Podstawy Programowania Komputerów

- Wykorzystywane struktury danych oraz zaproponowany algorytm powinien zostać zaakceptowany przez prowadzącego; w projekcie należy wykorzystać dynamiczne struktury danych (tj. listy, drzewa, stosy, kolejki...),
- Projekt powinien mieć strukturę modułową (pliki nagłówkowe (*.h/*.hpp), zawierające deklaracje, oraz implementacyjne (*.cpp), zawierające definicje),
- Komentarze powinny być zgodne ze standardem Doxygen (pełny opis techniczny),
- Dozwolone jest użycie kontenerów biblioteki stl (np. std::vector, std::list, itd.),
- **Terminy realizacji poszczególnych etapów projektu podane są na PZE**.

---

1. Gra kółko i krzyżyk z planszą 3+

Opracować program gry w kółko i krzyżyk dla dwóch graczy. Po uruchomieniu wprowadzane są parametry gry: rozmiar planszy (3-10) oraz liczba wygrywających znaków (przyjąć wartości 3 lub 5). Plansza jest wyświetlana w oknie konsoli, przykład dla rozmiaru 5 jest pokazany poniżej.

```
    A   B   C   D   E
  +---+---+---+---+---+
0 | O |   |   |   | X |
  +---+---+---+---+---+
1 | X |   |   |   |   |
  +---+---+---+---+---+
2 | O | X |   |   | O |
  +---+---+---+---+---+
3 |   |   | O |   |   |
  +---+---+---+---+---+
4 |   |   |   |   |   |
  +---+---+---+---+---+
```

Gracze na przemian stawiają na planszy własne symbole, a rozgrywka kończy się, kiedy dany gracz ustawi określoną liczbę znaków poziomo, pionowo lub na skos. Historia rozgrywki zapisywana jest w pliku tekstowym (możliwość przerwania rozgrywki, a następnie jej wznowienia).

Po zakończeniu rozgrywki cała historia zapisywana jest w pliku HTML (wraz z datą rozgrywki, nazwami graczy itp.). Sposób tworzenia prostej strony pokazany został w dodatku A.

Uwaga: można rozważyć grę z komputerem i zaproponować własny algorytm stawiania symbolu na planszy.

- Wykorzystywane struktury danych oraz zaproponowany algorytm powinien zostać zaakceptowany przez prowadzącego; w projekcie należy wykorzystać dynamiczne struktury danych (tj. listy, drzewa, stosy, kolejki...),
- Projekt powinien mieć strukturę modułową (pliki nagłówkowe (*.h/*.hpp), zawierające deklaracje, oraz implementacyjne (*.cpp), zawierające definicje),
- Komentarze powinny być zgodne ze standardem Doxygen (pełny opis techniczny),
- Dozwolone jest użycie kontenerów biblioteki stl (np. std::vector, std::list, itd.),
- **Terminy realizacji poszczególnych etapów projektu podane są na PZE**.

2. Symulator automatu do wydawania napojów

Opracować program do symulacji automatu do wydawania napojów. Stan automatu przechowywany jest w pliku tekstowym. Zapisane są w nim podstawowe informacje takie jak: liczba dostępnych napojów i ich rodzaje wraz z ceną (przyjąć, że liczba różnych rodzajów napojów może być do 20, a zasobniki na poszczególne napoje mogą zawierać maksymalnie 15 opakowań), zawartość kasetki z pieniędzmi oraz aktualny status automatu (brak pieniędzy w kasetce poniżej zdefiniowanej wartości, konieczność uzupełnienia zasobników z napojami itp.).

Program powinien zawierać proste menu: wyświetlane dostępne napoje (z liczbą dostępnych sztuk i cenami) oraz typowe opcje ANULUJ - rezygnacja z zakupu/wybranej opcji, WYBÓR NAPOJU (numer zasobnika), ZAPŁATA - podanie nominałów i liczby wrzuconych monet, OK - zatwierdzenie opcji, WYŁĄCZENIE - wyłączenie automatu/wyjście z symulatora i inne (w razie potrzeby). Automat powinien mieć możliwość zwrotu reszty w największych możliwych nominałach (o ile są dostępne), w przypadku braku możliwości wydania reszty pojawia się odpowiedni komunikat i operacja wydania napoju jest anulowana.

Uzupełnienie towaru i kasetki z pieniędzmi odbywa się poprzez edycję pliku zawierającego stan automatu.

Dodatkowo wszystkie sytuacje nietypowe (konieczność uzupełnienia zasobników lub kasetki z pieniędzmi, uzupełnienie towaru/kasetki z pieniędzmi) powinny być automatycznie raportowane w dodatkowym pliku tekstowym (włącznie z aktualną datą i godziną).

- Wykorzystywane struktury danych oraz zaproponowany algorytm powinien zostać zaakceptowany przez prowadzącego; w projekcie należy wykorzystać dynamiczne struktury danych (tj. listy, drzewa, stosy, kolejki...),
- Projekt powinien mieć strukturę modułową (pliki nagłówkowe (*.h/*.hpp), zawierające deklaracje, oraz implementacyjne (*.cpp), zawierające definicje),
- Komentarze powinny być zgodne ze standardem Doxygen (pełny opis techniczny),
- Dozwolone jest użycie kontenerów biblioteki stl (np. std::vector, std::list, itd.),
- **Terminy realizacji poszczególnych etapów projektu podane są na PZE**.

