливих варіантів для отримання оптимального результату.

Прикладом такого алгоритму може бути задача розкладання N предметів заданої ваги на дві купки максимально однакових за сумарною вагою, задача комівояжера знаходження найвигіднішого маршруту, що проходить через вказані міста по одному разу, якщо відома відстань між будь-якою парою цих міст.

Звичайно, що без знання відомих, розроблених класиками алгоритмів, розв'язання олімпіадних задач не можливе. Таких алгоритмів з різних розділів алгоритмічних задач чимало, знання всіх алгоритмів – це лише справа часу.

Частина цих алгоритмів використовується для розв'язування нескладних олімпіадних задач, а інша для більш серйозних, які пропонуються на олімпіадах високого рівня.

Однак, слід зазначити, що знання класичних алгоритмів, які можна знайти у книжках з олімпіадною тематикою, зовсім не дає гарантій успішного розв'язання навіть найпростіших задач, оскільки важливо не саме знання, а уміння їх доцільно використовувати.

Обмеження на час

У всіх попередньо розглянутих прикладах задач (мал.1,2,4,5) вказаний ще один параметр – ліміт часу 1 секунда. У задачі (мал.3) він ще менший – 0.18 секунд.

Для чого задається цей параметр? Яку корисну інформацію можна отримати, проаналізувавши його значення? Чи може ця інформація натякнути на вибір алгоритму розв'язання задачі?

Виявляється, що так, цей параметр несе в собі дуже багато цінної інформації!

Олімпіадні задачі з інформатики перевіряються на коректність роботи на авторських тестах з врахуванням часу їх виконання. Цей час, як бачимо, зазначається в умові задачі. Якщо на якомусь тесті програма учасника перевищує цей час, то її виконання переривається, відповідно вихідні дані відсутні і тест не зараховується. Швидше за все, у разі невірного або неоптимального алгоритму тести з невеликою кількістю вхідних даних вкладуться у відведений час, а для частини тестів з більшою кількістю вхідних даних їх обробка займе більше часу і задача буде перервана системою перевірки. Відповідно аналіз цих ситуацій повинен бути проведений і продовжена робота над оптимізацією або зміною алгоритма.

Вказаний ліміт часу може також підказати, який з алгоритмів сортування варто застосувати, якщо за умовою задачі необхідне упорядкування елементів. Досвітчені олімпіадники повинні знати, що існують різні за складністю методи сортування, серед них навіть є лінійні, однак область їх застосування дуже специфічна і залежить від особливостей вхідних даних. Можна навіть оцінити приблизний час

виконання того чи іншого алгоритму сортування на вказаних в умові задачі обсягах масивів даних та визначити, який з них найкраще застосувати. А якщо ж цей аналіз дасть негативний результат, то можна припустити, що сортування не потрібне і необхідно міняти напрям пошуку розв'язку задачі.

Можна навести декілька порад щодо оптимізації самого коду програми. Слід зауважити, що у даному разі не йдеться про сам алгоритм, а лише про його реалізацію мовою програмування.

У кожній мові програмування є можливість створювати ***функції*** як окремі підпрограми в основній програмі. Структура мови програмування С взагалі складається з послідовності функцій, серед яких є одна головна з іменем ***main***. Коди, написані цими мовами, мають максимально структурований читабельний вигляд. Але чи є використання функцій оптимальним для олімпіадних задач, коли у пріоритеті стоїть оптимізація за часом їх виконання?

Для того, щоб оцінити переваги та вади використання функцій в олімпіадних задач, слід нагадати процес їх виконання. Сама функція, записана у коді програми, не виконується до того часу, поки не буде здійснено її виклик. Після виклику цієї функції відбуваються наступні дії:

1) її вміст, як міні програми, копіюється у пам'ять комп'ютера;

2) у неї з викликаючої частини програми передаються значення фактичних параметрів;

3) виконуються всі оператори, описані у функції;

4) результати виконання функції передаються у викликаючу частину програми;

5) знищується адреса пам'яті комп'ютера, де розміщувалася дана функція.

Зрозуміло, що на виконання всіх цих п'яти описаних кроків йде певний час роботи програми. Якщо виклик функції є одноразовим або небагатократним, то це не впливає на загальний час виконання програми, але якщо функція викликається у циклі під час обробки N елементів послідовності, то це може мати негативні наслідки.

Говорячи про функції, не можна обійти увагою і питання використання ***рекурсій*** в олімпіадних задачах. Рекурсія – це та сама функція, яка викликає сама себе багатократно. Але, якщо функція після завершення роботи звільняє пам'ять, то рекурсія при кожному наступному виклику залишає у пам'яті свою попередню версію, що значно перевантажує стекову пам'ять комп'ютера. Тому використання рекурсій впливає не тільки на час виконання програми, але й на зайве використання пам'яті. Висновок може бути лише один: доцільність використання рекурсії в програмі повинна бути безумовно доведена!

Більшість умов задач передбачають роботу з масивами даних, що обумовлює використання ***циклів***. Складність програми визначається кількістю вкладених циклів, а саме добутком кількості повторів у кожному з них. Зрозуміло, що зменшення цієї величини призводить до значної оптимізації роботи програми.

При використанні ***вкладених розгалужень***, що мають більше двох вкладень, на саму гору краще виносити умову, ймовірність якої найбільша, на другому рівні має бути умова, що має наступну найбільшу ймовірність і т.д. Останньою залишається випадок з найнижчою ймовірністю зустрічання.

Прикладом може бути алгоритм бінарного пошуку елемента в упорядкованому масиві:

```
L=0;R=n-1; flag=0;
while(L<R && !flag)
{
   m=(L+R)/2;
   if (a[m]<x) L=m+1;
   else if (a[m]>x) R=m-1;
        else flag=1; // Умова a[m]=x має найменшу ймовірність!
}
```

Варто запобігати зайвої перевірки умов в операторах циклів. Наприклад, якщо відомо що шуканий елемент точно є в масиві то замість фрагмента циклу:

```
for (i=0; i<n; i++) if (a[i]==x) k=i;
printf("%d",k);
```

можна використати такий фрагмент:

```
for (i=0; a[i]!=x; i++);
```

printf(“%d”,i);
або такий:
i=0;
while (a[i++]!=x);
printf(“%d”,i);

Відомо, що мови програмування використовують ***бібліотеки,*** які є набором підпрограм та можуть бути використані для виконання конкретної задачі або для розширення функціональності цієї мови. В більшості випадків вони дають змогу програмістам значно скоротити час розробки прогрманого продукту, оскільки можуть використовувати готові компоненти у своїх проектах. Це також зменшує кількість помилок, які можуть виникнути при створенні нового коду, оскільки бібліотеки вже пройшли тестування і перевірку на помилки.

