POMORSKA LIGA ZADANIOWA *ZDOLNI Z POMORZA*

Konkurs dla uczniów szkół ponadpodstawowych i ponadgimnazjalnych województwa pomorskiego w roku szkolnym 2021/2022

Etap II-powiatowy

Przedmiot: Informatyka

Przed przystąpieniem do rozwiązywania zadań zapoznaj się z instrukcją.

INSTRUKCJA

1. Oprócz arkusza z treścią zadań otrzymujesz 2 załączniki niezbędne do rozwiązania zadania numer 2. Ich nazwy są określone w treści zadania. Przed przystąpieniem do rozwiązywania sprawdź, czy na pewno pobrała(e)ś te wszystkie pliki. Nie wolno używać własnych plików zamiast tych załączników wyraźnie wymienionych w treści zadania.
2. Zwróć uwagę, aby pliki zawierające rozwiązania oraz pliki z danymi testowymi (takie pliki będziesz tworzyć samodzielnie rozwiązując zadania 3, 4 oraz 5) miały zawartość i nazwy takie jakie określono w treściach zadań.
3. Nie przesyłaj do oceny innych plików niż te określone w treści zadań.
4. Pliki z rozwiązaniami przesyłasz organizatorom Pomorskiej Ligi Zadaniowej zgodnie z odrębną instrukcją.
5. Przy rozwiązywaniu zadań powinno się wykorzystywać te środowiska i narzędzia programistyczne, którymi posługujesz się w szkole lub w domu. W szczególności dopuszcza się następujące środowiska:
   a) systemy operacyjne – zarówno z grupy Windows, jak i dystrybucje systemu Linux,
   b) pakiety oprogramowania biurowego- Microsoft Office, ale również wersje otwarte np. Libra Office,

c) języki programowania – C/C++,C#, Free Pascal, Java,Python (kompilatory adekwatne do używanych środowisk systemu operacyjnego np. DEV, Code Block, Eclipse, GCC,G++ itp.),

d) wizualne środowiska programowania – Scratch, Logomocja lub inne mutacje LOGO,

e) nie określa się szczegółowo numerów wersji używanego oprogramowania, aby uczeń mógł je elastycznie dostosować do używanych w szkole, ale w przypadku języków programowania prosimy o dokładne podanie (np. w odrębnym pliku tekstowym) jaka wersja kompilatora (względnie jakie środowisko programistyczne) było wykorzystywane, aby adekwatnego użyć przy ocenie pracy z zastrzeżeniem punktu 6a.

6. W przypadku rozwiązań związanych z używaniem języków programowania:

a) powinno się używać kompilatorów bez ograniczonej dostępności (np. związanej z ich komercyjnym charakterem),

b) dopuszcza się używanie wyłącznie standardowej biblioteki (bibliotek) języka, nie jest dozwolone dołączanie zewnętrznych bibliotek np. crt, graph (poza sytuacjami wynikłymi z treści zadania np. próbą realizacji rysunku wynikającego z treści zadania),

c) nie jest dopuszczalne otwieranie przez program innych programów, plików (poza tymi z danymi wejściowymi i wyjściowymi), ani tworzenie nowych plików (np. tymczasowych) oraz tworzenie innych procesów lub wątków,

d) błąd kompilacji przy sprawdzaniu jest traktowany jako błąd składni i sprawdzający nie ma obowiązku dalszej analizy takiego rozwiązania choć może uwzględnić poprawny zarys samego algorytmu przyznając znacząco mniejszą liczbę punktów. Podobna uwaga dotyczy pojawienia się nietypowych błędów wykonania w trakcie uruchamiania programów (np. naruszenie zasad ochrony pamięci),

e) rozwiązania nie powinny wykorzystywać plików nagłówkowych typowych dla środowisk DOS/Windows np. conio.h lub windows.h (dotyczy języka C++),

f) rozwiązania nie powinny naruszać bezpieczeństwa systemowego w środowisku, w którym są sprawdzane.

7. Programy nie powinny zajmować się testowaniem poprawności danych. zakłada się, że ma ona miejsce.
8. Proszę zwrócić uwagę na samodzielność rozwiązań.

Życzymy powodzenia!

Zadanie 1

Pewien wykładowca wyższej uczelni podsumowuje prowadzony przez siebie przedmiot przy pomocy testu. Test rozwiązywany przez studentów składa się zawsze z 7 pytań. W każdym z pytań mamy 4 warianty odpowiedzi do wyboru. Cztery z tych pytań czyli pytania od 1 do 4, to pytania jednokrotnego wyboru (czyli tylko jeden z proponowanych wariantów odpowiedzi jest poprawny) , zaś trzy pozostałe (pytania 5-7) to pytania wielokrotnego wyboru. W pytaniach wielokrotnego wyboru zawsze przynajmniej jeden z wariantów odpowiedzi jest poprawny, ale poprawne mogą być nawet wszystkie. Sposób punktowania odpowiedzi studentów oraz wystawiania im ocen jest następujący:

a) za wskazanie jedynego poprawnego wariantu odpowiedzi w pytaniach od 1 do 4 student otrzymuje 1 pkt, za błędną odpowiedź 0 punktów,
b) w przypadku pytań wielokrotnego wyboru student otrzymuje 2 punkty jeżeli wskaże wszystkie poprawne warianty odpowiedzi. Jeżeli wskaże nie wszystkie poprawne warianty, ale nie będzie wśród wybranych wariantów odpowiedzi błędnej to uzyskuje liczbę punktów proporcjonalną do liczby wskazanych wariantów z poprawną odpowiedzią np. jeśli były 3 poprawne warianty , a student wskazał tylko jeden to otrzymuje (1/3)*2=0,67 (wyniki ułamkowe zaokrąglamy do dwóch miejsc po przecinku), a jeśli wskazał 1 wariant spośród dwóch poprawnych wariantów odpowiedzi to otrzymuje (1/2) *2=1 punkt. Student zawsze otrzymuje 0 punktów w przypadku pytań tego typu, jeśli wśród wskazanych przez niego wariantów odpowiedzi choć jeden będzie niepoprawny,
c) punkty uzyskane za poszczególne pytania są sumowane, a następnie wykładowca wystawia oceny stosując następującą skalę:

od 0, ale mniej niż 4.25 – ocena niedostateczna

od 4.25, ale mniej niż 6 – ocena dostateczna

od 6, ale mniej niż 7 – ocena dostateczna plus

od 7, ale mniej niż 8 – ocena dobra

od 8, ale mniej niż 9 – ocena dobra plus

od 9 – ocena bardzo dobra

d) studentom, którzy uzyskali ocenę niedostateczną, ale zdobyli przynajmniej 3,5 punktu wykładowca daje jeszcze dodatkową szansę zaliczenia przedmiotu zapraszając ich na rozmowę ustną zwaną w żargonie studenckim *Dopytką*. Studenci, którzy uzyskali mniej niż 3,5 punktu takiej szansy już nie mają i otrzymują definitywnie ocenę niedostateczną.