3. Książka adresowa

Napisać program do obsługi książki adresowej. Rekordy zapisywane są w pliku tekstowym (można dodać opcjonalnie proste szyfrowanie pliku, tak żeby nie można go było wprost przeglądać).

Każdy z rekordów powinien zawierać podstawowe pola takie jak:

- Typ (osoba prywatna, firma/instytucja, ...),
- Imię,
- Nazwisko,
- Nazwa firmy,
- Adres(y) (możliwość dodatnia kilku),
- Numer(y) telefonów (możliwość dodatnia kilku),
- Ważne daty (urodzenia, imienin, ...) (możliwość dodatnia kilku),
- Dodatkowe informacje.

Książka powinna zawierać proste menu w oknie konsoli pozwalające na przeglądanie, dodawanie, usuwanie, wyszukiwanie (wg określonego klucza) rekordów. Dodatkowo powinna być możliwość wyeksportowania zawartości całej książki do pliku HTML (prosty szablon pliku znajduje się w dodatku A) oraz powiadamianie o zgodności bieżącej daty z datami zapisanymi w rekordach (przypominajka o (zbliżających się) ważnych datach z podaniem liczby dni do lub po najbliższych wydarzeniach).

- Wykorzystywane struktury danych oraz zaproponowany algorytm powinien zostać zaakceptowany przez prowadzącego; w projekcie należy wykorzystać dynamiczne struktury danych (tj. listy, drzewa, stosy, kolejki...),
- Projekt powinien mieć strukturę modułową (pliki nagłówkowe (*.h/*.hpp), zawierające deklaracje, oraz implementacyjne (*.cpp), zawierające definicje),
- Komentarze powinny być zgodne ze standardem Doxygen (pełny opis techniczny),
- Dozwolone jest użycie kontenerów biblioteki stl (np. std::vector, std::list, itd.),
- **Terminy realizacji poszczególnych etapów projektu podane są na PZE**.

4. Gra w odgadywanie słów

Napisać program gry pozwalającej na odgadywanie słów. Słowa do gry dostępne są w pliku tekstowym.

Runda gry polega na wylosowaniu z pliku jednego słowa i wyświetleniu go na ekranie przy czym każda z liter na początku zastąpiona jest znakiem „*". Zadaniem gracza jest: podanie dowolnej litery, jeżeli jest ona w podanym wyrazie to zostaje odsłonięta (bez punktu, błędnie podana litera to odjęcie punktu) lub odgadnięcie całego wyrazu (punktacja taka jak liczba zasłoniętych liter).

Założenia wstępne (do modyfikacji): gra składa się z 10. rund o wzrastającym stopniu trudności (długość odgadywanego wyrazu, rozpoczynając np. od 3 i po każdych 2 kolejnych rundach jest zwiększana o 1) i zostaje zakończona po 10. rundzie lub gracz przegrywa, gdy zdobędzie np. -10pkt.

Zaproponować proste menu na każdym etapie gry. Przed rozpoczęciem gry uczestnik podaje swoją nazwę, a po zakończeniu jego rezultat zostaje zapisany w pliku HTML (prosty format pliku podany w dodatku A) z wynikami (10 pierwszych miejsc). W trakcie gry zapisywana jest historia rozgrywki pozwalająca na wstrzymanie i ponowne wznowienie gry.

- Wykorzystywane struktury danych oraz zaproponowany algorytm powinien zostać zaakceptowany przez prowadzącego; w projekcie należy wykorzystać dynamiczne struktury danych (tj. listy, drzewa, stosy, kolejki...),
- Projekt powinien mieć strukturę modułową (pliki nagłówkowe (*.h/*.hpp), zawierające deklaracje, oraz implementacyjne (*.cpp), zawierające definicje),
- Komentarze powinny być zgodne ze standardem Doxygen (pełny opis techniczny),
- Dozwolone jest użycie kontenerów biblioteki stl (np. std::vector, std::list, itd.),
- **Terminy realizacji poszczególnych etapów projektu podane są na PZE**.

5. Gra w życie (automat komórkowy)

Opracować program do symulacji automatu komórkowego na ograniczonej planszy o rozmiarze 80×25, zachowującego się wg reguł:

- Komórka ożywa, jeśli ma dokładnie trzech żywych sąsiadów.
- Komórka pozostaje żywa, jeśli ma dwóch lub trzech żywych sąsiadów.
- Komórka umiera z „samotności" (mniej niż dwóch sąsiadów) lub „przeludnienia" (więcej niż czterech sąsiadów).