Використання деяких з них обов’язкове, наприклад, такої, як стандартна бібліотека (standard library), що містить переважно механізми введення-виведення даних, структури даних та алгоритми, які часто застосовуються.

Окрім стандартної бібліотеки існують інші спеціальні бібліотеки, у яких підпрограми об’єднані за певною тематикою. Наприклад, у компіляторах мови С існують бібліотека математичних функцій **math.h**, бібліотека **string.h**, що містить функції для роботи з рядками.

Завжди є дуже велика спокуса їх використання. Однак, і тут варто оцінити доцільність цього сервісу. Справа в тім, що підключаючи бібліотеку, яка містить коди не тільки потрібних вам функцій, а і всі решту, що входять до цієї бібліотеки, ви значно збільшуєте обсяг вашої програми. Подальший виклик функції з цієї бібліотеки потребує часу на її пошук в бібліотеці, а далі виконання цієї функції, про яке йшлося вище. Отже, недоцільність використання бібліотек спричиняє не тільки втрати дорогоцінного часу виконання програми, але й втрати пам’яті, про які йтиметься далі.

Наведемо такий приклад. Для знаходження степеня числа 2 можна скористатися функцією **pow**(2,n), підключивши бібліотеку **math.h**, а можна просто написати цикл:

power=1;
for (i=1; i<=n; i++) power*=2;

Відкриємо невеличкий секрет: у бібліотеці **math.h** функція **pow**(2,n) для отримання результату виконує такий самий цикл!

Були випадки, коли використання бібліотечного алгоритму швидкого сортування не проходило за часом, а коли використали власний алгоритм **Qsort** безпосередньо в коді програми, то всі тести були зараховані.

Наведених вище прикладів можна перераховувати ще безліч, у кожного з вас з набуттям досвіду їх буде зв'являтися достатньо. Мабуть, саме тому все зазначене вище можна назвати ***мистецтвом програмування***, яке набувається власним досвідом і практикою.

Для всіх відомих вже розроблених оптимізаційних алгоритмів визначені оцінки складності їх виконання, які характеризуються кількістю виконуваних дій. Наприклад, щодо методів сортування, то прямі методи сортування послідовностей мають квадратичну складність, яка записується як $0(n^2)$ і це означає, що для упорядкування послідовності з n елементів необхідно виконати n^2 порівнянь. Удосконалені методи сортування мають значно кращу оцінку складності, яка визначається як $0(n \log n)$.

Коли розробляється алгоритм олімпіадної задачі, необхідно враховувати як оцінки використаних у ньому відомих алгоритмів, так і власних авторських фрагментів, щоб оцінити, чи вкладеться у відведений ліміт часу програма на максимально можливій кількості вхідних даних.

Обмеження на пам'ять

Про обмеження на пам'ять, яке є обов'язковою характеристикою будь-якої умови олімпіадної задачі, вже неодноразово згадувалося вище.

Нагадаємо про це ще раз тезисно:

1) обмеження на пам'ять разом із діапазонами зміни значень вхідних даних та їх кількості допомагає у визначенні типів даних;

2) зазначений обсяг пам'яті для виконання програми може допомогти у виборі алгоритму розв'язання задачі (доцільність використання масивів, вибір алгоритмів сортування тощо);

3) оптимізація використання циклів та розгалужень в них;

4) доцільність використання рекурсії;

5) доцільність використання бібліотек.

Знання класичних оптимізаційних алгоритмів дозволяє прискорити як час виконання програми за розробленим алгоритмом, так і зменшити використання обсягу пам'яті комп'ютера.

Вибір мови програмування

Зрозуміло, що учень, який зацікавився розв'язуванням олімпіадних задач не повинен зупинятися лише на вивчені тих мов програмування, які входять до шкільного курсу інформатики. Чим більше мов він знає, тим простіше йому буде оптимізувати свою роботу над задачами протягом олімпіадного туру.

На будь-якому етапі олімпіади учасникам пропонуються задачі різного рівня складності. Раніше їх складність можна було впізнати за оціночними балами та упорядкованістю за зростанням цієї складності. Однак останнім часом всі задачі оцінюються однаковою кількістю балів, переважно ця оцінка становить 100 балів і пропонуються вони учасникам в довільній послідовності. Тому складність задач доводиться визначати самому учаснику.

Стратегія виконання завдань, вибір мови програмування залежить від учасника олімпіади. Спробуємо дати декілька порад на прикладі двох мов програмування, які найчастіше обирають олімпіадники, Python та C++/C.

Мова програмування Python дає можливість розробляти дуже компактні і читабельні коди, однак вимагає обов'язкового використання бібліотек, підключення яких значно збільшує використання пам'яті комп'ютера. Мови програмування C++/C стосовно цього питання набагато ефективніша для використання у написанні саме олімпіадних задач. Тому можна порадити вибір такої стратегії: для нескладних задач свій вибір зупинити

на мові Python, оскільки ліміт пам'яті переважно дозволяє використовувати цю мову, а код пишеться компактно і швидко, а для складних задач з великою кількістю даних і з необхідністю максимально оптимізувати як час виконання, так і пам'ять комп'ютера, обирати мови програмування C++/C.

Дехто є прихильником мови програмування Java, яка значно запозичила синтаксис мов C та C++. Однак, Java одержала репутацію «повільної» мови і тому не дуже підходить до розв'язання саме олімпіадних задач. Наприклад, зустрічалися випадки, коли один і той самий код на Java не проходив за часом, а на C++/C успішно долав усі тести.

На останок, ще раз слід зазначити, що вибір мови програмування належить зробити самому олімпіаднику, а це можливо лише за умови його власного накопиченого досвіду щодо участі в олімпіадних змаганнях.

Середовище програмування

Успішним олімпіадником може стати лише той, хто не тільки вміє розробляти чудові алгоритми, але й використовувати всі можливості середовища програмування.

Середовище програмування – це комплекс програм, які надають користувачу достатній набір функцій, що дозволяють мінімізувати час для редагування програми, її налагодження, передбачають пошук синтаксичних та семантичних помилок, реалізують покрокове виконання програми, містять компілятор мови програмування для запуску програми та створення виконуваного коду тощо. Середовище програмування має графічний інтерфейс, який містить набір меню, панелей і вікон для розробки програм.