Ze względu na czytelne i stałe od lat reguły wykładowca chciałby zautomatyzować proces oceny swojego przedmiotu i przeprowadzać te ocenę za pomocą arkusza kalkulacyjnego, którym się posługuje. Wie, że dzięki możliwościom arkusza mógłby też uzyskać po wprowadzeniu danych związanych z testem zaliczeniowym kilka pożytecznych informacji podsumowujących.

Twoje zadanie polega na tym, aby pomóc wykładowcy w jego zamierzeniach przygotowując odpowiednie rozwiązania w arkuszu kalkulacyjnym (**ze względu na naszego wykładowcę nie wchodzi w grę napisanie programu w jakimkolwiek języku programowania – rozwiązanie musi być przedstawione za pomocą arkusza kalkulacyjnego).**

Przyjmijmy dwa założenia dotyczące przygotowywanego rozwiązania:

a) w arkuszu kalkulacyjnym zawsze będzie umieszczona informacja o wzorcu odpowiedzi dla danego wariantu testu w postaci jak w poniższym przykładzie:

Pyt1	Pyt2	Pyt3	Pyt4	Pytanie 5				Pytanie 6				Pytanie 7			
A	A	D	C	1	1	1		1		1			1	1	1

W przypadku pytań 1-4 wzorzec wskazuje po prostu poprawną odpowiedź (warianty odpowiedzi są oznaczone literami od A do D). W przypadku pytań wielokrotnego wyboru wstawienie cyfry 1 w odpowiedniej kratce wskazuje, że dany wariant odpowiedzi jest poprawny. W podanym przykładzie dla pytania 5

poprawnymi wariantami odpowiedzi były zatem warianty A, B i C (kratka D pozostaje pusta), dla pytania 6 warianty A i C, zaś dla pytania 7 warianty B,C oraz D,

b) do arkusza można wpisać ręcznie , albo wprowadzić z pliku tekstowego dane o wynikach maksymalnie 12 studentów (w grupach o takiej maksymalnej liczebności wykładowca przeprowadza test zaliczeniowy) wg następującego szablonu podanego przykładowo dla jednego studenta:

Kod osoby	Pyt1	Pyt2	Pyt3	Pyt4	Pytanie 5				Pytanie 6				Pytanie 7			
	A	B	D	D	1			1	1		1				1	1

Kod jest dowolnie ustalonym ciągiem liter i cyfr – nie wprowadzamy bezpośrednio nazwisk studentów. Litery dla pytań 1-4 oznaczają wariant odpowiedzi wskazany przez zdającego, a 1 ustawione w odpowiedniej kratce dla pytań 5-7 jakie warianty odpowiedzi zdający wybrał w przypadku pytań wielokrotnego wyboru. Zauważmy, że ten przykładowy zdający, gdyby faktycznie obowiązywał wzorzec poprawnych odpowiedzi przedstawiony w punkcie a) uzyskałby wg omówionych wcześniej zasad punktowania testu 2 punkty za odpowiedzi na pytania 1-4 (poprawnie odpowiedział tylko na pytania 1 oraz 3), za pytanie 5 – 0 punktów (błędnie wskazał wariant D), za pytanie 6 – 2 punkty (bezbłędnie wskazał poprawne warianty odpowiedzi), a za pytanie 7-1,33 punktu (wskazał 2 z 3 poprawnych wariantów odpowiedzi). Razem uzyskałby 5.33 punktu i ocenę dostateczną. Naturalnie student(ka) może w ogóle nie odpowiadać na dane pytanie, pozostawia wówczas kratkę pustą pytania (1-4) lub wszystkie kratki puste (pytania 5-7).

Dysponując w arkuszu kalkulacyjnym danymi o wzorcu poprawnych odpowiedzi i danymi o wynikach grupy co najwyżej 12 studentów (te wyniki w celach testowych możesz sobie przyjmować dowolnie) rozwiąż w tym samym arkuszu następujące problemy tzn. spraw, aby zastosowane przez Ciebie formuły, funkcje itp. funkcjonowały poprawnie dla każdej kolejnej grupy studentów (nowe wyniki studentów i nowy wzorzec odpowiedzi). Nie zapomnij, aby wyniki były prawidłowe

także wtedy, gdy grupa niekoniecznie liczy 12 osób, ale wprowadzono dane mniejszej grupy osób.

I. Przypisz każdej z osób w grupie zdających poprawną liczbę punktów obliczoną wg opisanych wyżej kryteriów oraz ocenę wg podanej wyżej skali.
II. Spowoduj, aby w kolumnie na prawo od oceny studenta z oceną niedostateczną pojawiał się automatycznie tekst „Dopytka" jeżeli wg opisanych wyżej kryteriów student(ka) na nią zasłużył(a).
III. Przedstaw na wykresie ile mamy w danej grupie poszczególnych ocen traktując „Dopytkę" jako osobna kategorię, inną niż ocena niedostateczna bez prawa do rozmowy ustnej.
IV. Przedstaw na wykresie jak wygląda procentowo proporcja osób, które przedmiot zaliczyły do tych, które go nie zaliczyły („Dopytka" w tym punkcie to formalnie brak zaliczenia).
V. **Wskaźnik łatwości pytania** to wyrażony w procentach stosunek liczby punktów, które uzyskali razem **wszyscy** odpowiadający na dane pytanie do maksymalnej liczby punktów jaką teoretycznie razem **wszyscy** odpowiadający na to pytanie mogli zdobyć. Oblicz na podstawie danych grupy wskaźniki łatwości poszczególnych pytań oraz przedstaw je na wykresie.
VI. Wykładowca zastanawia się, czy nie wprowadzić od kolejnego roku dość surowego kryterium na podstawie, którego osoba, która dwukrotnie w pytaniach wielokrotnego wyboru (czyli w dwóch z trzech takich pytań) zdobywa 0 punktów (czyli wskazuje choć jeden fałszywy wariant odpowiedzi) nie uzyskuje zaliczenia bez względu na uzyskaną sumę punktów z pozostałych pytań. Naturalnie nadal nie uzyskiwaliby zaliczenia Ci, którzy osiągnęli mniej niż 4,25 punktu. Spraw, aby w arkuszu pojawiała się uzupełniająca informacja o liczbie osób, które nie zaliczyłyby przedmiotu, gdyby obowiązało to kryterium (*Dopytka* przy tym problemie nie ma znaczenia- jeśli ktoś nie uzyskał przynajmniej 4.25 punktu to w tym punkcie uwzględniamy go wśród osób, które nie zaliczają przedmiotu).

