Dorota Roman-Jurdzińska
Elżbieta Wierzbicka

MOCNE I SŁABE STRONY WYKSZTAŁCENIA INFORMATYCZNEGO MATURZYSTÓW

W tym roku po raz pierwszy informatyka mogła być wybrana jako przedmiot obowiązkowy. Być może dla niektórych maturzystów stała się „wyjściem awaryjnym" – mając niewielką wiedzę z innych przedmiotów, wybrali egzamin z informatyki, bo „znają się na komputerze", i przystąpili do egzaminu bez gruntownego przygotowania.

Z drugiej strony wiadomo, że mamy wśród młodzieży osoby zdolne i zainteresowane informatyką, które jednak nie wybierają tego przedmiotu na maturze, stawiając raczej na matematykę i fizykę, bo zachęca ich do tego system rekrutacji na wyższe uczelnie.

Czego oczekuje się od zdających na egzamin maturalny z informatyki?
Zgodnie ze standardami egzaminacyjnymi, zdający powinni wykazać się kompetencjami w następujących obszarach wiedzy i umiejętności:

1. **orientacja w tematyce informatycznej:** sprzęt i technologia, reprezentacje danych, systemy operacyjne i sieci, narzędzia programistyczne, bezpieczeństwo danych oraz użytkowników sieci
2. **myślenie algorytmiczne:** analizowanie działania podanych algorytmów, ocena ich poprawności i złożoności, znajomość klasycznych algorytmów prowadzących do rozwiązania typowych prostych problemów, przedstawienie własnej konstrukcji algorytmicznej stosownej do zadanego problemu
3. **sprawne poruszanie się w oceanie zalewającej nas informacji** przy zastosowaniu dostępnych narzędzi programowych, manipulacja dużymi ilościami danych, wybór z nich tego, co potrzebne, tworzenie zestawień (w tym: wizualizacji), wyciąganie wniosków ilościowych i jakościowych.

Najmocniejszą stroną zdających był bez wątpienia obszar trzeci – przetwarzanie informacji przy pomocy narzędzi technologii informacyjnej.
Prawie wszyscy wykazali się umiejętnością pracy z bazą danych. Zaimportowali dane źródłowe do tabel, ustanowili relacje między tabelami, wyciągnęli z nich dane spełniające postawione kryteria i utworzyli niezbędne podsumowania.
Większość zdających radziła sobie dobrze z arkuszem kalkulacyjnym. Prawdopodobnie zaskoczyła ich w tym roku treść zadania, wymagająca zastosowania arkusza do rozwiązania problemu natury matematycznej, ale poradzili sobie bardzo dobrze. Umieją importować dane, przetwarzać je przy pomocy rozmaitych formuł z różnych dziedzin: matematycznych, tekstowych i statystycznych oraz tworzyć wykresy.
Na średnim poziomie zdający rozwiązali zadania reprezentujące obszar pierwszy – orientacja w tematyce informatycznej. Wyniki są zróżnicowane i pokazują, że wiedza niektórych zdających jest powierzchowna. Potrafią przeliczyć bity na bajty i zamienić liczbę z postaci dziesiętnej na binarną. Gdy jednak pytanie sięga trochę głębiej: dotyczy kodowania binarnego liczb ze znakiem, w standardzie U2, pojawia się wiele błędnych odpowiedzi.
W obszarze algorytmiki i programowania widoczne jest największe zróżnicowanie poziomu zdających. Analiza działania pojedynczej prostej pętli na ogół kończyła się sukcesem, ale już wiedza o najprostszych algorytmach klasycznych była niewielka, a umiejętność tworzenia własnych sensownych algorytmów i programów opanowali tylko nieliczni zdający. Byli i tacy, którzy zaskakiwali pomysłowością w tym zakresie.
Przyjrzyjmy się szczegółowo, jak zdający radzili sobie z poszczególnymi zadaniami egzaminacyjnymi.

Poziom podstawowy

Zadanie pierwsze polegało na analizie działania algorytmu sprawdzającego, czy podany niepusty ciąg liczb dodatnich jest rosnący. Trzeba było prześledzić jego działanie na konkretnych danych i zmodyfikować algorytm tak, aby znajdował najlepszy wynik zawodnika w podanym niepustym ciągu liczb dodatnich.

W podpunkcie a) należało uzupełnić specyfikację, a konkretnie podać, kiedy algorytm poda odpowiedź TAK, a kiedy NIE. Część zdających miała z tym problem, można było odnieść wrażenie, że z pojęciem specyfikacji spotkali się na maturze pierwszy raz.

Największy problem sprawił maturzystom podpunkt b), w którym należało podać, ile razy dla konkretnych danych zostanie wykonany w algorytmie krok 3. Część zdających omijała ten podpunkt albo podawała liczbę, z której wynikało, że wykonywali pętlę z kroku 4 niezależnie od tego, czy warunek "następna jest większa od aktualna" w instrukcji warunkowej był prawdziwy, czy fałszywy.

W punkcie c) należało zauważyć, że jeśli usuniemy zakończenie algorytmu w przypadku niespełnienia warunku w kroku 4, a krok 4.2 będzie wykonywany aż do zakończenia algorytmu, czyli stanie się krokiem 5, to zmienna *aktualna* będzie wskazywała największą wartość znalezioną w niepustym skończonym ciągu liczb dodatnich i tą zmienną należało wypisać na zakończenie działania algorytmu w kroku 2 (zamiast TAK).

Zadanie nie wymagało znajomości konkretnych algorytmów ze standardów wymagań egzaminacyjnych, a jedynie logicznego myślenia i umiejętności modyfikacji i analizowania pętli. Należy zauważyć, że dwa pierwsze punkty miały naprowadzić maturzystę na odpowiedź do punktu c).

Zadanie drugie dotyczyło systemu dwójkowego. Algorytm pozycyjnej reprezentacji liczb występuje w standardach wymagań egzaminacyjnych z informatyki na poziomie podstawowym. Zagadnienie to pojawia się po raz pierwszy na lekcjach informatyki już na poziomie gimnazjum.

W punkcie a) zdający mieli za zadanie uzupełnić tabelkę z cennikiem, czyli dwie liczby dziesiętne zamienić na system dwójkowy, a dwie liczby zapisane w systemie dwójkowym zamienić na system dziesiętny. Większość zdających poprawnie zamieniało ceny na system dziesiętny; gorzej radzili sobie z zamianą cen z systemu dziesiętnego na system dwójkowy.

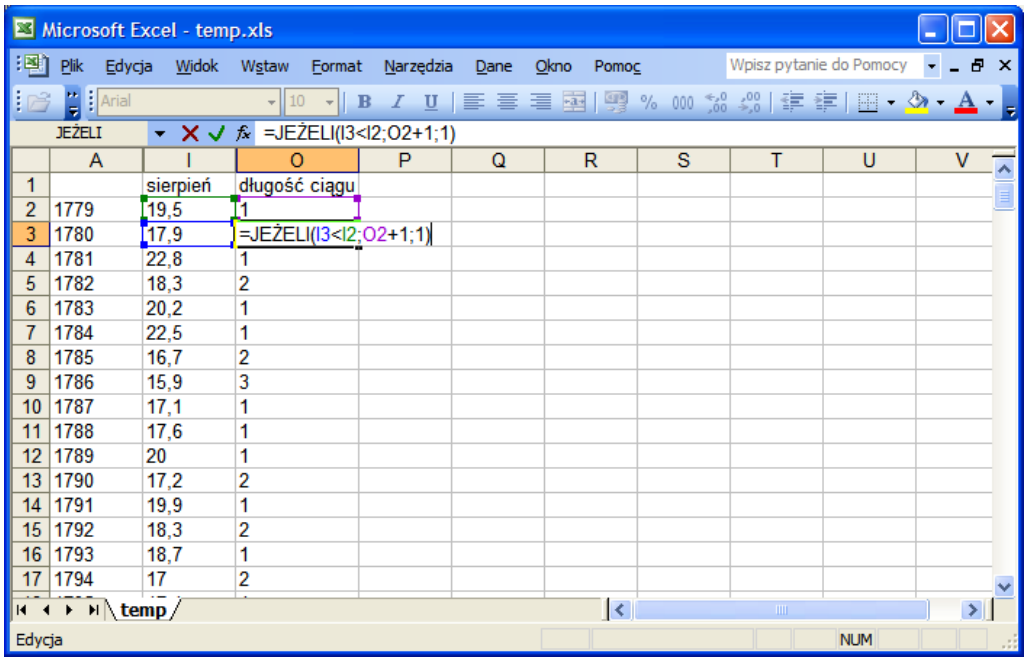
Podpunkt b) polegał na napisaniu algorytmu zamiany liczb z systemu dwójkowego na system dziesiętny z uwzględnieniem warunku, że ceny są podawane z dokładnością do dwóch miejsc po przecinku. W tym przypadku zdający miał już podaną pełną specyfikację. Jednak niewielu uczniów podjęło próbę rozwiązania tego zadania. Maturzyści, którzy podjęli się napisania algorytmu, najlepiej radzili sobie z zamianą na system dziesiętny części całkowitej liczby. Dziwić może słabe radzenie sobie z częścią dziesiętną, gdyż dotyczyła ona tylko dwóch miejsc po przecinku i jej wszystkie wartości można było poznać już w punkcie a): 00 to 0,00; 01 to 0,25; 10 to 0,50; a 11 to 0,75. Nasuwa się wniosek, iż maturzyści potrafią w praktyce zastosować algorytm zamiany liczby zapisanej w systemie dwójkowym, natomiast nie radzą sobie z zapisem formalnym tego algorytmu.