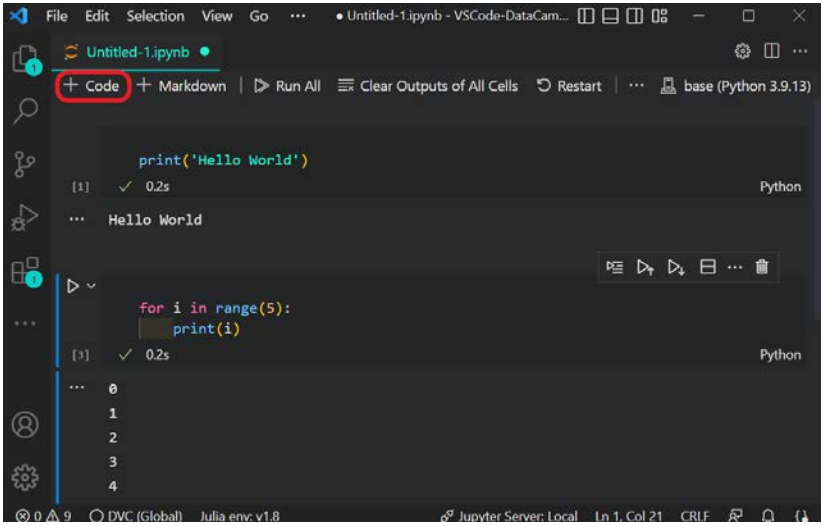
На першому ж кроці набору коду програми треба раціонально використовувати можливості текстового редактора, вбудованого у середовище, щодо копіювання та переміщення, вставлення та вилучення фрагментів тексту, використовуючи при цьому «гарячі» клавіші, що значно прискорює процес створення коду програми.

Не менш важливим є знання і використання можливостей контролю поточних значень змінних та «гарячих» клавіш для

покрокового виконання алгоритму. Саме це заміняє використання в коді фрагментів програми, що призначені для виведення поточних проміжних результатів, які потім необхідно обов'язково закоментовувати і на що витрачається дорогоцінний час роботи над самою програмою.

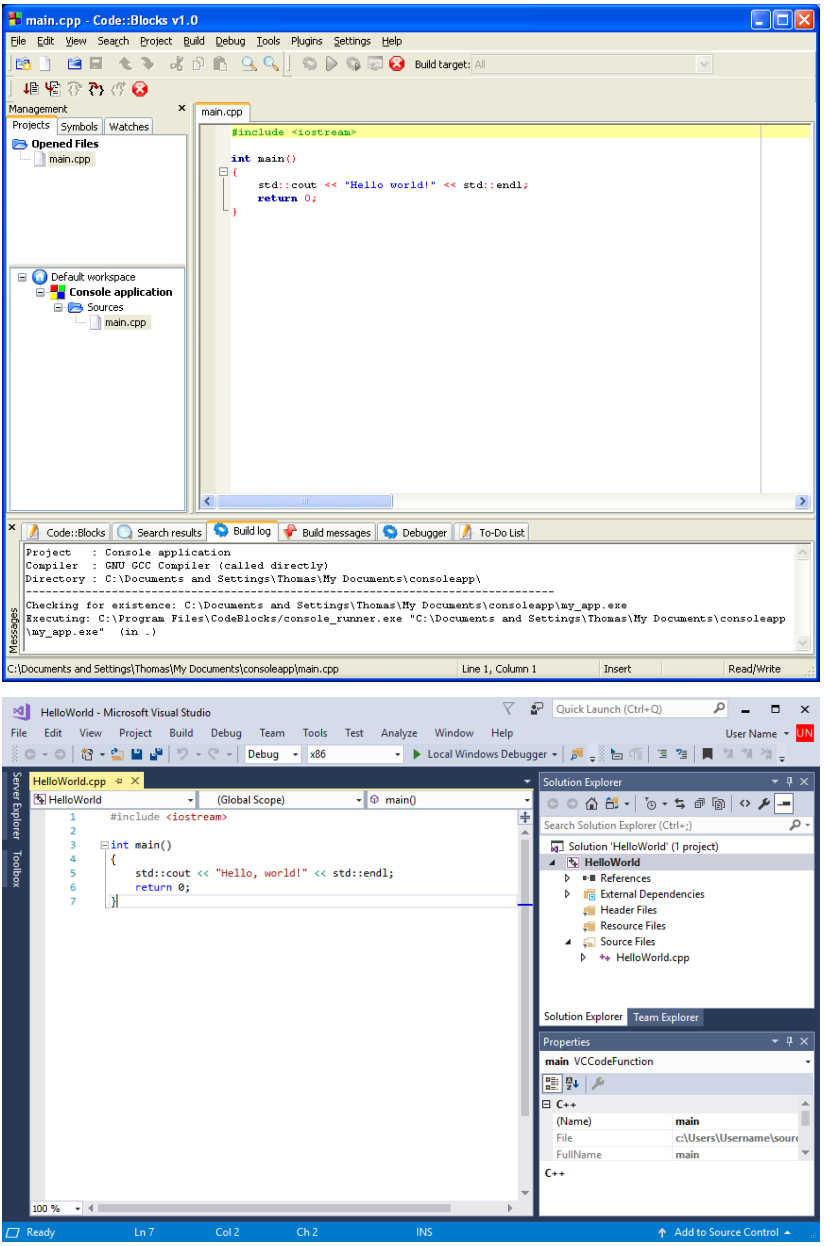
На сьогоднішній день існує достатній вибір середовищ програмування для одних і тих самих мовних платформ. Наприклад, для мов програмування C, C++, Python можна перерахувати цілий спектр середовищ програмування, для яких ці мови є базовими.

Під час знайомства з мовою програмування Python переважно використовуються середовища Python IDLE, PyCharm, CodingGround. Однак, спектр найбільш популярних середовищ програмування мовою Python значно ширший і до нього можна віднести Visual Studio Code, PyDEV, WingWare, Komodo IDE, Eric, Eclipse, Geany, Spyder, PyScripter тощо(мал.6).