Stan początkowy kolonii jest podany w pliku tekstowym w formacie 25 linii po 80 znaków lub możliwe jest wygenerowanie przez program losowej kolonii (podając liczbę osobników do wygenerowania). Można założyć, że znak „." określa puste miejsce, natomiast „X" osobnika (sprawdzić poprawność formatu pliku wejściowego). Podczas działania programu kolejne pokolenia wyświetlane są w oknie konsoli (z pewnym opóźnieniem), dodatkowo historia kolonii jest zapamiętywana i zapisywana w pliku HTML (uproszony format podany w dodatku A). Program kończy symulację po określonej liczbie iteracji lub w sytuacji kiedy kolonia przechodzi w stan stabilny.

Podstawy Programowania Komputerów

- Wykorzystywane struktury danych oraz zaproponowany algorytm powinien zostać zaakceptowany przez prowadzącego; w projekcie należy wykorzystać dynamiczne struktury danych (tj. listy, drzewa, stosy, kolejki...),
- Projekt powinien mieć strukturę modułową (pliki nagłówkowe (*.h/*.hpp), zawierające deklaracje, oraz implementacyjne (*.cpp), zawierające definicje),
- Komentarze powinny być zgodne ze standardem Doxygen (pełny opis techniczny),
- Dozwolone jest użycie kontenerów biblioteki stl (np. std::vector, std::list, itd.),
- **Terminy realizacji poszczególnych etapów projektu podane są na PZE**.

6. Labirynt

W pliku tekstowym zapisana jest mapa labiryntu w postaci znaków ASCII (spacja oznacza wolną przestrzeń, po której można się poruszać, natomiast za pomocą znaków '-', '+' oraz '|' oznaczane są ściany; do oznaczania ścian można użyć innego dowolnego znaku np. '#'). Przykładowa mapa pokazana jest poniżej.

```
+---+-+---+-------+
|  S| |   |       |
| +-+ +-- | +-+-- |
| |   |     |M    |
| | | | +---+ +---+
|   |   |         |
| --+-+ +-+ | +-+ |
|   | |   | | | | |
| | | +-- +-+ | | |
| | |           | |
| | | --+ --+---+ |
|   |   |   |     |
+---+---+---+-----+
```

Opracować program, zadaniem którego będzie znalezienie (najkrótszej) drogi prowadzącej od punktu startu (oznaczony jako S) do punktu mety (oznaczony jako M). Wynik działania programu powinien być wyświetlony w konsoli oraz dodatkowo zostać zapisany w pliku w postaci mapy wraz z zaznaczoną trasą w postaci punktów. Wynik, oprócz mapy z trasą, powinien zawierać pojedyncze kroki poruszania się po labiryncie (w postaci współrzędnych punktów (x, y)).

- Wykorzystywane struktury danych oraz zaproponowany algorytm powinien zostać zaakceptowany przez prowadzącego; w projekcie należy wykorzystać dynamiczne struktury danych (tj. listy, drzewa, stosy, kolejki...),
- Projekt powinien mieć strukturę modułową (pliki nagłówkowe (*.h/*.hpp), zawierające deklaracje, oraz implementacyjne (*.cpp), zawierające definicje),
- Komentarze powinny być zgodne ze standardem Doxygen (pełny opis techniczny),
- Dozwolone jest użycie kontenerów biblioteki stl (np. std::vector, std::list, itd.),
- **Terminy realizacji poszczególnych etapów projektu podane są na PZE**.

7. Obliczanie wyrażeń

W pliku tekstowym zapisane są działania na liczbach (dostępne operacje: dodawanie, odejmowanie, mnożenie, dzielenie, potęgowanie, pierwiastkowanie...; każde z działań w osobnej linii). Przykładowa zawartość pliku:

```
((2+7)/3+(14-3)*4)/2
```

Wynikiem działanie programu jest plik tekstowy HTML (uproszczona postać pliku pokazana w dodatku A) z obliczonymi wynikami (wykorzystać konwersję działania do postaci Odwrotnej Notacji Polskiej) oraz zapisem działania w notacji ONP. Dla podanego przykładu jego zawartość powinna wyglądać:

```
((2+7)/3+(14-3)*4)/2
2 7 + 3 / 14 3 - 4 * + 2 /
= 23.5
```

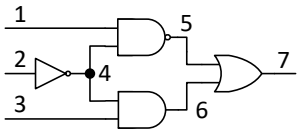
Program dodatkowo powinien wykrywać źle podane działania (np. brak nawiasów, dzielenie przez zero, niepoprawny format liczb, niepoprawne znaki, itp.).

Podstawy Programowania Komputerów

- Wykorzystywane struktury danych oraz zaproponowany algorytm powinien zostać zaakceptowany przez prowadzącego; w projekcie należy wykorzystać dynamiczne struktury danych (tj. listy, drzewa, stosy, kolejki...),
- Projekt powinien mieć strukturę modułową (pliki nagłówkowe (*.h/*.hpp), zawierające deklaracje, oraz implementacyjne (*.cpp), zawierające definicje),
- Komentarze powinny być zgodne ze standardem Doxygen (pełny opis techniczny),
- Dozwolone jest użycie kontenerów biblioteki stl (np. std::vector, std::list, itd.),
- **Terminy realizacji poszczególnych etapów projektu podane są na PZE**.

