dr inż. Jolanta Kawulok

Tematy programów z PK2

Program z podstaw C++ ma być oparty o kilka powiązanych klas (min 5) i obejmujący zagadnienia:

- dziedziczenie,
- operatory,
- strumienie/pliki,
- pamięć dynamiczna (preferowane obslugiwanie przez inteligentne wskazniki),
- polimorfizm
- tylko dla chętnych bardziej zaawansowane elementy C++

W programie można wykorzystać wczytywanie i zapis binarny plików (np. stanu gry, bazy danych), jak i pliki tekstowe (np. wyniki gier, wyniki zapytanie do bazy). Oprócz wczytywania z pliku, program może wczytywać z klawiatury wybrane elementy. Przy wykorzystywaniu plików należy sprawdzać, czy istnieją, czy nie są puste. Program powinien się skompilować zarówno w trybie Debug i Release (jeśli jest wykorzytywane Visual Studio). Należy sprawdzić, czy nie ma wycieków pamięci.

Elementy sprawozdania:

- Strona tytułowa (autor, tytuł, data, prowadzący)
- Temat
- Analiza tematu (doprecyzowanie tematu, uzasadnienie wyboru klas, algorytmów, bibliotek, itp.)
- Spec. zewnętrzna (instrukcja dla użytkownika, przykłady działania, zrzuty ekranu)
- Spec. wewnętrzna (klasy, znaczenie obiektu, powiązania z innymi klasami, istotne pola i metody, diagram hierarchii klas, istotne struktury danych i algorytmy, wykorzystane techniki obiektowe, ogólny schemat działania programu)
- Testowanie i uruchamianie (opis napotkanych trudności, naprawionych błędów)

Tematy projektów mogą dotyczyć prostych baz danych (2-3 tabele), terminarza, prostych gier do zrealizowania w konsoli i bez AI (sztucznej inteligencji). Przykładowe tematy poniżej, dopuszczane są podobne inne tematy VI. Należy na oddawanie projektu przygotować przykłady działania programu (np. pliki pomocnicze z dużą ilością danych). Program powinien działać z linii komend (warto przygotować sobie od razu gotowe polecenia z użycia wybranych parametrów).

Orientacyjny plan pracy nad projektem:

- do zajęć nr 2 (ćw. 1) należy wgrać na platformę pdf z rozszerzonym opisem tematu (opisany ogólny schemat działania programu, opis zadania z pdf można modyfikować, zastanowić się co warto jeszcze dodać, usunąć, zmienić) i szkicem interfejsu programu, który albo zostanie zaakceptowany, albo będzie trzeba z prowadzącym uzgodnić konieczne poprawki. Za ten etap pracy otrzymujemy do 1 pkt
- do zajęć nr 4 należy wgrać na platformę pdf z opisem istotnych klas (ogólnym—powiązania klas, istotne najważniejsze pola i metody klas) i struktur danych programu. Opis albo zostanie zaakceptowany, albo będzie trzeba uzgodnić z prowadzącym konieczne poprawki. Za ten etap pracy otrzymujemy do 3 pkt
- gotowy program należy pokazac/obronic na ostatnich zajęciach i w ciągu tygodnia dostarczyć sprawozdanie. Proszę przy oddawaniu (pokazywaniu) programu mieć już przygotowany **schemat diagramu klas** (może być jako obrazek lub w pdfie).

Program oceniany jest do 8 pkt, a sprawozdanie do 3 pkt (program oceniany jest po otrzymaniu sprawozdania). Razem za program, jego przygotowanie (2 etapy) oraz sprawozdanie można zdobyć 15 pkt.

Spis treści

Część I
Gry

1 Lis i gęsi

Celem projektu jest napisanie programu służącego do grania w grę „Lis i gęsi" dla dwóch użytkowników.

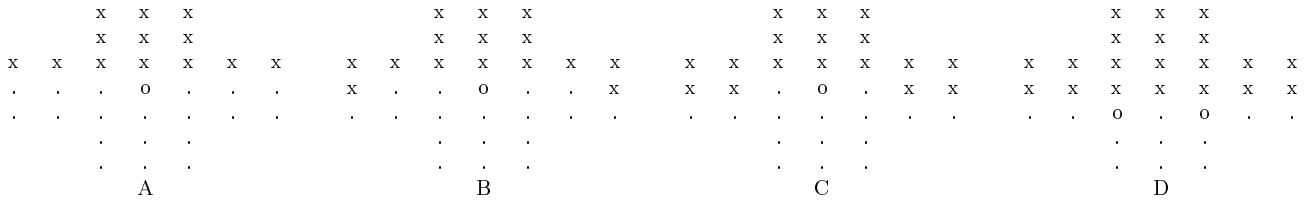
Do gry wykorzystywana jest plansza w kształcie krzyża o 33 polach, 13 pionków „gęsi" oraz jeden pionek „lisa". Pierwszy ruch należy zawsze do gęsi. Zarówno gęsi, jak i lis mogą się poruszać o jedno pole w dowolną stronę w rzędzie lub kolumnie, jeśli jest to pole wolne (w podstawowej wersji nie mogą poruszać się po przekątnych). Lis może bić gęś przez przeskoczenie przez nią na następne, wolne pole za nią w linii prostej (podobnie jak w warcabach). Możliwe jest bicie więcej niż jednej gęsi w jednym ruchu – zawsze pojedynczo, przeskakując przez nie kolejno zgodnie z zasadami bicia. Zabite gęsi są zdejmowane z planszy i już na nią nie wracają. Bicie nie jest obowiązkowe. Gęsiom nie wolno bić lisa – wygrywają, gdy uda im się unieruchomić lisa w taki sposób, by nie mógł wykonywać ruchów.

