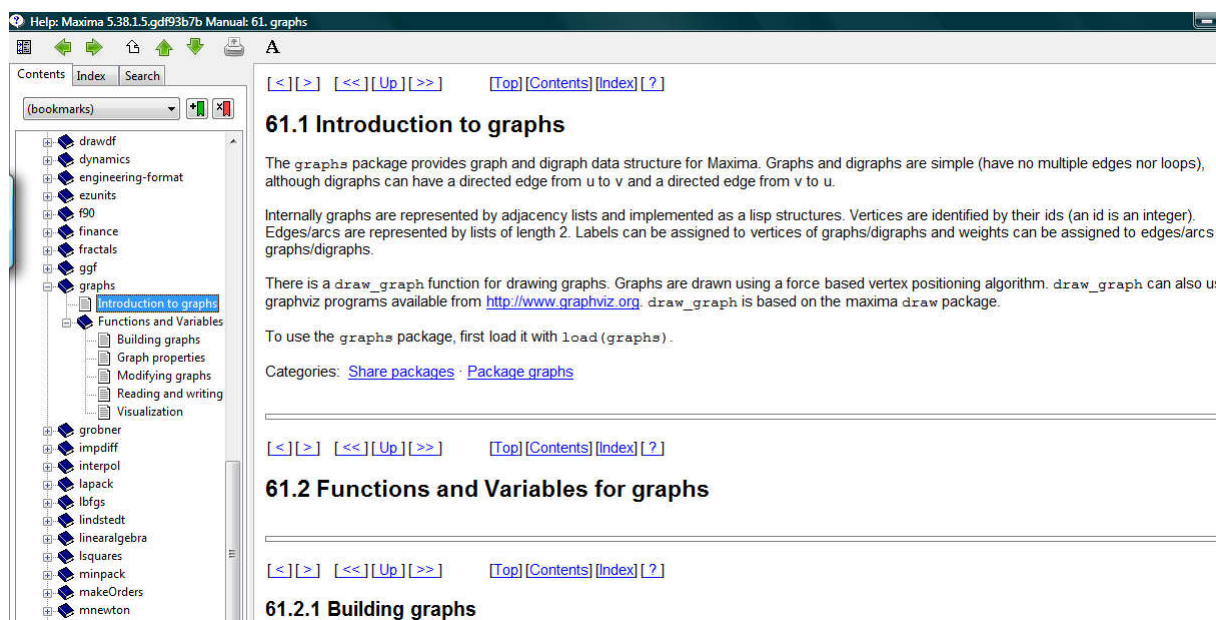


## Ćwiczenie lab. Badanie wybranych właściwości drzew

Celem tego ćwiczenia jest poznanie podstawowych właściwości grafów spójnych bez cykli, zwanych drzewami. Jeżeli drzewo jest  $n$ -wierzchołkowe, to na pewno posiada  $n-1$  krawędzi, które łączą jego wierzchołki w spójną całość i krawędzie nie tworzą żadnych cykli.

Każda krawędź drzewa jest mostem, tzn. usunięcie jakiegokolwiek krawędzi rozspaja drzewo, tworząc dwa mniejsze drzewa. Pojedynczy wierzchołek traktowany jest jako drzewo, a powstaje on wtedy, kiedy jako most usuwa się krawędź sąsiadującą z wierzchołkiem stopnia 1. Dodanie dowolnej krawędzi do drzewa powoduje powstanie dokładnie jednego cyklu. Między dowolną parą wierzchołków w drzewie istnieje dokładnie jedna droga.

Uruchomić program Maxima i wczytać w nim bibliotekę grafową instrukcją `load(graphs)`. Wczytać także bibliotekę kombinatoryczno-grafową w wersji 2.7 lub wyższej. W systemie pomocy programu Maxima zapoznać się z funkcjami biblioteki grafowej, przeznaczonymi do tworzenia grafów, badania ich właściwości, modyfikowania oraz wizualizacji.



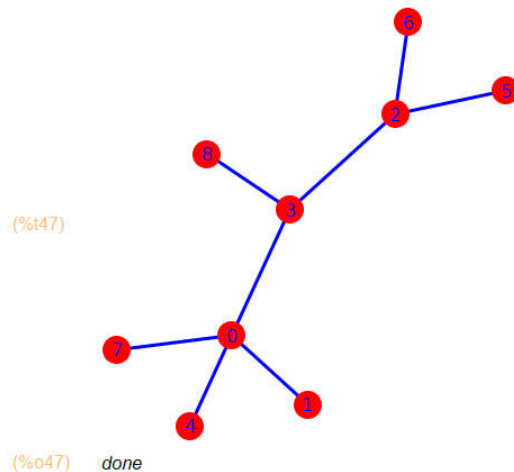
### 1. Wyznaczanie kodu Prüfera dla drzewa

Funkcją `random_tree()` wylosować drzewo nieskierowane o liczbie wierzchołków większej niż 7 i narysować je funkcją `draw_graph()`, wraz z numerami wierzchołków. Wyznaczyć na kartce papieru kod Prüfera tego drzewa pamiętając, że każde drzewo oznakowane (zaetykietowane) posiada kod zbudowany z  $n-2$  liczb, gdzie  $n$  to liczba wierzchołków drzewa. Zauważyć, że numer wierzchołka występuje w kodzie Prüfera o 1 mniej razy niż wynosi stopień danego wierzchołka.