8. Symulacja układów kombinacyjnych

W pliku tekstowym opisany jest układ cyfrowy złożony z dwuwejściowych bramek logicznych AND, OR, NAND, NOR, XOR oraz jednowejściowej NOT. Połączenie wejść i wyjść bramek jest traktowane jako węzeł. Przykładowo dla układu jak na rysunku



zawartość pliku, opisująca ten układ, jest następująca:

```
IN 1 2 3
OUT 7
NOT 2 4
NAND 1 4 5
AND 3 4 6
OR 5 6 7
```

W pierwszej linii podane są numery węzłów będących wejściem układu. W drugiej linii numery węzłów będące wyjściem układu. Każda następna linia zawiera opis jednej bramki w postaci:

```
typ_bramki węzeł_wejściowy [węzeł_wejściowy] wyjście
```

Program powinien w pliku wyjściowym zapisać stan wyjścia układu dla wszystkich kombinacji stanów wejść np.

```
IN3 IN2 IN1 OUT
 0   0   0   1
 0   0   1   0
 0   1   0   1
 0   1   1   1
 1   0   0   1
 1   0   1   1
 1   1   0   1
 1   1   1   1
```

(stany wyjścia OUT zgodne z układem podanym na rysunku)

Podstawy Programowania Komputerów

- Wykorzystywane struktury danych oraz zaproponowany algorytm powinien zostać zaakceptowany przez prowadzącego; w projekcie należy wykorzystać dynamiczne struktury danych (tj. listy, drzewa, stosy, kolejki...),
- Projekt powinien mieć strukturę modułową (pliki nagłówkowe (*.h/*.hpp), zawierające deklaracje, oraz implementacyjne (*.cpp), zawierające definicje),
- Komentarze powinny być zgodne ze standardem Doxygen (pełny opis techniczny),
- Dozwolone jest użycie kontenerów biblioteki stl (np. std::vector, std::list, itd.),
- **Terminy realizacji poszczególnych etapów projektu podane są na PZE**.

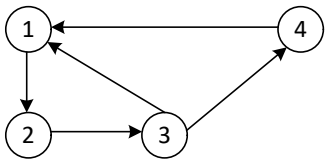
---

9. Wyznaczanie cykli w grafie skierowanym

W pliku tekstowym opisany jest graf skierowany w postaci par numerów wierzchołków A -> B lub C <- D, gdzie A i D są wierzchołkami początkowymi, natomiast B i C końcowymi. Przykładowa zawartość pliku:

```
1 -> 2
2 -> 3
3 -> 1
3 -> 4
1 <- 3
1 <- 4
```

odpowiada grafowi skierowanemu



Napisać program, który dla grafu skierowanego, opisanego w pliku, określa jego cykle (także podaje czy jest acykliczny) i zapisuje wyniki w tekstowym pliku wyjściowym.

Podstawy Programowania Komputerów

- Wykorzystywane struktury danych oraz zaproponowany algorytm powinien zostać zaakceptowany przez prowadzącego; w projekcie należy wykorzystać dynamiczne struktury danych (tj. listy, drzewa, stosy, kolejki...),
- Projekt powinien mieć strukturę modułową (pliki nagłówkowe (*.h/*.hpp), zawierające deklaracje, oraz implementacyjne (*.cpp), zawierające definicje),
- Komentarze powinny być zgodne ze standardem Doxygen (pełny opis techniczny),
- Dozwolone jest użycie kontenerów biblioteki stl (np. std::vector, std::list, itd.),
- **Terminy realizacji poszczególnych etapów projektu podane są na PZE**.

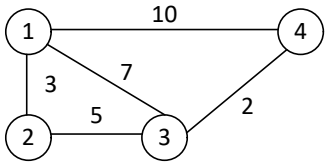
---

10. Wyznaczanie najkrótszych ścieżek w grafie

W pliku tekstowym opisany jest graf w postaci par numerów wierzchołków A, B oraz wagą krawędzi. Przykładowa zawartość pliku:

```
1, 2, 3
1, 3, 7
1, 4, 10
2, 3, 5
3, 4, 2
```

odpowiada grafowi nieskierowanemu.



Napisać program, który dla grafu, opisanego w pliku, wyznaczy wszystkie najkrótsze ścieżki od danego wierzchołka do pozostałych i zapisze wyniki w pliku tekstowym.

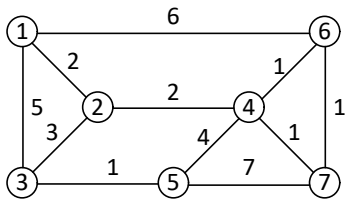
Podstawy Programowania Komputerów

- Wykorzystywane struktury danych oraz zaproponowany algorytm powinien zostać zaakceptowany przez prowadzącego; w projekcie należy wykorzystać dynamiczne struktury danych (tj. listy, drzewa, stosy, kolejki...),
- Projekt powinien mieć strukturę modułową (pliki nagłówkowe (*.h/*.hpp), zawierające deklaracje, oraz implementacyjne (*.cpp), zawierające definicje),
- Komentarze powinny być zgodne ze standardem Doxygen (pełny opis techniczny),
- Dozwolone jest użycie kontenerów biblioteki stl (np. std::vector, std::list, itd.),
- **Terminy realizacji poszczególnych etapów projektu podane są na PZE**.