Grający lisem wygrywa, gdy uda mu się zbić co najmniej osiem gęsi — umożliwia mu to swobodne poruszanie się po planszy i przy dobrej grze uniemożliwia zablokowanie. W praktyce często gra się aż do zbicia wszystkich gęsi lub do czasu, aż gęsi same się poddadzą.

W podstawowej wersji pionki gęsi na planszy ustawia się według schematu na rys 1 A. Lis stoi zawsze na środku planszy. W programie zaimplementować również możliwość wybrania innych opcji:

- Poruszanie się dodatkowo po przekątnych liniach
- Rozpoczęcie gry z lisem ustawionym w dowolnym miejscu planszy
- Wykorzystanie 15 gęsi (rys 1 B) lub 17 gęsi (rys 1 C)
- Wykorzystanie dwóch lisów i 20 gęsi (rys 1 D)

Po uruchomieniu programu gęsi już są ustawione na swoim miejscu, w wersji podstawowej lis również znajduje się na planszy.



Rysunek 1: Gra lis i gęsi

Program powinien mieć również możliwość zapisu stanu gry w sposób binarny, a następnie po wczytaniu pliku możliwość kontynuacji gry.

Program powinien być uruchamiany z linii poleceń, gdzie są podawane parametry programu. Należy też zastosować parametry domyślne (np. nazwa pliku z zapisem stanu gry). Przykłady uruchomienia programu:

`>gra -in gra_2019_10_01.dat` — kontunacja gry na podstawie pliku „gra_2019_10_01.dat"

`>gra -num 17` — uruchomienie gry z 17 gęsiami, jako wartość domyślna brak możliwości poruszania się po przekątnych, jeśli gra zostanie przerwana zostanie zapisana do pliku o nazwie domyślnej

`>gra -diag 1 -out moj_katalog\moja_gra.dat -num 19` — uruchomienie gry z możliwością poruszania się po przekątnych, uruchomienie gry z 19 gęsiami, zapis przerwanej gry do pliku „moj_katalog moja_gra.dat"

(Opis problemu na podstawie `http://rygalo.com/2014/04/21/lis-i-gesi-w-fortecy/`

2 Gangsterskie porachunki

Celem projektu jest napisanie programu służącego do grania w grę „Gangsterskie porachunki" dla dwóch użytkowników.

Do gry wykorzystywana jest plansza w kształcie kwadratu 5x5, po 5 pionków (gangsterów) dla każdego gracza oraz jeden dodatkowy pionek „walizka". Pionki gangsterów umieszczone są wzdłuż przeciwległych boków planszy. Na środku znajduje się pionek-walizka z pieniędzmi (rysunek 2). Gracz zwycięża, jeżeli uda mu się wprowadzić walizkę na jedno z pól zajmowanych na początku przez gangsterów lub wtedy, gdy przeciwnik nie może wykonać ruchu walizką albo gangsterem.

W każdej turze gracz przesuwa najpierw walizkę, a później jednego z gangsterów. Ruch może odbyć się pionowo, poziomo lub na ukos. Pole może zajmować tylko jeden pionek. Zabronione jest przeskakiwanie przez pionki. Walizkę trzeba przesunąć zawsze o maksymalną liczbę pól w danym kierunku, to jest aż do granicy planszy lub do pola sąsiadującego z polem zablokowanym przez inny pionek.

```
x . . . o
x . . . o
x . $ . o
x . . . o
x . . . o
```

Rysunek 2: Gangsterskie porachunki

Program powinien również mieć możliwość zapisu stanu gry w sposób binarny, a następnie po wczytaniu pliku możliwość kontynuacji gry.

Program powinien być uruchamiany z linii poleceń, gdzie są podawane parametry programu. Należy też zastosować parametry domyślne (np. nazwa pliku z zapisem stanu gry). Przykłady uruchomienia programu:

`>gra -in gra_2019_10_01.dat` — kontunacja gry na podstawie pliku „gra_2019_10_01.dat"
`>gra -out moj_katalog\moja_gra.dat` — zapis przerwanej gry do pliku „moj_katalog moja_gra.dat"
`>gra` — jeśli gra zostanie przerwana zostanie zapisana do pliku o domyślnej nazwie

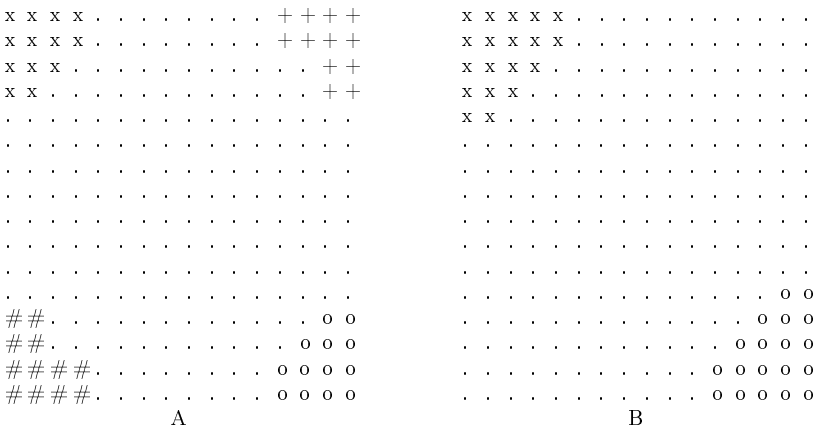
3 Halma

Celem projektu jest napisanie programu służącego do grania w grę „Halma” dla 2-4 użytkowników.

