Zadanie 1. Zachłanne kolorowanie grafu

Napisz funkcje realizujące następujące operacje na liście sąsiedztwa wierzchołków grafu:

= dodawanie wierzchołka,
= usuwanie wierzchołka,
= dodawanie krawędzi,
= usuwanie krawędzi,
= czytelne wypisanie listy krawędzi.

Zaimplementuj algorytm zachłannego kolorowania grafu reprezentowanego jako **lista sąsiedztwa wierzchołków**. Kolory przydzielone wierzchołkom zapisz w dynamicznie utworzonym wektorze zawierającym tyle elementów ile jest wierzchołków w grafie.

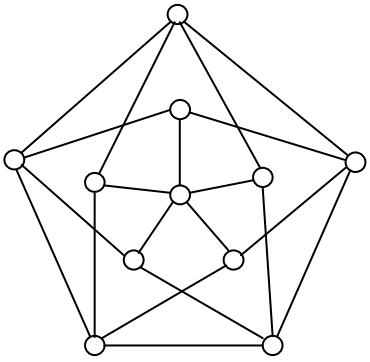
Algorytm zachłannego kolorowania przypisuje kolejnym wierzchołkom grafu kolory, stosując dla każdego z nich zasadę przydzielania możliwie najmniejszego koloru nieużytego dotychczas w sąsiedztwie.

Algorytm Greedy – kolorowanie zachłanne

```
(1)         Dopóki są w grafie niepokolorowane wierzchołki wykonuj co następuje:
(1.2)            weź dowolny niepokolorowany wierzchołek v
(1.3)            kolor <- 1
(1.4)            Dopóki v jest niepokolorowany wykonuj co następuje:
(1.4.1)                Jeżeli istnieje sąsiad wierzchołka v pokolorowany
                       kolorem kolor
(1.4.1.1)                     to kolor <- kolor + 1  // weź następny kolor
(1.4.1.2)                     w przeciwnym wypadku pokoloruj v kolorem kolor
```

Przykład 1 : Kolorowanie zachłanne grafu będącego ścieżką o czterech wierzchołkach

Kolejne wierzchołki ścieżki to v_1, ..., v_4. Jeżeli algorytm koloruje w kolejności v_1, v_4, v_2, v_3 to użyje kolorów 1, 1, 2, 3. Jeżeli jednak wierzchołki będą kolorowane w kolejności v_1, v_2, v_3, v_4 to otrzymamy pokolorowanie optymalne przy pomocy dwóch kolorów. Zawsze da się znaleźć takie uporządkowanie wierzchołków, które daje optymalne pokolorowanie ale nie jest to łatwe nawet wtedy, gdy liczba wierzchołków jest stosunkowo niewielką liczbą.



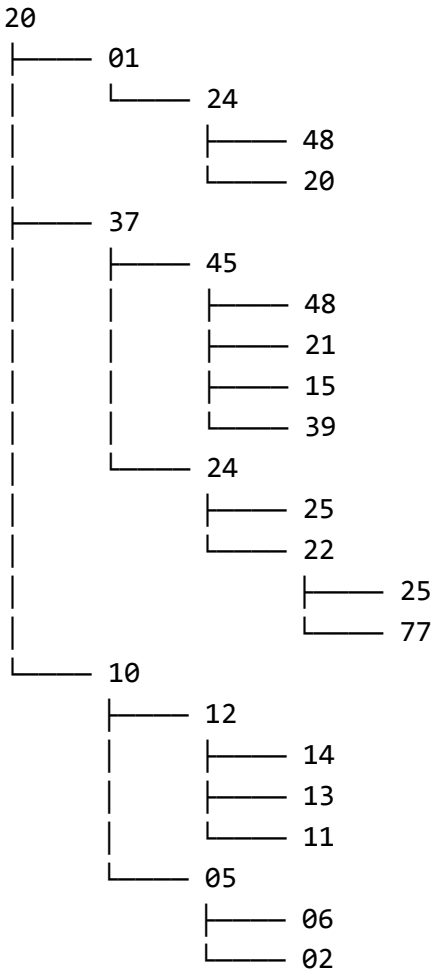
Ćwiczenie: Jaka jest najmniejsza liczba kolorów konieczna do pokolorowania poniższego grafu? Czy algorytm zachłanny zawsze poda dla tego grafu rozwiązanie optymalne?