Pierwsze zadanie arkusza II na poziomie podstawowym było typowym zadaniem do rozwiązania w arkuszu kalkulacyjnym. W pliku tekstowym podano średnie miesięczne temperatury w Warszawie w latach 1779–2006. W pierwszych trzech podpunktach należało wydobyć informacje o najwyższych i najniższych średnich rocznych i miesięcznych temperaturach. Trzeba też było sporządzić wykres punktowy ilustrujący najwyższe i najniższe średnie temperatury od stycznia do grudnia. Do realizacji tych poleceń wystarcza podstawowa znajomość funkcji arkusza kalkulacyjnego. W szczególności, rozwiązanie punktów a) i b) można uzyskać, wyliczając średnie roczne temperatury (funkcja ŚREDNIA) ze średnich temperatur miesięcznych, a potem korzystając z funkcji MIN i MAX w zakresie komórek odpowiadających średnim temperaturom rocznym. Maturzyści poradzili sobie z tymi podpunktami dobrze.

Trudniejszym i mniej standardowym zadaniem był ostatni podpunkt d), w którym trzeba było znaleźć najdłuższy podciąg malejący w ciągu utworzonym ze średnich miesięcznych temperatur sierpnia. Podpunkt ten sprawił maturzystom największe trudności, rozwiązały go nieliczne osoby. Jedno z możliwych rozwiązań mogło polegać na utworzeniu dodatkowej kolumny. W kolumnie tej, korzystając z funkcji JEŻELI, umieszczamy

- w pierwszym wierszu liczbę 1
- w każdym następnym wierszu: wartość o jeden większą od wartości w poprzednim wierszu (jeśli bieżąca wartość temperatury w sierpniu jest mniejsza od poprzedniej) lub wartość 1, w przeciwnym przypadku.

Największa wartość w tak zdefiniowanej kolumnie odpowiadać będzie wielkości najdłuższego podciągu malejącego. Fragment przykładowego rozwiązania prezentujemy na rysunku.



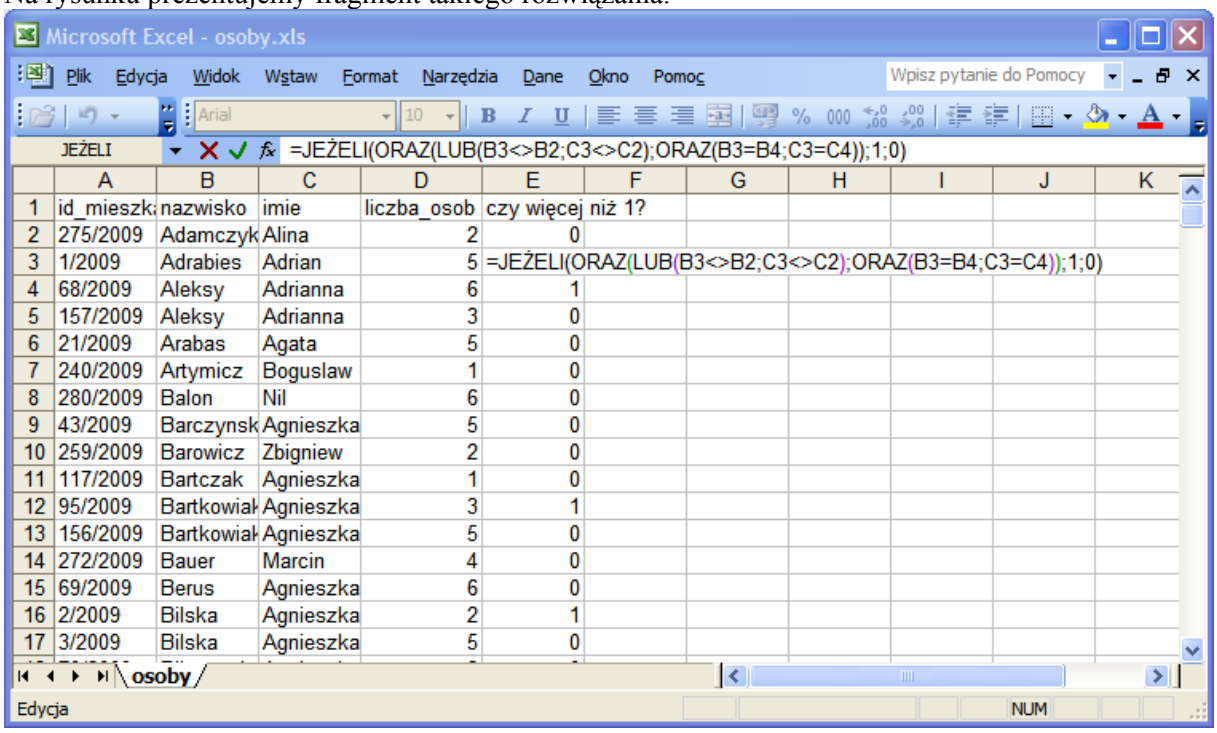
Zadanie 5 miało charakter programistyczny. Wyróżniało się ono tym, iż w treści wymagano wprost, aby rozwiązanie było uzyskane przy pomocy samodzielnie napisanego programu komputerowego. Sam problem jest bardzo prosty, do rozwiązania wystarczy umiejętność sprawdzania pierwszości liczby (ćwiczona już w szkole podstawowej!) oraz sprawdzenia, czy podana liczba jest kwadratem innej liczby naturalnej. Jedynej trudności upatrywać tu można w „tłumaczeniu" znanych uczniom prostych algorytmów na poprawnie działający program w konkretnym języku programowania. Okazało się, że dla maturzystów wybierających poziom podstawowy była to ogromna trudność! Tylko nieliczni rozwiązali to zadanie. Jak wynika z jego treści, ocenie podlega nie tylko odpowiedź programu dla dostarczonych danych, ale również jego poprawność dla dowolnych danych spełniających podaną specyfikację (chodzi tutaj m.in. o wyeliminowanie rozwiązań, w których program wybiera kwadraty liczb naturalnych, natomiast pierwszość sprawdzana jest ręcznie).

Ostatnie zadanie drugiego arkusza miało charakter bazodanowy. Dane do zadania umieszczono w dwóch plikach opisujących mieszkania i ich właścicieli. Pliki te odpowiadały dwóm tabelom relacyjnej bazy danych, powiązanym poprzez identyfikator mieszkania. Z tego powodu naturalne jest tutaj zastosowanie aplikacji bazodanowej, realizującej automatycznie takie powiązania. Wprawni użytkownicy arkuszy kalkulacyjnych poradzą sobie z takim zadaniem dobrze, korzystając z różnych funkcji wyszukiwania (np. funkcja WYSZUKAJ.PIONOWO). Co więcej, dane do tego konkretnego zadania były skonstruowane w ten sposób, że kolejne wiersze pliku `adres.txt` (od pierwszego do trzysetnego) odpowiadały kolejnym wierszom pliku `osoby.txt`. Połączenie plików można więc było uzyskać w arkuszu kalkulacyjnym przez proste kopiowanie i wklejanie.