Do gry wykorzystywana jest plansza w kształcie kwadratu 16x16. Przy grze dwu lub trzy osobowej każdy z graczy posiada po 19 pionków (rys 3 A), przy grze czteroosobowej po 13 (rys 3 B). Pionki mogą się poruszać na dowolne sąsiednie pole poziomo, pionowo po skosach w tył lub w przód. Inną możliwością ruchu jest przeskoczenie nad innym pionkiem własnym lub przeciwnika. Skok można wykonać, jeśli za sąsiednim pionkiem jest wolne pole. Skok należy wykonać w linii prostej. Można skakać pionowo, poziomo, po skosach w tył i w przód. Jeśli po jednym skoku ten sam pionek ma możliwość wykonania następnego to można go wykonać w tym samym ruchu. Pionki przeskoczone nie są zbijane. Gracze wykonują po jednym ruchu albo skoku na przemian.

Celem gry jest doprowadzenie wszystkich swoich pionków do przeciwległego narożnika tzw. obozu. Wygrywa ten gracz, który pierwszy to osiągnie.

Pionek, który wszedł do docelowego obozu przeciwnika już nie może być z niego zabrany. Jeśli gracz przeciwny pozostawi w domu jakieś swoje pionki, to jego przeciwnik wygra po zajęciu wszystkich pozostałych miejsc w jego obozie (narożniku).



Rysunek 3: Halma

Program powinien również mieć możliwość zapisu stanu gry w sposób binarny, a następnie po wczytaniu pliku możliwość kontynuacji gry.

Program powinien być uruchamiany z linii poleceń, gdzie są podawane parametry programu. Należy też zastosować parametry domyślne (np. nazwa pliku z zapisem stanu gry). Przykłady uruchomienia programu:

`>gra -in gra_2019_10_01.dat` — kontunacja gry na podstawie pliku „gra_2019_10_01.dat”

`>gra -out moj_katalog\moja_gra.dat` — domyślna liczba graczy (np. 2), zapis przerwanej gry do pliku „moj_katalog moja_gra.dat”

`>gra -num 3` — liczba graczy wynosi 3, jeśli gra zostanie przerwana zostanie zapisana do pliku o domyślnej nazwie

4 Statki

Celem projektu jest napisanie programu służącego do grania w grę „Statki” dla 2 użytkowników.

Gra toczy się na planszy kwadratowej o rozmiarach podanych przez użytkownika. Również jako parametry gry podawane są liczby statków o poszczególnych rozmiarach (dane te są wczytywane z osobnego pliku).

Statki ustawiane są w pozycji pionowej lub poziomej, tak aby nie stykały się one ze sobą w żaden sposób (ani bokami, ani rogami).

Program powinien również mieć możliwość zapisu stanu gry w sposób binarny, a następnie po wczytaniu pliku możliwość kontynuacji gry.

Program powinien być uruchamiany z linii poleceń, gdzie są podawane parametry programu. Należy też zastosować parametry domyślne (np. nazwa pliku z zapisem stanu gry). Przykłady uruchomienia programu:

`>gra -in gra_2019_10_01.dat` — kontunacja gry na podstawie pliku „gra_2019_10_01.dat”

`>gra -innum liczba_statkow.txt -out moj_katalog\moja_gra.dat -n 12` — informacje o wielkości i liczbie statków znajdują się w pliku „liczba_statkow.txt”, przerwana gry jest zapisywana do pliku „moj_katalog moja_gra.dat”, długość boku planszy wynosi 12

Przykładowy plik „liczba_statkow.txt”:

```
4 1
3 2
3 3
5 1
```

gdzie pierwsza kolumna oznacza wielkość statku, a druga jego liczbę.

5 Wąż

Celem projektu jest napisanie programu służącego do grania w popularną grę „Wąż". W grze należy kierować wężem, tak aby połykał pojawiające się jedzenie na planszy. Każdy posiłek wydłuża węża. Gra jest skończona, gdy wąż dotknie ściany lub swojego ogona.

W parametrach programu należy dobierać szybkość poruszania się węża, wielkość okna, jak również początkową wielkość węża, nazwę użytkownika (można również wprowadzić własne dodatkowe parametry).

Program powinien również mieć możliwość zapisu stanu gry w sposób binarny, a następnie po wczytaniu pliku możliwość kontynuacji gry. Na koniec programu jest zapisywany (lub dopisywany do istniejącego pliku) wynik wraz z nazwą użytkownika

Program powinien być uruchamiany z linii poleceń, gdzie są podawane parametry programu. Należy też zastosować parametry domyślne (np. nazwa pliku z zapisem stanu gry). Przykłady uruchomienia programu:

`>gra -in gra_2019_10_01.dat` — kontunacja gry na podstawie pliku „gra_2019_10_01.dat"

`>gra -out moj_katalog\moja_gra.dat -n 30` — zapis przerwanej gry do pliku „moj_katalog moja_gra.dat", długość boku planszy wynosi 30

Część II
Grafy

6 Mapa miasta

Celem projekt jest wyznaczanie najkrótszej trasy pomiędzy dwoma punktami w mieście. Ulice w mieście leżą tylko na linii północ-południe oraz wschód-zachód, jednak w mieście istnieją również ścieżki po których poruszają się piesi.

W celu wyznaczenia trasy należy wykorzystać algorytm Dijkstry wyszukujący najkrótszej ścieżki w grafie. Na wejście wymagane są trzy pliki. Pierwszy zawiera id, nazwę punktu i jego współrzędne, drugi plik zawiera binarną tablicę kwadratową z informacją, czy pomiędzy dwoma punktami znajduje się połączenie (tablica nie jest symetryczna), a trzeci plik zawiera listę tras do wyznaczenia. Każda linijka zawiera jedną trasę w postaci <punkt początkowy> <punkt docelowy>. Przykładowe pliki:

```
1 Dom 0 10
2 Apteka 10 12
3 Policja 6 7
4 Sklep 11 5
5 Szkoła 1 1
6 Szpital 14 0
```

```
0 1 0 0 0 0
1 0 1 1 0 0
1 1 0 1 1 0
0 1 1 0 1 1
0 0 1 1 0 1
0 0 0 1 1 0
```

```
Dom Szpital
Szpital Dom
```

Odległości pomiędzy miastami jest wyznaczana za pomocą metryki euklidesowej dla pieszych, a za pomocą metryki Manhattan dla samochodów.