Żeby wyznaczyć kod Prüfera dla drzewa, stosujemy następujący algorytm:

- szukamy wierzchołka stopnia 1, czyli liścia, o najmniejszym numerze,
- numer wierzchołka, do którego prowadzi krawędź z tego liścia zapisujemy do kodu,
- usuwamy z drzewa liść o najmniejszym numerze (wraz z jego krawędzią),
- jeżeli w drzewie została tylko jedna krawędź (dwa wierzchołki), wówczas algorytm kończy działanie, w przeciwnym wypadku przechodzimy do punktu a).

```
(%i47) n : 9$
drzewo : random_tree(n);
draw_graph(drzewo, show_id=true, vertex_size=4, edge_width=3, edge_color=blue);
(drzewo) GRAPH(9 vertices, 8 edges)
```



Jaki jest związek sekwencji stopni wierzchołków drzewa z jego kodem Prüfera? Sekwencję stopni wierzchołków grafu można wyznaczyć za pomocą funkcji `degree_sequence(gr)`.

**Uwaga!** Jeżeli funkcja `draw_graph()` nie działa poprawnie (nie rysuje grafu, tylko informuje o błędzie), wówczas w jej opcjach należy dopisać `vertex_color = red` (albo inny kolor). Tak można „oszukać” funkcję `draw_graph()`. Problem ten występuje w starszych wersjach Maximy.

Biblioteka grafowa `graphs` nie zawiera funkcji wyznaczania kodu Prüfera. Biblioteka kombinatoryczno-grafowa ma taką funkcję. Trzeba pamiętać, że wierzchołki grafu w funkcjach biblioteki kombinatoryczno-grafowej numeruje się od 1, a nie od 0, jak w bibliotece `graphs`. Ponadto w bibliotece k.-g. graf definiuje się za pomocą tylko listy krawędzi. Listy wierzchołków nie podaje się. W związku z tym wyznaczenie kodu Prüfera, w zależności od tego, czy najmniejszy numer wierzchołka w drzewie wynosi 0 czy 1, będzie wyglądać następująco:

```
(%i77) Drzewo : edges(drzewo);
kodP : prufer_code_for_tree(Drzewo,0);
Drzewo : edges(drzewo)+1;
kodP : prufer_code_for_tree(Drzewo,1);

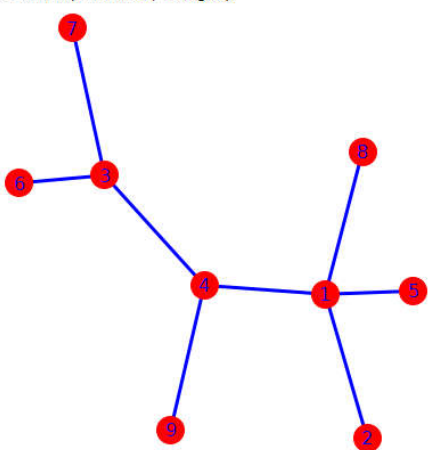
(Drzewo) [[2,6],[0,7],[2,5],[0,4],[0,1],[2,3],[3,8],[0,3]]
(kodP) [0,0,2,2,3,0,3]
(Drzewo) [[3,7],[1,8],[3,6],[1,5],[1,2],[3,4],[4,9],[1,4]]
(kodP) [1,1,3,3,4,1,4]
```

```
Drzewo : edges(drzewo);
kodP : prufer_code_for_tree(Drzewo,0);
Drzewo : edges(drzewo)+1;
kodP : prufer_code_for_tree(Drzewo,1);
```

Aby narysować badane drzewo z numeracją wierzchołków od 1, należy utworzyć nową definicję drzewa za pomocą funkcji `create_graph()`. Pobiera ona dwa zasadnicze argumenty: listę wierzchołków i listę krawędzi (każda krawędź to lista długości 2). Listy te łatwo obliczyć, wystarczy do list wierzchołków i krawędzi grafu z numeracją wierzchołków od 0 dodać 1. Wtedy wszystkie liczby na tych listach zwiększą się o 1. Ciąg instrukcji prowadzących do narysowania drzewa z numeracją wierzchołków od 1 może mieć postać:

```
(%i87) wierzcholki : vertices(drzewo)+1;
krawedzie : edges(drzewo)+1;
drzewo2 : create_graph(wierzcholki, krawedzie, directed=false);
draw_graph(drzewo2, show_id=true, vertex_size=4, edge_width=3, edge_color=blue);

(wierzcholki) [9,8,7,6,5,4,3,2,1]
(krawedzie) [[3,7],[1,8],[3,6],[1,5],[1,2],[3,4],[4,9],[1,4]]
(drzewo2) GRAPH(9 vertices, 8 edges)
```



```
(%o87) done
```

Jak już wcześniej obliczono, kod Prüfera dla tego drzewa to ciąg [1, 1, 3, 3, 4, 1, 4].

```
wierzcholki : vertices(drzewo)+1;
krawedzie : edges(drzewo)+1;
drzewo2 : create_graph(wierzcholki, krawedzie, directed=false);
draw_graph(drzewo2, show_id=true, vertex_size=4, edge_width=3, edge_color=blue);
```

## 2. Wyznaczanie drzewa z kodu Prüfera

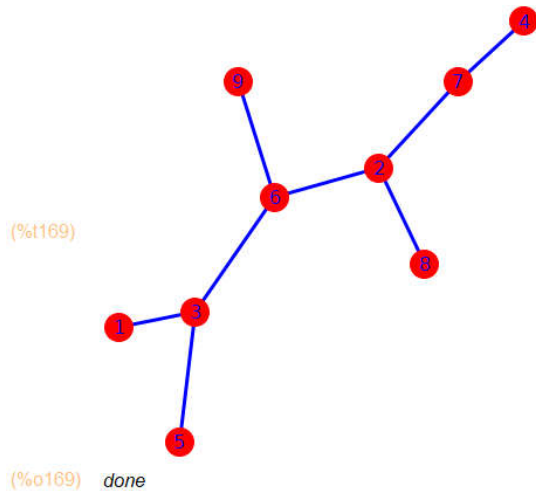
Skopiować podane dalej instrukcje do Maximy i wylosować 7-liczbowy kod Prüfera, utworzony z liczb ze zbioru { 1, ..., 9 }. To, czy liczba wystąpi w kodzie i ile razy, zależy od wyników losowania za pomocą funkcji `random()`. Jak widać, zmienna `ile_liczb` przechowuje ilość liczb w kodzie. Liczba wierzchołków drzewa będzie o 2 większa, czyli wyniesie  $7 + 2 = 9$ . Stosując algorytm poznany na wykładzie, wyznaczyć drzewo odpowiadające wylosowanemu kodowi.