VII. Spowoduj, aby po przyciśnięciu umieszczonego w arkuszu dodatkowego przycisku w wybranym przez Ciebie miejscu arkusza (albo w nowym arkuszu) generował się nadający się później do wydruku i przedstawienia studentom raport z zaliczenia przedmiotu. Raport winien dla każdej osoby przedstawiać: kod studenta, liczbę punktów, ocenę i tzw. diagnozę. Diagnoza to jedna z trzech informacji : *Zaliczył(a), Dopytka, Nie zaliczył(a)* (w tym punkcie *Dopytka* to ponownie odrębna kategoria) Naturalnie na potrzeby raportu obowiązują standardowe kryteria, a nie te planowane dopiero na kolejne lata w punkcie VI. Uwaga ! W raporcie generowanym w arkuszu dopuszczamy puste wiersze wynikające z mniejszej od 12 liczby zdających w danej grupie- osoba, która potem zechce sobie wydrukować taki obszar może sobie przecież zaznaczyć do wydruku tylko wiersze jej niezbędne.

Do oceny oddajesz plik(i) zawierający komputerową realizację obliczeń, na podstawie której uzyskasz odpowiedzi na wszystkie postawione w części problemy. Nazwa tego pliku to *Zadanie1* .

10 punktów

Zadanie 2

W plikach ***Zalacznik-Zadanie2-uczniowe.txt*** oraz ***Zalacznik-Zadanie2-wyniki.txt*** umieszczono fikcyjne dane dotyczące pewnego ogólnopolskiego konkursu informatycznego, przeprowadzonego 10 lat temu dla uczniów szkół ponadgimnazjalnych, ale regulamin dopuszczał również start gimnazjalistów. W pierwszym z plików znajdują się w kolejnych wierszach oddzielone znakiem tabulacji podstawowe dane dotyczące każdego z uczestników konkursu (*identyfikator ucznia;imię;nazwisko;nazwa szkoły;miejscowość;klasa;okręg).* W drugim pliku w kolejnych wierszach znajdują się oddzielone średnikiem identyfikator ucznia oraz liczby punktów uzyskane przez danego ucznia za rozwiązania kolejnych zadań w konkursie (zadań było 5, a każde było punktowane w skali od 0 do 100).

Na podstawie danych umieszczonych w załączonych plikach rozwiąż następujące problemy :

1. Utwórz zestawienie pokazujące ilu startujących pochodzi z różnych okręgów.

2. Przyjmując, że tylko imiona dziewcząt kończą się na literę „a" oblicz ile dziewcząt wzięło udział w konkursie.
3. Utwórz zestawienie pokazujące do okręgów o jakim numerze należą wszystkie miasta reprezentowane przez uczestników konkursu. W zestawieniu miasta nie mogą się powtarzać i powinny być pokazane w kolejności alfabetycznej (rosnąco).
4. Utwórz zestawienie zawierające nazwiska najlepszych zawodników (zawodniczek) w każdym z okręgów, które znajdują się w danych. W każdym okręgu jest tylko jedna taka osoba.
5. Utwórz zestawienie, które dla każdego miasta występującego w danych pokaże średni wynik uzyskany przez startujących z tego miasta oraz liczbę zawodników to miasto reprezentujących. Średnie należy podać z dokładnością do 2 miejsc po przecinku.
6. Opierając się na definicji łatwości zadania (pytania) podanej w **Zadaniu 1** utwórz zestawienie zawierające wartość współczynników łatwości dla wszystkich 5 zadań.
7. Oblicz ilu uczniów gimnazjum wzięło udział w konkursie. Przy czym przyjmujemy, że uczeń reprezentuje gimnazjum tylko jeśli w nazwie szkoły pojawia się słowo Gimnazjum oraz że przedstawiciele Zespołu Szkół UMK Gimnazjum i Liceum Akademickie w Toruniu nie byli uczniami gimnazjum, ale liceum w tym zespole.
8. Utwórz zestawienie pokazujące średnie wyniki dla poszczególnych okręgów, ale w rozbiciu na klasy, które reprezentowali uczestnicy. W tym zestawieniu uwzględniamy wyłącznie uczniów szkół ponadgimnazjalnych.
9. Uczniowie będą kwalifikowani do kolejnego etapu konkursu. Nie wiemy jednak jaki będzie punktowy próg kwalifikacyjny. Skonstruuj narzędzie, które pozwoli zadawać pytanie z parametrami. Jednym z parametrów będzie identyfikator osoby, a drugim hipotetyczny próg punktowy. Takie narzędzie pozwoli na sprawdzanie, czy osoba o identyfikatorze podawanym przez uruchamiającego zapytanie kwalifikują się do kolejnego etapu zależnie od progu punktowego, który też określi uruchamiający poprzez potwierdzające wyświetlenie jego imienia i nazwiska jeśli się rzeczywiście kwalifikuje.

Do oceny w tej części zadania oddajesz plik(i) zawierający komputerową realizację obliczeń, na podstawie której uzyskasz odpowiedzi na wszystkie postawione problemy. Nazwa tego pliku to *Zadanie2* lub jeśli jest ich więcej *to* kolejne pliki mają nazwy *Zadanie 2a, Zadanie2b itd.* Ponadto załączasz plik tekstowy *Zadanie2-odpowiedzi.txt* zawierający odpowiedzi dla problemów 2,4,6,7 i 8.