Wynikiem działania programu jest plik wyjściowy zawierający przebieg wyznaczonej trasy, z podaniem całkowitej odległości oraz poszczególnych punktów trasy, np. trasa dla pieszych:

```
trasa: Dom → Szpital: 23.1001
Dom → Apteka → Sklep → Szpital

trasa: Szpital → Dom: 17.9243
Szpital → Sklep → Policja →Dom
```

Należy wziąć pod uwagę, że może nie istnieć możliwość wyznaczenia zadanej trasy.

Program powinien być uruchamiany z linii poleceń, gdzie są podawane parametry programu (powinna być możliwość podawania dowolnej kolejności parametrów):

`-coor` plik wejściowy z współrzędnymi
`-tab` plik wejściowy z tablicą połączeń
`-q` plik wejściowy z trasami do wyznaczenia
`-out` plik wynikowy z wyznaczonymi trasami

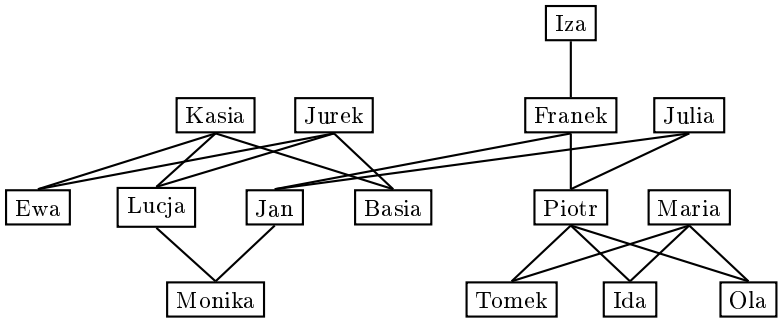
7 Drzewo genealogiczne

Celem projekt jest stworzenie drzewa genealogicznego pewnej rodziny, a następnie wyszukiwanie zadanych relacji.

Na wejście wymagane są dwa pliki. Pierwszy plik w każdym wierszu zawiera informację: rodzic dziecko. Drugi plik zawiera zapytanie postaci: <imie> <relacja>, gdzie relacja jest jedną z definiowanych określeń, m.in.: matka, ojciec, rodzic, siostra, brat, rodzeństwo, dziadek, babcia, dziadkowie, kuzyn, kuzynka, kuzynostwo, syn, córka, dziecko, wnuczek, wnuczka, wnuki. Płeć osoby jest określana na podstawie imienia (żeńskie kończą się na literę 'a').

Poniżej przykładowy plik wejściowy, z pokazaniem drzewa genealogicznego (grafu), który opisuje.

Lucja Monika
Jan Monika
Kasia Ewa
Kasia Lucja
Kasia Basia
Jurek Ewa
Jurek Lucja
Jurek Basia
Franek Jan
Franek Piotr
Julia Jan
Julia Piotr
Piotr Tomek
Piotr Ida
Piotr Ola
Maria Tomek
Maria Ida
Maria Ola
Iza Franek



Należy zwrócić uwagę, aby nazwy osób się nie powtarzały.

Przykładowe zapytania:

Monika kuzynostwo
Jan dziadek
Julia wnuczka

Plikiem wyjściowym są znalezione wyniki:

Monika kuzynostwo: Tomek, Ida, Ola
Jan dziadek: brak informacji
Julia wnuczka: Monika, Ida, Ola

Program powinien być uruchamiany z linii poleceń, gdzie są podawane parametry programu (powinna być możliwość podawania dowolnej kolejności parametrów):

`-rel` plik z relacjami
`-q` plik wejściowy z relacjami do wyznaczenia
`-out` plik wynikowy z wyznaczonymi relacjami

8 Sieć komputerowa

Celem projekt jest stworzenie sieci komputerowej między mieszkańcami bloku. W tym celu należy położyć kable tak, aby wszyscy mieszkańcy byli ze sobą połączeni. Wystarczy, aby mieszkańcy byli połączeni ze sobą za pośrednictwem innej osoby, bez konieczności, aby jeden użytkownik był połączony kablami z wszystkimi pozostałymi osobami. W celu stworzenia sieci należy wykorzystać **algorytm Kruskala** tworzący drzewo rozpinające.

Na wejście wymagane są 2 pliki. Pierwszy plik zawiera współrzędne wszystkich mieszkańców. Przykładowy plik:

```
1 Kowalski 0 10
2 Nowak 10 12
3 Malicki 6 7
4 Duda 11 5
5 Blat 1 1
6 Dudek 14 0
```

Drugi plik zawiera informacje o połączeniach pomiędzy mieszkańcami, które nie mogą zostać zrealizowane. Przykładowy plik:

```
1 2
2 3
2 5
2 6
3 4
4 6
```

Plikiem wyjściowym jest lista odcinków (par dwóch punktów) z informacją o kolejności wyznaczania odcinków na których zostaną położone kable wraz z sumaryczną długością wszystkich kabli. Program powinien być uruchamiany z linii poleceń, gdzie są podawane parametry programu (powinna być możliwość podawania dowolnej kolejności parametrów):

`-coor` plik wejściowy z współrzędnymi mieszkańców
`-tab` plik wejściowy z informacją o braku możliwości położenia kabli
`-out` plik wynikowy z informacją o położeniu kabli

9 Sieć ścieżek

Celem projekt jest stworzenie sieci dróg pomiędzy domkami znajdującymi się w górach. Koszt budowy drogi w górach jest bardzo drogi, stąd założono, że należy zbudować jak najmniejszą liczbę dróg, tak aby zminimalizować koszty, jednak, aby była możliwość komunikacji pomiędzy wszystkimi domkami. W celu zaprojektowania położenia dróg należy wykorzystać **algorytm Prima** tworzący drzewo rozpinające. Zaczynając budowę drogi od domku wybranego przez użytkownika.