Мал.6

Аналогічно можна перерахувати середовища програмування, якими користуються програмісти, які пишуть коди на C та C++. Це Visual Studio, CodeBlocks, Eclipse, Qt Creator та інші (мал.7).


main.cpp - Code::Blocks v1.0
File Edit View Search Project Build Debug Tools Plugins Settings Help
Build target: All
Management
Projects Symbols Watches
Opened Files
main.cpp
Default workspace
Console application
Sources
main.cpp
main.cpp
#include <iostream>
int main()
{
std::cout << "Hello world!" << std::endl;
return 0;
}
Code::Blocks Search results Build log Build messages Debugger To-Do List
Project : Console application
Compiler : GNU GCC Compiler (called directly)
Directory : C:\Documents and Settings\Thomas\My Documents\consoleapp\
Checking for existence: C:\Documents and Settings\Thomas\My Documents\consoleapp\my_app.exe
Executing: C:\Program Files\CodeBlocks/console_runner.exe "C:\Documents and Settings\Thomas\My Documents\consoleapp\my_app.exe" (in .)
Messages
C:\Documents and Settings\Thomas\My Documents\consoleapp\main.cpp
Line 1, Column 1
Insert
Read/Write
HelloWorld - Microsoft Visual Studio
Quick Launch (Ctrl+Q)
File Edit View Project Build Debug Team Tools Test Analyze Window Help
User Name
Debug
x86
Local Windows Debugger
Server Explorer
Toolbox
HelloWorld.cpp
HelloWorld
(Global Scope)
main()
#include <iostream>
int main()
{
std::cout << "Hello, world!" << std::endl;
return 0;
}
100 %
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'HelloWorld' (1 project)
HelloWorld
References
External Dependencies
Header Files
Resource Files
Source Files
HelloWorld.cpp
Solution Explorer
Team Explorer
Properties
main VCCodeFunction
C++
(Name)
main
File
c:\Users\Username\sourc
FullName
main
C++
Ready
Ln 7
Col 2
Ch 2
INS
Add to Source Control

Мал.7

У наведених переліках не ставилося завдання ранжувати середовища програмування за популярністю чи рівнем фаховості використання. Головне – розуміння того, що таких платформ є багато, деякі з них платні, деякі безкоштовні, всі вони орієнтовані на певний користувацький рівень (початківці чи висококваліфіковані програмісти) та сферу розробки програмного продукту. Окрім цього останнім часом популярності набули і додалися також online середовища програмування. Тому завданням користувача є вибір того середовища програмування, яке найкраще підійде для роботи з олімпіадними завданнями та відповідатиме технічним параметрам його комп'ютера.

Перед початком олімпіади учасники отримують можливість роботи на робочих комп'ютерах у тестовому режимі, під час якого вони мають можливість ознайомитися з середовищами програмування, які встановлені організаторами змагань. Найчастіше ці середовища є найуживанішими і в учасників не виникає питань щодо роботи з ними. Порадою може бути лише наступне: скористатися додатковим часом перед початком олімпіади, щоб налаштувати обране середовище програмування під власні потреби.

Слід також не забувати, що перед відправкою програми на перевірку необхідно закоментувати всі відлагоджувальні оператори. Деякі середовища програмування передбачають використання затримки виконання програми для того, щоб можна було аналізувати на екрані монітора отримані результати. Якщо ці оператори чи функції не закоментувати, то система перевірки програми не отримає вихідних даних за відведений час і задача буде знята з виконання.

Системи перевірки олімпіадних задач

Історія проведення Всеукраїнських олімпіад з інформатики сягає далекого 1989 року, коли вона була вперше проведена у м.Чернівцях.

На той час комп'ютеризація навчальних закладів була на початковому рівні, та й ті персональні комп'ютери, які повезло

мати лише деяким навчальним закладам, були різноманітні за своїми технічними можливостями.

Перші олімпіади проводилися у «паперовому» варіанті, а перевірка відбувалася відповідно «читанням» представлених алгоритмів.

З часом журі перейшло до тестової перевірки завдань учасників, що дало змогу саму олімпіаду з інформатики зробити максимально відкритою, оскільки кожний учасник протягом виконання завдань має змогу в онлайн-режимі слідкувати за процесом перевірки, аналізувати свої помилки, коректувати коди програм, відправляючи їх на повторну перевірку.

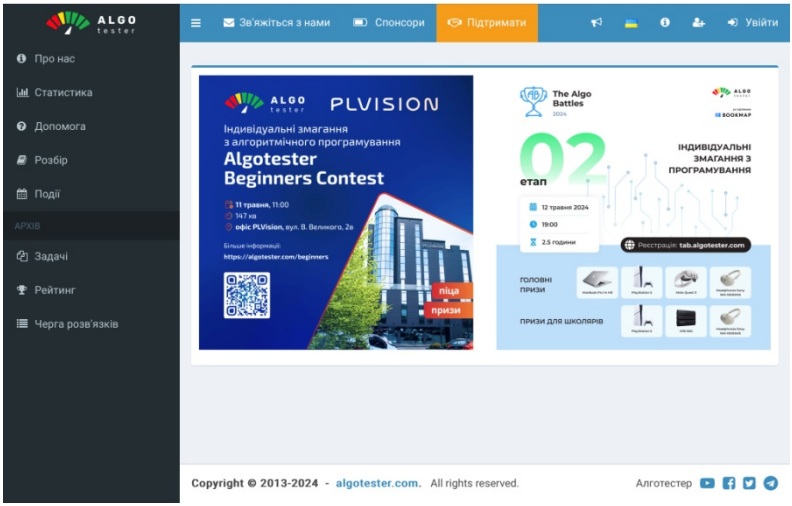
На початках учасники виконували завдання у локальній мережі, до якої був підключений сервер з автоматизованою програмою перевірки олімпіадних завдань. Згодом перевірка завдань перейшла у дистанційний формат в онлайн режимі з використанням потужних серверів, де на той час використовувалась система автоматичної перевірки програм Ejudge. Це дало змогу розширити географію використання цих серверів для проведення Всеукраїнських олімпіад не тільки IVетапу, а й II та III етапів олімпіад з інформатики.

Головною перевагою автоматизованої перевірки завдань учасників є те, що по завершенні туру олімпіади кожний з них знає свої підсумкові бали за тур, після чого під час розбору задач їх авторами може оцінити свої розв'язки.

Український сайт https://algotester.com/uk - це сучасна платформа **Алготестер** автоматичного тестування та проведення змагань зі спортивного програмування з навчальними матеріалами у вільному доступі (мал.8).

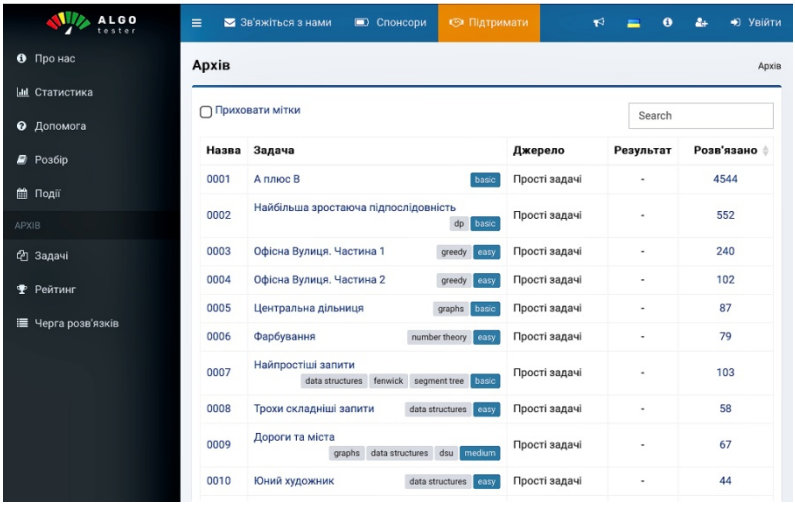
Основними напрямками роботи команди розробників платформи **Алготестер є п**ідготовка учасників до олімпіадних змагань з програмування, зокрема до обласних учнівських олімпіад та студентських ICPC командних змагань.