Warto zwrócić uwagę, że podpunkty a) i d) nie wymagały korzystania z powiązania obu tabel: w podpunkcie a) należało ustalić osoby występujące w pliku `osoby.txt` więcej niż raz, a podpunkt (d) wymagał zliczenia kobiet, co w tym przypadku oznaczało liczbę osób o imieniu kończącym się literą „a". Jednak właśnie te punkty okazały się najtrudniejsze; poprawnie rozwiązało je mniej niż 20% maturzystów. Może to wynikać z tego, że ustalenie liczby osób o różnych imionach i nazwiskach (podpunkt a)) wymaga użycia raczej niestandardowych, choć prostych operacji. Rozwiązanie tego podpunktu w arkuszu kalkulacyjnym mogło polegać na przykład na

- posortowaniu osób wg nazwiska i imienia
- zliczaniu liczby wierszy, w których imię i nazwisko są takie same jak w wierszu następnym i inne niż w wierszu poprzednim.

Na rysunku prezentujemy fragment takiego rozwiązania.



	A	B	C	D	E	F	G	H	I	J	K
1	id_mieszk	nazwisko	imie	liczba_osob	czy więcej niż 1?						
2	275/2009	Adamczyk	Alina	2	0						
3	1/2009	Adrabies	Adrian	5	=JEŻELI(ORAZ(LUB(B3<>B2;C3<>C2);ORAZ(B3=B4;C3=C4));1;0)						
4	68/2009	Aleksy	Adrianna	6	1						
5	157/2009	Aleksy	Adrianna	3	0						
6	21/2009	Arabas	Agata	5	0						
7	240/2009	Artymicz	Boguslaw	1	0						
8	280/2009	Balon	Nil	6	0						
9	43/2009	Barczynsk	Agnieszka	5	0						
10	259/2009	Barowicz	Zbigniew	2	0						
11	117/2009	Bartczak	Agnieszka	1	0						
12	95/2009	Bartkowiak	Agnieszka	3	1						
13	156/2009	Bartkowiak	Agnieszka	5	0						
14	272/2009	Bauer	Marcin	4	0						
15	69/2009	Berus	Agnieszka	6	0						
16	2/2009	Bilska	Agnieszka	2	1						
17	3/2009	Bilska	Agnieszka	5	0						

Podpunkt d) wymagał użycia funkcji tekstowych, co być może było dodatkowym utrudnieniem (w Excelu wystarczy użyć funkcji PRAWY, która umożliwia wydobycie ustalonej liczby skrajnie prawych znaków z tekstu). Drobnym utrudnieniem był też fakt, że ta sama osoba może występować w zestawieniu więcej niż jeden raz.

Jeżeli dane z plików `osoby.txt` i `adres.txt` zostały połączone, podpunkty b) i c) stają się dość łatwe. W podpunkcie b) liczymy powierzchnię przypadającą na osobę na podstawie pola *metraż mieszkania* i *liczba osób zamieszkujących to mieszkanie*, a do rozwiązania zadania używamy funkcji filtrowania dostępnych zarówno w aplikacjach bazodanowych jak i w arkuszu kalkulacyjnym. W podpunkcie c) również wystarczy proste filtrowanie. Wyniki uzyskane przez maturzystów w punktach b) i c) są dużo wyższe niż w podpunktach a) i d).

Podsumowując, możemy zauważyć, że dwa spośród trzech zadań w arkuszu II poziomu podstawowego można rozwiązać za pomocą arkusza kalkulacyjnego. Znajomość funkcji arkusza umożliwia uzyskanie 22 punktów z 30 punktów możliwych do uzyskania w tej części egzaminu.

Poziom rozszerzony

W zadaniu pierwszym tegoroczni maturzyści nieźle poradzili sobie z analizą algorytmu zawierającego zmienne proste w pojedynczej pętli (zadanie testowe 1a). Na ogół poprawnie śledzili wartości zmiennych w kolejnych iteracjach, wyznaczali wartość logiczną warunkującą powtarzanie pętli oraz zliczali ilość powtórzeń pętli.

Zdający poprawnie przeliczali bity na bajty, kilobajty i megabajty (zadanie 1b). Poprawnie wybierali też odpowiednik wartości liczby całkowitej bez znaku, wyrażonej w systemie dziesiętnym, przekształconej na system dwójkowy, czwórkowy, ósemkowy i szesnastkowy (zadanie 1c). Reprezentacja liczby całkowitej ze znakiem, wyrażona w systemie binarnym U2, sprawiła jednak problem znacznej liczbie zdających. Widać, że nie wszyscy ćwiczyli to w szkole (zadanie 1d). Hasło „*Schemat Hornera*" większość zdających poprawnie skojarzyła z zastosowaniem do obliczania wartości wielomianu przy minimalnej liczbie operacji mnożenia (zadanie 1e).

W zadaniu drugim − Punkty kratowe − zdający na ogół poprawnie wyznaczali wynik liczbowy – ilość punktów kratowych dla podanych dwóch wartości liczbowych promienia koła (zadanie 2a). Konstrukcja algorytmu, który wyznacza liczbę punktów kratowych dla dowolnej danej wartości promienia, okazała się zadaniem ciekawym, obfitującym w różnorodność rozwiązań (zadanie

2b). Wielu zdających wykonało szkice graficzne, zaznaczyło na rysunku punkty kratowe i próbowało odnaleźć jakąś prawidłowość. Niektórzy szukali formuły rekurencyjnej, inni, o zacięciu naukowym, próbowali zapisać ogólny wzór wiążący liczbę punktów kratowych z promieniem okręgu, niestety – nieskutecznie.

Wśród rozwiązań poprawnych dominowało przeglądanie kolejnych punktów na siatce układu współrzędnych i sprawdzanie, czy ich odległość od początku układu nie przekracza *R*. Zdający konstruowali pętlę przebiegającą po wartościach całkowitych współrzędnej poziomej *x* z zakresu $x \in <-R;R>$, w której na różne sposoby zliczali punkty o współrzędnej pionowej *y* całkowitej, spełniającej warunek $y <= \sqrt{R^2 - x^2}$. Niektórzy wykorzystywali symetrię problemu, skutecznie przyspieszając działanie algorytmu. Ciekawa była grupa rozwiązań oparta na koncepcji zliczania, ile jest liczb *L* takich, że kwadrat każdej z nich nie przekracza wartości danej: $L^2 <= R^2$.

W innych koncepcjach rozwiązań zdający najczęściej zliczali poprawnie punkty kratowe położone na osiach *x* i *y*, natomiast pozostałe punkty bywały zliczane błędnie.

Zadanie świetnie nadaje się do wykorzystania podczas lekcji. Podane wcześniej uczniom do przemyślenia (zadane do domu?) powinno sprowokować dyskusję nad różnymi wariantami rozwiązania.

Przykładowe warianty rozwiązania zadania 2:

```
  int R, N;
  cout<<"Podaj R=";
  cin>>R;
// sprawdza wszystkie punkty w kwadracie opisanym na kole o promieniu R
  N=0;
  for (int i=-R; i<=R; i++) {
        for (int j=-R; j<=R; j++)
          if (sqrt(i*i+j*j)<=R) N++;
        }
  cout<<"Liczba punktow = "<<N<<endl;
// dla każdego x ∈ (-R,R) szuka największej współrzędnej y, mieszczącej się w kole o promieniu R
  N=0;
  for (int i=-R; i<=R; i++) {
        int j=(int)sqrt(R*R-i*i);
        N=N+2*j+1;
        }
  cout<<"Liczba punktow = "<<N<<endl;
// podobna koncepcja jak wyżej, z użyciem pętli wewnętrznej zliczającej ilość punktów w wierszu
  int i=1;
  N=0;
  while (i<=R) {
        int k=R;
        while (sqrt(i*i+k*k)>R) k--;
        N=N+2*k+1;
        i++;
        }
  N=2*N+2*R+1;
  cout<<"Liczba punktow = "<<N<<endl;
// zlicza punkty należące do koła w obszarze jednej ćwiartki koła i mnoży przez 4
  int a=0;
  N=1;
  if (R!=0) {
     for (int i=R; i>0; i--) {
```

```
            for (int j=R; j>0; j--) {
                 if (i*i+j*j<=R*R) a++;
            }
     }
      a=a+R;
      N=N+4*a;
    }
     cout<<"Liczba punktow = "<<N<<endl;
// zlicza punkty w 1 ćwiartce, wykorzystując funkcję pomocniczą
     N=4*R+1;
     int r_kw=R*R;
     int kwadrat=1;
     int suma_cwiartki=0;
     for (int i=1; kwadrat<r_kw; ) {
         suma_cwiartki+=ile_poteg(r_kw-kwadrat);
         i++;
         kwadrat=i*i;
         }
     N=N+4*suma_cwiartki;
     cout<<"Liczba punktow = "<<N<<endl;
```

```
// funkcja pomocnicza ile_poteg zwraca ilość liczb o takiej własności, że kwadrat liczby
// jest mniejszy od wartości danej
int ile_poteg(int dana) {
     int wynik=0, i=1;
     while(i*i<=dana) {wynik++; i++;}
     return wynik;
}
```

Wersja opisowa algorytmu rozwiązania zadania 2.