Na wejście wymagane są 2 pliki. Pierwszy plik zawiera nazwiska rodzin zamieszkujących w domkach. Przykładowy plik:

1 Kowalscy
2 Malinowscy
3 Maliccy
4 Nowakowscy
5 Wojciechowscy
6 Lewandowcy

Drugi plik zawiera macierz trójkątną z informacją o kosztach wybudowania drogi pomiędzy poszczególnymi domkami. Przykładowy plik:

```
    1   2   3   4   5   6
1   --  --  --  --  --  --
2   100 --  --  --  --  --
3   150 200 --  --  --  --
4   40  130 44  --  --  --
5   90  60  295 333 --  --
6   70  78  234 456 700 --
```

Plikiem wyjściowym jest lista odcinków (par dwóch punktów) z informacją w jaki sposób była budowana droga oraz z sumarycznym kosztem wybudowania dróg.

Program powinien być uruchamiany z linii poleceń, gdzie są podawane parametry programu (powinna być możliwość podawania dowolnej kolejności parametrów):

`-name` plik wejściowy z mieszkańcami
`-tab` plik wejściowy z tablicą połączeń
`-out` plik wynikowy z informacją o budowie dróg

Część III
Bazy danych/terminarze

10 Dziennik lekcyjny

Celem projektu jest stworzenie dziennika lekcyjnego, składającego się z dwóch tabel. Pierwsza zawiera informacje o uczniach (PESEL, imię, nazwisko, data urodzenia, adres zamieszkania, itp), gdzie ich id jest numer PESEL. Druga tablica zawiera informacje o ocenach (id ucznia, nazwa przedmiot, ocena, opis zadania, itp).

Dane do programu są dostarczane w postaci dwóch plików tekstowych. Dla pierwszej tabeli np.

99101215381, Judyta, Kuc, 12.10.1999, Akademicka, 30/2, Gliwice
99120217386, Maciej, Adamczyk, 02.12.1999, Bony, 4/3, Gliwice
99081627482, Piotr, Kit, 16.08.1999, Niepodległości, 6, Knurów
99020217181, Iza, Malce, 02.02.1999, Bony, 6/6, Gliwice

Dla drugiej tabeli np.

99101215381, matematyka, 4, odpowiedź ustna
99101215381, matematyka, 3.5, sprawdzian
99120217386, matematyka, 4.5, sprawdzian
99081627482, fizyka, 3, kartkówka

Program powinien umożliwić generowanie zadanych zdefiniowanych (przez programistę) informacji, np. średnia ocen z danego przedmiotu, lista uczniów urodzonych w danym przedziale czasowym, lista uczniów mieszkających na takiej samej ulicy. Lista uczniów, których średnia z danego przedmiotu jest wyższa niż zadana ocena, itp.

Program powinien być uruchamiany z linii poleceń, gdzie są podawane parametry programu (powinna być możliwość podawania dowolnej kolejności parametrów):

`-inper` plik wejściowy z opisem osób
`-insub` plik wejściowy z opisem przedmiotów
`-out` plik wyjściowy z informacjami

11 Biblioteka książek

Celem projektu jest stworzenie biblioteki książek, składającego się z trzech tabel. Pierwsza zawiera informacje o osobach (PESEL, imię, nazwisko, data urodzenia, adres zamieszkania, itp), gdzie ich id jest numer PESEL. Druga tablica zawiera informacje o książkach (id książki, tytuł, autor, data wydania, liczba stron, itp). Trzecia tabela zawiera informacje o wypożyczeniach (id osoby, id książki, data wypożyczenia, data zwrotu).

Dane do programu są dostarczane w postaci trzech plików tekstowych (analogicznie tworzonych jak w zadaniu 10.

Program powinien umożliwić generowanie zadanych zdefiniowanych (przez programistę) informacji, np. liczba wszystkich wypożyczonych książek przez daną osobę, wypożyczonych w zadanym przedziale czasowym, lista osób, która wypożyczyła kiedykolwiek zadaną książkę, itp.

Program powinien być uruchamiany z linii poleceń, gdzie są podawane parametry programu (powinna być możliwość podawania dowolnej kolejności parametrów):

`-inper` plik wejściowy z opisem osób
`-inbook` plik wejściowy z opisem książek
`-inbor` plik wejściowy z opisem wypożyczeń
`-out` plik wyjściowy z informacjami

12 Rejestr samochodów

Celem projektu jest stworzenie rejestru samochodów, składającego się z trzech tabel. Pierwsza zawiera informacje o właścielach aut (PESEL, imię, nazwisko, data urodzenia, adres zamieszkania, itp), gdzie ich id jest numer PESEL. Druga tablica zawiera informacje o samochodach (id samochodu, marka, kolor, rocznik, itp). Trzecia tabela zawiera informacje o relacjach dwóch tabel – jedna osoba może mieć kilka samochodów oraz jeden samochód może mieć kilka właścicieli.

Dane do programu są dostarczane w postaci dwóch plików tekstowych (analogicznie tworzonych jak w zadaniu 10. Trzeci plik zawiera dwie kolumny: id właściciela i id samochodu, np.:

```
79101215381 1
69101215381 1
70120217386 2
86081627482 3
86081627482 4
```

Program powinien umożliwić generowanie zadanych zdefiniowanych (przez programistę) informacji, np. lista osób posiadająca samochód marki Volvo, lista osób posiadająca jakikolwiek samochód mająca mniej niż 28 lat, itp.

Program powinien być uruchamiany z linii poleceń, gdzie są podawane parametry programu (powinna być możliwość podawania dowolnej kolejności parametrów):

`-inper` plik wejściowy z opisem osób
`-incar` plik wejściowy z opisem samochodów
`-inrel` plik wejściowy z opisem relacji
`-out` plik wyjściowy z informacjami

13 Przychodnia

Celem projektu jest stworzenie systemu dla przychodni do rejestracji pacjentów. Można założyć, że na każdego pacjenta jest 15 min i w takich odstępach czasowych można się rejestrować. W programie należy wykorzystać dwie tabele. Pierwsza zawiera informacje o lekarzach (PESEL, imię, nazwisko, specjalizacja, dni i godziny kiedy przyjmuje), gdzie ich id jest numer PESEL. Druga tablica zawiera informacje o zarejestrowanych pacjentach (id pacjenta, id lekarza, data, godzina).

Dane do programu są dostarczane w postaci jednego pliku tekstowego dotyczącego lekarzy (analogicznie tworzonych jak w zadaniu 10). Po uruchomieniu programu pacjent może wybrać specjalizację lekarza lub odrazu konkretnego lekarza, do którego chce się udać, zakres dat, który mu odpowiada na wizytę. Następnie z wygenerowanej listy wybiera odpowiedni dla niego termin. Również powinna być możliwość rezygnacji z wizyty.

Dodatkowo program powinien umożliwiać zapisywanie powstałej bazy do pliku binarnego, jak również odczyt i dopisanie dodatkowych informacji.

Program powinien być uruchamiany z linii poleceń, gdzie są podawane parametry programu. Należy też zastosować parametry domyślne (np. nazwa pliku binarnego wyjściowego zawierająca bazę).

`-in` plik wejściowy z bazą danych (binarny)
`-out` plik wyjściowy z bazą danych (binarny)
`-indoc` plik wejściowy z informacją o lekarzach

14 Apteka

Celem projektu jest stworzenie systemu dopasowywania leków do pacjentów. W programie należy wykorzystać dwie tabele. Pierwsza zawiera informacje o lekach (id, nazwa, koszt, przedział wiekowy pacjenta). Druga tabela zawiera listę objawów choroby wraz z informacją o zalecanych lekach (id leku).

Oba rodzaje danych wczytywane są do programu z dwóch osobnych plików. Dodatkowo program powinien umożliwiać zapisywanie powstałej bazy do pliku binarnego, jak również odczyt i dopisanie dodatkowych informacji.

W programie użytkownik podaje informacje o swoim wieku oraz o objawach choroby (np. kaszel suchy, gorączka, zatkany nos) w odpowiedzi otrzymuje propozycje jednego lub kilku zestawów leków, wraz z sumą kosztów. Wyniki są zapisywane do pliku wyjściowego.

Program powinien być uruchamiany z linii poleceń, gdzie są podawane parametry programu. Należy też zastosować parametry domyślne (np. nazwa pliku binarnego wyjściowego zawierająca bazę).

`-in` plik wejściowy z bazą danych (binarny)
`-out` plik wyjściowy z bazą danych (binarny)
`-inmed` plik wejściowy z informacją o lekach
`-insym` plik wejściowy z powiązaniami leków i objawów
`-outset` plik wyjściowy z listą proponowanych leków

15 Detektyw

Celem projektu jest stworzenie systemu odnajdywania przestępcy na podstawie posiadanej aktualnej bazy danych o ich charakterystycznych cechach oraz na podstawie rysopisu dokonanego przez ofiarę. W programie należy wykorzystać dwie tabele. Pierwsza tabela zawiera charakterystykę przestępców (gdzie ich id jest numer PESEL), np. płeć, rok urodzenia, wzrost, waga, kolor oczu, kolor włosów, długość włosów, narodowość, posiadanie okularów, miejsce posiadanie widocznego znamienia (może być ich kilka). Dane do programu dostarczane są w pliku tekstowym. Opis poszukiwanego przestępcy również jest dostarczany w postaci pliku, rysopis nie musi zawierać wszystkich możliwych cech.

Ponieważ poszczególne cechy nie są równoważne, należy w programie zamieścić dodatkowe informacje o ich wagach. Przykładowo zgodność kolorów włosów jest ważniejsza, niż zgodność w noszeniu okularów. Również należy uwzględnić, że łatwiej przed dokonaniem przestępstwa ściąć włosy niż je wydłużyć lub łatwiej pomylić Słowaka z Polakiem, niż z Algierczykiem.

Druga tabela zawiera informacje o popełnionych już przestępstwach przez daną osobę (wykorzystujemy nr pesel). Np. data i opis przestępstwa.

Program może działać w dwóch trybach:

1. Na wyjście zwracać plik zawierający listę (np. 5) najlepiej dopasowanych przestępców do rysopisu wraz z informacją o zgodności cech i różnicach (wykorzystujemy plik z rysopisem).

2. Na wyjście zwracać plik zawierający informacje o przestępcy — rysopis, listę popełnionych przestępstw.

Program powinien być uruchamiany z linii poleceń, gdzie są podawane parametry programu.

`-inPer` plik wejściowy z opisem przestępców
`-inCri` plik wejściowy z opisem przestępstw
dla opcji 1: `-q` plik wejściowy z rysopisem przestępcy wykonany przez ofiarę
dla opcji 2: `-id` numer Pesel
`-out` plik wyjściowy w zależności od wybranej opcji

Część IV
Dopasowanie ciągów symboli

16 Tablica sufiksów

Celem projektu jest stworzenie programu służącego do poszukiwania wzorów w tekście przy wykorzystaniu tablicy sufiksów.

Tablica sufiksowa (SA, ang. *suffix array*) jest to tablica zawierająca indeksy leksykograficznie posortowanych wszystkich sufiksów ciągu S długości n. W budowie SA wykorzystywany jest dodatkowy symbol spoza alfabetu, który zaznacza koniec ciągu, stąd długość tablicy wynosi $n + 1$.

Na rysunku 4 przedstawiono tablicę sufiksową (SA) wraz z pokazanymi posortowanymi sufiksami dla przykładowego ciągu $S =$'she sells sea shells' z dodanym na końcu symbolem '$'.

i	$SA[i]$	Posortowane sufiksy
0	9	sea shells$
1	3	sells sea shells$
2	13	shells$
3	19	$
4	12	a shells$
5	2	e sells sea shells$
6	11	ea shells$
7	5	ells sea shells$
8	15	ells$
9	1	he sells sea shells$
10	14	hells$
11	6	lls sea shells$
12	16	lls$
13	7	ls sea shells$
14	17	ls$
15	8	s sea shells$
16	18	s$
17	10	sea shells$
18	4	sells sea shells$
19	0	she sells sea shells$

Rysunek 4: Tablice posortowanych wszystkich sufiksów (SA) dla $S'=$'she sells sea shells$'

Wyszukiwanie dokładnych wzorców P w S odbywa się poprzez znalezienie wszystkich sufiksów, których prefiks wynosi P. Ponieważ SA jest posortowaną leksykograficznie tablicą sufiksów, poszukiwanie wzorca P można przeprowadzić za pomocą binarnego poszukiwania. W tym celu należy znaleźć najmniejszy indeks i, taki że prefiks ciągu suf_i (i-ty najmniejszy leksykograficznie sufiks) wynosi P, nie spełnienie tego warunku oznacza, że P nie występuje w S. W przypadku znalezienia i spełniającego powyższy warunek, należy znaleźć największy indeks j ($\geq i$), taki że prefiks suf_i wynosi P. Wszystkie elementy w zakresie $SA[i..j]$ zawierają pozycje P w S.

W celu przyspieszenia działania programu, chętni w implementacji algorytmu mogą dodatkowo zbudować i wykorzystać tablicę określaną jako Lcp, która zawiera długości najdłuższych wspólnych prefiksów pomiędzy kolejnymi parami sufiksów z SA.

Program na wejście otrzymuje dwa pliki, jeden zawiera tekst w którym będzie poszukiwany wzorze. Przy wczytywaniu tego pliku pomijane są białe znaki oprócz znaku spacji. Dzięki tej operacji można wykorzystywać teksty zawarte w kilku liniach. Przykładowy plik:

```
Litwo, Ojczyzno moja! ty jesteś jak zdrowie;
Ile cię trzeba cenić, ten tylko się dowie,
Kto cię stracił. Dziś piękność twą w całej ozdobie
Widzę i opisuję, bo tęsknię po tobie.
```

Drugi plik zawiera listę poszukiwanych wzorów. Wracając do przykładu z rysunku 4, lista ta może wyglądać następująco:

```
se
set
he
he
```

Należy zwrócić uwagę, że we wzorcach też są uwzględniane spacje.

Plikiem wyjściowym jest lista wszystkich wzorców wraz z podaniem jego wystąpienia w przeszukiwanym tekście. Dla powyższego przykładu będzie ona wyglądać następująco:

```
se: 10, 4
set: brak
he : brak
he: 1, 14
```

Program powinien być uruchamiany z linii poleceń, gdzie są podawane parametry programu.

`-t` plik wejściowy z przeszukiwanym tekstem
`-p` plik wejściowy z poszukiwanymi wzorcami
`-o` plik wyjściowy

17 Podobieństwo ciągów

Celem projektu jest stworzenie programu służącego do porównywania dwóch ciągów symboli przy wykorzystaniu programowania dynamicznego.

W metodzie tej wykorzystuje się dwie macierze. Pierwsza jest to dwuwymiarowa macierz wyników M^s (ang. *scoring matrix*). Poszczególne komórki tej macierzy są uzupełniane krokowo na podstawie porównywania badanych miejsc w obu ciągach, z uwzględnieniem poprzednio wyliczonych wartości, tj:

$$M^s(i,j) = \max \begin{cases} M^s(i-1,j-1) + \delta(S_1[i], S_2[j]) & \textit{dopasowanie/niedopasowanie} \\ M^s(i,j-1) + \delta(-, S_2[j]) & \textit{delecja} \\ M^s(i-1,j) + \delta(S_1[i], -) & \textit{insercja} \end{cases} \quad (1)$$

Funkcja δ zwraca koszt zmiany edycyjnej, jak i punkty za dopasowanie.

Druga macierz (optymalnych decyzji) w poszczególnych komórkach zawiera informację, w jaki sposób dana komórka w M^s została uzupełniona — z którego wiersza równania (1) została pobrana największa wartość. Optymalna ścieżka dopasowania jest wyznaczana poprzez działanie w odwrotnym kierunku niż budowanie macierzy — wykorzystując drugą macierz przejścia.

Do porównywania dwóch ciągów symboli można wykorzystać algorytm Needlemana-Wunscha, gdzie dopasowanie ciągów odbywa się od początku do końca obu ciągów, czyli od lewego górnego rogu do prawego dolnego w macierzach dopasowania. Pierwsza kolumna i pierwszy wiersz macierzy M^s jest wypełniony kosztami wprowadzenia przerw. Optymalny wynik dopasowania znajduje się w prawym dolnym rogu ($M^s(|S_1|, |S_2|)$), zatem ścieżka dopasowania prowadzi od tego pola do pola $M^s(0,0)$, przesuwając się wzdłuż pól, z których pochodził wynik (na podstawie drugiej macierzy).