Zastosowania: Przydział kanałów komunikacyjnych, szeregowanie zadań, konstruowanie rozkładów zajęć, itp.

UWAGA: nie stosuj STL

Zadanie 2. Ortogonalna wizualizacja drzewa

Napisz program wyświetlający drzewo w przedstawionej poniżej postaci. Przyjmij, że drzewo reprezentowane jest przez listy sąsiedztwa wierzchołków, z jednym wierzchołkiem wyróżnionym jako korzeń oraz, że wysokość drzewa pozwala na jego narysowanie na ekranie o szerokości 80 znaków.



Ciekawostka: reprezentacja ta stosowana jest przez polecenie 'tree' interpretera poleceń systemu operacyjnego Windows do wyświetlania struktury folderów.

Uwaga: nie stosuj tablicy do tworzenia "obrazu" drzewa oraz biblioteki STL

Zadanie 3. Lista liniowa dwukierunkowa

Zaimplementuj podstawowe operacji tworzenia i modyfikowania listy liniowej dwukierunkowej:

= dodawanie/usuwanie pierwszego lewego/ pierwszego prawego elementu,

= dodawanie elementu za elementem wskazywanym,

= usuwanie wskazywanego elementu,

= wyszukiwanie elementu o zadanej wartości atrybutu,

= wypisanie elementów listy,

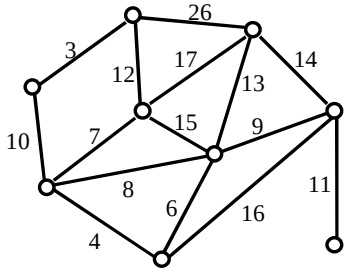
= zwalnianie pamięci zajmowanej przez listę,

= sortowanie listy metodą bąbelkową ze zmianą kierunku przeglądania.

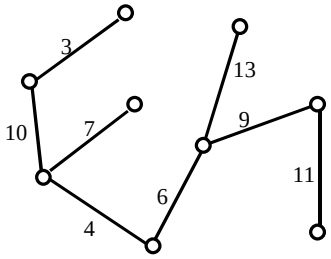
Przykład kompletnej implementacji dla listy jednokierunkowej oraz kilku funkcji dla listy dwukierunkowej znajdziesz na slajdach z wykładu POP.

Zadanie 4. Sieć kolejowa

W pewnej okolicy ludzie mieszkający w kilku miastach zapragnęli, aby pomiędzy miastami można było podróżować koleją. Zlecili więc pewnemu przedsiębiorcy budowlanemu budowę sieci kolejowej. Roztargnieni mieszkańcy podali kwotę, którą zapłacą za wybudowanie sieci ale postawili tylko jeden wymóg. Chcieli aby z każdego miasta można było dojechać pociągiem do dowolnego innego miasta. Przedsiębiorca był chytry. Zbudował sieć o najmniejszej długości (najtańszą), choć niekoniecznie najwygodniejszą dla mieszkańców których bardziej interesowałoby np. zminimalizowanie średniego czasu podróży niż sumaryczna długość wszystkich połączeń. To co wytyczył przedsiębiorca budowlany nazywane jest *najtańszym drzewem rozpinającym* grafu będącego modelem dla tego problemu (graf ten nie musi posiadać wszystkich krawędzi ze względu na pewne dodatkowe uwarunkowania takie jak np. ukształtowanie terenu).



Spójny graf z wagami na krawędziach



Najtańsze drzewo rozpinające (koszt: 63)

Algorytm KRUSKALA – wyznaczanie najtańszego drzewa rozpinającego

```
(1)     Zbiór krawędzi posortuj rosnąco według wag
(2)     Kolejno, w niemalejącym porządku wag, wykonuj dla każdej krawędzi co
        następuje:
(2.1)           Sprawdź czy dodanie krawędzi do drzewa powoduje powstanie cyklu
                i jeżeli nie to dodaj ją do drzewa
```

Napisz program wyznaczający najtańsze drzewo rozpinające grafu przy pomocy **algorytmu Kruskala**. Graf reprezentuj jako dynamiczną listę liniową jednokierunkową o elementach reprezentujących krawędzie (numer pierwszego wierzchołka, numer drugiego wierzchołka, waga krawędzi). Do reprezentacji drzewa możesz zastosować podobną strukturę.