Funkcja biblioteki kombinatoryczno-grafowej, przeznaczona do wyznaczania drzewa z kodu, nazywa się `tree_from_prufer_code(kod_Prüfera, v_start, czy_dac_wagi)`. Wyznaczyć drzewo z kodu Prüfera przy użyciu tej funkcji. Stosując funkcje `create_graph()` oraz `draw_graph()` utworzyć i narysować to drzewo wraz z numerami wierzchołków w Maximie. Poeksperymentować z opcjami funkcji `draw_graph()`. Spróbować samodzielnie zaprogramować w Maximie algorytm odkodowywania drzewa. Podać przykład kodu Prüfera o długości 7, który jest nieprawidłowy i wyjaśnić, dlaczego jest niepoprawny?

```
ile_liczb : 7$
kodPruf : makelist(1+random(ile_liczb+2), i, 1, ile_liczb);
krawedzie_drzewa : tree_from_prufer_code(kodPruf,1, false);
wierzcholki_drzewa : unique(flatten(krawedzie_drzewa));
drzewko : create_graph(wierzcholki_drzewa, krawedzie_drzewa);
draw_graph(drzewko, show_id=true, vertex_size=4, edge_width=3, edge_color=blue);
```

```
(%i169) ile_liczb : 7$
kodPruf : makelist(1+random(ile_liczb+2), i, 1, ile_liczb);
krawedzie_drzewa : tree_from_prufer_code(kodPruf, 1, false);
wierzcholki_drzewa : unique(flatten(krawedzie_drzewa));
drzewko : create_graph(wierzcholki_drzewa, krawedzie_drzewa);
draw_graph(drzewko, show_id=true, vertex_size=4, edge_width=3, edge_color=blue);

(kodPruf) [3,7,3,6,2,2,6]
(krawedzie_drzewa) [[3,1],[7,4],[3,5],[6,3],[2,7],[2,8],[6,2],[6,9]]
(wierzcholki_drzewa) [1,2,3,4,5,6,7,8,9]
(drzewko) GRAPH(9 vertices, 8 edges)
```

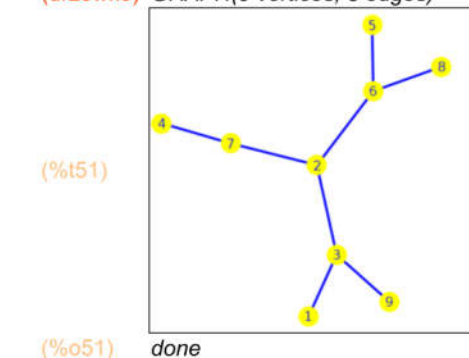


Funkcją `random_permutation()` dokonać permutacji wylosowanego kodu i wyznaczyć nowe drzewo.

```
NowyKodPruf : random_permutation(kodPruf);
krawedzie_drzewa : tree_from_prufer_code(NowyKodPruf, 1, false);
wierzcholki_drzewa : unique(flatten(krawedzie_drzewa));
drzewko : create_graph(wierzcholki_drzewa, krawedzie_drzewa);
draw_graph(drzewko, show_id=true, vertex_size=4, edge_width=3, edge_color=blue, vertex_color=yellow);
```

```
(%i51) NowyKodPruf : random_permutation(kodPruf);
krawedzie_drzewa : tree_from_prufer_code(NowyKodPruf, 1, false);
wierzcholki_drzewa : unique(flatten(krawedzie_drzewa));
drzewko : create_graph(wierzcholki_drzewa, krawedzie_drzewa);
draw_graph(drzewko, show_id=true, vertex_size=4, edge_width=3, edge_color=blue, vertex_color=yellow);

(NowyKodPruf) [3,7,6,2,6,2,3]
(krawedzie_drzewa) [[3,1],[7,4],[6,5],[2,7],[6,8],[2,6],[3,2],[3,9]]
(wierzcholki_drzewa) [1,2,3,4,5,6,7,8,9]
(drzewko) GRAPH(9 vertices, 8 edges)
```



Warto zauważyć, że permutacja liczb w kodzie nie zmienia stopni wierzchołków drzewa, tzn. w obu drzewach wierzchołki o tych samych numerach mają te same stopnie. Nie można jednak uznać, że te dwa drzewa oznakowane są izomorficzne (jednakowe). Widać to po listach krawędzi tych drzew. Nie są one identyczne (nawet przy zaniedbaniu kolejności krawędzi). Na przykład krawędź [3, 5]

nie występuje w drugim drzewie. Możliwe jest więc, że po zmianie kolejności liczb w kodzie zmieni się struktura drzewa, tak jak w tym przypadku. Od czego to zależy? O tym, że te dwie struktury są różne świadczy choćby fakt, że najdłuższe ścieżki w obu drzewach mają różne długości: 5 i 4. Narysować wszystkie możliwe różne struktury drzewa (czyli drzewa nieoznakowane) o takiej sekwencji stopni wierzchołków, jaka wynika z kodu Prüfera. Ile jest tych drzew? Czy da się jakoś obliczyć ich liczbę?