---

11. Projekt sieci przewodowej

Należy zaprojektować sieć przewodową łączącą n hubów przy zużyciu jak najmniej kabla. Nie wszystkie pary hubów dają się połączyć, gdyż są za bardzo oddalone od siebie. Pomiędzy parami hubów, które możemy połączyć, znane są wymagane długości kabla. Nie można tworzyć nowych rozgałęzień przy tworzeniu sieci. Wszystkie możliwe połączenia, dla przykładowej topologii pokazanej na rysunku (huby oznaczone są kółkami, natomiast przy połączeniach podane są odległości)



opisane są w pliku tekstowym w postaci

```
1, 6, 6
1, 2, 2
1, 3, 5
2, 3, 3
2, 4, 2
3, 5, 1
4, 5, 4
4, 6, 1
4, 7, 1
5, 7, 7
6, 7, 1
```

przy czym pierwsza i druga wartość określają numery łączonych hubów, natomiast trzecia jest odległością między nimi. Program powinien znaleźć sieć łączącą huby tak, aby możliwy był transport danych między wszystkimi parami (z użyciem hubów pośredniczących) i wyniki zapisać w pliku (w formacie podanym jak wyżej).

Podstawy Programowania Komputerów

- Wykorzystywane struktury danych oraz zaproponowany algorytm powinien zostać zaakceptowany przez prowadzącego; w projekcie należy wykorzystać dynamiczne struktury danych (tj. listy, drzewa, stosy, kolejki...),
- Projekt powinien mieć strukturę modułową (pliki nagłówkowe (*.h/*.hpp), zawierające deklaracje, oraz implementacyjne (*.cpp), zawierające definicje),
- Komentarze powinny być zgodne ze standardem Doxygen (pełny opis techniczny),
- Dozwolone jest użycie kontenerów biblioteki stl (np. std::vector, std::list, itd.),
- **Terminy realizacji poszczególnych etapów projektu podane są na PZE**.

---

12. Kompresja plików metodą Huffmana

Opracować program do kompresji (dekompresji) plików metodą Huffmana. Podczas kompresji tworzony jest słownik, zapisywany w pliku, wykorzystywany podczas dekompresji. Program powinien być wywoływany z parametrami: tryb (kompresja czy dekompresja), plik wejściowy, plik wyjściowy, plik słownika.

Podstawy Programowania Komputerów

- Wykorzystywane struktury danych oraz zaproponowany algorytm powinien zostać zaakceptowany przez prowadzącego; w projekcie należy wykorzystać dynamiczne struktury danych (tj. listy, drzewa, stosy, kolejki...),
- Projekt powinien mieć strukturę modułową (pliki nagłówkowe (*.h/*.hpp), zawierające deklaracje, oraz implementacyjne (*.cpp), zawierające definicje),
- Komentarze powinny być zgodne ze standardem Doxygen (pełny opis techniczny),
- Dozwolone jest użycie kontenerów biblioteki stl (np. std::vector, std::list, itd.),
- **Terminy realizacji poszczególnych etapów projektu podane są na PZE**.

---

13. Kurier

Kurier ma za zadanie zawieść towar do klientów w różnych lokalizacjach i powrócić do miejsca, z którego wyjechał. Kurier musi odwiedzić każdego klienta raz i tylko raz. Należy znaleźć zamkniętą najkrótszą drogę, która umożliwia odwiedzenie wszystkich klientów. W pliku wejściowym zapisane są długości dróg pomiędzy miastami. Drogi zapisane są w następujący sposób `klientA - klientB, odległość`. Niektóre drogi nie są symetryczne, tzn. jest pewna różnica między drogą tam a z powrotem. Zapis `klientC -> klientD, odległość`, oznacza długość drogi jednokierunkowej od klienta C do klienta D. Poszczególne drogi znajdują się w kolejnych liniach pliku tekstowego. Przykładowa zawartość pliku wygląda następująco:

```
1 - 2, 60
1 - 3, 50
2 -> 4, 4.5
4 -> 2, 7.5
...
```

Nie jest podana liczba dróg. Jeżeli nie jest możliwe wyznaczenie drogi, program zgłasza odpowiedni komunikat. Wyniki powinny zostać wyświetlone w oknie konsoli i zapisane w pliku tekstowym.

Podstawy Programowania Komputerów

- Wykorzystywane struktury danych oraz zaproponowany algorytm powinien zostać zaakceptowany przez prowadzącego; w projekcie należy wykorzystać dynamiczne struktury danych (tj. listy, drzewa, stosy, kolejki...),
- Projekt powinien mieć strukturę modułową (pliki nagłówkowe (*.h/*.hpp), zawierające deklaracje, oraz implementacyjne (*.cpp), zawierające definicje),
- Komentarze powinny być zgodne ze standardem Doxygen (pełny opis techniczny),
- Dozwolone jest użycie kontenerów biblioteki stl (np. std::vector, std::list, itd.),
- **Terminy realizacji poszczególnych etapów projektu podane są na PZE**.