Krok 1. Przygotowanie tablicy o rozmiarze Z*Z, gdzie Z jest równe całkowitej części liczby R.
Krok 2. Wypełnienie tablicy kwadratami odległości punktów kratowych od środka układów współrzędnych, utożsamiając ją z I ćwiartką kartezjańskiego układu współrzędnych. Pole w lewym dolnym rogu tablicy odpowiada punktowi o współrzędnych (1,1).
Krok 3. Podniesienie liczby R do kwadratu.
Krok 4. Wyznaczenie ile liczb wpisanych do tablicy jest mniejszych co do wartości od kwadratu liczby R.
Krok 5. Pomnożenie liczby otrzymanej w poprzednim kroku przez 4.
Krok 6. Dodanie do liczby otrzymanej w poprzednim kroku czterokrotnosci całkowitej części liczby R.
Krok 7. Dodanie do otrzymanego wyniku liczby 1. Otrzymana liczba jest wynikiem.

Celem zadania 3a było sprawdzenie, czy zdający rozumie i potrafi prześledzić ciąg wywołań rekurencyjnych podanej funkcji dla konkretnych danych liczbowych. Większość zdających bez problemu poradziła sobie z tą częścią zadania, choć trafiały się również błędne odpowiedzi.

W zadaniu 3b zdający powinni zaproponować konstrukcję algorytmu NWD w postaci iteracyjnej. Zdawałoby się, że jest to powszechnie znany i najprostszy z klasycznych algorytmów. A jednak spora liczba zdających nie umiała tego zrobić. Okazało się nawet, że nie wszyscy zdający potrafią sformułować specyfikację problemu. To nie powinno się zdarzyć i świadczy o brakach w edukacji informatycznej i matematycznej zdających.
Osoby, które przedstawiły rozwiązanie, sięgały na ogół po starożytny algorytm Euklidesa, z zastosowaniem odejmowania lub dzielenia modulo zmiennych *a* i *b*.

```
      int a,b,nwd;
// wariant z dzieleniem modulo
      while (b!=0) {
            int c=a%b;
            a=b;
            b=c;
      }
      nwd=a;
```

```
// wariant z odejmowaniem
while (a!=b)
      if (a>b) a=a-b; else b=b-a;
nwd=a;
```

Szczególnie interesujący był taki zwięzły zapis:

```
      while (a*b != 0)
           if (a>b) a=a%b; else b=b%a;
      nwd=a+b;
```

Ale nie brakło innych rozwiązań, ciekawych, zwracających poprawny wynik.

Zdarzały się rozwiązania naiwne, przeczesujące kolejno cały zakres liczb pomiędzy wartościami 1 (jeden) i *min(a,b)*. Jeśli poszukiwania podzielnika przebiegały od góry tego zakresu, od największych liczb, wówczas po znalezieniu pierwszego wspólnego podzielnika liczb *a* i *b* można było zakończyć algorytm.

```
      if (b<a) nwd=b; else nwd=a;
      while ((a%nwd!=0 || b%nwd!=0) && (nwd>1)) nwd--;
```

Zdarzało się także, że zdający stosowali algorytm znany ze szkoły podstawowej: rozkładali na czynniki obie liczby *a* i *b*, a następnie wyszukiwali i mnożyli wspólne podzielniki. Oczywiście ich trud został doceniony.
Problem wyznaczenia największego wspólnego podzielnika, podobnie jak zadanie 2, znakomicie nadaje się do dyskusji w szkole. Przy ocenie różnych wariantów rozwiązania powinno się brać pod uwagę prostotę algorytmu i optymalizację czasu działania.
Zainteresowanym uczniom warto pokazać także rozszerzony algorytm Euklidesa. Nie jest trudny, a przecież choć taki archaiczny, przydaje się aktualnie w kryptografii przy wyznaczaniu klucza publicznego i prywatnego.

Zadanie 4 – smok iteracyjny – nie okazało się trudne dla zdających. Większość osób rozwiązała je przy pomocy arkusza kalkulacyjnego:

	A	B	C	D
1	x	y		
2	1	1		
3	=JEŻELI(D3=0;(-0,4*A2)-1;0,76*A2-0,4*B2)	=JEŻELI(D3=0;(-0,4*B2)+0,1;0,4*A2+0,76*B2)	=LOS()	=JEŻELI(C3<0,5;0;1)
4	=JEŻELI(D4=0;(-0,4*A3)-1;0,76*A3-0,4*B3)	=JEŻELI(D4=0;(-0,4*B3)+0,1;0,4*A3+0,76*B3)	=LOS()	=JEŻELI(C4<0,5;0;1)

Wybór układu równań w kolejnych iteracjach realizowano losowo przy pomocy formuły *=LOS()*. Otrzymane wyniki *(x,y)* posłużyły następnie jako źródło danych dla wykresu typu *XY-Punktowy*.

Rozmiary smoka: minimalne, maksymalne oraz średnie wartości współrzędnych *x* i *y*, a także ich zaokrąglenie do wymaganej dokładności wyznaczano przy pomocy formuł arkusza kalkulacyjnego:

	D	E
	=ŚREDNIA(A101:A5000)	=ZAOKR(D2;2)
	=ŚREDNIA(B101:B5000)	=ZAOKR(D3;2)
	=MAX(A101:A5000)	=MIN(A101:A5000)
	=MAX(B101:B5000)	=MIN(B101:B5000)
	=ZAOKR(D5;1)	=ZAOKR(E5;1)
	=ZAOKR(D6;1)	=ZAOKR(E6;1)

Pewną niedogodność odczuli zdający, którzy, pracując na sprzęcie o stosunkowo słabej wydajności obliczeniowej, wybrali typ wykresu *XY-Punktowy*, ale z liniami łączącymi punkty (nie powinni byli tego robić, smok miał składać się z samych punktów). W tym przypadku czas oczekiwania na utworzenie wykresu wyraźnie wydłużał się.

Obraz wykresu w formie osobnego pliku graficznego realizowano zwykle jako zrzut ekranu, wklejony następnie do okna dowolnego programu graficznego (choćby Paint-a), gdzie został odpowiednio wykadrowany i zapisany w wybranym formacie graficznym.

Pewna grupa zdających zdecydowała się rozwiązać zadanie drogą programowania w wybranym języku. Oto przykładowe rozwiązanie w języku C++:

```
int main()
{
    int los;
    double x=1, y=1;
    srand(time(0));
    double p1=0,p2=0;
    double min1,min2,max1,max2;
    ofstream zapisz("smok.txt");

    for(int i=0;i<=5000;i++)  {

     los=rand()%2;
     if (los==1) {
              x=(-0.4)*x-1;
              y=(-0.4)*y+0.1;
           }
           else {
              x=0.76*x-0.4*y;
              y=0.4*x+0.76*y;
           }
     zapisz<<x<<"\t"<<y<<endl;

     if(i==100)  {
              min1=x;
              max1=x;
              min1=y;
              min2=y;
     }
     if (i>=100)  {
               if (x<min1) min1=x;
               if (x>max1) max1=x;
               if (y<min2) min2=y;
               if(y>max2)  max2=y;
               p1=p1+x;
               p2=p2+y;
     }
    }
    zapisz<<"\n\nx sr="<<p1/(a-100)<<", y sr="<<p2/(a-100)<<endl;
```

```
    zapisz<<"Minimalna wartosc x="<<min1<<" a y="<<min2;
    zapisz<<"Maksymalna wartosc x="<<max1<<"a y="<<max2<<"."<<endl;

zapisz.close();
return 0;
}
```

Obliczone wartości współrzędnych punktów (*x,y*) były zapisywane w pliku tekstowym, każda para liczb w osobnej linii pliku. Następnie dane te importowano do arkusza kalkulacyjnego i sporządzano z nich wykres.