Uwaga ! Odpowiedzi umieszczone w pliku *Zadanie2-odpowiedzi.txt* nie będą mogły być uznane nawet jeśli będą poprawne, o ile nie znajdą potwierdzenia i odzwierciedlenia w zawartości pliku(ów) z komputerową realizacją obliczeń.

11 punktów

Zadanie 3

Słowo x jest anagramem słowa y jeżeli oba słowa mają tę samą długość i słowo x składa się z dokładnie takich samych znaków jak słowo y. Anagramami są np. słowa *arak* i *kara*.

Andrzej rozwiązywał problem polegający na tym, aby podać numery miejsc, w których zaczynają się kolejne podsłowa pewnego słowa *a*, które są anagramami innego (krótszego lub co najwyżej równego) słowa *b*. Uwaga w tym zadaniu przed podsłowo pewnego słowa będziemy rozumieć dowolny ciąg złożony z **kolejnych** znaków tego słowa (także samo słowo).

Np. dla a=*opryropabrpo* i b=*por* takimi numerami byłyby (numeracja znaków od 0) 0, 4,9, gdyż kolejne trzyznakowe podsłowa słowa a (tylko rozważanie podsłów o takiej długości ze względu na długość słowa b ma sens) to: **opr**, pry, ryr, yro, **rop**, opa, pab,abr, brp i **rpo**, a tylko te wyróżnione pogrubieniem są anagramami słowa por i zaczynają się w słowie a kolejno właśnie od miejsc 0, 4 i 9.

Andrzej nie bardzo chciałby się męczyć sprawdzaniem dla kolejnych możliwych do utworzenia podsłów sprawdzaniem czy są one anagramami innego słowa więc kiedy zapoznał się ze znanym algorytmem Rabina-Karpa wyszukiwania wzorca w tekście wpadł na pomysł, że aby sprawdzić czy dwa słowa są swoimi anagramami wyznaczy dla każdego z nich sumę kodów ASCII tworzących je znaków. Słowa zbudowane z tych samych znaków (nawet

występujących w innej kolejności) muszą bowiem mieć takie sumy równe. Ten wniosek jest prawdziwy, ale euforia Andrzeja była chwilowa, gdyż wspomniana zasada nie działa w przeciwną stronę. Może być tak, że słowa mają identyczne sumę kodów ASCII dla swoich znaków, ale wcale nie będą swoimi anagramami. Oto przykład:

baca suma kodów Ascii znaków wynosi 391

babb suma kodów ASCII to również 391

a przecież te słowa nie są anagramami !

Mimo wszystko Andrzej stwierdził, że sumowanie kodów ASCII można wykorzystać do selekcji negatywnej, jeżeli sumy kodów ASCII znaków dla dwóch słów nie będą równe, to takich słów nie trzeba już sprawdzać dalej, bo nie będą na pewno swoimi anagramami.

Z tych obserwacji dla jego problemu powstał taki ogólny algorytm, który dalej będziemy nazywać algorytmem Andrzeja:

1. Stwórz listę podsłów słowa *a* o tej samej długości jak słowo b i dla każdego oblicz sumę kodów ASCII jego znaków.
2. Tylko w przypadku podsłów, dla których obliczona suma kodów ASCII jego znaków jest równa sumie kodów ASCII słowa *b* dokonaj znaną Ci metodą sprawdzenia czy dane podsłowo i słowo *b* mogą być anagramami (bo już wiemy, że sama równość sum kodów ASCII znaków sprawdzanych słów nie wystarcza) i jeśli tak jest zapamiętaj miejsce, w którym w *a* zaczyna się podsłowo. Podsłowami *a* , dla których suma kodów ASCII znaków nie jest równa sumie kodów ASCII znaków słowa *b* nie zajmuj się w ogóle.

Ten algorytm należy wykorzystać w zadaniu do poszukiwania anagramów dla nieco innej sytuacji. Napisz program, który wczytuje z pliku tekstowego m słów (złożonych tylko z małych, dużych liter oraz cyfr, ale bez spacji) o długości dokładnie n znaków (czyli powstaje z tego swego rodzaju tabela dwuwymiarowa m*n znaków) oraz pewne słowo wzorcowe x (także złożonego tylko z małych, dużych liter oraz cyfr - bez spacji). Program wykorzystując opisany wyżej algorytm Andrzeja powinien rozwiązać następujące problemy:

a) podać dla każdego z m słów osobno numery miejsc, w których rozpoczynają się podsłowa będące anagramami x (numeracja znaków w słowie od 0). Jeżeli dla

pewnego ze słów nie ma takich podsłów w ogóle jedynym wynikiem będzie dla niego liczba -1,

b) podać dla każdej kolumny tabeli (m*n) osobno numery miejsc, w których rozpoczynają się podsłowa będące anagramami x (numeracja znaków w kolumnie od 0). Uwaga ! Zerową kolumnę tworzą znaki numer 0 wszystkich tekstów , kolumnę nr 1 znaki numer 1 wszystkich tekstów itd. Jeżeli dla pewnej kolumny nie ma takich podsłów w ogóle to jedynym wynikiem będzie liczba -1,

c) w przypadku, gdy m=n i nasza tabela będzie kwadratowa podać osobno dla tekstów utworzonych ze znaków obu przekątnych numery miejsc, w których rozpoczynają się w tych tekstach podsłowa będące anagramami x. Numerację znaków tekstu z lewej przekątnej zaczynamy od 0, który to numer ma znak położony w lewym górnym rogu naszej tabeli, ostatni numer (m-1 lub n-1) ma znak w prawym dolnym rogu tabeli. Numerację znaków tekstu z prawej przekątnej zaczynamy od 0, który to numer ma znak położony w prawym górnym rogu naszej tabeli, ostatni numer (m-1 lub n-1) ma znak w lewym dolnym rogu tabeli. Jeżeli dla danej przekątnej nie ma takich podsłów, które są anagramami x jedynym wynikiem będzie dla niej liczba -1,