---

14. Połączenia kolejowe

Napisać program, który umożliwia znalezienie najkrótszej trasy kolejowej między dwoma wybranymi miastami. Miasta połączone są trasami (jednokierunkowe) o podanej długości. Plik mapy tras ma następującą postać (w każdej linii podana jest jedna trasa): `miasto_początkowe miasto_końcowe odległość`. Przykładowy plik z trasami (dla kilku wybranych tras)

```
Katowice Krakow 70
Krakow Tarnow 70
Tarnow Jaslo 50
Katowice Gliwice 22
Lodz Poznan 205
Gliwice Katowice 22
Katowice Czestochowa 70
Czestochowa Lodz 120
Lodz Torun 165
Krakow Katowice 70
Gliwice Wroclaw 180
```

Drugim plikiem wejściowym jest plik z trasami do wyznaczenia. Każda linia pliku zawiera jedną trasę w postaci: `miasto_początkowe miasto_końcowe` (podobnie jak w pliku z podanymi trasami, ale bez określania odległości). Wynikiem działania programu jest plik wyjściowy z wyznaczonymi trasami, tzn. podana jest nazwa trasy, całkowita długość, a potem poszczególne odcinki z długościami, np.

```
Katowice --> Torun (355 km)
---------------------------------
Katowice --> Czestochowa 70
Czestochowa --> Lodz 120
Lodz --> Torun 165
```

.

- Wykorzystywane struktury danych oraz zaproponowany algorytm powinien zostać zaakceptowany przez prowadzącego; w projekcie należy wykorzystać dynamiczne struktury danych (tj. listy, drzewa, stosy, kolejki...),
- Projekt powinien mieć strukturę modułową (pliki nagłówkowe (*.h/*.hpp), zawierające deklaracje, oraz implementacyjne (*.cpp), zawierające definicje),
- Komentarze powinny być zgodne ze standardem Doxygen (pełny opis techniczny),
- Dozwolone jest użycie kontenerów biblioteki stl (np. std::vector, std::list, itd.),
- **Terminy realizacji poszczególnych etapów projektu podane są na PZE**.

15. Program szyfrujący pliki tekstowe

Napisać program, który szyfruje (deszyfruje) pliki tekstowe metodą Vigénre'a. Po wywołaniu programu jako parametry należy podać tryb (szyfrowanie czy deszyfrowanie), nazwę pliku jawnego, nazwę pliku zaszyfrowanego oraz klucz szyfrująco-deszyfrujący. W celu oznaczenia pliku zaszyfrowanego można wprowadzić dodatkowy znacznik tekstowy, wtedy nie będzie konieczne podanie trybu. Rozszerzyć działanie programu na pliki tekstowe zawierające wszystkie znaki ASCII.

Podstawy Programowania Komputerów

- Wykorzystywane struktury danych oraz zaproponowany algorytm powinien zostać zaakceptowany przez prowadzącego; w projekcie należy wykorzystać dynamiczne struktury danych (tj. listy, drzewa, stosy, kolejki...),
- Projekt powinien mieć strukturę modułową (pliki nagłówkowe (*.h/*.hpp), zawierające deklaracje, oraz implementacyjne (*.cpp), zawierające definicje),
- Komentarze powinny być zgodne ze standardem Doxygen (pełny opis techniczny),
- Dozwolone jest użycie kontenerów biblioteki stl (np. std::vector, std::list, itd.),
- **Terminy realizacji poszczególnych etapów projektu podane są na PZE**.

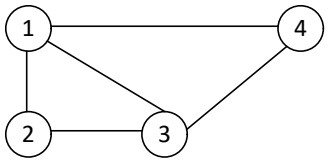
---

16. Kolorowanie grafu nieskierowanego

W pliku tekstowym opisany jest graf w postaci par numerów wierzchołków `A - B` połączonych krawędzią. Przykładowa zawartość pliku:

```
1 - 2
1 - 3
1 - 4
2 - 3
3 - 4
```

odpowiada grafowi nieskierowanemu pokazanemu na rysunku



Napisać program, który dla grafu, opisanego w pliku, pokoloruje jego wierzchołki (dwa wierzchołki połączone krawędzią muszą mieć inne kolory) przy użyciu minimalnej liczby kolorów. Wyniki wyświetlić w oknie konsoli oraz dodatkowo zapisać w pliku tekstowym w postaci np.

```
kolor A: 1
kolor B: 2, 4
kolor C: 3
```

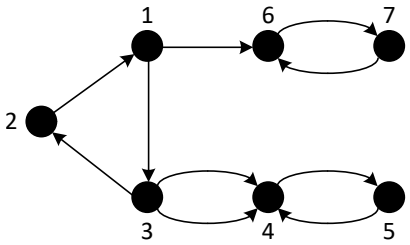
.