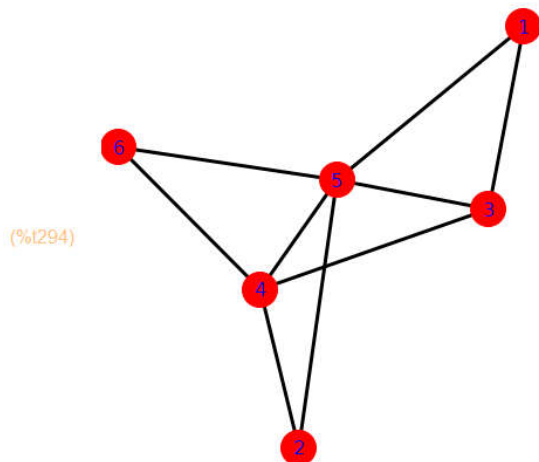
### 3. Wyznaczanie drzew rozpinających grafu nieskierowanego

Przepisać poniższe instrukcje do Maximy i wylosować graf na sześciu lub siedmiu wierzchołkach, z prawdopodobieństwem wystąpienia każdej krawędzi wynoszącym około 0.65. Wylosowany graf zmodyfikować tak, aby numery wierzchołków zaczynały się od 1, a nie od 0. Dla wygenerowanego grafu wyznaczyć na kartce papieru liczbę drzew rozpinających metodą macierzy Laplace'a oraz metodą graficzną ~~(jeśli była na wykładzie)~~. Następnie sprawdzić poprawność analitycznego rozwiązania w Maximie, wykorzystując funkcje `laplacian_matrix()` oraz `adjoint()`.

```
(%i293) do (  
    graf : random_graph(6, 0.65),  
    if is_connected(graf) then return(0)  
)$  
graf : create_graph(vertices(graf)+1, edges(graf)+1);  
(graf) GRAPH(6 vertices, 9 edges)
```

```
do (  
    graf : random_graph(6, 0.65),  
    if is_connected(graf) then return(0)  
)$  
graf : create_graph(vertices(graf)+1, edges(graf)+1);  
draw_graph(graf, show_id=true, vertex_size=5, edge_color=black, edge_width=3);
```

```
(%i294) draw_graph(graf, show_id=true, vertex_size=5, edge_color=black, edge_width=3);
```



```
print_graph(graf);  
macierz_sasiedztw : adjacency_matrix(graf);  
macierz_laplacea : laplacian_matrix(graf);  
macierz_dopelnien_algebraicznych : adjoint(macierz_laplacea);  
liczba_drzew_rozpinajacych : macierz_dopelnien_algebraicznych[1][1];
```

```
(%i308) print_graph(graf);
macierz_sasiedztw : adjacency_matrix(graf);
macierz_laplacea : laplacian_matrix(graf);
macierz_dopelnien_algebraicznych : adjoint(macierz_laplacea);
liczba_drzew_rozpinajacych : macierz_dopelnien_algebraicznych[1][1];
```

**Graph on 6 vertices with 9 edges.**

**Adjacencies:**

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

```
(%o304) done
```

```
(%o304) done
```

```
(macierz_sasiedztw)
[ 0 1 1 0 0 0 ]
[ 1 0 1 1 1 1 ]
[ 1 1 0 1 1 0 ]
[ 0 1 1 0 0 1 ]
[ 0 1 1 0 0 0 ]
[ 0 1 0 1 0 0 ]
```

```
(macierz_laplacea)
[ 2 -1 -1 0 0 0 ]
[ -1 5 -1 -1 -1 -1 ]
[ -1 -1 4 -1 -1 0 ]
[ 0 -1 -1 3 0 -1 ]
[ 0 -1 -1 0 2 0 ]
[ 0 -1 0 -1 0 2 ]
```

```
(macierz_dopelnien_algebraicznych)
[ 52 52 52 52 52 52 ]
[ 52 52 52 52 52 52 ]
[ 52 52 52 52 52 52 ]
[ 52 52 52 52 52 52 ]
[ 52 52 52 52 52 52 ]
[ 52 52 52 52 52 52 ]
```

```
(liczba_drzew_rozpinajacych)
52
```

Funkcją `all_spanning_ditrees(digraf, [vroot])` wygenerować wszystkie drzewa rozpinające badanego grafu, a następnie część lub wszystkie drzewa narysować funkcją `draw_graph()` jako wyróżnione krawędzie w badanym grafie. Funkcja `all_spanning_ditrees(digraf, [vroot])` wyznacza wszystkie skierowane drzewa, ale jeżeli graf jest nieskierowany i każda jego krawędź zostanie przekonwertowana na dwa przeciwnie skierowane łuki, to funkcja zwróci drzewa skierowane do lub od korzenia `vroot`, które można traktować jako drzewa nieskierowane. Przekonać się, że dla grafu nieskierowanego, przekonwertowanego na graf skierowany, bez względu na wybór wierzchołka `vroot`, zostaną wygenerowane te same zestawy drzew rozpinających. Zmieniać argument `vroot`, a także jego znak i sprawdzać, czy rzeczywiście liczba drzew i wyznaczone drzewa nie zależą od niego.

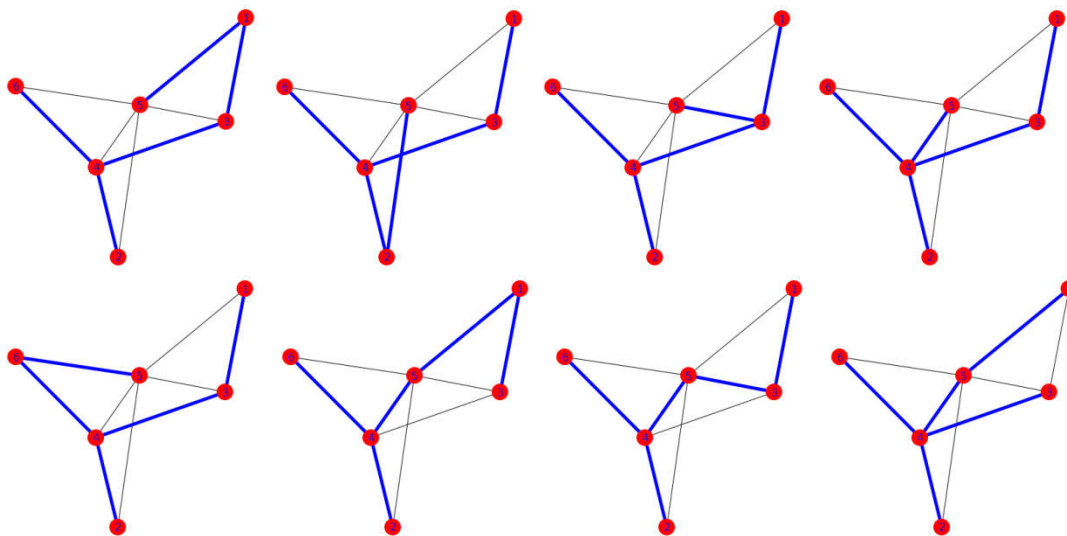
```
graf_z_kraw : edges(graf);
digraf : graph_to_digraph(graf_z_kraw);
vroot : 1;
drzewa : all_spanning_ditrees(digraf, vroot)$
ile : length(drzewa);
print_list(drzewa);
for i:1 thru 8 do
(
drze : copypaste(drzewa[i]),
draw_graph(graf, show_id=true, vertex_size=4, show_edges=drze, show_edge_width=5)
)$
```



```
(%i423) graf_z_kraw : edges(graf);
digraf : graph_to_digraph(graf_z_kraw);
vroot : 1;
drzewa : all_spanning_ditrees(digraf, vroot)$
ile : length(drzewa);
print_list(drzewa);
for i:1 thru 8 do
(
  drze : copylist(drzewa[i]),
  draw_graph(graf, show_id=true, vertex_size=4, show_edges=drze, show_edge_width=5)
)$
(graf_z_kraw) [[1,3],[1,5],[2,4],[2,5],[3,4],[3,5],[4,5],[4,6],[5,6]]
(digraf) [[1,3],[3,1],[1,5],[5,1],[2,4],[4,2],[2,5],[5,2],[3,4],[4,3],[3,5],[5,3],[4,5],[5,4],[4,6],[6,4],[5,6],[6,5]]
(vroot) 1
(ile) 52

[[6,5],[2,4],[3,1],[4,3],[5,4]]
[[6,4],[2,4],[3,1],[4,3],[5,1]] [[6,5],[2,4],[3,1],[4,5],[5,1]]
[[6,4],[2,4],[3,1],[4,3],[5,2]] [[6,5],[2,4],[3,1],[4,5],[5,3]]
[[6,4],[2,4],[3,1],[4,3],[5,3]] [[6,5],[2,4],[3,1],[4,6],[5,1]]
[[6,4],[2,4],[3,1],[4,3],[5,4]] [[6,5],[2,4],[3,1],[4,6],[5,3]]
[[6,4],[2,4],[3,1],[4,3],[5,6]] [[6,5],[2,4],[3,4],[4,5],[5,1]]
[[6,4],[2,4],[3,1],[4,5],[5,1]] [[6,5],[2,4],[3,4],[4,6],[5,1]]
[[6,4],[2,4],[3,1],[4,5],[5,3]] [[6,5],[2,4],[3,5],[4,3],[5,1]]
[[6,4],[2,4],[3,4],[4,5],[5,1]] [[6,5],[2,4],[3,5],[4,5],[5,1]]
[[6,4],[2,4],[3,5],[4,3],[5,1]] [[6,5],[2,4],[3,5],[4,6],[5,1]]
[[6,4],[2,4],[3,5],[4,5],[5,1]] [[6,5],[2,5],[3,1],[4,2],[5,1]]
[[6,4],[2,5],[3,1],[4,2],[5,1]] [[6,5],[2,5],[3,1],[4,2],[5,3]]
[[6,4],[2,5],[3,1],[4,2],[5,3]] [[6,5],[2,5],[3,1],[4,3],[5,1]]
[[6,4],[2,5],[3,1],[4,3],[5,1]] [[6,5],[2,5],[3,1],[4,3],[5,3]]
[[6,4],[2,5],[3,1],[4,3],[5,3]] [[6,5],[2,5],[3,1],[4,3],[5,4]]
[[6,4],[2,5],[3,1],[4,3],[5,4]] [[6,5],[2,5],[3,1],[4,5],[5,1]]
[[6,4],[2,5],[3,1],[4,3],[5,6]] [[6,5],[2,5],[3,1],[4,5],[5,3]]
[[6,4],[2,5],[3,1],[4,5],[5,1]] [[6,5],[2,5],[3,1],[4,6],[5,1]]
[[6,4],[2,5],[3,1],[4,5],[5,3]] [[6,5],[2,5],[3,1],[4,6],[5,3]]
[[6,4],[2,5],[3,4],[4,2],[5,1]] [[6,5],[2,5],[3,4],[4,2],[5,1]]
[[6,4],[2,5],[3,4],[4,5],[5,1]] [[6,5],[2,5],[3,4],[4,5],[5,1]]
[[6,4],[2,5],[3,5],[4,2],[5,1]] [[6,5],[2,5],[3,4],[4,6],[5,1]]
[[6,4],[2,5],[3,5],[4,3],[5,1]] [[6,5],[2,5],[3,5],[4,2],[5,1]]
[[6,4],[2,5],[3,5],[4,5],[5,1]] [[6,5],[2,5],[3,5],[4,3],[5,1]]
[[6,5],[2,4],[3,1],[4,3],[5,1]] [[6,5],[2,5],[3,5],[4,5],[5,1]]
[[6,5],[2,4],[3,1],[4,3],[5,2]] [[6,5],[2,5],[3,5],[4,6],[5,1]]
[[6,5],[2,4],[3,1],[4,3],[5,3]] (%o422) done
```

Zmodyfikować skrypt (%i423) tak, aby zamiast funkcji `all_spanning_ditrees()` można było użyć funkcji `all_spanning_trees()`. Trzeba pamiętać, że ta druga funkcja wymaga podania grafu nieskierowanego, tzn. nie należy konwertować krawędzi na łuki. Najlepiej zmodyfikować ten skrypt, pracując na jego kopii. Dopisać instrukcje, które wykażą, że wygenerowane listy drzew w obu przypadkach będą identyczne. Najpierw pewnie trzeba będzie posortować te listy, a być może i konieczne będzie posortowanie krawędzi w listach. Ewentualnie można zamienić listy na zbiory, aby pozbyć się problemu kolejności wierzchołków w listach 2-elementowych.



Wygenerować kody Prüfera dla wszystkich drzew rozpinających i przeanalizować je.

```
(%i431) makelist(prufer_code_for_tree(Drz), Drz, drzewa);
(%o431) [[4,1,3,4],[3,4,2,4],[3,4,3,4],[3,4,4,4],[3,4,4,6],[4,1,5,4],[3,4,5,4],[5,4,4,4],[5,4,3,4],[5,4,5,4],[1,5,2,4],[3,5,2,4],[5,1,3,4],[3,5,3,4],[3,5,4,4],[3,5,4,6],[5,1,5,4],[3,5,5,4],[5,4,2,4],[5,5,4,4],[5,5,2,4],[5,5,3,4],[5,5,5,4],[4,3,1,5],[3,4,2,5],[3,4,3,5],[3,4,4,5],[4,1,5,5],[3,4,5,5],[4,1,5,6],[3,4,5,6],[5,4,4,5],[5,4,4,6],[5,4,3,5],[5,4,5,5],[5,4,5,6],[1,5,2,5],[3,5,2,5],[5,3,1,5],[3,5,3,5],[3,5,4,5],[5,1,5,5],[3,5,5,5],[5,1,5,6],[3,5,5,6],[5,4,2,5],[5,5,4,5],[5,5,4,6],[5,5,2,5],[5,5,3,5],[5,5,5,5],[5,5,5,6]]
```

Narysować na kartce papieru wszystkie nieoznakowane drzewa  $n$ -wierzchołkowe (tu  $n=6$ ), gdzie  $n$  to liczba wierzchołków w każdym drzewie rozpinającym. Narysować tylko te drzewa nieoznakowane, dla których istnieje co najmniej jedno oznakowane drzewo rozpinające w badanym grafie. Czy wystąpiły wszystkie nieoznakowane drzewa  $n$ -wierzchołkowe? Ile i których nie ma?

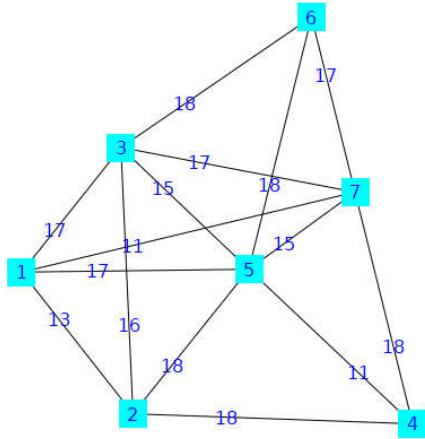
Wygenerować wszystkie drzewa rozpinające grafu pełnego o 4 wierzchołkach. Graf pełny zdefiniować funkcją `complete_graph(4)` z biblioteki `graphs`, tzn. zastąpić losowanie grafu podaniem konkretnego grafu instrukcją `graf : complete_graph(4)`. Sprawdzić, czy liczba drzew rozpinających w grafie pełnym jest zgodna z tw. Cayleya-Hamiltona, tzn. czy wynosi  $n^{n-2}$ , gdzie  $n$  to liczba wierzchołków grafu. Zmieniać liczbę wierzchołków grafu pełnego od 3 do 6 i obserwować liczbę drzew zwróconą przez funkcję `adjoint()`. Obejrzyć rysunki drzew.

#### 4. Wyznaczanie minimalnego drzewa rozpinającego w grafie ważonym

Przepisać, podany na następnej stronie, kod do Maximy i wylosować graf ważony dla tego zadania. Na kartce papieru wyznaczyć minimalne drzewo rozpinające tego grafu na dwa sposoby, algorytmem Prima i Kruskala. W Maxymie, za pomocą funkcji `minimum_spanning_tree_kruskal(graf_wazony, [pokaz])` oraz `minimum_spanning_tree_prim(graf_wazony, [zmienne])`, wyznaczyć minimalne drzewa rozpinające, porównać je ze sobą oraz z drzewem z obliczeń ręcznych. Powinno wyjść jedno i to samo drzewo, chociaż może się zdarzyć, szczególnie wtedy, kiedy funkcja wagowa nie jest różnowartościowa, że wyjdą różne drzewa, jednak na pewno będą miały taką samą sumę wag krawędzi. Wykonać również obliczenia za pomocą funkcji `minimum_spanning_tree()` z biblioteki `graphs`, która zwraca listę krawędzi drzewa jako obiekt typu `graph`. Narysować minimalne drzewo rozpinające w grafie, tzn. narysować graf tak, aby krawędzie drzewa były wyróżnione (kolorem i grubością). Pokazać wagi krawędzi i numery wierzchołków. Użyć odpowiednich opcji funkcji `draw_graph()`! Na koniec zbadać wpływ wierzchołka, od którego rozpoczyna się algorytm Prima, na kolejne kroki tego algorytmu i na efekt końcowy, czyli mst. Czy zawsze wychodzi to samo drzewo?



```
do
(
  graf_wazony : random_graph(7, 0.6),
  wierzcholki_wazonego : vertices(graf_wazony)+1,
  krawedzie_wazonego : edges(graf_wazony)+1,
  graf_wazony : create_graph(wierzcholki_wazonego, krawedzie_wazonego),
  for kraw in krawedzie_wazonego do set_edge_weight(kraw, random(10)+10, graf_wazony),
  if is_connected(graf_wazony) then return(0)
)$
draw_graph(graf_wazony, show_id=true, vertex_size=4, show_weight=true, vertex_type=filled_square, vertex_color=cyan);
```



done

```
do
(
  graf_wazony : random_graph(7, 0.6),
  wierzcholki_wazonego : vertices(graf_wazony)+1,
  krawedzie_wazonego : edges(graf_wazony)+1,
  graf_wazony : create_graph(wierzcholki_wazonego, krawedzie_wazonego),
  for kraw in krawedzie_wazonego do set_edge_weight(kraw, random(10)+10, graf_wazony),
  if is_connected(graf_wazony) then return(0)
)$
draw_graph(graf_wazony, show_id=true, vertex_size=4, show_weight=true, vertex_type=filled_square, vertex_color=cyan);
```

Wyznaczenie minimalnego drzewa rozpinającego za pomocą algorytmów Prima i Kruskala:

```
(%i541) krawedzie_wazonego : makelist([kra, get_edge_weight(kra, graf_wazony)], kra, edges(graf_wazony));
mst_prim : minimum_spanning_tree_prim(krawedzie_wazonego, 1, pokaz);
mst_kruskal : minimum_spanning_tree_kruskal(krawedzie_wazonego, pokaz);

(krawedzie_wazonego) [[1,2],13],[1,3],17],[1,5],17],[1,7],11],[2,3],16],[2,4],18],[2,5],18],[3,5],15],[3,6],18],[3,7],17],[4,5],11],[4,7],18],
],[5,6],18],[5,7],15],[6,7],17]]

Zbiór S      Zbiór V\S      Przekrój (S, V\S)      Krawędź lekka
-----
[1]          [2,3,4,5,6,7]          [[1,2],13],[1,3],17],[1,5],17],[1,7],11]      [[1,7],11]
[1,7]        [2,3,4,5,6]          [[1,2],13],[1,3],17],[1,5],17],[7,3],17],[7,4],18],[7,5],15],[7,6],17]      [[1,2],13]
[1,7,2]      [3,4,5,6]          [[1,3],17],[1,5],17],[7,3],17],[7,4],18],[7,5],15],[7,6],17],[2,3],16],[2,4],18],[2,5],18]      [[7,5],15]
[1,7,2,5]    [3,4,6]          [[1,3],17],[7,3],17],[7,4],18],[7,6],17],[2,3],16],[2,4],18],[5,3],15],[5,4],11],[5,6],18]      [[5,4],11]
[1,7,2,5,4]  [3,6]          [[1,3],17],[7,3],17],[7,6],17],[2,3],16],[5,3],15],[5,6],18]      [[5,3],15]
[1,7,2,5,4,3] [6]          [[7,6],17],[5,6],18],[3,6],18]      [[7,6],17]

(mst_prim) [[1,7],11],[1,2],13],[7,5],15],[5,4],11],[5,3],15],[7,6],17],82]
```

Krawędzie grafu posortowane rosnąco (niemalejąco) względem wag:

```
[[1,7],11],[4,5],11],[1,2],13],[3,5],15],[5,7],15],[2,3],16],[1,3],17],[1,5],17],[3,7],17],[6,7],17],[2,4],18],[2,5],18],[3,6],18],[4,7],18],[5,6],18]]
```

Krawędzie drzewa	Zbiory wierzchołków
-----	[[1],[2],[3],[4],[5],[6],[7]]
[[1,7],11]	[[1,7],[2],[3],[4],[5],[6]]
[[4,5],11]	[[1,7],[2],[3],[4,5],[6]]
[[1,2],13]	[[1,7,2],[3],[4,5],[6]]
[[3,5],15]	[[1,7,2],[3,4,5],[6]]
[[5,7],15]	[[1,7,2,3,4,5],[6]]
[[6,7],17]	[[1,7,2,3,4,5,6]]

```
(mst_kruskal) [[1,7],11],[4,5],11],[1,2],13],[3,5],15],[5,7],15],[6,7],17],82]
```

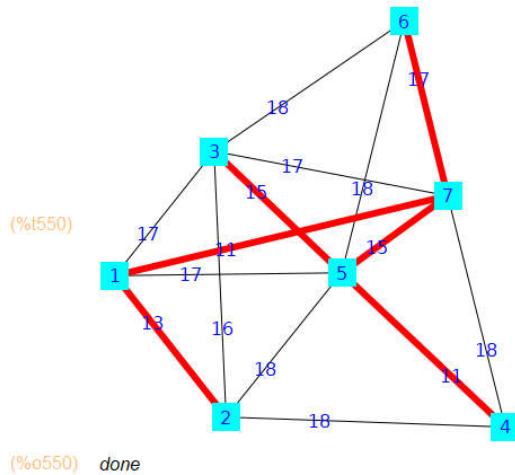
```
krawedzie_wazonego : makelist([kra, get_edge_weight(kra, graf_wazony)], kra, edges(graf_wazony) );
mst_prim : minimum_spanning_tree_prim(krawedzie_wazonego, 1, pokaz);
mst_kruskal : minimum_spanning_tree_kruskal(krawedzie_wazonego, pokaz);
```

```
(%i559) Tree : minimum_spanning_tree(graf_wazony); edges(Tree);
```

```
(Tree) GRAPH(7 vertices, 6 edges)
```

```
(%o559) [[6,7],[3,5],[5,7],[1,2],[1,7],[4,5]]
```

```
(%i550) draw_graph(graf_wazony, show_id=true, vertex_size=4, show_weight=true, vertex_type=filled_square, vertex_color=cyan,
show_edges=makelist(kraw[1], kraw, mst_prim[1]), show_edge_color=red, show_edge_width=6);
```



```
draw_graph(graf_wazony, show_id=true, vertex_size=4, show_weight=true, vertex_type=filled_square, vertex_color=cyan,
show_edges=makelist(kraw[1], kraw, mst_prim[1]), show_edge_color=red, show_edge_width=6);
```

W pewnych zastosowaniach może być potrzebne maksymalne drzewo rozpinające badanego grafu. Jak je wyznaczyć? Co zmodyfikować w grafie i w kodzie dla Maximy i jak, żeby to drzewo maksymalne wyznaczyć? Czy zanegowanie wag wszystkich krawędzi grafu wystarczy? A może trzeba jeszcze dodać do tych zanegowanych wag największą wagę dodatnią (i jeszcze +1)? Wtedy wszystkie wagi będą dodatnie. Czy znalezione drzewo minimalne w tym zmodyfikowanym grafie będzie drzewem maksymalnym w grafie oryginalnym?

Opracować zestaw instrukcji dla Maximy, który wyznaczy wszystkie ważone drzewa rozpinające badanego grafu i posortuje je rosnąco według wag (waga drzewa to suma wag krawędzi tworzących to drzewo). Można np. najpierw wygenerować wszystkie drzewa nieważone funkcją `all_spanning_ditrees()`, a następnie dla każdego drzewa z listy obliczyć sumę wag krawędzi i „doczepić” tę wagę do drzewa umieszczając każde drzewo w nowej liście, w której oprócz krawędzi drzewa znajdzie się także waga drzewa. Następnie funkcją `sort()` trzeba posortować tak powstałą listę. I w końcu odczytać pierwszy i ostatni elementy tej listy, w celu wyznaczenia drzewa minimalnego i maksymalnego.

Przykładowe rozwiązanie tego problemu przedstawiono na następnej stronie. Przeanalizować uważnie poszczególne instrukcje. Przyspieszyć działanie tego kodu poprzez użycie kwadratowej macierzy wag dla grafu ważonego i zrezygnowanie `m.in.` z funkcji `sublist_indices()`. Przyspieszyć kod poprzez rezygnację z sortowania wszystkich drzew, tzn. znaleźć na liście tylko drzewo maksymalne oraz minimalne. Jeszcze raz zmodyfikować omawiany skrypt tak, aby zamiast `all_spanning_ditrees()` można było użyć funkcji `all_spanning_trees()`.

```
(%i784) i:1$ suma:0$
krawedzie_wazonego;
krawedzie_bez_wag : makelist(kraw[1], kraw, krawedzie_wazonego);
wagi : makelist(kraw[2], kraw, krawedzie_wazonego);
wszystkie_drzewa : all_spanning_ditrees(graph_to_digraph(krawedzie_bez_wag),1)$
liczba_drzew : length(wszytkie_drzewa);
liczba_drzew;
for i:1 thru liczba_drzew do
(
d : copylst(wszytkie_drzewa[i]), suma : 0,
for kra in d do
suma : suma+wagi[ sublist_indices(krawedzie_bez_wag, lambda([a], is(setify(a)=setify(kra))))[1] ],
wszytkie_drzewa[i] : [d, suma]
)$
posortowane : sort(wszytkie_drzewa, lambda([a,b], a[2]<b[2]))$
first(posortowane);
last(posortowane);
```

```
i:1$ suma:0$
krawedzie_wazonego;
krawedzie_bez_wag : makelist(kraw[1], kraw, krawedzie_wazonego);
wagi : makelist(kraw[2], kraw, krawedzie_wazonego);
wszystkie_drzewa : all_spanning_ditrees(graph_to_digraph(krawedzie_bez_wag),1)$
liczba_drzew : length(wszytkie_drzewa);
liczba_drzew;
for i:1 thru liczba_drzew do
(
d : copylst(wszytkie_drzewa[i]), suma : 0,
for kra in d do
suma : suma+wagi[ sublist_indices(krawedzie_bez_wag, lambda([a], is(setify(a)=setify(kra))))[1] ],
wszytkie_drzewa[i] : [d, suma]
)$
posortowane : sort(wszytkie_drzewa, lambda([a,b], a[2]<b[2]))$
first(posortowane);
last(posortowane);
```

```
(%o775) [[[1,2],13],[[1,3],17],[[1,5],17],[[1,7],11],[[2,3],16],[[2,4],18],[[2,5],18],[[3,5],15],[[3,6],18],[[3,7],17],[[4,5],11],[[4,7],18],
[[5,6],18],[[5,7],15],[[6,7],17]]
(krawedzie_bez_wag) [[1,2],[1,3],[1,5],[1,7],[2,3],[2,4],[2,5],[3,5],[3,6],[3,7],[4,5],[4,7],[5,6],[5,7],[6,7]]
(wagi) [13,17,17,11,16,18,18,15,18,17,11,18,18,15,17]
(liczba_drzew) 1543
(%o780) 1543
(%o783) [[[7,1],[2,1],[3,5],[4,5],[5,7],[6,7]],82]
(%o784) [[[7,4],[2,5],[3,6],[4,2],[5,1],[6,5]],107]
```