W tej grupie rozwiązań powtarzał się problem, z którym niektórzy zdający nie poradzili sobie i nie utworzyli wykresu. Otóż program główny zapisywał do pliku wartości *x* i *y* jako liczby rzeczywiste z kropką dziesiętną. W tej postaci były one importowane do arkusza kalkulacyjnego, który, ze względu na znak kropki, traktował dane jako łańcuchy znaków, a nie wartości liczbowe, więc nie mógł z nich utworzyć wykresu. Wystarczyłoby w arkuszu kalkulacyjnym zamienić wszystkie znaki kropki na przecinek (przy pomocy narzędzia *Edycja / Zamień*), lecz część osób nie dostrzegła takiej możliwości.

Zdający, którzy tworzyli rozwiązanie w Pascal-u, programowali bezpośrednio rysowanie obrazu smoka, wykorzystując procedury graficzne interfejsu BGI.

Zadanie 5 – pary słów – polegało na przetwarzaniu tekstów. Operacje porównywania łańcuchów znaków, wycinania podłańcucha, wyszukiwania wzorca znaków w łańcuchu, itp. mają szerokie znaczenie praktyczne. Stanowią podstawę działania edytorów tekstów, wyszukiwarek, programów tłumaczących. Inżynieria genetyczna również sięga w swoich metodach do wycinania i sklejania wybranych sekwencji chromosomów. Istnieje już wiele pomysłowych i szybkich algorytmów. Zadanie to nie wymagało bynajmniej ich znajomości, ale i tak okazało się najtrudniejsze w tegorocznym arkuszu maturalnym. Wielu zdających w ogóle nie podjęło próby jego rozwiązania.
Spośród osób, które rozwiązywały zadanie 5, duża grupa poprawnie wyszukała i zliczyła palindromy (zadanie 5a). Było to typowe szkolne zadanie i zdający poradzili sobie z nim na drodze programowania. W Excelu na razie nie ma prostej formuły odwracającej kolejność liter w słowie (można było użyć funkcji *StrReverse* języka VisualBasic, ale na tegorocznej maturze nie zdarzało się takie rozwiązanie).
Wśród rozwiązań tego problemu dominowały dwie koncepcje

- zbudowanie nowego słowa o odwróconej kolejności znaków w stosunku do danego słowa, a następnie porównanie obu tych słów (programiści w Delphi wykorzystali funkcję *ReverseStrin*g() do odwracania kolejności znaków w słowie)
- porównywanie kolejnych znaków w danym słowie: pierwszego z ostatnim, drugiego z przedostatnim itd.

Przykłady rozwiązań z prac zdających:

w języku Java:

```
   private static boolean CzyPalindrom(String slowo){
        String tmp = "";
        for(int i = slowo.length()-1; i >= 0; i--) {
            tmp += slowo.charAt(i);
        }
        return slowo.equals(tmp);
    }
```

w języku C++

```
     int len = strlen(slowo);
     bool flaga=true;
     for (int j=0;j<(len/2);++j)
     {
          if(slowo[j]!=slowo[len-1-j]) {flaga=false; break;}
```

```
}
if (flaga) pal++;
```

W niektórych rozwiązaniach spośród tych, które porównywały poszczególne znaki w słowie, indeks pętli przebiegał po całej długości słowa (zamiast tylko do połowy), niepotrzebnie dublując operacje porównania tych samych znaków – na przykład pierwszego z ostatnim, a później ostatniego z pierwszym.

Następne punkty zadania 5 wymagały wykonania operacji na łańcuchach znaków: wycięcia wybranego fragmentu łańcucha znaków, wyszukania pozycji podłańcucha w łańcuchu znaków i porównywania łańcuchów.

Wszystkie dostępne na maturze języki programowania, a także arkusz kalkulacyjny, posiadają narzędzia ułatwiające tę pracę. Ale zdający w większości nie znają (a może tylko nie umieją stosować?) funkcji wbudowanych w typ/klasę String, w szczególności nie znają:

— funkcji służących do wycięcia znaków z łańcucha *A*
- w Pascalu jest to funkcja *Copy()*
- w C++ jest *A.substr()*
- w języku Java jest metoda *A.substring()*
- w arkuszu kalkulacyjnym są to formuły *=LEWY(), =PRAWY(), =FRAGMENT.TEKSTU()* lub *=MID()* w OpenOffice

— funkcji służących do wyszukiwania i określenia pozycji łańcucha *B* wewnątrz łańcucha *A*:
- w Pascalu jest to funkcja *Pos(B,A)*
- w C++ jest *A.find(B)*
- w języku Java jest *A.indexOf(B)*
- w arkuszu kalkulacyjnym jest formuła *=ZNAJDŹ(B;A)*.

Znajomość tych funkcji znakomicie ułatwiłaby rozwiązanie zadania 5.

Ponadto maturzyści piszący własne programy mieli tendencję do przechowywania łańcuchów znaków w tablicach typu *char* zamiast w zmiennych typu *String*, co jeszcze bardziej utrudniło im zaprogramowanie żądanych operacji na tekstach.

Oto przykłady rozwiązań zadania 5b – zliczanie takich par słów, gdzie słowo *A* zawiera w sobie słowo *B*:

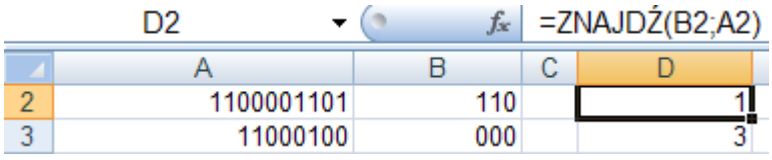
w języku C++ – fragment kodu który zlicza takie pary słów *A* i *B,* że *A* zawiera *B*

```
int len = strlen(slowo);
int len2 = strlen(slowo2);
int BwewA = 0;
for (int x=0; x<len; ++x)
 {
      bool BcA=true;
      for (int y=0; y<len2; ++y)
            if (slowo2[y]!=slowo[x+y]) {BcA=false; break;}
      if (BcA) BwewA++; break;}
}
```

w języku Java – funkcja, która testuje, czy słowo *B* jest zawarte w słowie *A*:

```
private static boolean CzyBzawartewA(String a, String b){
        return a.indexOf(b) != -1;
    }
```

w Excelu – formuła *=ZNAJDŹ()* zwraca pozycję łańcucha *B* w słowie *A*



W dalszej części zadania 5 należało wyszukać wspólne prefiksy i sufiksy słów *A* i *B*, tak by można było utworzyć najkrótszy łańcuch *C* zawierający słowo *A* i słowo *B*.
Oto przykład kodu w języku Java, który zwraca długość *L* największego sufiksu słowa *B,* równego prefiksowi słowa *A*:

```
String A,B;
int ilewspolnychBA() {
      int na=A.length();
      int nb=B.length();
      int L=0;
      for (int k=nb-1; k>0; k--) {
            String SB=B.substring(nb-k,nb);   // k znaków z końca łańcucha B
            String SA=A.substring(0,k);       // k znaków z poczatku łańcucha A
                  if (SA.compareTo(SB)==0) {L=k; break;}   // porównaj
            }
      return L;
}
```

Z innej pracy: przykład kodu w języku Java, który wyszukuje najdłuższy wspólny sufiks *B* i prefiks *A*, a następnie buduje słowo *C,* łącząc początkowy fragment słowa *B* (bez sufiksu) ze słowem *A*.