Сайт містить навчальні матеріали, відео-лекції українською мовою у поєднанні з відповідними задачами архіву, що дозволяють дистанційно вивчати алгоритми як початківцям так і досвідченим програмістам.



Мал.8

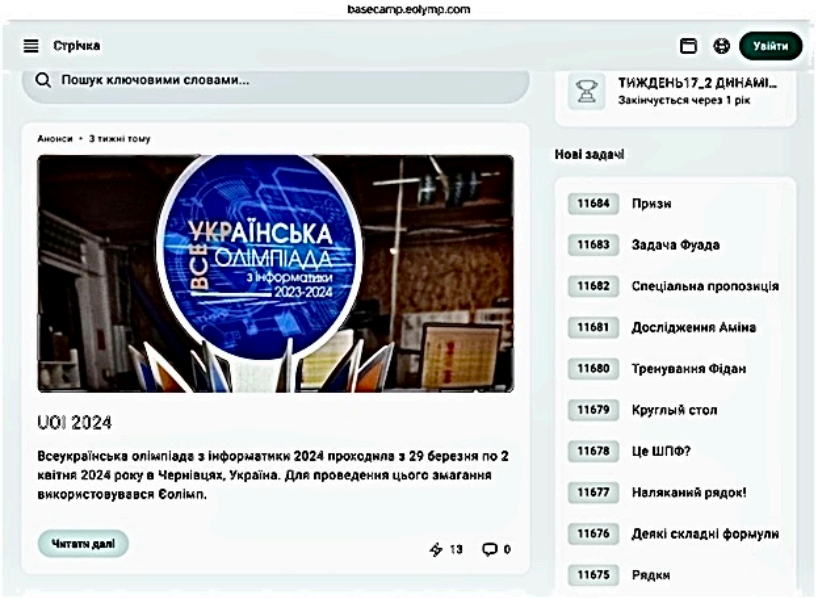
Система автоматичного тестування розв'язків **Алготестер** містить великий набір цікавих алгоритмічних задач різноманітної складності; є платформою для підготовки задач, для проведення змагань з олімпіадного програмування за правилами ICPC та IOI, для проведення оптимізаційних змагань.



Мал.9

Меню сайту надає можливість розв’язання достатньої кількості олімпіадних задач (мал.9), перегляду черги розв’язків під час їх виконання з вказанням позитивного/помилкового результату та турнірної таблиці.

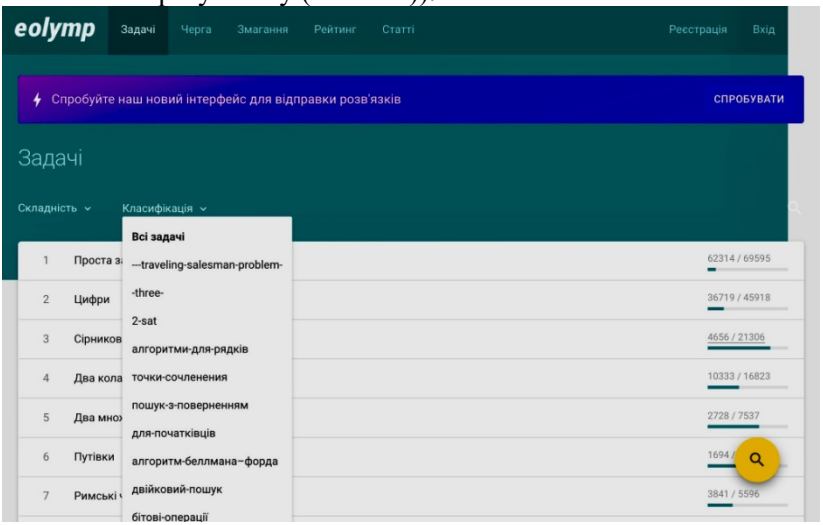
Ще одна система українських розробників **EOlymp** (**eolymp.com**) – це система для проведення олімпіад з програмування у дистанційному форматі, на базі якої останнім часом проводяться Всеукраїнські олімпіади з інформатики (мал.10). Проект створений у межах Державної програми «Інформаційно-комунікаційні технології в освіті і науці» 2006-2010 року Житомирським державним університетом імені Івана Франка.



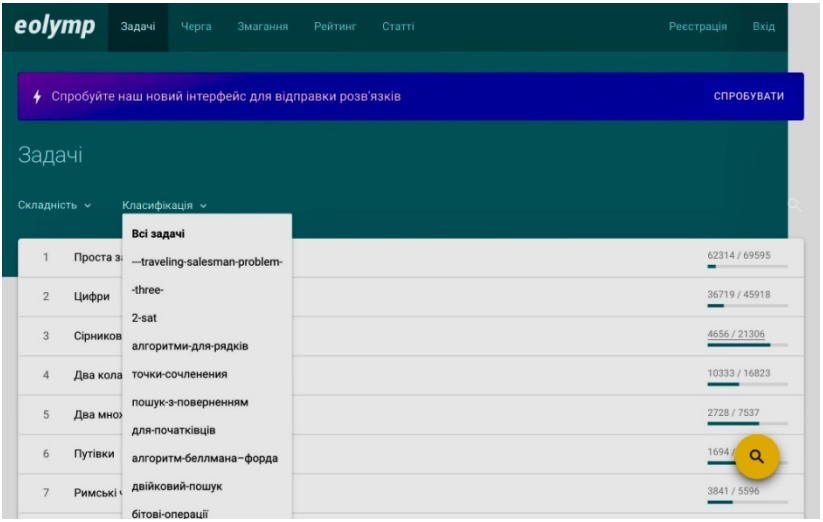
Мал.10

Опція меню середовища «Задачі» (мал.11а)) дозволяє обрати зареєстрованим користувачам для підготовки та тренування будь-яку задачу (мал.2-5) з великого переліку відсортованих за складністю або тематикою олімпіадних задач з інформатики та, користуючись інструментом «Черга»,

відслідковувати всі етапи її перевірки, зокрема й отримання остаточного результату (мал.11б)).