d) jeżeli słowo x ma nieparzystą liczbę znaków to ze słowa x utworzymy symbol podobny do litery T np. gdyby było to słowa barka wyglądałby on tak:

b a r k a
b
a
r
k
a

Program powinien obliczyć w ilu miejscach, w których taką literę da się umieścić w tabeli m*n znaków występujące w tabeli w tym miejscu litery w poziomie (górna „krawędź" T) tworzą anagram x oraz niezależnie litery w pionie („nóżka" T, bez znaku należącego do górnej krawędzi T) też tworzą anagram (może inny) słowa x . Dla przykładu, gdyby w pewnym miejscu tabeli istniał następujący układ:

k a r a b
b
a
r

a
k

to mamy do czynienia z symbolem (literą T), który spełnia te założenia dla słowa *barka*, gdyż słowa *karab* oraz *barak* są niezależnie od siebie anagramami słowa *barka*. By lepiej objaśnić sformułowanie dotyczące miejsc, w których w tabeli m*n może się zmieścić litera T utworzona wg zasad jak wyżej przyjrzyjmy się poniższemu schematowi . W nim m=5 , n=6, słowo x ma 3 znaki, a znakiem „*” oznaczono znaki tekstów. Znaki celowo przedstawione z dodatkowym cieniowaniem pokazują dwa wybrane położenia w których można umieścić literę T (wszystkie powinien znaleźć program- dla ułatwienia w tym schemacie jest ich 8, niektóre się „przysłaniają”). Przyjmijmy, że nie wchodzi np. w grę obracania, czy też stawianie odwrotnie litery T. Rozważamy tylko takie położenie dwóch słów względem siebie jak na poniższym schemacie i w tych położeniach szukamy dwóch anagramów słowa x wg podanego wyżej opisu.

*	*	*	*	*	*
*	*	*	*	*	*
*	*	*	*	*	*
*	*	*	*	*	*
*	*	*	*	*	*

e) dla wszystkich realizowanych poszukiwań anagramów w punktach a-c łącznie (o ile poszukiwania w punktach c występują , bo one zależą od wartości m, n) należy policzyć ile razy wymyślona pierwotnie przez Andrzeja metoda zawiodła tzn. obliczone sumy kodów ASCII dla dwóch sprawdzanych słów były równe , a słowa jednak po do dodatkowym sprawdzeniu anagramami się nie okazały. Nie uwzględniamy w wyniku dotyczącym tego punktu „poszukiwań” litery T w tablicy m*n, aby nie narzucać algorytmu dla tych „poszukiwań”.

Dane wejściowe

W pliku *teksty.txt* w pierwszym wierszu umieszczono dwie liczby naturalne. Pierwsza oznacza liczbę tekstów **m** umieszczonych w kolejnych wierszach (m jest równe co najmniej 5, ale nie jest większe niż 50) , druga oznacza długość każdego tekstu, która jest identyczna i równa **n** (n jest równe przynajmniej 2, ale nie jest większe niż 100). W drugim wierszu podane jest wzorcowe słowo **x** również składające się tylko z małych liter, dużych liter oraz cyfr (bez spacji) o długości przynajmniej 2, ale nie przekraczającej 25 znaków. Przyjmijmy, że długość słowa x jest zawsze mniejszy niż m i mniejsza niż n. W kolejnych m wierszach umieszczono teksty o długości n złożone wyłącznie z małych liter, dużych liter i cyfr-bez spacji.

Dane wyjściowe

W pliku *anagramy.txt* w pierwszych m wierszach znajdują się oddzielone spacją liczby naturalne z zakresu od 0 do n-2 wskazujące numery miejsc, na których w kolejnych tekstach rozpoczynają się podsłowa będące anagramami x znalezione przy pomocy algorytmu Andrzeja albo liczba -1 jeżeli w danym tekście nie ma podsłowa, które byłoby anagramem x (pierwszy wiersz odpowiada tekstowi numer 1, drugi tekstowi numer 2 itd.). W następnych n wierszach znajdują się oddzielone spacją liczby naturalne z zakresu od 0 do m-2 wskazujące numery miejsc, na których w kolejnych kolumnach tablicy m*n rozpoczynają się podsłowa będące anagramami x znalezione przy pomocy algorytmu Andrzeja albo liczba -1 jeżeli w danej kolumnie nie ma podsłowa, które byłoby anagramem x. Jeżeli m=n to w kolejnych dwóch wierszach znajdują się oddzielone spacją liczby naturalne z zakresu od 0 do n-1 wskazujące numery miejsc, na których w pierwszym z tych 2 wierszy w tekście zbudowanym ze znaków z lewej przekątnej, a w drugim w tekście zbudowanym ze znaków z prawej przekątnej rozpoczynają się podsłowa będące anagramami x znalezione przy pomocy algorytmu Andrzeja. Jeżeli na danej przekątnej żadne z podsłów utworzonych na podstawie tekstu z tej przekątnej nie jest anagramem x to w odpowiadającym tej przekątnej wierszu pliku *anagramy.txt* pojawia się liczba -1 . Liczba -1 powinna się pojawić w obu tych wierszach odpowiadających przeszukiwaniu przekątnych także wtedy, gdy m nie jest równe n

i nie jest realizowane poszukiwanie anagramów wzdłuż przekątnej. W przedostatnim wierszu pliku *anagramy.txt* powinna być umieszczona jedna liczba naturalna odpowiadająca na pytanie w ilu miejscach, w których da się umieścić w tabeli m*n literę T zbudowaną z x (szczegóły w punkcie d) znaki ją tworzące stanowią niezależnie od siebie dwa anagramy słowa x (szczegóły w punkcie d). Jeżeli x nie ma nieparzystej liczby znaków to w przedostatnim wierszu pojawia się liczba -1.W ostatnim wierszu pliku znajduje się jedna liczba naturalna stanowiąca odpowiedź na pytanie w ilu przypadkach obliczone sumy kodów ASCII dla dwóch sprawdzanych słów były równe , a słowa jednak po do dodatkowym sprawdzeniu anagramami się nie okazały (uzyskana wartość winna odnosić się do poszukiwań anagramów związanych z wszystkimi wcześniej realizowanymi poleceniami, poza „poszukiwanie" anagramów związanych z literą T).