Uwaga: podczas wybierania kolejnych krawędzi do drzewa może się zdarzyć, że zanim algorytm zakończy działanie drzewo nie będzie spójne (będzie zbiorem drzew, czyli lasem). Zastanów się nad możliwością zaznaczania (numerowania) wierzchołków drzew w lesie tak aby było wiadomo które wierzchołki są z drzewa 1 (nadaj im numer 1), które z drzewa 2, itd. Dodając kolejną krawędź będziemy mogli wykryć cykle sprawdzając czy połączy ona wierzchołki tego samego czy różnych drzew. Nie zapomnij o przenumerowaniu wierzchołków, gdy połączysz dwa drzewa w jedno.

Uwaga: nie stosuj STL

Zadanie 5. Operacje na drzewie binarnych poszukiwań BST

Napisz program zawierający funkcje pozwalające na tworzenie drzewa BST o unikalnych wartościach elementów, funkcję usuwania z drzewa węzła o podanej wartości oraz funkcję drukującą drzewo w podanej poniżej postaci.

Wskazówka: zastanów się nad wykorzystaniem jednego z klasycznych sposobów przeglądania drzew binarnych (inorder, preorder, postorder).

Przykład:

```
                    48
               45
                    42
          37
                    25
               24
                    21
     20
                    14
               12
                    11
          10
                    06
               05
                    02
```

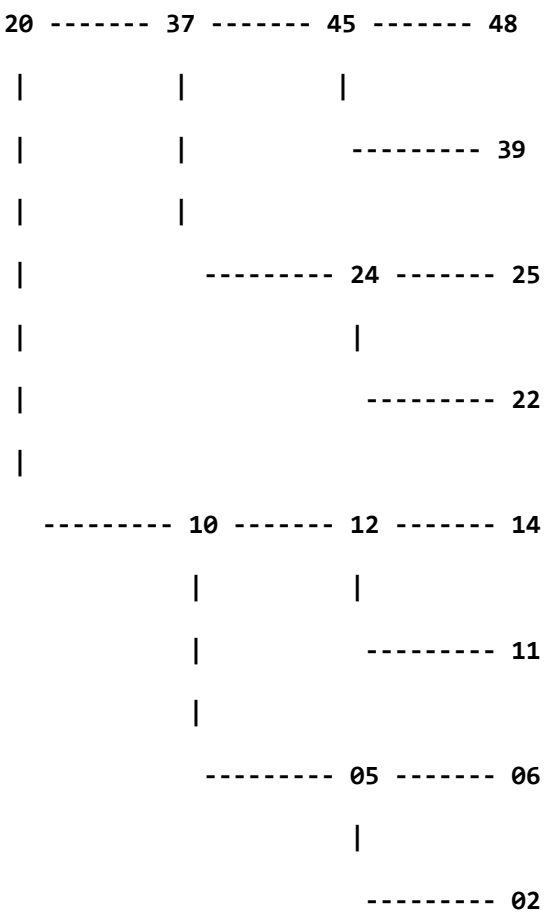
Uwagi:

1. Nie stosuj tablicy do zapamiętania "obrazu" drzewa.
2. Drzewo w przykładzie jest drzewem "pełnym". Zadbaj, aby twój program działał dla każdego drzewa, czyli również wtedy, gdy nie wszystkie liście są na tym samym poziomie.
3. Nie stosuj biblioteki STL

Zadanie 6. Operacje na drzewie binarnych poszukiwań BST

Napisz program zawierający funkcje pozwalające na tworzenie drzewa BST o unikalnych wartościach elementów oraz funkcję usuwania z drzewa węzła o zadanej wartości. Następnie dodaj funkcję wyznaczającą wysokość drzewa i zastosuj ją do drukowania drzewa na ekranie. w postaci podanej poniżej.

Wskazówka: zastanów się nad wykorzystaniem jednego z klasycznych sposobów przeglądania drzew binarnych (inorder, preorder, postorder).