Poniżej przedstawiono przykładową macierz wyników wraz z pokazaniem, z których komórek były pobrane wartości do wyliczenia kolejnych. Czerwone strzałki ilustrują najlepsze dopasowanie. Za dopasowane symbole przyznawane był 1 punkt, za zmianę symbolu -1, a za wprowadzenie przerwy -2.

		G	C	C	C	T	A	G	C	G
	0	-2	-4	-6	-8	-10	-12	-14	-16	-18
G	-2	1	-1	-3	-5	-7	-9	-11	-13	-15
C	-4	-1	2	0	-2	-4	-6	-8	-10	-12
G	-6	-3	0	1	-1	-3	-5	-5	-7	-9
C	-8	-5	-2	1	2	0	-2	-4	-4	-6
A	-10	-7	-4	-1	0	1	1	-1	-3	-5
A	-12	-9	-6	-3	-2	-1	2	0	-2	-4
T	-14	-11	-8	-5	-4	-1	0	1	-1	-3
G	-16	-13	-10	-7	-6	-3	-2	1	0	0

Rysunek 5: Macierz wykorzystywana przy algorytmie Needlemana-Wunscha
(źródło: http://www.ibm.com/developerworks/library/j-seqalign/)

Plikiem wyjściowym jest wydruk znalezionego dopasowania w następujący sposób (dla ciągów kurczaczek i kubraczek) :

```
k u - r c z a c z e k
| |   |     | | | | |
k u b r - - a c z e k
```

Program powinien być uruchamiany z linii poleceń, gdzie są podawane parametry programu.

`-M` punty za dopasowanie
`-MM` punty za niedopasowanie
`-G` punty za przesunięcie
`-t1` plik wejściowy z przeszukiwanym tekstem numer 1
`-t2` plik wejściowy z przeszukiwanym tekstem numer 2
`-o` plik wyjściowy

Część V
Automaty komórkowe

18 Symulator zarażania

Celem projektu jest zasymulowanie zarażanie się katarem przy wykorzystaniu automatów komórkowych, na prostokątnej płaszczyźnie o wymiarach MxN. Na początku katar posiada tylko jedna osoba, po upływie kolejnej jednostki czasowej chora osoba ma p szans na zarażenie każdej z osobna osoby z sąsiedztwa von Neumanna (4 komórek przylegających do niej bokami) oraz $0.7p$ szans na zarażenie pozostałych czterech osób (rogi sąsiedztwo Moore'a — 4 komórek przylegających do niej bokami), gdzie p jest ustalonym parametrem. Po siedmiu jednostkach czasowych, osoba z katarem jest odporna na zarażenia przez 4 jednostki czasowe (sama też nie zaraża), a następnie całkowicie staje się zdrowa. Po każdej jednostce czasowej powinna być zapisywana informacja z liczbą osób posiadającą katar.

Program powinien być uruchamiany z linii poleceń, gdzie są podawane parametry programu.

`-N` szerokość płaszczyzny na której znajdują się osoby
`-M` wysokość płaszczyzny na której znajdują się osoby
`-i` liczba iteracji
`-p` prawdopodobieństwo zarażenia
`-o` plik wyjściowy z listą liczb osób posiadającą katar, po danej iteracji

19 Misjonarze i kanibale

Celem projektu jest zasymulowanie współistnienia dwóch populacji – misjonarzy i kanibali – na planszy kwadratowej o boku N. Na jednej komórce planszy może znajdować się naraz kilka osobników. Osobniki poruszają się w każdej jednostce czasowej o jedno pole w dowolnym kierunku.

Tylko kanibale się rozmnażają, aby mogli to uczynić musi płodna kobieta i mężczyzna znaleźć się na jednym polu. Oboje osiągają dojrzałość do rozmnażania w wieku ustalonym przez użytkownika, dodatkowo kobieta po urodzeniu przestaje być płodna na zadaną liczbę iteracji. Prawdopodobieństwo urodzenia chłopca lub dziewczynki jest takie samie.

Misjonarz zostanie zjedzony przez kanibali jeśli w jego sąsiedztwie jest nie mniej kanibali niż misjonarzy. Kanibal zostanie nawrócony jeśli w jego sąsiedztwie jest nie mniej misjonarzy niż kanibali. Kanibale podczas rozmnażania nie są brane pod uwagę do otoczenia. Długość życia osobników z danej populacji jest także parametrem programu. Jeśli na jednym polu znajdzie się więcej niż określona liczba osobników to część z nich umiera z przeludnienia.

Pierwotna liczba kobiet i mężczyzn kanibali, jaki i misjonarzy są parametrami programu. Istnieje kilka sposobów rozmieszczenia populacji przy uruchomieniu programu: losowe wymieszanie, na przeciwległych bokach, na przeciwległych kątach, w 10 jednorodnych skupiskach, w 10 wymieszanych skupiskach, itp.

Gra kończy się z zadaną liczbą iteracji lub jak wszystkie osobniki wymrą. Na wyjście zwracana jest lista liczb poszczególnych typów osobników po każdej jednostce czasowej.

Program powinien być uruchamiany z linii poleceń, gdzie są podawane parametry programu. Należy też zastosować parametry domyślne (np. początkowa liczba poszczególnych osobników).

Część VI

Lista przykładowych innych tematów

20. Baza danych/terminarz imprez masowych (koncertów, meczów) z możliwością rezerwacji
21. Sklep z ubraniami
22. Jubiler
23. Terminarz mechanika samochodowego
24. Biuro podróży
25. Książka kucharska
26. Wypożyczalnia pojazdów drogowych i sprzętu wodnego
27. Inne automaty komórkowe