Przykład

Jeżeli w pliku *teksty.txt* mamy:

4 6
alo
kalosz
aloalo
alacja
pelbng

to w pliku *anagramy.txt* powinniśmy otrzymać

1
0 1 2 3
-1
-1
-1
-1
0 1
-1

-1
-1
-1
-1
1
1

Komentarz do wyników: W tej tabeli litera T utworzona ze słowa *alo* na zasadach opisanych w zadaniu może być w tablicy 4*6 utworzonej z podanych tekstów umieszczona 4 razy. Jednak tylko układ:

a l o
o
a
l

(można go stworzyć z 1,2,3 znaku pierwszego słowa- numeracja od zera oraz 1,2,3 znaku w 2 kolumnie -numeracja od zera) spełnia opisane w zadaniu założenia w stosunku do wzorca *alo*. Ostatni wynik równy 1 wynika z kolei z jednego z podciągów w ostatnim wierszu. W słowie *pelbng* mamy podciąg *lbn* . Suma kodów ASCII dla tego podciągu to 308, a więc dokładnie tyle samo co dla wzorcowego słowa *alo*. Tymczasem słowa te nie są swoimi anagramami, a więc mamy jedyny dla tych danych przypadek, w którym pierwotna metoda Andrzeja zawiodła.

Do oceny oddajesz plik zawierający kod programu o nazwie *Zadanie3*.

10 punktów

Zadanie 4

Dowolna liczbę rzeczywistą (całkowitą też) da się przedstawić w postaci:

$c.u * 10^{potega}$

gdzie c oznacza dowolną cyfrę większą od 0, u oznacza część ułamkową liczby, która może składać się z samych zer, a potęga to dowolny wykładnik całkowity. Liczby ujemne są dodatkowo poprzedzone znakiem minus.

Przykłady:

$18.45=1,845*10^{1}$

$-0.032=-3.2*10^{-2}$

$12=1.2*10^{1}$

$3=3*10^{0}$

Kłopot jest tylko z zerem (zastanów się dlaczego) , które dlatego w tym zadaniu pominiemy. Na potrzeby tego zadania ustaloną przez nas postać nazywać będziemy znormalizowaną.

Napisz program, w którym zdefiniujesz na swoje potrzeby:

- sprowadzanie dowolnej liczby do postaci znormalizowanej,

- 4 podstawowe działania arytmetyczne (dodawanie, odejmowanie, mnożenie i dzielenie) realizowane na liczbach znormalizowanych w taki sposób, aby wynik miał również postać znormalizowaną.

Twój program powinien odczytywać plik tekstowy, w którym znajdują się trzy wiersze. Pierwszy określa ilość liczb n. W drugim jest n dowolnych liczb różnych od zera (które nie są podane w postaci znormalizowanej) oddzielonych od siebie spacją. W trzecim znajduje się n-1 operatorów symbolizujących tylko 4 działania arytmetyczne (+, - , *, /) też oddzielonych spacją. Program powinien zamienić liczby na postać znormalizowaną oraz wykorzystując operatory zbudować wyrażenie. Operatory winny być czytane od lewej do prawej i umieszczane między liczbami. Następnie program powinien obliczyć wartość skonstruowanego wyrażenia wykonując działania w kolejności wynikające z priorytetu operatorów definiowanego w matematyce. Program powinien w kolejnych wierszach pliku wynikowego zapisywać aktualny stan wyrażenia (zaczynając w pierwszym od jego postaci początkowej) po kolejnej operacji, a na końcu zapisać w postaci znormalizowanej tylko jeden wynik będący ostatecznym efektem obliczeń przeprowadzonych dla wyrażenia. W każdym wierszu liczby przedstawione powinny być w postaci znormalizowanej.

Jeśli np. w pliku mamy w wierszu z liczbami 2 13.24 15, a w wierszu z operatorami + *

to zgodnie zasadami wynikającymi z matematyki (kolejność działań) oraz uwzględniając, że liczby muszą być przedstawiane w postaci znormalizowanej w pliku wynikowym najpierw powinien pojawić się wiersz z odczytaną postacią wyrażenia czyli:

$2*10^0+1.324*10^1*1.5*10^1$

Ponieważ najpierw należy wykonać mnożenie wiec kolejny wiersz będzie wyglądał tak:

$2*10^0+ 1.986*10^2$

A w następnym powinniśmy już ujrzeć ostateczny wynik (naturalnie w postaci znormalizowanej):

$2.006*10^2$

Dane wejściowe:

Plik tekstowy ***normalizacja.txt*** zawierający w pierwszym wierszu liczbę naturalną **n (**równą przynajmniej 2, ale nie większą niż 15**),** w drugim n liczb rzeczywistych różnych od zera i nie większych (co do wartości bezwzględnej) od 10^{12} oddzielonych spacją, a w trzecim n-1 operatorów też oddzielonych spacją. Operator może reprezentować tylko jeden z symboli: + dodawanie, - odejmowanie, * mnożenie, / dzielenie. Nie ma żadnych nawiasów. Przyjmujemy, że działania są dobrze określone i zawsze wykonywalne dla podanych w pliku liczb (nie ma zatem np. niedozwolonego dzielenia przez zero, ani nie rozważamy przekroczenia zakresu liczb).

Dane wyjściowe:

Plik ***wynik.txt*** składający się z n wierszy. W pierwszym wierszu powinno się znaleźć wyrażenie zbudowane na podstawie danych z pliku ***normalizacja.txt*** przy czym operatory, które znajdą się pomiędzy liczbami winny być odczytywane od lewej do prawej, a liczby należy przedstawić w opisanej w zadaniu postaci znormalizowanej. Kolejne wiersze powinny pokazywać aktualny stan wyrażenia po realizacji jednego wynikającego z kolejności określonej w matematyce działania. Liczby niezmiennie są przedstawiane w postaci znormalizowanej, przy czym ujemne (ujemna cyfra c z ogólnej formuły przedstawionej na początku zadania) winny być przedstawione w nawiasie, a wykładnik potęgowania dla

ułatwienia też zapisujemy w nawiasie bez względu na jego znak. W ostatnim wierszu powinna się znaleźć tylko jedna liczba przedstawiona w postaci znormalizowanej będąca ostatecznym wynikiem wyrażenia.

Przykład

Jeżeli w pliku ***normalizacja.txt*** mamy następujące dane:

4

12 2 -0.24 6.5

/ + *

to w pliku ***wynik.txt*** powinniśmy otrzymać:

1.2*10 (1)/ 2*10(0)+ (-2.4)*10(-1) * 6.5 *10(0)

6*10(0)+(-2.4)*10(-1) * 6.5 *10(0)

6*10(0)+(-1.56*10(0))

4.44*10(0)

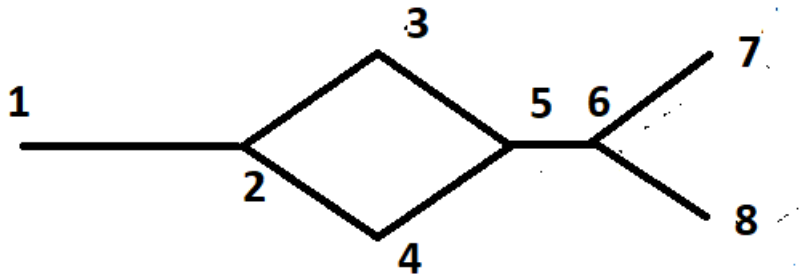
Do oceny oddajesz plik zawierający kod programu o nazwie *Zadanie4.*

10 punktów

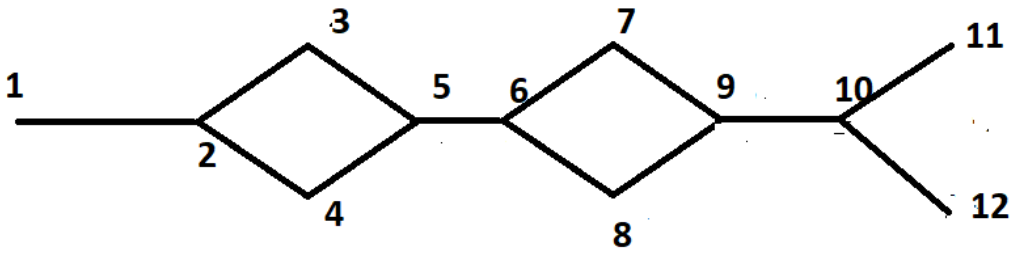
Zadanie 5

Grupa naukowców prowadziła doświadczenia polegające na obserwowaniu poruszania się niewielkich żyjątek po specjalnie przygotowanym torze.

Tor ma następujący kształt:



Przy czym to tylko jeden z wariantów toru o współczynniku skomplikowania 1. Współczynnik skomplikowania to liczba charakterystycznych wypustek w kształcie rombu.
Tor o stopniu skomplikowania 2 wyglądałby zatem następująco:



Maksymalny współczynnik skomplikowania toru może wynosić 3. Kolejnymi liczbami na obu rysunkach oznaczono tzw. punkty węzłowe toru. Numeracja jest dość charakterystyczna. Numerujemy od lewej do prawej, ale w przypadku wypustek, najpierw numer otrzymuje górny wierzchołek, a dopiero potem dolny. Na końcu toru znajdują się dwa tzw. punkty zejścia, niższy numer przypisujemy zawsze górnemu punktowi, a wyższy dolnemu.
Na podstawie obserwacji poruszania się żyjątek, ale także w oparciu o pewne własne zasady wypracowano model ruchu tychże żyjątek oparty na następujących założeniach:

a) chwila, w której zaczynamy obserwację ruchu danej grupy żyjątek ma numer zero i w chwili o tym numerze może najszybciej wyruszyć na tor pierwsze żyjątko,
b) żyjątka wysyłane są na tor w dowolnych odstępach czasu będących liczbami naturalnymi, ale w tym samym momencie nigdy nie startują dwa żyjątka jednocześnie

co oznacza, że najkrótszy możliwy odstęp czasowy pomiędzy kolejnymi żyjątkami wyruszającymi na tor wynosi 1,

c) żyjątko porusza się w ten sposób, że jeżeli nie ma żadnych przeszkód to przejście pomiędzy dwoma punktami węzłowymi zajmuje mu 1 jednostkę czasu- np. jeśli w punkcie węzłowym nr 2 żyjątko jest w chwili czasu 2 i nie ma żadnych przeszkód, aby iść dalej to w punkcie węzłowym numer 3 żyjątko będzie w chwili czasu 3,

d) przy wypustkach żyjątko wybiera najpierw drogę do punktu górnego, a jeśli to niemożliwe drogę do dolnego punktu węzłowego,

e) jeśli w danym momencie żyjątko nie może przemieścić się wzdłuż toru do kolejnego punktu węzłowego (kiedy może tak być podamy w następnym punkcie) to czeka 1 jednostkę czasu i ponownie próbuje wyruszyć w drogę, jeśli to nadal niemożliwe czeka kolejną jednostkę czasu i powtarza to tak długo, aż ruch po torze w stronę punktu zejścia będzie znowu możliwy,

f) gdyby nikt nie zakłócał ruchu żyjątek to zgodnie z dotąd przedstawionymi zasadami wędrujące żyjątka zachowywałyby między sobą odstęp przynajmniej 1 jednostki czasu i wędrowałyby bez zatrzymań. Obserwatorzy prowokują jednak (dzięki substancji zapachowej) zatrzymanie się żyjątek-tzw. postój kontrolowany. Teoretycznie mogą zatrzymać każde żyjątko dowolną liczbę razy na jego trasie w dowolnej chwili **poza chwilą 0, chwilą, w której żyjątko znajdzie się na końcu toru tuż przed punktami zejścia (punkty węzłowe numer 6 na pierwszym oraz 10 na drugim z powyższych rysunków) oraz chwilą, w której żyjątko dotrze do punktu zejścia**. Postój kontrolowany w danym punkcie węzłowym nigdy nie trwa jednak dłużej niż jedną jednostkę czasu i chwili po jego zakończeniu też nie można wymuszać kolejnego postoju kontrolowanego (co innego postój niekontrolowany wynikający z zatoru na trasie -punkt e). Po upływie czasu postoju żyjątko potrafi „otrząsnąć" się z efektu jaki wywołał zapach i próbuje ruszyć dalej. To właśnie wskutek zatrzymań na trasie mogą powstać zatory i żyjątka nie mogące się przemieszczać oczekują w punktach węzłowych zgodnie z zasadami opisanymi w poprzednim punkcie. Jest ciekawe, że żyjątko wie kiedy nie przemieści się do kolejnego punktu, bo wyczuwa, że będące w