Мал.11а)



Мал.11б)

Система **EOlymp** дозволяє також створювати код програми безпосередньо у цьому середовищі, обирати компілятор для її його виконання та переглядати результати тестування на системі тестів, розроблених для даної задачі.

Вибір компілятора (інтерпретатора)

Як зазначалося вище, до складу кожного середовища програмування, у якому працює учасник олімпіади, входить компілятор або інтерпретатор.

Компілятор – це програма, яка переводить код, написаний мовою високого рівня, у машинний код. У результаті створюється виконуваний файл, наприклад, в ОС Windows з розширенням *.exe, який у решті решт запускається на виконання.

На відміну від компілятора, який перетворює весь код на машинну мову, ***інтерпретатор*** виконує код порядково.

Відмінність між компіляторами та інтерпретаторами:

– інтерпретатор виконує одну інструкцію за раз, транслюючи і виконуючи її, а потім переходячи до наступної. Компілятор же транслює всю програму відразу, а потім виконує її;

– компілятор генерує звіт про помилки після трансляції всієї програми, тоді як інтерпретатор припиняє трансляцію після першої знайденої помилки;

– компілятор потребує більше часу на аналіз і обробку мови високого рівня порівняно з інтерпретатором;

– час виконання коду компілятора швидший, ніж в інтерпретатора, не тільки через час аналізу й обробки, а й тому, що програма вже скомпільована в машинну мову.

Завдання компілятора/інтерпретатора, тобто транслятора, перекласти програму з мови високого рівня у машинний код. Наприклад, оператор цикла **for** (i=0; i<n; i++) s+=a; буде замінений на цілу послідовність машинних операцій типу:

– визначити адреси змінних, які використовуються в операторі;

– занести за цими адресами відповідні початкові значення;

– порівнювати значення змінних *i* та *n*;

– змінювати значення змінних *s* та *i*.

І це лише дуже обмежена кількість описаних операцій, якими заміняється оператор циклу мовою машинних інструкцій, зважаючи на те, що на низькому рівні комп'ютер може лише порівнювати вміст значень визначених областей пам'яті, здійснювати перехід до іншої частини коду, виконувати операцію додавання. Будь-яка виконувана програма складається з послідовності таких атомарних машинних операцій.

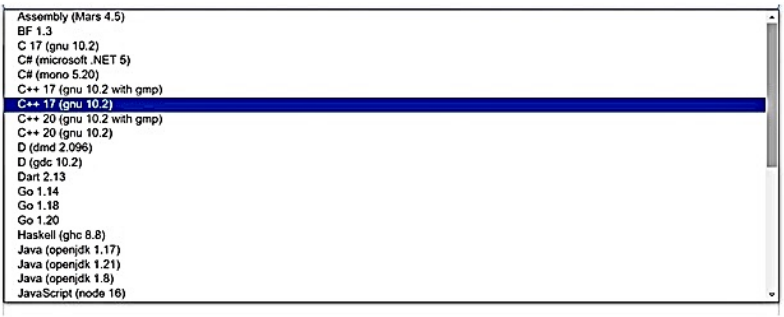
Проаналізувавши наведену вище інформацію, можна дійти висновку, що швидкість роботи машинного коду, яким буде замінено програму, написану мовою високого рівня, залежатиме саме від того, якою кількістю машинних інструкцій заміняються ті чи інші оператори вашої програми обраним компілятором чи інтерпретатором. Компілятори та інтерпретатори – це ті ж самі програми, які мають також ознаки часової та ємнісної складності.

Для виконання програм, написаних мовами програмування C та C++, використовуються компілятори, а для мови Python – інтерпретатори. Є різні розробники цих програм. Для того, щоб визначитися з оптимальним вибором, необхідно врахувати такі чинники:

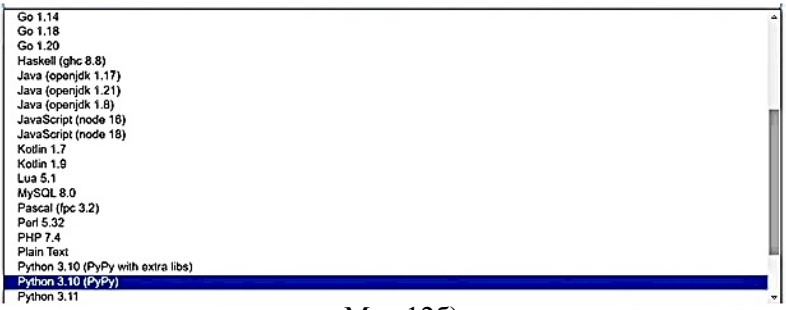
– компілятор/інтерпретатор у системі перевірки олімпіадних завдань має збігатися або бути подібним до того, який використовувався у середовищі програмування, де розроблявся код;

– серед усіх підходящих компіляторів/інтерпретаторів обрати той, що створює оптимальніший за часом виконання виконуваний код.

До прикладу, система перевірки олімпіадних завдань **EOlymp** пропонує достатній вибір компіляторів та інтерпретаторів мов програмування високого рівня (мал.12а),б)).



Мал.12а)



Мал.12б)

Отже, підсумовуючи все вище зазначене, слід зважати на те, що передумовою успішного розв’язання олімпіадної задачі є:

— уважне прочитання умови задачі;

— коректне визначення типів змінних;

— розробка алгоритму задачі з урахуванням ліміту часу та обсягу пам’яті, які зазначені в умові задачі;

— визначення мови програмування для розв’язання конкретної задачі;

— коректний вибір компілятора у системі перевірки завдань;

— тестування програми на коректно підібраних власних тестах, що відповідають умові задачі.

Олімпіадні задачі

В цьому розділі ви можете познайомитися з деякими олімпіадними задачами з інформатики різного рівня складності, які пропонувалися учням на Всеукраїнських олімпіадах II та III етапів і на заочних олімпіадах останніми роками. Матеріал навмисне розбитий на три частини. Перша містить тільки умови задач. Якщо у вас є ідеї щодо їх розв'язання – беріться до роботи. Якщо ж ви засумніваєтеся у їх вірності або ж навпаки захочете переконатися у їх правильності – зверніться до другої частини. Там ви знайдете невелику допомогу щодо розв'язку кожної задачі. Коли вам знадобиться більш серйозна допомога або ж з'явиться бажання порівняти авторський розв'язок зі своїм – відкрийте третю частину. Можливо ваш розв'язок виявиться кращим!