Podstawy Programowania Komputerów

- Wykorzystywane struktury danych oraz zaproponowany algorytm powinien zostać zaakceptowany przez prowadzącego; w projekcie należy wykorzystać dynamiczne struktury danych (tj. listy, drzewa, stosy, kolejki...),
- Projekt powinien mieć strukturę modułową (pliki nagłówkowe (*.h/*.hpp), zawierające deklaracje, oraz implementacyjne (*.cpp), zawierające definicje),
- Komentarze powinny być zgodne ze standardem Doxygen (pełny opis techniczny),
- Dozwolone jest użycie kontenerów biblioteki stl (np. std::vector, std::list, itd.),
- **Terminy realizacji poszczególnych etapów projektu podane są na PZE**.

17. Drogi

W pewnym państwie jest n miast. Miasta są połączone jednokierunkowymi drogami. Każda droga łączy tylko dwa miasta i nie przechodzi przez żadne inne. Niestety, istniejąca sieć dróg nie zawsze pozwala dojechać z wybranego miasta do każdego innego. Ze względu na wysoki koszt budowy nowych dróg należy wyznaczyć minimalną liczbę jednokierunkowych dróg, które trzeba zbudować, by usunąć opisany problem.
Napisać program, który wczyta z pliku tekstowego opis istniejącej sieci dróg, następnie wyznaczy minimalną liczbę nowych dróg, a wynik zapisze w pliku tekstowym. Dla przykładowej pierwotnej mapy dróg pokazanej na rysunku



dane w pliku mogą mieć postać

```
7
11
1->3
3->2
2->1
3->4
4->5
5->4
3->4
1->6
6->7
7->6
```

przy czym w pierwszej linii zapisano liczbę miast, w drugiej aktualną liczbę połączeń, a następnie drogi jednokierunkowe łączące pary miast.

- Wykorzystywane struktury danych oraz zaproponowany algorytm powinien zostać zaakceptowany przez prowadzącego; w projekcie należy wykorzystać dynamiczne struktury danych (tj. listy, drzewa, stosy, kolejki...),
- Projekt powinien mieć strukturę modułową (pliki nagłówkowe (*.h/*.hpp), zawierające deklaracje, oraz implementacyjne (*.cpp), zawierające definicje),
- Komentarze powinny być zgodne ze standardem Doxygen (pełny opis techniczny),
- Dozwolone jest użycie kontenerów biblioteki stl (np. std::vector, std::list, itd.),
- **Terminy realizacji poszczególnych etapów projektu podane są na PZE**.

18. Transport

Napisać program, który zaplanuje przewożenie przedmiotów z jednego magazynu do drugiego wg. określonego kryterium tak, by liczba przejazdów była jak najmniejsza. Opis przedmiotów znajduje się w pliku tekstowym, w którym każda linia ma format
`<przedmiot> <gabaryt> <waga> <cena> <liczba sztuk>`
Przykładowy plik może wyglądać następująco (dla kilku wybranych przedmiotów)

```
tablet 2 0.6 860 3
komputer 5 7.5 2200 1
monitor 3 5.0 750 4
aparat 1 0.2 1000 2
```

Jako parametry wywołania programu podawane są: nazwa pliku opisującego przedmioty oraz parametry platformy do przewożenia przedmiotów (maksymalne sumaryczne gabaryty `-g[gabaryt]`, maksymalna sumaryczna waga `-w[waga]`, maksymalna sumaryczna cena `-c[cena]`). Program powinien sprawdzać poprawność podawanych parametrów.
W pliku wyjściowym powinna zostać zapisana liczba kursów realizowanych pomiędzy magazynami z wyszczególnieniem przewożonych przedmiotów wraz z informacją o przewożonym ładunku.

- Wykorzystywane struktury danych oraz zaproponowany algorytm powinien zostać zaakceptowany przez prowadzącego; w projekcie należy wykorzystać dynamiczne struktury danych (tj. listy, drzewa, stosy, kolejki...),
- Projekt powinien mieć strukturę modułową (pliki nagłówkowe (*.h/*.hpp), zawierające deklaracje, oraz implementacyjne (*.cpp), zawierające definicje),
- Komentarze powinny być zgodne ze standardem Doxygen (pełny opis techniczny),
- Dozwolone jest użycie kontenerów biblioteki stl (np. std::vector, std::list, itd.),
- **Terminy realizacji poszczególnych etapów projektu podane są na PZE**.

19. Połączenia międzymiastowe

W pliku tekstowym umieszczone zostały informacje dotyczące czasu przejazdu pomiędzy miastami, przy założeniu, że czas przejazdu z miasta A do B oraz z B do A może być różny (inne drogi lub przeszkody na drodze przejazdu). Przykładowy plik z danymi może mieć postać (komentarz w pliku rozpoczyna się od znaku średnika i powinien być ignorowany):

```
; polaczenia miedzy miastami z~czasem przejazdu
; z~miasta, do miasta, czas
Katowice,Krakow,65
Krakow,Katowice,60
Katowice,Gliwice,28
Gliwice,Katowice,35
Warszawa,Krakow,130
Katowice, Warszawa,150
Katowice,Czestochowa,76
Czestochowa,Katowice,80
```

Napisać program, który wyznaczy wszystkie możliwe trasy (z podaniem czasu przejazdu) pomiędzy dwoma wybranymi miastami (miasta proponowane są w oparciu o zawartość pliku).
Wyniki wyświetlić w oknie konsoli oraz zapisać w pliku (format zapisu do wyboru). Program powinien ponadto sprawdzić poprawność danych w pliku oraz wskazać miasta, z których możliwy jest tylko wyjazd lub do których można tylko dojechać (błędnie opracowany plik z danymi).

Podstawy Programowania Komputerów

- Wykorzystywane struktury danych oraz zaproponowany algorytm powinien zostać zaakceptowany przez prowadzącego; w projekcie należy wykorzystać dynamiczne struktury danych (tj. listy, drzewa, stosy, kolejki...),
- Projekt powinien mieć strukturę modułową (pliki nagłówkowe (*.h/*.hpp), zawierające deklaracje, oraz implementacyjne (*.cpp), zawierające definicje),
- Komentarze powinny być zgodne ze standardem Doxygen (pełny opis techniczny),
- Dozwolone jest użycie kontenerów biblioteki stl (np. std::vector, std::list, itd.),
- **Terminy realizacji poszczególnych etapów projektu podane są na PZE**.

20. Program szyfrujący pliki tekstowe ENIGMA

Napisać program, który szyfruje (deszyfruje) pliki tekstowe z wykorzystaniem algorytmu ENIGMY. Założenia: szyfrowane są wyłącznie znaki A-Z oraz cyfry (pozostałe znaki nie będą występować); liczba bębnów szyfrujących 3-5; parametry maszyny zapisane w pliku tekstowym (liczba bębnów, połączenia w bębnach i w bębnie odwracającym; zaproponować format zapisu). Początkowe położenie bębnów szyfrujących należy podać z klawiatury.

Podstawy Programowania Komputerów

- Wykorzystywane struktury danych oraz zaproponowany algorytm powinien zostać zaakceptowany przez prowadzącego; w projekcie należy wykorzystać dynamiczne struktury danych (tj. listy, drzewa, stosy, kolejki...),
- Projekt powinien mieć strukturę modułową (pliki nagłówkowe (*.h/*.hpp), zawierające deklaracje, oraz implementacyjne (*.cpp), zawierające definicje),
- Komentarze powinny być zgodne ze standardem Doxygen (pełny opis techniczny),
- Dozwolone jest użycie kontenerów biblioteki stl (np. std::vector, std::list, itd.),
- **Terminy realizacji poszczególnych etapów projektu podane są na PZE**.

---

21. Gra Mastermind

Napisać program gry planszowej Mastermind. Zasady gry:

- Program losuje 4 pionki z zestawu 6 kolorów (kodowane w konsoli jako cyfry lub wybrane litery, można używać kolorów w konsoli),
- Gracz odgaduje w kolejnych krokach kolory pionków, program podpowiada trafienia, które kodowane są za pomocą białych i czarnych minipionków (biały - kolor i położenie pionka trafione, czarny - kolor trafiony, ale nie położenie; kolejność położenia minipionków może być przypadkowa),
- Gra zostaje zakończona po odgadnięciu prawidłowego położenia wszystkich pionków.

Przed rozpoczęciem gry w programie należy podać nazwę gracza, a po rozgrywce, w pliku HTML, zapisywany jest wynik gracza (ranking 10. najlepszych miejsc - wygrana w jak najmniejszej liczbie kroków). Dodatkowo w pliku tekstowym skojarzonym z danym graczem zapisywana jest historia rozgrywki.

Przykład:

```
A B C D    (wylosowane pionki, niewidoczne dla gracza)
-------
E F C A~xo.. (1. runda, "x" jeden pion trafiony, "o" drugi nie te położenie)
A F C E xx.. (2. runda, "x" dwa piony trafione)
A D C B xxoo (3. runda, "x" dwa piony trafione, "o" dwa nie te położenie)
A B C D xxxx (4. runda, "x" wszystkie piony trafione)
```

.

Dodatek A

Przykład strony WWW zawierającej tekst oraz prostą tabelkę, zapisaną w formacie html.

```
<!DOCTYPE html>
<html lang="pl">

<head>
    <title>Tytuł Strony</title>
</head>

<body>
    <h1>Nagłówek</h1>
    <p>Tekst na stronie, a poniżej tabelka.</p>

    <style> td { border: 1px solid black; width: 50px; text-align: center} </style>
    <!-- ustawienie czarnego obramowania komórek tabeli w CSS -->

    <table>
    <tr>
       <td>X</td> <td></td> <td>O</td>
    </tr>
    <tr>
       <td>X</td> <td>O</td> <td></td>
    </tr>
    <tr>
       <td></td> <td>O</td> <td></td>
    </tr>
    </table>

</body>

</html>
```

Wygląd strony w przeglądarce.

Nagłówek

Tekst na stronie, a poniżej tabelka.

X		O
X	O	
	O	