nim inne żyjątko pozostanie tam na postój (obserwatorzy spekulują, że również do poprzedniego punktu węzłowego dociera impuls zapachowy),

g) żyjątka mogą zejść z trasy w jednym z dwóch punktów zejścia. Przy czym tak naprawdę są z toru zdejmowane w tych punktach dla ich bezpieczeństwa. W punktach zejścia oraz punktach węzłowych poprzedzających punkty zejścia jak wspomniano w poprzednim punkcie nie może być postojów kontrolowanych Po opuszczeniu toru przez żyjątko dany punkt zejścia jest wolny i gotowy do przyjęcia kolejnego żyjątka. Żyjątka przy punktach końcowych zachowują się podobnie jak przy wypustkach. Najpierw próbują zejść z toru za pośrednictwem górnego (na rysunku) punktu zejścia. Jeżeli to zejście jest zajęte (co ma miejsce wtedy kiedy żyjątko jest w punkcie węzłowym poprzedzającym punkty zejścia, a w tej samej chwili czasu w tym punkcie, w którym miałoby zejść z toru jest inne żyjątko) to próbuje zejść z toru przez drugi punkt zejścia. Gdyby wszystkie punkty były zajęte, ze względu na postój końcowy innych żyjątek to żyjątko naturalnie czeka, aż dostanie się do któregoś z punktów zejścia będzie możliwe.

Dodatkowo przyjmujemy, że chwile wyruszania żyjątek na tor, a także chwile realizacji postojów kontrolowanych są tak dobrane, że nigdy nie zdarzy się, aby żyjątko później wyruszające na trasę wyprzedzi lub stanie się przeszkodą dla żyjątka, które wyruszyło wcześniej.

Twoje zadanie polega na napisaniu programu symulującego ruch żyjątek wg opisanych wyżej zasad. Z pliku tekstowego program winien odczytać informację o stopniu skomplikowania toru, liczbie żyjątek, chwilach czasu , w których kolejne żyjątka wyruszają na tor oraz chwilach czasu, w których dla niektórych żyjątek zaplanowano postoje kontrolowane równe 1 jednostce czasu. Program powinien dla każdego z żyjątek odpowiedzieć, **w której chwili czasu opuściło ono tor oraz przez punkt zejścia o jakim numerze to nastąpiło.**

Dane wejściowe:

Plik tekstowy *zyjatka.txt* , w którego pierwszym wierszu zapisano liczbę naturalną równą przynajmniej 1, a mniejszą od 4 oznaczającą stopień skomplikowania toru. W drugim wierszu zapisano liczbę naturalną **n** oznaczającą liczbę żyjątek (dopuszczalne wartości od 2 do 30).

W kolejnych n wierszach zapisano dwie liczby naturalne oddzielone spacją. Pierwsza oznacza numer chwili, w której dane żyjątko wyruszyło na tor ,a druga liczba oznacza liczbę postoi kontrolowanych, które zorganizowano danemu żyjątku na torze za pomocą impulsu zapachowego. Para liczb w trzecim wierszu dotyczy pierwszego żyjątka, w czwartym drugiego itd., aż do wiersza n+2, dane w którym para dotyczy ostatniego n-tego żyjątka. Numery chwil, w których żyjątka wyruszają na tor tworzą ciąg rosnący, zaś druga liczba w każdym z wierszy (liczba postoi kontrolowanych) jest nie większa od 3, a może również wynosić 0. W kolejnych n wierszach (począwszy od wiersza n+3, który dotyczy pierwszego żyjątka, n+4 drugiego, a ostatni 2n+2 ostatniego) zapisano oddzielony spacją rosnący ciąg liczb naturalnych stanowiący numery chwil, w których danemu żyjątku zorganizowano postój kontrolowany. Najmniejsza liczba w tym ciągu musi być większa niż numer chwili, w której żyjątka startuje z punktu węzłowego 1, a różnica pomiędzy elementami tego ciągu musi być większa niż 1. Ilość liczb w ciągu z danego wiersza jest zgodną z określoną wcześniej liczba postojów kontrolowanych dla danego żyjątka. Jeżeli dla danego żyjątka w ogóle nie zaplanowano takich postojów to w odpowiadającym mu wierszu pojawia się tylko liczba -1. **Można założyć, że dla żadnego żyjątka żadna z przypisanych mu chwil postoju kontrolowanego nie przypadnie w punkcie węzłowym poprzedzającym dojście do punktów zejścia, ani w samych punktach zejścia, a także to, że chwile realizacji postojów kontrolowanych są tak dobrane, że nigdy nie zdarzy się, aby żyjątko później wyruszające na trasę wyprzedzi lub stanie się przeszkodą dla żyjątka, które wyruszyło wcześniej.**

Dane wyjściowe

W pliku *czasy.txt* będzie n wierszy. Pierwszy odpowiada pierwszemu żyjątku, drugi drugiemu , a ostatni ostatniemu n-temu. W każdym z tych wierszy znajdują się oddzielone spacją dwie liczby. Pierwsza oznacza numer chwili czasu, w której dane żyjątko opuściło tor, a druga numer punktu zejścia za pośrednictwem, którego to uczyniło.

Przykład

Dla danych wejściowych umieszczonych w pliku ***zyjatka.txt :***

2

6

1 2

2 0

4 1

6 2

9 0

10 0

2 4

-1

5

7 9

-1

-1

plik ***czasy.txt*** powinien mieć postać:

11 11

12 12

13 11

16 11

17 12

18 11

Do oceny oddajesz plik zawierający kod programu o nazwie *Zadanie5.*

Uwaga: W zadaniu 5 nie można implementować modelu w żadnym środowisku typu symulacyjnego. Jedynym rozwiązaniem, które będzie uwzględniane przy ocenie jest rozwiązanie w postaci programu napisanego w jednym z dopuszczonych w regulaminie języków programowania.

9 punktów

Razem w całym zestawie zadań 50 punktów