Умови задач

1. Визначити довжину та координати інтервалу з цілочисловими координатами кінців, на який припадають значення елементів дійсної послідовності $x_1,x_2,\ldots x_n$ ($1\le n\le 10^9, |x_i|\le 10^5$).

2. Задане натуральне число N. Знайти всі цілі додатні числа, що не перевищують N та діляться на кожну із своїх цифр.

3. Таблиця футбольного чемпіонату задана квадратною матрицею порядку n, в якій всі елементи, що належать головній діагоналі, дорівнюють нулеві, а кожний елемент, що не належить головній діагоналі, рівний 2, 1 або 0 (кількості очок, що набрані в грі: 2 – виграш, 1 – нічия, 0 – програш). Визначити:

1) кількість команд, які мають більше перемог, ніж поразок;

2) номери команд, що пройшли чемпіонат без поразок;

3) чи є хоча б одна команда, що виграла більше половини ігор.

4. Задані натуральне число n ($n>1$) та послідовність $a_1,a_2,\ldots,a_n$. Знайти найдовшу зростаючу підпослідовність $b_j,b_{j+1},\ldots,b_{j+k}$ ($1\le j$, $2\le k<$n) заданої послідовності. Якщо їх декілька, то вивести елементи першої з них і останньої.

5. Многокутник заданий координатами своїх вершин (x_1,y_1), $(x_2,y_2),\ldots,(x_n,y_n)$. Визначити кількість прямих кутів заданого многокутника і координати відповідних вершин.

6. Дано натуральні числа*n, m* ($m \le n$) і дійсні числа $a_1,a_2,\ldots,a_n$. Визначити середнє арифметичне чисел $a_1,\ldots,a_{m-1},a_{m+1},\ldots,a_n$..

7. Дано два натуральних числа *a* і *b*. Обчислити значення *a*/*b* з точністю до *n* знаків після коми ($n \le 1000$).

8. Дано масив натуральних чисел розмірністю *n*m*. Прямокутником у масиві вважатимемо групу сусідніх елементів одного значення, що разом утворюють прямокутник розміром *k*l* ($1<k \le n, 1<l \le m$). Прямокутником не вважається група елементів, що належать іншому прямокутнику. Обчислити кількість прямокутників у даному масиві.

Вхідні дані	*Вихідні дані*
5 5 1 1 2 2 4 1 1 3 3 3 2 2 3 3 3 2 2 5 5 4 2 2 5 6 4	3

9. Дано натуральне число *n*. Визначити, скільки різних цифр зустрічається в його десятковому записові?

10. Дано натуральне число *N*. Побудувати всі такі трійки натуральних чисел a_1,a_2,a_3, для яких одночасно виконуються дві умови: $a_1 \le a_2 \le a_3$ та $a_1+a_2+a_3=N$.

11. Дано деякий текст та словник з *N* слів. Визначити, які слова і в якій кількості можна утворити із букв заданого тексту.

12. Заданий одновимірний масив цілих чисел A_i, де $i=1,2,\ldots,n$ та $0<A_i \le n$. Надрукувати елементи масиву за таким правилом:

— першим елементом завжди є A_1;

— значення поточного виведеного елемента масиву є порядковим номером наступного елемента цього масиву.

Процес завершується тоді, коли буде надруковано останній елемент масиву, або ж порядковий номер виведеного елемента співпаде з його значенням.

або через одну, не потрапивши при цьому у воду. За яку найменшу кількість стрибків жабеня може досягти своєї мети? А може взагалі йому це не вдасться?

Відомо, що у лівому верхньому та у правому нижньому кутах болота обв'язково є купини.

Вхідні дані	*Вихідні дані*
3 3 1 0 1 1 1 1 10 1	3

50. Деяка послідовність двійкових цифр одержується таким чином:

— першою записується цифра 1;

— для отримання наступної послідовності цифр необхідно всі попередні інвертувати, тобто замість 0 записати 1 і навпаки.

Наприклад, на 5-му кроці формування послідовності вона матиме такий вигляд: 1 0 0 1 0 1 1 0| 0 1 1 0 1 0 0 1.

Визначити, яка двійкова цифра знаходитиметься на k-му місці в даній послідовності.

Вхідні дані	*Вихідні дані*
5 {k}	0

Алгоритми задач

1. Алгоритм задачі передбачає введення дійсних значень для елементів послідовності. Тому для визначення цілочислового інтервалу, на який вони припадають, достатньо визначити мінімальне і максимальне значення цієї послідовності та знайти найбільше ціле, що не перевищує мінімум, і найменше ціле, що перевищує максимум. До речі, при складанні даного алгоритму можна не використовувати масив, а знаходити максимальне та мінімальне значення під час введення елементів послідовності.

2. Особливість цього алгоритму полягає в тому, що нам не відома розмірність числа. Для виділення цифр необхідно організувати цикл, в якому одночасно перевірятиметься і поділ самого числа на його цифри. Удосконалення алгоритму полягає у використанні циклу з перед- або післяумовою, оскільки при

визначенні першої цифри числа, на яку поділ не можливий, подальша перевірка не має змісту.

3. Таблиця результатів футбольного чемпіонату є симетричною відносно головної діагоналі і кожний *i*-ий рядок (або стовпчик) її містить інформацію про результати участі в чемпіонаті *i*-ої команди. Тому для отримання першого результату необхідно в кожному рядку визначити, яких значень більше «2» чи «0», другого результату – відсутність значень «0», третього – для яких рядків кількість значень «0» перевищує $(n-1)/2$.

4. В основі алгоритму лежить підрахунок кількості виконаних умов $a_i<a_{i+1}$ і пошуку максимального значення серед цієї кількості. Для визначення першої підпослідовності необхідно перевіряти умову $k>max$, а для останньої відповідно $k\geq max$. Для запам'ятовування самих підпослідовностей достатньо зберігати порядковий номер їх початку та довжину k. Алгоритм є однопрохідним.

5. Алгоритм базується на теоремі Піфагора. Беручи послідовно по три вершини многокутника і перевіряючи наявність умови існування прямокутного трикутника, підраховуємо їх кількість. Алгоритм працює як для опуклих, так і для неопуклих многокутників.

6. Алгоритм задачі є класичним зразком знаходження середнього арифметичного значення деякої послідовності чисел. Необхідно лише виключити з отриманої суми значення a_m (алгоритм masiv_1) або зразу ж його не сумувати (алгоритм masiv_2). Ще одна відмінність алгоритму masiv_1 від алгоритму masiv_2: в першому варіанті використаний масив, в другому – покрокове введення даних і одночасне обчислення результату.