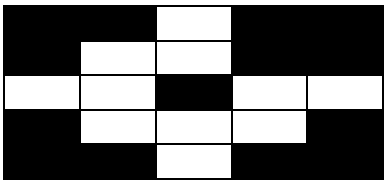
Uwagi:

1. Nie stosuj tablicy do zapamiętania "obrazu" drzewa.
2. Drzewo w przykładzie jest drzewem "pełnym". Zadbaj aby twój program działał dla każdego drzewa, czyli również wtedy gdy nie wszystkie liście są na tym samym poziomie.
3. Nie stosuj biblioteki STL

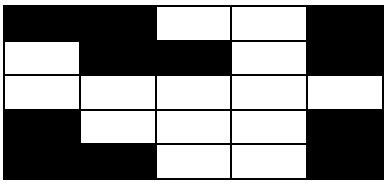
Zadanie 7. Poszukiwanie drogi

Napisz program wyznaczający drogę robota, którego zadaniem jest dotarcie od punktu A do punktu B na mapie składającej się z 8x4 losowo wybieranych segmentów (mogą się powtarzać):

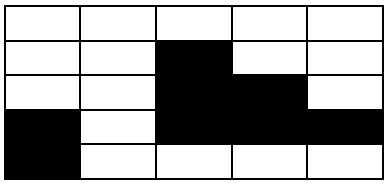
Segment A:



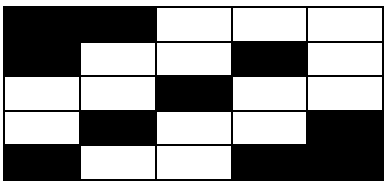
Segment B:



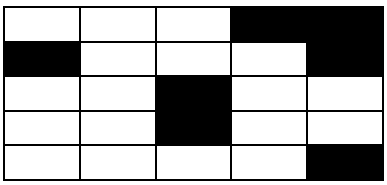
Segment C:



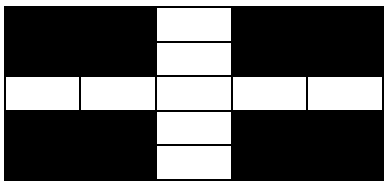
Segment D:



Segment E:



Segment F:



Robot może się poruszać tylko po jasnych polach. Punkty A i B na mapie ustawiane są losowo.

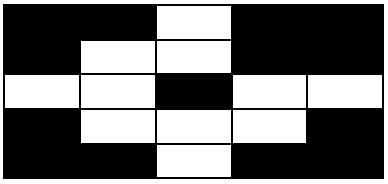
Uwaga: mapę reprezentuj jako graf o wierzchołkach w jasnych polach oraz krawędziach łączących wierzchołki, gdy odpowiadające im pola posiadają wspólny bok. Graf zapisz jako dynamiczne listy sąsiedztwa wierzchołków. Do rozwiązania zadania wykorzystaj algorytm przeszukiwania wszerz (**BFS**).

Uwaga: Nie stosuj biblioteki STL

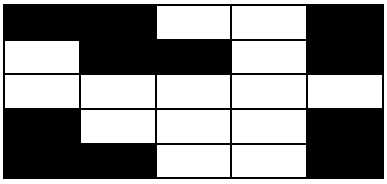
Zadanie 8. Poszukiwanie drogi

Napisz program wyznaczający drogę robota, którego zadaniem jest dotarcie od punktu A do punktu B na mapie składającej się z 8x4 losowo wybieranych segmentów (mogą się powtarzać):

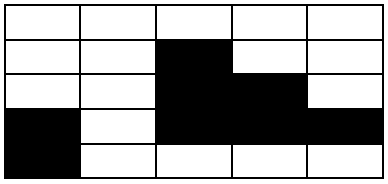
Segment A:



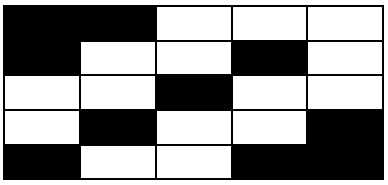
Segment B:



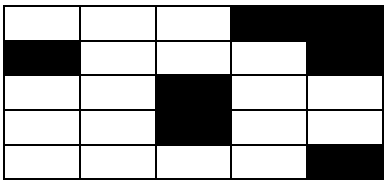
Segment C:



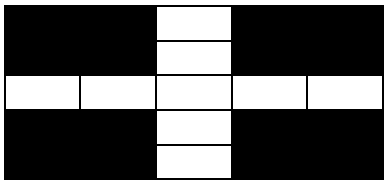
Segment D:



Segment E:



Segment F:



Robot może się poruszać tylko po jasnych polach. Punkty A i B na mapie ustawiane są losowo.

Uwaga: mapę reprezentuj jako graf o wierzchołkach w jasnych polach oraz krawędziach łączących wierzchołki, gdy odpowiadające im pola posiadają wspólny bok. Graf zapisz jako dynamiczne listy sąsiedztwa wierzchołków. Do rozwiązania zadania wykorzystaj algorytm przeszukiwania w głąb (**DFS**).