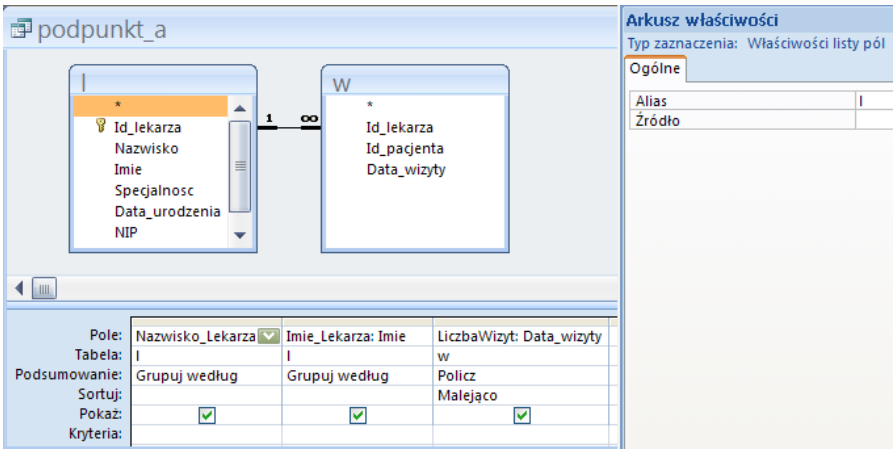
```
private static String StworzC(String a, String b) {

   if (CzyBzawartewA(a, b)) return a;

   for (int i = Math.min(b.length()-1, a.length()-1); i > 0; i--)
     if (a.substring(0, i).equals(b.substring(b.length()-i)))
           {
            return b.substring(0, b.length()-i).concat(a);
           }
   return "";
}
```

W poprawnym rozwiązaniu należałoby jeszcze sprawdzić możliwość połączenia słów w odwrotnej kolejności: *A* z *B* (być może istnieje dłuższy wspólny sufiks *A* i prefiks *B*) i wybrać korzystniejszy wariant.

W zadaniu szóstym dane z trzech plików tekstowych należało połączyć w jedną relacyjną bazę danych. W większości przypadków zdający potrafili to zrobić. Wykorzystując program bazodanowy z pakietu biurowego (na ogół był to program Microsoft Access), zdający

- zaimportowali dane z trzech plików i utworzyli z nich trzy tabele
- zdefiniowali klucze główne w tabelach
- połączyli tabele relacjami typu *jeden-do wielu*
- utworzyli kwerendy wybierające i podsumowujące, z których uzyskali odpowiedzi na pytania postawione w zadaniu.

Nieliczni zdający mieli problem z utworzeniem relacji, zdarzały się błędy lub brak utworzonych relacji, ale byli i tacy, którzy bardzo sprawnie posługiwali się oprogramowaniem, definiując nawet aliasy dla tabel, aby ułatwić sobie konstruowanie kwerend. Na przykład zdający, zamiast używać długiej(?) pełnej nazwy tabeli *lekarze,* zdefiniował krótki, jednoliterowy alias *l* i użył go w kwerendzie.

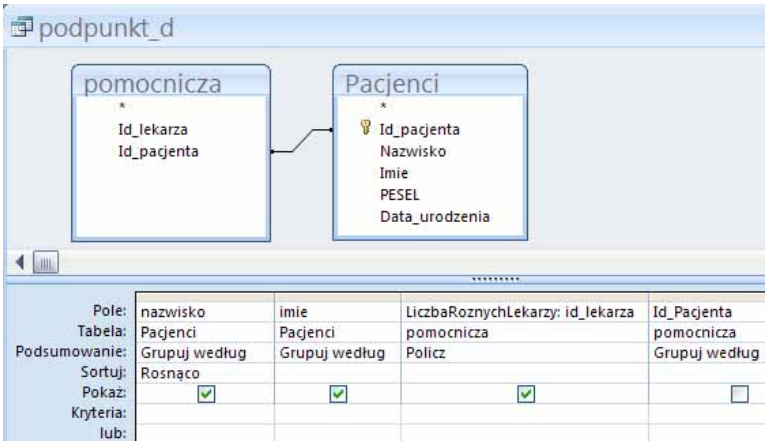


Zdarzały się przypadki, że zdający poprawnie wyselekcjonowali dane w odpowiedzi na pytanie, ale nie uporządkowali ich w wymagany sposób, tracąc cenne punkty. Przyczyną było prawdopodobnie roztargnienie i nieuważne czytanie treści zadania, bowiem zdarzało się to w jednym podpunkcie zadania, a w innym już nie, w tej samej pracy egzaminacyjnej.

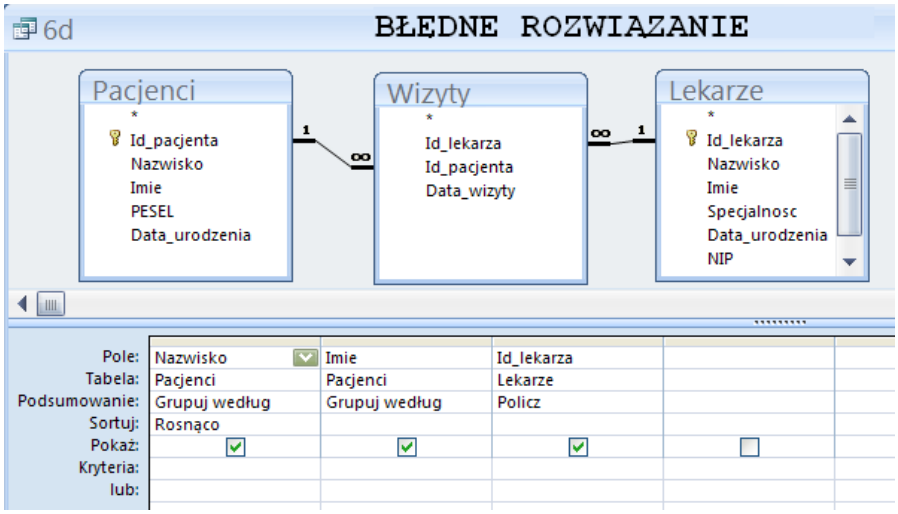
Jedyny poważniejszy problem napotkali zdający w zadaniu 6d), gdzie należało utworzyć dla każdego pacjenta zestawienie zawierające informację, u ilu lekarzy się on leczył. Nie wszyscy wykonali to zadanie poprawnie.

Najprościej było rozwiązać problem w dwóch etapach:

1. kwerenda pomocnicza grupuje identyfikatory pacjentów oraz identyfikatory lekarzy, u których leczyli się ci pacjenci
2. właściwa kwerenda grupuje pacjentów i zlicza identyfikatory lekarzy:

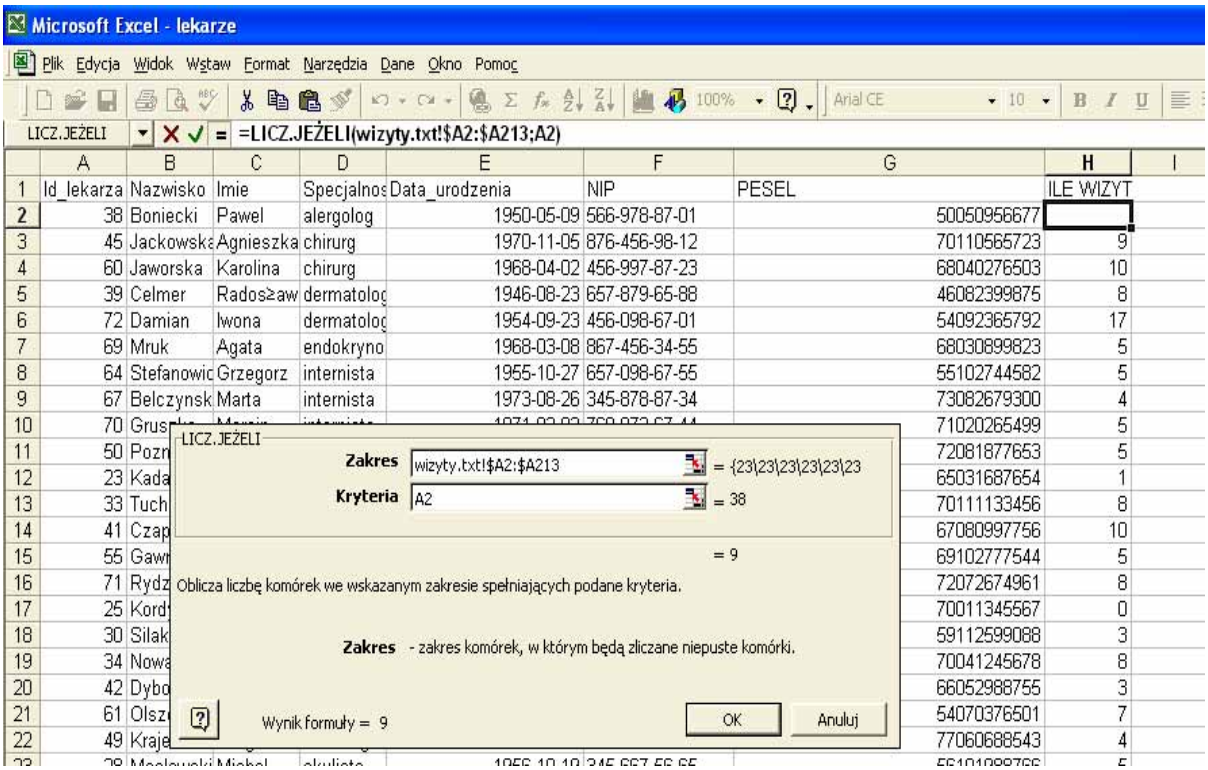


Dużo osób wykonało to zadanie błędnie, pomijając etap grupowania lekarzy. Zdający skonstruowali tylko jedną kwerendę, która zliczyła wszystkie wizyty każdego pacjenta bez grupowania ich względem lekarzy:



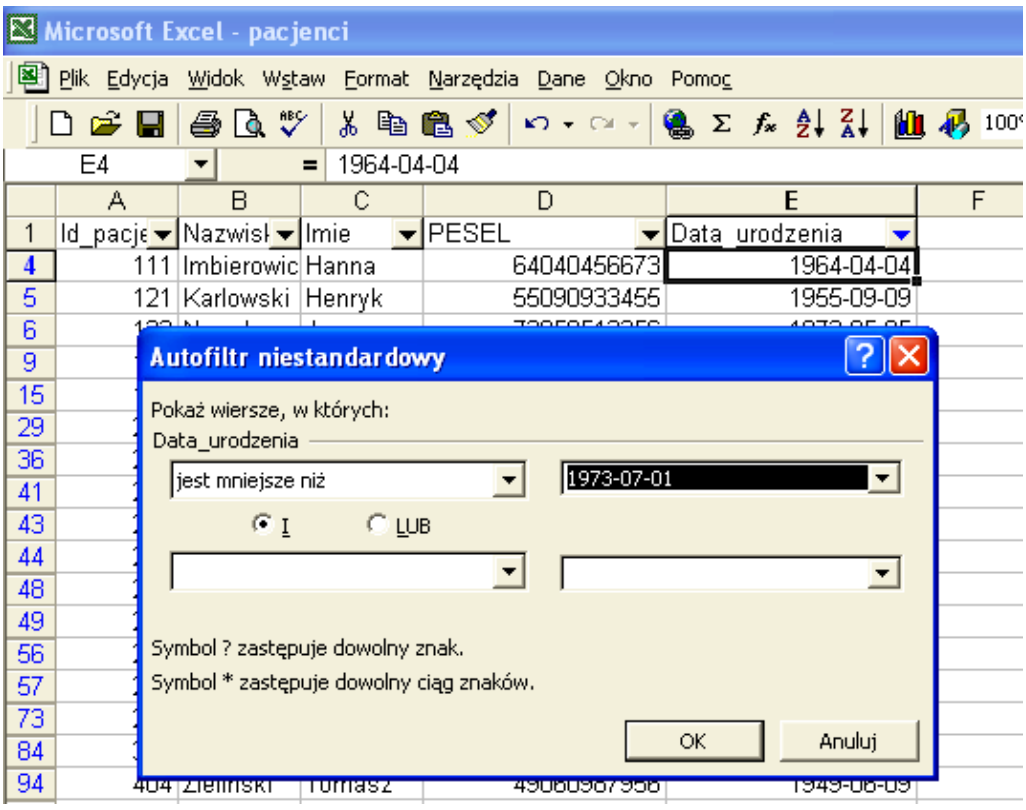
Zauważmy raz jeszcze, że bardzo często maturalne zadania bazodanowe można w łatwy sposób rozwiązać za pomocą arkusza kalkulacyjnego, bez korzystania z bardzo zaawansowanych opcji (typu tabele przestawne).

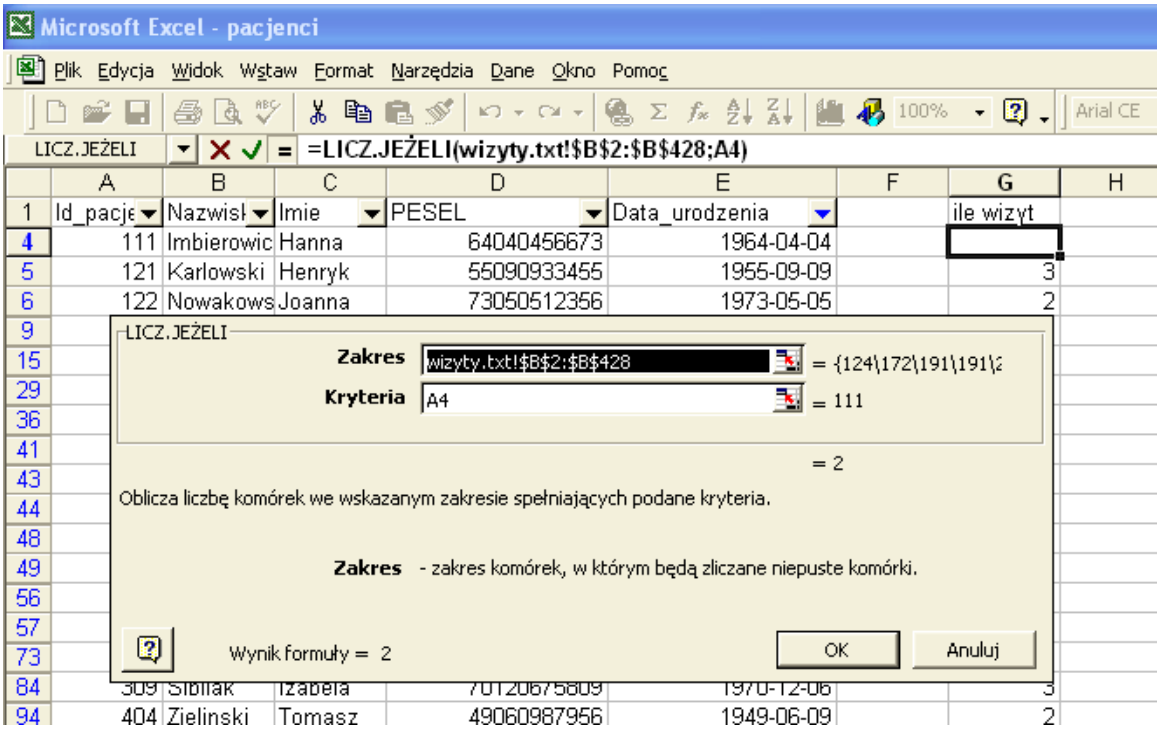
W podpunkcie a) należało utworzyć zestawienie zawierające nazwiska i imiona lekarzy oraz liczbę wizyt przeprowadzonych przez każdego lekarza. Zestawienie miało być posortowane wg liczby wizyt. Żeby rozwiązać to zadanie przy pomocy arkusza kalkulacyjnego, trzeba wykorzystać 2 pliki: `wizyty.txt` oraz `lekarze.txt`. Najpierw trzeba wczytać te pliki do arkusza kalkulacyjnego. Potem wystarczy zliczyć, ile razy pojawia się odpowiedni identyfikator lekarza z pliku `lekarze.txt` w pliku `wizyty.txt`, wykorzystując funkcję LICZ.JEŻELI, wpisując w kryteria adres komórki z identyfikatorem lekarza. Po zastosowaniu sortowania i wybraniu odpowiednich kolumn otrzymujemy żądane w zadaniu zestawienie. Na rysunku przedstawiamy przykład fragmentu rozwiązania.