7. Наведений алгоритм реалізує правило поділу в стовпчик двох натуральних чисел. Під час покрокового поділу отримувані цифри результату виводяться на екран монітора. При такій реалізації точність результату не має ніяких обмежень. Слід лише врахувати варіанти ділення чисел. Наприклад, 10:4=1.25

Розглянемо елемент з порядковим номером k=12, значення якого дорівнює 0. Група елементів від 9 до 16 була породжена елементами від 1 до 8, тобто саме серед цих елементів треба шукати предка зі значенням 1 для елемента з порядковим номером k=12. Його порядковий номер у групі від 1 до 8 визначається як k=4. У свою чергу група з 4-х елементів, куди входить елемент з порядковим номером 4, була отримана дописуванням половини елементів до двох лівих елементів і його предок у цій лівій частині стоїть на другому місці k=2. Але, щоб повернутися до елемента, який породив усю послідовність, треба ще раз зробити поділ навпіл і потрапити у елемент з порядковим номером k=1.

Порахуваємо, скільки переміщень було зроблено за порядковими номерами масиву: $12 \Rightarrow 4 \Rightarrow 2 \Rightarrow 1$. Як бачимо, відбулося 3 перетворення, що відповідає попередньому нашому аналізу, і дійсно на 12-і позиції знаходиться значення 0.

Як видно з описаних попередньо послідовностей дій нас не цікавить значення елемента на новому місці, а лише його новий порядковий номер у зменшеній вдвічі послідовності елементів, відраховуючи від початку цієї послідовності. Це пояснюється тим, що інверсне відображення поточного елемента було породжене його предком у лівій половині, нумерація елементів у якій починається з 1.

Процес перерахунку порядкового номера шуканого елемента відповідає кількості інверсій початкового значення 1.

Розв'язки задач

У даному розділі пропонуються коди програм до відповідних задач мовами C та C++. Вони можуть бути використані читачами як приклад до складання власних варіантів розв'язків, їх покращення та удосконалення.

1.
```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
int main()
{
   float x, min, max;int n, i, a, b;
```

```
   printf("Задайте кількість елементів послідовності: ");
   scanf("%d", &n);
   printf("Задайте 1-й елемент послідовності: ");
   scanf("%f", &x);
   min=max=x;
   for (i=2; i<=n; i++)
   {
      printf("Задайте %d-й елемент послідовності: ", i);
      scanf("%f", &x);
      if (x<min) min=x;
      if (x>max) max=x;
   }
   x=modf(min,&n);        // Функція modf(x,&y) повертає дробову
   a=(min<0)?n-1:n;       // частину числа x, а у змінній nзберігає
   x=modf(max,&n);        // його цілу частинуразом зі знаком
   b=(max<0)?n:n+1;                                  // самого числа.
   printf("Межі інтервалу: [%.f .. %.f]\n",a,b);
   return 0;
}
```

2.

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
   int N, cipher, i, buffer, flag;
   printf("Введіть число: ");
   scanf("%d", &N);
   for (i=1; i<=N; i++)
   {
      flag=1;
      buffer=i;
      while (buffer>0 && flag)
      {
         cipher=buffer%10;
         if (buffer%cipher!=0) flag=0;
         buffer=buffer/10;
      }
      if (flag) printf("%d  ",i);
   }
```

```
   return 0;
}
```

3.
```
#include <stdio.h>
#include <stdlib.h>
int main()
{
   FILE *f_in= fopen("football.in","r");
   FILE *f_out=fopen("football.out","w");
   int a[100][100], b[100];
   int n, i, j, count=0, m=0, k1, k2, flag=0;
   fscanf(f_in,"%d",&n);
   for (i=0; i<n; i++)
     for (j=0; j<n; j++)
       fscanf(f_in,"%d",&a[i][j]);
   for (i=0; i<n; i++)
   {
      k1=0; k2=0;
      for (j=0; j<n; j++)
        if (a[i][j]==2) k1++;
        else if (a[i][j]==0) k2++;
      if (k1>k2-1) count++;
      if (k2==1) b[m++] = i;
      if (k1>(n-1)/2) flag=1;
   }
   fprintf(f_out,"%d\n",count);
   for (i=0; i<m; i++) fprintf(f_out,"%d ",b[i]+1);
   fprintf(f_out,"\n");
   if (flag) fprintf(f_out,"YES");
   else fprintf(f_out,"NO");
   fclose(f_out);
   return 0;
}
```

4.
```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    FILE *f_in=fopen("sequence.in","r");
```

```
    FILE *f_out=fopen("sequence.out","w");
    double a[100];
    int n,i,first=0,last=0,max=2,k,m;
   fscanf(f_in,"%d",&n);
   for (i=0; i<n; i++) fscanf(f_in,"%lf",&a[i]);
   i=1;
   while (i<n)
   {
      fscanf(f_in,"%lf",&a[i]);
      k=0;
      if (a[i-1]<a[i]) m=i-1;
      while (a[i-1]<a[i] && i<n)
      {
         i++; k++;
         fscanf(f_in,"%lf",&a[i]);
      }
      if (k+1>max) { max=k+1; first=m; }
      if (k+1>=max) last=m;
      i++;
   }
   if (max>2)
     if (first!=last)
     {
        for (i=0; i<max; i++) fprintf(f_out,"%.2lf ",a[first + i]);
        fprintf(f_out,"\n");
        for (i=0; i<max; i++) fprintf(f_out,"%.2lf ",a[last + i]);
      }
      else for (i=0; i<max; i++) fprintf(f_out,"%.2lf ",a[first + i]);
   fclose(f_out);
   return 0;
}
```

5.

```
#include <stdio.h>
#include<stdlib.h>
float len(floata, floatb, floatc, floatd)
{
    return (a-c)*(a-c)+(b-d)*(b-d);
}
```

Зміст

Навчальне видання

Караванова Тетяна Петрівна

ОЛІМПІАДНА ІНФОРМАТИКА

Навчальний посібник

Відповідальний за випуск – ***І. М. Черевко***
Літературний редактор – ***О. В. Лупул***
Технічний редактор – ***Т. П. Караванова***
Дизайн обкладинки – ***У. М. Мельник***

Підписано до друку 11.06.2024. Формат 60x84/16.
Папір офсетний. Друк різографічний. Умов.-друк. арк. 14,7.
Обл.-вид. арк. 13,7. Тираж 30. Зам. Н-051.
Видавництво та друкарня Чернівецького національного університету
імені Юрія Федьковича.
58002, Чернівці, вул. Коцюбинського, 2.
e-mail: ruta@chnu.edu.ua

Свідоцтво суб'єкта видавничої справи ДК № 891 від 08.04.2002.