Uwaga: Nie stosuj biblioteki STL

Zadanie 9. Sieć wodociągowa

Napisz program projektujący sieć wodociągową dostarczającą wodę do miast.

Wodociąg może być zlokalizowany tylko wzdłuż dróg łączących miasta.

Woda do danego miasta może być dostarczana tylko jednym wodociągiem - inaczej mówiąc wodociągi nie mogą tworzyć zamkniętej pętli. Zauważ, że tak postawiony problem polega na wyznaczeniu drzewa rozpinającego grafu reprezentującego sieć dróg.

Przykład:

```
A ---- B ---- C
|              |
D ---- E ---- F
|      |      |
G ---- H      I
```

gdzie pojedyncze kreski (- oraz |) oznaczają połączenia drogowe.

```
A ==== B ==== C
||             |
D ==== E ==== F
||     |      ||
G ==== H      I
```

gdzie podwójne kreski (= oraz ||) oznaczają sieć wodociągową.

Program powinien umożliwić:

1. Wygenerowanie mapy miast o zadanych wymiarach n x m , wraz z losowo wybranymi drogami tworzącymi sieć ortogonalną tzn., dopuszczalne są wyłącznie połączenia prostopadłe pomiędzy miastami w sąsiednich wierszach/kolumnach.
2. Sprawdzenie spójności sieci.
3. Automatyczne wygenerowanie połączeń wodociągowych na podstawie połączeń drogowych.
4. Wypisanie mapy sieci wodociągowej.

Uwaga: do sprawdzenia spójności jak i do wyznaczenia sieci wodociągów należy wykorzystać algorytm przeszukiwania w głąb (**DFS**).

Uwaga: Nie stosuj biblioteki STL

Zadanie 10. Sieć wodociągowa

Napisz program projektujący sieć wodociągową dostarczającą wodę do miast.

Wodociąg może być zlokalizowany tylko wzdłuż dróg łączących miasta.

Woda do danego miasta może być dostarczana tylko jednym wodociągiem - inaczej mówiąc wodociągi nie mogą tworzyć zamkniętej pętli. Zauważ, że tak postawiony problem polega na wyznaczeniu drzewa rozpinającego grafu reprezentującego sieć dróg.

Przykład:

```
A ---- B ---- C
|              |
D ---- E ---- F
|      |      |
G ---- H      I
```

gdzie pojedyncze kreski (- oraz |) oznaczają połączenia drogowe.

```
A ==== B ==== C
||             |
D ==== E ==== F
||     |      ||
G ==== H      I
```

gdzie podwójne kreski (= oraz ||) oznaczają sieć wodociągową.

Program powinien umożliwić:

1. Wygenerowanie mapy miast o zadanych wymiarach n x m , wraz z losowo wybranymi drogami tworzącymi sieć ortogonalną tzn., dopuszczalne są wyłącznie połączenia prostopadłe pomiędzy miastami w sąsiednich wierszach/kolumnach.
2. Sprawdzenie spójności sieci.
3. Automatyczne wygenerowanie połączeń wodociągowych na podstawie połączeń drogowych.
4. Wypisanie mapy sieci wodociągowej.

Uwaga: do sprawdzenia spójności jak i do wyznaczenia sieci wodociągów należy wykorzystać algorytm przeszukiwania wszerz (**BFS**)

Uwaga: Nie stosuj biblioteki STL

Zadanie 11. Drzewo genealogiczne

Napisz program umożliwiający tworzenie, edycję i wyświetlanie drzewa genealogicznego.

Program powinien realizować następujące funkcje:

1. Dodawanie, usuwanie, edycja węzłów reprezentujących pary (osoby).
2. Dodawanie małżonka osoby o podanym imieniu oraz dziecka pary.
3. Drukowanie całego drzewa w postaci zbliżonej do podanej w poniższym przykładzie.
4. Drukowanie imion dzieci lub wnuków.

Dla uproszczenia przyjmij że imiona są unikalne.

Przykład:

```
Jan  + Maria
  |
   --- Andrzej
  |
   --- Zenon  +  Anna
         |
          --- Ewa
         |
          --- Marek
  |
   --- Teresa
```

Do reprezentacji drzewa zastosuj następującą strukturę danych: