Maciej BONIECKI[1]
Zenon GNIAZDOWSKI[2]
Tomasz NOWAKOWSKI

WPROWADZENIE W ROZWIĄZYWANIE KONKURSOWYCH ZADAŃ PROGRAMISTYCZNYCH

Streszczenie

W artykule na kilku przykładach przedstawiono typowe problemy pojawiające się w trakcie rozwiązywania konkursowych zadań algorytmicznych, a także sposoby ich pokonywania.

Abstract

In this paper, some typical problems that become visible in time of algorithmic competition and methods of their resolution are showed.

1. WSTĘP

Pomimo wielkiej popularności studiów informatycznych, liczba studentów zainteresowanych algorytmiką nie jest imponująca. Pojawia się pytanie, dlaczego tak się dzieje? Zapewne przyczyn jest sporo. Doświadczenie pokazuje, że jednym z problemów – na jaki napotykają studenci – są bariery psychologiczne. Dobrym sposobem ich przełamywania jest osiąganie sukcesów. Sukcesy pojawiają się, gdy istnieje motywacja. Można to uzyskać przez rozwiązywanie zadań algorytmicznych, które są poddawane osądowi zewnętrznemu, a więc przez udział w konkursach programistycznych. Pojawiająca się w takich sytuacjach rywalizacja może mobilizować do dalszej pracy nad własnym rozwojem. Przy okazji, dla niektórych studentów jest to także wprowadzenie w pewną formę działalności naukowo-badawczej.

[1] Maciej Boniecki i Tomasz Nowakowski są studentami Warszawskiej Wyższej Szkoły Informatyki. Obydwaj uczestniczą w pracach Koła Naukowego Miłośników Algorytmów w WWSI.
[2] Dr hab. inż. Zenon Gniazdowski jest profesorem w WWSI i opiekunem KNMA w WWSI.

Na samym początku udziału w konkursach pojawiają się problemy, które – jeśli nie zostaną przezwyciężone – rodzą zniechęcenie. Dla przykładu, początkujący programista wie jak w istocie powinien wyglądać algorytm, a nie rozumie podstawowych pojęć pojawiających się w zadaniach konkursowych, takich jak wczytywanie ze standardowego wejścia czy wypisywanie na standardowym wyjściu (nawet pomimo tego, że operacje te wykorzystuje w prawie każdym swoim programie). Brakuje jasnych wskazówek, które wprowadzałyby początkujących informatyków w świat zawodów programistycznych.

Głównym celem tego artykułu jest zapełnienie tej luki informacyjnej. Będzie to wprowadzenie w rozwiązywanie zadań konkursowych. Zostanie to pokazane na przykładach zadań, przy rozwiązywaniu, których pojawią się problemy typowe dla początkujących uczestników zawodów programistycznych. Problemy te to błędna specyfikacja wejścia i wyjścia, istnienie wielu zestawów danych testowych oraz błędy przekroczenia dopuszczalnego czasu działania programu lub przekroczenia zakresu zmiennych. Zostaną one zidentyfikowane, a następnie rozwiązane.

Autorzy zdają sobie sprawę, iż w swojej analizie dotykają tylko niektórych kwestii, nie będąc w stanie wyczerpać całego zagadnienia. Mają przy tym nadzieję, że ich artykuł może okazać się użyteczny.

2. UWAGI FORMALNE

Zadania konkursowe są zapisywane w typowej formie, na którą w pierwszej kolejności warto zwrócić uwagę. Treść zadań bywa podzielona na pięć części: ***Opis***, ***Zadanie***, ***Wejście***, ***Wyjście*** oraz ***Przykład***. Taki podział spotykany jest w większości zadań programistycznych.

- Część pierwsza zadania, zatytułowana jako „***Opis***” zazwyczaj zawiera jakąś historię będąca tłem literackim dla prezentowanego problemu. W opisie oprócz historii, autorzy często umieszczają wskazówki pomocne przy rozwiązywaniu zadania. Nierzadko są one dosyć dobrze zakamuflowane tak, że podczas pierwszego czytania nie przykuwają uwagi zawodników.
- W części drugiej, zatytułowanej „***Zadanie***”, jasno sprecyzowano, co tworzony program ma robić.
- Części o nazwie „***Wejście***” i „***Wyjście***” specyfikują, jakie dane i w jakiej postaci będą przekazywane do programu, a także co powinno zostać wypisane na ekranie. Początkującym programistom należy przy okazji wyjaśnić czym jest standardowe wejście i wyjście. W zawodach algorytmicznych przez wczytanie danych ze standardowego wejścia rozumie się wprowadzenie danych za pomocą klawiatury, analogicznie wypisanie na standardowym wyjściu jest wypisaniem informacji na ekranie monitora.

– W części o nazwie „***Przykład***” autorzy zadania dołączają przykładowe dane wejściowe, a także poprawny wynik dla tych danych. Wyjątkiem są zadania, w których podanie przykładu mogłoby zbytnio ułatwić rozwiązanie problemu. Warto odnotować, że przykładowe dane podane w zadaniu są zazwyczaj pierwszym testem sprawdzanym przez sędziów lub (tak jest w większości zawodów) przez aplikację sprawdzającą. Z tego powodu, zawodnicy zawsze powinni sprawdzać przed wysłaniem rozwiązania, czy ich program radzi sobie z tymi testami. Przetestowanie rozwiązania może mieć w niektórych zawodach kluczowe znaczenie. Przykładowo, w Akademickich Mistrzostwach Polski w Programowaniu Zespołowym, za przesłanie niepoprawnego rozwiązania, do ogólnego czasu poświęconego na rozwiązywanie zadań, każdorazowo doliczane jest *20* minut karnych. Wielu niedoświadczonych zawodników pomija testowanie i zaraz po napisaniu rozwiązania, wysyła je do programu sprawdzającego. Efekt takiego postępowania łatwo przewidzieć.

W większości zawodów algorytmicznych organizowanych w Polsce można programować w jednym z trzech języków programowania: C, C++ lub Pascalu. W zawodach międzynarodowych organizatorzy coraz częściej rezygnują z Pascala na rzecz takich języków jak Java czy C#. W rozwiązaniach tworzonych dla potrzeb zawodów algorytmicznych, kodowanych w językach C i C++ widoczne są pewne różnice. Pierwsza z nich dotyczy nazw standardowych bibliotek. W języku C++ nazwy bibliotek są pozbawione rozszerzenia „.h”, zaś nazwy plików nagłówkowych znanych z biblioteki języka C są poprzedzone literą „c”. Przykładowo, nazwa „stdio.h” użyta w C, musi zostać zamieniona na „cstdio” w C++. Kolejną różnicą jest używana w C++ (a nieznana w języku C) biblioteka wejścia/wyjścia „iostream” zawierająca strumienie „cout” oraz „cin” służące do wczytywania i wypisywania danych. Niestety, podczas wczytywania dużej ilości danych, wydajność tej biblioteki jest gorsza w porównaniu z funkcjami printf() i scanf() znanymi z biblioteki standardowej języka C. Programiści C++ mają, więc dwie możliwości. Przy pomocy instrukcji ios_base::sync_with_stdio(false) mogą wyłączyć synchronizację ze strumieniami z „cstdio”, albo mogą korzystać z wyżej wymienionych funkcji printf() i scanf(). Przedstawione różnice są na tyle mało istotne, że autorzy niniejszego artykułu zdecydowali się ograniczyć do przykładów kodowanych w języku C, nie zmniejszając przez to ogólności rozważań. Przedstawione tu uwagi mogą być przydatne przy pisaniu programów także w Pascalu.

Należy też wspomnieć o programach sprawdzających rozwiązania. Programy te mają zazwyczaj przygotowany zestaw plików tekstowych zawierających testy czyli dane jakie zostaną przekazane do programu, a także zestaw plików z wzorcowymi wynikami wygenerowanymi dla tych testów. W trakcie sprawdzania, wyniki działania

testowanego programu są zapisywane do pliku tekstowego. Program sprawdzający automatycznie porównuje zawartości pliku z wynikami i pliku zawierającego wzorcowe wyniki dla poszczególnych testów.

Działanie programu sprawdzającego można we własnym zakresie częściowo symulować, wywołując skompilowany program w wierszu polecenia:

```
C:\>Zadanie1.exe < test1.in > test1.out
```

W poleceniu tym „***Zadanie1.exe***" jest nazwą pliku wykonywalnego otrzymanego po skompilowaniu programu, „***test1.in***" jest nazwą pliku z przykładowymi danymi dla programu, zaś „***test1.out***" jest nazwą pliku do którego zostaną skierowane wyniki działania programu. Otrzymany w efekcie wykonania polecenia plik „***test1.out***" może być porównany z plikiem z poprawnymi wynikami dla danych testowych.

3. SPECYFIKACJA WEJŚCIA I WYJŚCIA PROGRAMU

Aby wygenerować przykładowy plik z danymi testowymi, trzeba zapoznać się ze specyfikacją wejścia oraz wyjścia podaną w treści zadania. Jest to kluczowy fragment zadania, ponieważ dokładnie precyzuje, co należy wczytać a co wypisać w rozwiązaniu. Jeżeli programista nie zastosuje się do umieszczonych tam wytycznych, nie ma szans na zaliczenie zadania. W podanym niżej przykładowym zadaniu przez kolejne propozycje rozwiązania, zostanie pokazane, jak powinna wyglądać specyfikacja wejścia i wyjścia do programu.

Zadanie 1

Opis

Bliźniacy Romek i Tomek pomimo swojego młodego wieku uwielbiają rozwiązywać proste zagadki matematyczne, jakie przygotowują im rodzice. Pewnego dnia tata maluchów dał im zadanie, aby zsumowali wiek rodziców. Niestety nasi bohaterowie ze względu na młody wiek kiepsko radzą sobie z obliczeniami, których zakres wyniku przekracza 20. W związku z powyższym poprosili właśnie Ciebie o pomoc.

Zadanie

Napisz program, który:
- Wczyta za standardowego wejścia wiek mamy i taty bliźniaków.
- Wypisze na standardowym wyjściu sumę wieku rodziców.

Wejście

W pierwszej i jedynej linii wejścia znajdują się dwie liczby naturalne A, B, $20 \leq A, B \leq 100$, oddzielone pojedynczą spacją oznaczające odpowiednio wiek mamy oraz wiek taty.

Wyjście

Na wyjściu powinna zostać wypisana jedna liczba naturalna C będąca sumą wczytanych liczb A i B.

Przykład

Wejście:

25 27
Wyjście:

52

Rozwiązanie

Treścią zadania jest wczytanie dwóch liczb naturalnych ze standardowego wejścia i zwrócenie ich sumy na standardowe wyjście. Oto przykładowy kod rozwiązania powyższego zadania:

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>

int main() {
        int a, b;
        printf("Podaj wiek mamy: ");
        scanf("%d", &a);
        printf("Podaj wiek taty: ");
        scanf("%d", &b);
        printf("Laczny wiek rodzicow wynosi: %d\n", a + b);
        getch();
        return EXIT_SUCCESS;
}
```

Podany kod poprawnie rozwiązuje zadanie: po wczytaniu wieku obydwojga rodziców wypisuje sumę oznaczającą łączny ich wiek. Niestety, nie przeszedłby on pomyślnie testów podczas zawodów. Dlatego w dalszej części stopniowo będą eliminowane błędy rozwiązania tak, aby doprowadzić rozwiązanie do poprawnej postaci.

W pierwszej kolejności trzeba wiedzieć, jakie biblioteki są możliwe do użycia podczas zawodów. Zazwyczaj lista ogranicza się do biblioteki standardowej danego języka programowania, pomniejszonej o funkcje tworzenia plików, uruchamiania nowych procesów itp. Z tego powodu, próba dołączenia niedozwolonych plików nagłówkowych zazwyczaj kończy się błędem kompilacji. Po wysłaniu przedstawionego wyżej rozwiązania, również pojawiłby się błąd kompilacji. Dzieje się tak za sprawą biblioteki conio.h (Console Input/Output). Ta użyteczna biblioteka[3] nie jest częścią standardowej biblioteki języka C. Kod programu należy zmodyfikować tak, aby pozbyć się funkcji getch() i pliku nagłówkowego conio.h. W tym celu funkcja getch() zostanie zastąpiona zdefiniowaną w pliku nagłówkowym stdlib.h funkcją system() z argumentem „pause". Biblioteka stdlib.h – w przeciwieństwie do biblioteki conio.h – jest standardową biblioteką języka C. Nowy kod programu prezentuje się następująco:

```
#include <stdio.h>
#include <stdlib.h>

int main() {
        int a, b;
        printf("Podaj wiek mamy: ");
        scanf("%d", &a);
        printf("Podaj wiek taty: ");
        scanf("%d", &b);
        printf("Laczny wiek rodzicow wynosi: %d\n", a + b);
        system("pause");
        return EXIT_SUCCESS;
}
```

Także i to rozwiązanie nie przejdzie testów. W tym wypadku można się spodziewać dwóch komunikatów o błędach. Pierwszy z nich może dotyczyć użycia niedozwolonej funkcji system(), umożliwiającej wywoływanie poleceń systemowych. Ponieważ przy zawodach na dużą skalę, ich uczestnicy mogą mieć najróżniejsze pomysły, organizatorzy dla bezpieczeństwa (i własnego spokoju) zazwyczaj wyłączają tę funkcję oraz funkcje umożliwiające tworzenie plików lub plików tymczasowych.

[3] Biblioteka ta zawiera kilka użytecznych funkcji, takich, jak np. wykorzystywana w pokazanym rozwiązaniu funkcja getch(). Z tego powodu jest ona często wykorzystywana przez programistów.

W przypadku gdyby funkcja system() była funkcją dozwoloną, wtedy autor rozwiązania otrzymałby komunikat o błędzie przekroczenia limitu czasu dla danego zadania. W zawodach programistycznych rozwiązania są testowane nie tylko pod kątem poprawności, ale również wydajności. Głównym miernikiem wydajności jest właśnie łączny czas działania programu. Aktualne rozwiązanie przekroczyłby wszelkie limity czasowe. Przyczyną takiego stanu rzeczy jest wywołanie funkcji system(“pause”). Polecenie “pause” powoduje, że program czeka aż użytkownik wciśnie dowolny przycisk na klawiaturze, a dopiero później kończy swoje działanie. Ponieważ zadanie automatycznie jest sprawdzane poprzez program sprawdzający, testowany program nigdy nie doczekałby się wciśnięcia klawisza. Aplikacja sprawdzająca uznałaby, że rozwiązanie jeszcze nie dokonało wszystkich obliczeń i zwróciłaby błąd przekroczenia limitu czasu. W związku z powyższym należy pamiętać, aby w programach nigdy nie oczekiwać innej interakcji niż ta opisana w części zadania zatytułowanej „wejście”.

Usunięcie wywołania funkcji system(“pause”) ma także negatywny skutek. Ponieważ zaraz po zakończeniu obliczeń i wypisaniu wyniku znika okienko, dlatego nie można obejrzeć wyników działania programu. W tym wypadku są możliwe dwa rozwiązania. Pierwszym z nich jest zatrzymanie instrukcji system(“pause”) i wyrzucenie jej dopiero przed wysłaniem rozwiązania. Drugim sposobem jest uruchamianie programu bezpośrednio w wierszu poleceń. Niezależnie od tego, które rozwiązanie zostanie wybrane, aktualna wersja programu przed wysłaniem do sprawdzenia prezentuje się następująco:

```
#include <stdio.h>
#include <stdlib.h>

int main() {
        int a, b;
        printf(“Podaj wiek mamy: “);
        scanf(“%d”, &a);
        printf(“Podaj wiek taty: “);
        scanf(“%d”, &b);
        printf(“Laczny wiek rodzicow wynosi: %d\n”, a + b);
        return EXIT_SUCCESS;
}
```

Jednakże, nie jest to jeszcze koniec problemów. Z pozoru program działa poprawnie: nie używa niedozwolonych bibliotek, nie przekracza limitów czasowych. Pomimo tego rozwiązanie nadal zawiera błędy. Należy zwrócić uwagę, że programista nie zastosował się do wytycznych zawartych w specyfikacji ***Wejście*** i „***Wyjście***”

w treści zadania. Napisano tam, że wejście składa się tylko z jednej linii, w której wpisane zostaną dwie liczby naturalne, tak więc przykładowy plik test1.in mógłby się prezentować tak:

```
29 30
```

Po zapisaniu pliku *test1.in* i wywołaniu programu w sposób podany w punkcie drugim, w pliku *test1.out* otrzymano by komunikat o następującej treści:

```
Podaj wiek mamy: Podaj wiek taty:
Laczny wiek rodzicow wynosi: 59
```

Tekst ten jest tym, co zostało wypisane na standardowym wyjściu. Widać, że nie jest to tekst oczekiwany przez autorów zadania. W specyfikacji wyjścia napisano, że na standardowym wyjściu (czyli na ekranie) powinna zostać wypisana tylko jedna liczba naturalna będąca sumą dwóch wczytanych liczb. Błąd polega na wypisywaniu komunikatów pomocnych użytkownikowi takich jak "*Podaj wiek mamy:* ". Komunikaty tego typu są całkowicie niezrozumiałe dla programów, które sprawdzają rozwiązania. Kiedy zamiast oczekiwanej liczby 59 program sprawdzający napotyka na tekst "*Podaj wiek mamy:* " od razu traktuje to jako błąd obliczeń i zwraca wynik "Błędna odpowiedź". Należy zatem pozbyć się z kodu zbędnych komunikatów. Poprawione rozwiązanie, które teraz powinno zostać zaakceptowane przez program sprawdzający, prezentuje się następująco:

```
#include <stdio.h>
#include <stdlib.h>

int main() {
        int a, b;
        scanf("%d", &a);
        scanf("%d", &b);
        printf("%d\n", a + b);
        return EXIT_SUCCESS;
}
```

Na koniec warto podkreślić kilka istotnych spraw. Po pierwsze należy zwracać uwagę na to czy podczas zakończenia wykonania programu, zwracana jest wartość *0*. Oznacza to, że w trakcie wykonania wszystko przebiegło pomyślnie. Drugą rzeczą wartą odnotowania jest sposób wczytywania przez program danych wejściowych. W specyfikacji wyjścia jest napisane, że liczby znajdują się w jednej linii i są oddzielone spacją, tymczasem użycie dwóch funkcji scanf() sugeruje jakoby znajdowały się

one w osobnych wierszach. W praktyce jednak umieszczenie funkcji scanf() nie ma znaczenia przy wczytywaniu danych. Zarówno wprowadzenie liczb w dwóch wierszach jak i w jednym zostanie poprawnie obsłużone przez pokazane tu rozwiązanie.

4. WIELE ZESTAWÓW DANYCH

W treści następnego zadania nowością jest wprowadzenie zestawów danych. Celem takiego zabiegu jest umożliwienie przetestowania poprawności i wydajności algorytmu dla różnych danych wejściowych podczas jednego wykonania programu.

Zadanie 2

Opis

Jaś jest uczniem szkoły podstawowej. Jednym z jego ulubionych przedmiotów jest matematyka. Nasz bohater stara się być jak najlepiej przygotowany na każdą lekcję, w związku z czym często prosi tatę, aby ten pomógł mu się przygotować. W tym tygodniu nauczycielka Jasia zapowiedziała sprawdzian z tabliczki mnożenia do 100. Jaś bardzo chciałby dostać 5, niestety ma poważny problem. Jego tata musiał dzień przed sprawdzianem zostać dłużej w pracy i nie pomoże mu w przygotowaniach. W związku z tym poprosił właśnie Ciebie o napisanie programu, który pomoże Jasiowi w przygotowaniach.

Zadanie

Napisz program, który dla każdego zestawu danych:
- Wczyta ze standardowego wejścia dwie liczby naturalne.
- Wypisze na standardowym wyjściu ich iloczyn.

Wejście

W pierwszej linii wejścia znajduje się dokładnie jedna liczba naturalna N, $1 \leq N \leq 100$, oznaczająca ilość zestawów danych. W każdej z następnych *N* linii znajdują się dwie liczby naturalne A, B, $1 \leq A, B \leq 10$, oddzielone pojedynczą spacją będące czynnikami iloczynu.

Wyjście

Dla każdego zestawu danych należy wypisać w osobnej linii jedną liczbę naturalną będącą iloczynem wczytanych liczb A i B.

Przykład

Wejście:

3
2 10
9 9
7 4

Wyjście:

20
81
28

Rozwiązanie

Poza wprowadzeniem wielu zestawów danych, poniższe zadanie jest bardzo podobne do zadania poprzedniego. Stąd, nie będą wymagane znaczące zmiany w treści programu. Kod rozwiązania zadania prezentuje się następująco:

```
#include <stdio.h>

int main(void) {
        int i, c;
        int czynniki[100][2];
        scanf("%d", &c);
        for (i = 0; i < c; i++) {
                  scanf("%d %d", &czynniki[i][0], &czynniki[i][1]);
        }
        for (i = 0; i < c; i++) {
                  printf(„%d\n", czynniki[i][0] * czynniki[i][1]);
        }
        return 0;
}
```

W pokazanym rozwiązaniu, do przechowywania wprowadzanych danych została użyta dwuwymiarowa tablica liczb całkowitych. Dane dla kolejnych zestawów są wprowadzane w pierwszej pętli for, a następnie w drugiej pętli for wykonywane są obliczenia i wynik wypisywany jest na ekranie. Ważnym elementem, na który należy zwracać uwagę w przypadku zadań z zestawami danych jest przechodzenie do nowej

linii po wypisaniu każdego z wyników. Opuszczenie przejścia do nowej linii (albo przynajmniej spacji) spowoduje sklejenie się ze sobą wszystkich wypisanych iloczynów, a co za tym idzie brak akceptacji rozwiązania przez automat sprawdzający zadania.

Pokazany na początku artykułu sposób symulacji działania programu sprawdzającego gwarantował całkowite oddzielenie danych wczytywanych od wypisywanych. W przypadku zadania o bliźniakach nie miało to żadnego znaczenia, jednak w momencie, kiedy wprowadzone zostały zestawy danych, może to umożliwić napisanie rozwiązania w znacznie prostszy sposób. Oto kod drugiej wersji zadania z mnożeniem:

```
#include <stdio.h>

int main(void) {
        int i, a, b, c;
        scanf(„%d”, &c);
        for (i = 0; i < c; i++) {
                scanf(„%d %d”, &a, &b);
                printf(„%d\n”, a * b);
        }
        return 0;
}
```

Dzięki oddzieleniu standardowego wejścia i wyjścia, program nie musi najpierw wczytywać wszystkich zestawów danych, a dopiero potem dokonywać obliczeń. Iloczyn liczb A i B może zostać policzony i wyświetlony na ekranie zaraz po ich wczytaniu. Z punktu widzenia aplikacji sprawdzającej efekt będzie ten sam jak wtedy, gdy wczytywanie danych i wypisywanie wyników odbywają się w dwóch różnych pętlach.

5. PRZEKROCZENIE CZASU ORAZ ZAKRESU ZMIENNYCH

Często spotykanym i trudnym do wykrycia błędem jest przekraczanie dopuszczalnego czasu działania programu lub przekraczanie zakresu zmiennych. Problem ze znalezieniem takich błędów polega na tym, iż algorytm może być poprawny i działać bezbłędnie dla wielu zestawów danych testowych. W przypadku przekroczenia zakresu, mogą to być niemal wszystkie dane wejściowe poza danymi ekstremalnymi. Dla pokazania tego typu problemów zostanie zaprezentowany kolejny przykład.

Zadanie 3

Limit czasu: 2s

Zadanie

Napisz program, który dla każdego zestawu danych:
- Wczyta ze standardowego wejścia jedną liczbę naturalną N.
- Obliczy i wypisze na standardowym wyjściu sumę wszystkich liczb od 1 do N włączając obydwie te liczby.

Wejście

Pierwsza linia zawiera dokładnie jedną liczbę naturalną M, $1 \leq M \leq 200000$, będącą liczbą zestawów danych. W kolejnych M liniach występują poszczególne zestawy danych. Każdy zestaw składa się z jednej liczby naturalnej N, $1 \leq N \leq 65535$.

Wyjście

Dla każdego zestawu danych program powinien wypisać na standardowym wyjściu sumę wszystkich liczb naturalnych od 1 do N. Gwarantujemy, że wartość bezwzględna sumy nie przekroczy 2^{31}.

Przykład

Wejście:

```
4
3
2
5
8
```

Wyjście:

```
6
3
15
36
```

Rozwiązanie

Autor zadania zagwarantował, iż „wartość bezwzględna sumy nie przekroczy 2^{31}", czyli wynik obliczeń nie przekroczy zakresu zmiennej typu *long int*. Zatem

oczywistym rozwiązaniem będzie wykonanie sumowania liczb w pętli od *1* do *N*. Istnieje wtedy pewność zarówno co do poprawności wyniku jak i co do tego, że żadna ze zmiennych nie przekroczy zakresu typu *long int*. Łatwo jednak w tym przypadku obliczyć ilość operacji jakie będzie musiał wykonać program, przy maksymalnym rozmiarze danych wejściowych (*M=200000*, *N=65535*) będzie to *M·N=13 107 000 000* operacji. Uwzględniając ograniczenie czasowe wynoszące *2* sekundy, z pewnością rozwiązanie to nie zostanie zaakceptowane, ponieważ program przekroczy limit czasu. Poszukać należy zatem innego, szybszego rozwiązania. Z pomocą przychodzi tu wzór na sumę ciągu arytmetycznego:

$$\sum_{i=1}^{n} i = 1+2+3+...+n = \frac{(1+n)n}{2} \tag{1}$$

Złożoność programu przy wykorzystaniu takiego wzoru jest niewielka i wynosi 3 działania dla każdej zmiennej, zatem limit czasu przestaje być problemem. Dla pewności należy sprawdzić tę metodę dla przykładowych danych:

$$\sum_{i=1}^{3} i = 1+2+3 = \frac{(1+3)3}{2} = 6$$

$$\vdots$$

$$\sum_{i=1}^{8} i = 1+2+3+4+5+6+7+8 = \frac{(1+8)8}{2} = 36$$

Wyniki pokrywają się z podanymi przez autora zadania, zatem z dużym prawdopodobieństwem rozwiązanie jest poprawne. Po przełożeniu tego rozwiązania na język C i wysłaniu go do serwisu w celu weryfikacji poprawności, program uzyska status błędnej odpowiedzi. Po przeczytaniu takiego komunikatu mając jednocześnie pewność co do poprawności zaimplementowanego algorytmu zastanowić się należy nad maksymalną liczbą, jaką program może uzyskać podczas wykonywania obliczeń. Aby upewnić się, że to jest przyczyną otrzymanego statusu, należy uruchomić program i jako dane wejściowe podać maksymalne możliwe wartości.

Autor zadania zadeklarował maksymalną sumę nie większą niż zakres *long int*. Deklaracja ta dotyczy jednak samego wyniku. Wykorzystując wzór na sumę ciągu arytmetycznego, w trakcie obliczeń można przekroczyć zakres *long int*. Dla maksymalnej wartości *N=65535*, zgodnie z wzorem (1) najpierw jest wykonywane mnożenie *65535·65536*. Jego wynikiem jest liczba *4 294 901 760*. Liczba ta jest prawie dwukrotnie większa od maksymalnej wartości jaką może przechowywać *long int*: w trakcie obliczeń nastąpiło przekroczenie zakresu dla tego typu. Otrzymany wynik pośredni jest interpretowany jako bardzo duża liczba ujemna (zjawisko to jest

określane jako „przekręcanie" się zmiennej). Po podzieleniu przez *2*, jest to nadal liczba ujemna. W przypadku zadania w którym nie może pojawić się liczba ujemna (program w żadnym miejscu nie wykonuje odejmowania, a autorzy gwarantują na wejściu liczby dodatnie) pewnym jest, że któraś z zadeklarowanych zmiennych ma zbyt mały zakres.

Rozwiązaniem tego problemu może być użycie zmiennej typu *unsigned long int*, która jest w stanie przechowywać dwukrotnie większe wartości niż *long int* lub zamiana kolejności wykonywania działań. Jeżeli we wzorze (1) najpierw parzysty czynnik zostanie podzielony przez *2* a dopiero potem zostanie wykonane mnożenie, to nie będzie problemu z przekroczeniem zakresu danych. Oto poprawny kod do zadania trzeciego:

```
        #include <stdio.h>

int main(void) {
        long i, m, n;
        scanf(„%ld", &m);
        for (i = 0; i < m; i++) {
                scanf(„%ld", &n);
                if (n % 2 == 0) {
                        printf(„%ld\n", n / 2 * (n + 1));
                } else {
                        printf(„%ld\n", (n + 1) / 2 * n);
                }
        }
        return 0;
}
```

6. PODSUMOWANIE

W artykule na kilku przykładach przedstawiono typowe problemy pojawiające się w trakcie rozwiązywania konkursowych zadań algorytmicznych, a także sposoby ich pokonywania. W pierwszej kolejności zwrócono uwagę na formę zadań, używane języki programowania oraz sposób testowania zadań konkursowych. Z kolei zauważono, że z punktu widzenia początkującego uczestnika konkursu najistotniejsze są trzy sprawy: błędna specyfikacja wejścia i wyjścia programu, istnienie wielu zestawów danych testowych oraz błędy przekroczenia dopuszczalnego czasu działania programu lub przekroczenia zakresu zmiennych. W artykule krok po kroku pokazano, jak z tymi problemami należy sobie radzić.