W podpunkcie b) trzeba było podać zestawienie pacjentów urodzonych przed 1 lipca 1973 oraz liczbę wizyt lekarzy. Aby rozwiązać to zadanie w arkuszu kalkulacyjnym, wykorzystujemy opcję AUTOFILTRU oraz, podobnie jak w podpunkcie a), funkcję LICZ.JEŻELI, z tym, że tym razem

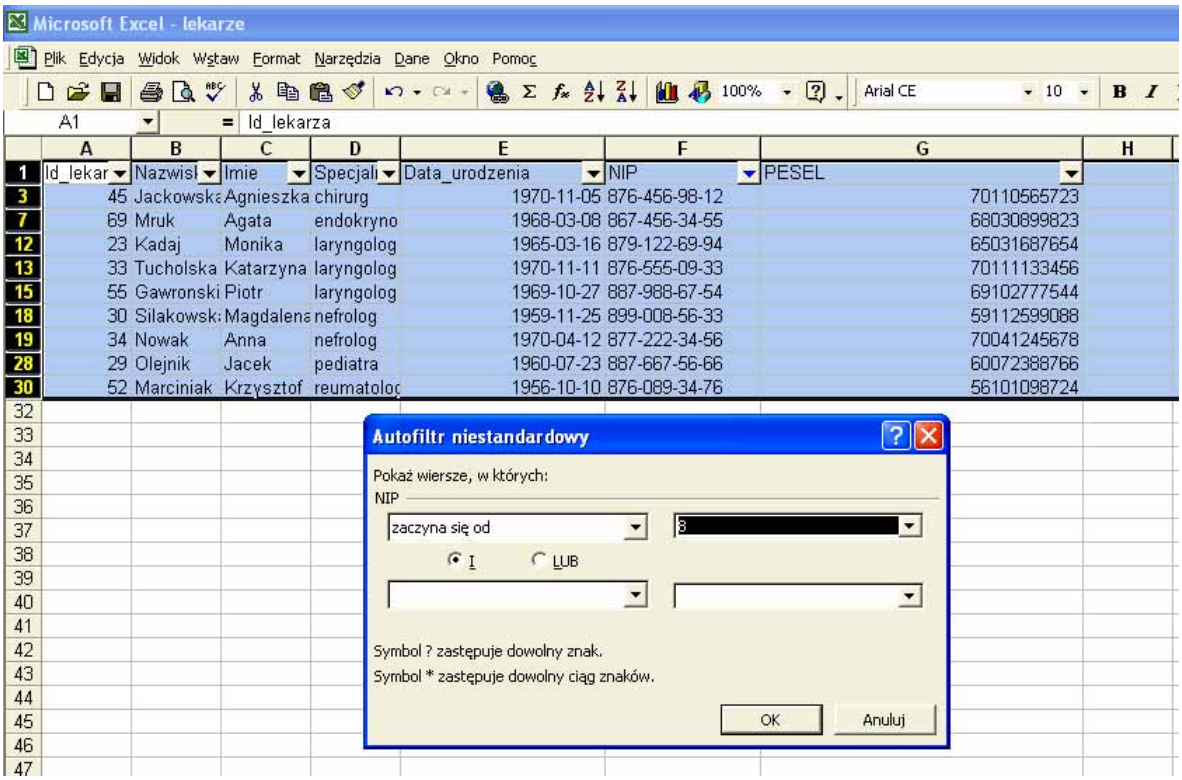
parametrem w kryteriach będzie adres komórki z identyfikatorem pacjenta. Po posortowaniu otrzymanego zestawienia uzyskujemy rozwiązanie za 3 punkty. Fragment rozwiązania przedstawiony jest na rysunkach:





Podpunkt c) był zadaniem bardzo prostym, ponieważ wymagał wykorzystania informacji tylko z jednego pliku `lekarze.txt`. Z tego powodu za zadanie można było otrzymać jedynie 2 punkty. W rozwiązaniu należało podać zestawienie lekarzy, których numer NIP rozpoczyna się od cyfry 8. Na początku sortujemy lekarzy według specjalności. Potem przydaje się funkcja

AUTOFILTR (autofiltr niestandardowy) do wybrania lekarzy z odpowiednim numerem NIP. Na rysunku przedstawiamy fragment rozwiązania.



Podpunkt d), w którym należało podać zestawienie zawierające informacje, u ilu lekarzy leczył się każdy pacjent, również można było rozwiązać za pomocą arkusza kalkulacyjnego. Najprościej ponownie wykorzystać funkcję LICZ.JEŻELI do sprawdzenia, ile razy dany identyfikator pacjenta znalazł się w pliku `wizyty.txt`. Jednak takie rozwiązanie będzie prawidłowe tylko przy założeniu, że pacjent odwiedzał danego lekarza tylko jeden raz, dlatego należy ten podpunkt rozwiązać inaczej, na przykład stosując wspomniane wcześniej autofiltr i odpowiednie sortowanie.

Zdarzyło się także rozwiązanie zadania 6 prezentujące zupełnie nowy typ podejścia do rozwiązania problemu. Zdający rozwiązał poprawnie całe zadanie, nie korzystając z żadnego programu bazodanowego. Samodzielnie napisał i uruchomił program w języku C++, aplikację konsoli, w której pobrał dane ze źródłowych plików tekstowych, a wyniki zapisał również w plikach tekstowych. Oczywiście, odbyło się to dużym nakładem jego pracy.

Podsumowanie

Jak ukierunkować przygotowanie następnego rocznika maturzystów do egzaminu z informatyki? Przede wszystkim warto zachęcić uczniów do rozwiązania zadań z wybranych arkuszy maturalnych z lat poprzednich. I to zanim podejmą decyzję o przystąpieniu do egzaminu.

Wystarczyło uzyskać 30%, aby zdać egzamin. Oznacza to poprawne wykonanie półtora zadania z drugiej, praktycznej części egzaminu, które niewiele wykraczają poza podstawowy poziom nauczania technologii informacyjnej, oraz sensowne rozpoczęcie pracy nad dowolnym innym zadaniem arkusza. To nie są wygórowane wymagania.

Uzyskanie punktów za odpowiedzi do pytań testowych nie jest łatwe, wymaga bowiem zaznaczenia wszystkich poprawnych odpowiedzi do każdego pytania. Na ogół zdający mając fragmentaryczną wiedzę, zaznaczają jedną z poprawnych odpowiedzi, ale nie zaznaczają wszystkich i w związku z tym tracą punkt za całe pytanie. Taki system punktacji różnicuje dobrze przygotowanych do egzaminu i tych, którzy mają tylko powierzchowną wiedzę na dany temat.

Często jednak zdający tracą pojedyncze punkty przez nieuwagę i niestaranność w opracowaniu odpowiedzi, zapominając o posortowaniu wyników kwerendy w wymagany sposób lub nie przedstawiając wyników liczbowych z żądaną dokładnością, lub nie opisując osi wykresu, itp.

Algorytmika jest „piętą Achillesową” większości zdających. Niektórzy nie potrafią nawet poprawnie sformułować specyfikacji problemu. A przecież skrupulatne wypisanie listy: „co jest dane, a co należy obliczyć” powinno być rutynowym wstępem do rozwiązania każdego zadania, znanym z lekcji fizyki, chemii czy matematyki.

Często zdający nie znają prostych klasycznych algorytmów. A przecież warto je znać, aby nie „wyważać otwartych drzwi” w konstruowaniu rozwiązań dla popularnych problemów. Warto także docenić ich pomysłowość i wzorując się na nich, uczyć się tworzenia algorytmów szybkich, nieobciążających nadmiernie pamięci komputera i elegancko sformułowanych.

Od zdających, którzy sięgają po najwyższe oceny, wymagana jest umiejętność rozwiązywania problemów drogą samodzielnego programowania. Bez względu na wybór języka większość zdających programistów ograniczała się do stosowania wyłącznie najprostszych typów zmiennych i ewentualnie tablic, mimo że C++, język najczęściej używany przez maturzystów, a także Java, dają o wiele większe możliwości. Tymczasem bardzo rzadko w rozwiązaniach pojawia się *vector* czy definicja własnej klasy.

Zdający nie wykorzystują podstawowych funkcji – metod zdefiniowanych w standardowych środowiskach programistycznych, za to mozolnie budują własne bloki instrukcji realizujące potrzebne operacje. Boleśnie odczuli to podczas rozwiązywania zadania 5.

Dobrze, że to potrafią zrobić, ale po co? Współczesne środowiska IDE rozwijają się w takim kierunku, aby dostarczyć programiście jak najwięcej gotowych „klocków” do składania: gotowych klas i interfejsów, z mnóstwem wbudowanych w nie funkcji. Oferują także rozbudowane systemy pomocy, które na bieżąco podpowiadają programiście, wyświetlając listę właściwości i metod dostępnych dla zmiennej każdego typu. Nie trzeba wcale uczyć się tego na pamięć, wystarczy umieć skorzystać z możliwości edytorów IDE. I takie właśnie podejście warto pokazać w szkole naszym najlepszym uczniom interesującym się informatyką, a w szczególności programowaniem.