# Matematyka dyskretna 1 – laboratorium komputerowe

## Ćwiczenie lab. nr 1. **Badanie wybranych właściwości permutacji**

Uruchomić program Maxima (w odpowiedniej wersji) i wczytać w nim najnowszą bibliotekę kombinatoryczno-grafową (pakiet skompilowanych procedur i funkcji, zawartych w pliku binarnym o nazwie permutacje.fasl), zgodnie ze wskazówkami podanymi na końcowych stronach pliku PDF, dokumentującego funkcje zaimplementowane w bibliotece. Plik permutacje.fasl, wraz z opisem biblioteki (plik PDF), pobrać jako jeden plik zip ze strony pei.prz.edu.pl, klikając zakładkę Dydaktyka --> Materiały dla studentów, i po zalogowaniu się (login: pei, hasło: d…) » 1 EF-DI » Matematyka Dyskretna 1 » Lato2021 » LaboratoriumKomputerowe. Zapoznać się z komputerowymi sposobami zapisu permutacji i podstawowymi zasadami wykonywania obliczeń w programie Maxima. Zasadniczo, każde zadanie należy najpierw rozwiązać „ręcznie" na kartce papieru, a następnie na komputerze. Oba podejścia muszą dać ten sam wynik. Do ładnego wypisania listy permutacji lub innej listy można użyć funkcji print_list() albo print_list2(). Przez ładne wypisanie elementów listy rozumie się tutaj wypisanie ich jeden pod drugim, a nie jeden za drugim.

**Zad. 0.** Dana jest losowa permutacja f należąca do $S_6$, $S_7$ albo $S_8$, wylosowana za pomocą funkcji p_random(n) dla n = 6, 7 albo 8. Przedstawić (zapisać lub narysować) tę permutację we wszystkich siedmiu postaciach: na diagramie Venna, w postaci funkcyjnej, 1-wierszowej, 2-wierszowej, cyklowej, grafowej i macierzowej. Korzystając z funkcji konwersji (cycles2perm() i innych) biblioteki kombinatoryczno-grafowej, wykonać odpowiednie obliczenia na komputerze. Nie wszystkie wymienione sposoby zapisu permutacji są zaimplementowane, tzn. diagram Venna, postać funkcyjna, 2-wierszowa oraz grafowa nie są. **Uwaga:** losowa permutacja, wyznaczona przez funkcję p_random(n), jest zwracana w zapisie cyklowym. Przykład losowania permutacji należącej do $S_8$:

```
(%i82)  n : 8$
        f : p_random(n);
(f)     [[1,2,4],[5,7,6,8]]
```

**Zad. 1.** Dla losowej permutacji p, wygenerowanej funkcją p_random(n), dla n = 8, 9 albo 10, zapisanej 1-wierszowo albo cyklowo, wykonać wymienione niżej polecenia. Potwierdzić rozwiązanie ręczne odpowiednią funkcją z biblioteki kombinatoryczno-grafowej.

a) wyznaczyć Sn, do którego należy p, czyli wyznaczyć n
b) wyznaczyć punkty stałe permutacji p
c) wyznaczyć permutację odwrotną do p
d) wyznaczyć permutację, której postać 1-wierszowa powstała z zapisania od końca postaci 1-wierszowej permutacji p; porównać zapisy cyklowe obu permutacji
e) wyznaczyć typ permutacji p
f) wyznaczyć rząd permutacji p
g) zapisać permutację p w kanonicznej postaci cyklowej
h) cyklicznie przesunąć permutację p w lewo lub w prawo (np. o 3 elementy)
i) wyznaczyć liczbę wszystkich cykli w permutacji p
j) wyznaczyć liczbę cykli parzystej długości w permutacji p
k) wyznaczyć liczbę cykli nieparzystej długości w permutacji p
l) wyznaczyć wszystkie inwersje w permutacji p

m) wyznaczyć liczbę inwersji w permutacji p

n) wyznaczyć wektor inwersyjny permutacji p

o) wyznaczyć znak permutacji p na 5 sposobów (komputerowo tylko jedną funkcją):

$\mathrm{sgn}(p) = (-1)^{\mathrm{I}(p)}$

$\mathrm{sgn}(p) = (-1)^{n \pm \mathrm{c}(p)}$

$\mathrm{sgn}(p) = (-1)^{\mathrm{cpd}(p)}$

$\mathrm{sgn}(p) = (-1)^{\mathrm{T}(p)}$

$\mathrm{sgn}(p) = \det(\mathbf{A})$

gdzie:

$n$ to numer $S_n$, do którego należy permutacja $p$,

$\mathrm{I}(p)$ to liczba inwersji w permutacji $p$,

$\mathrm{c}(p)$ to liczba wszystkich cykli w permutacji $p$ zapisanej w postaci złożenia rozłącznych cykli, z uwzględnieniem cykli długości 1, nawet tych pominiętych,

$\mathrm{cpd}(p)$ to liczba cykli parzystej długości w permutacji $p$, zapisanej w postaci złożenia rozłącznych cykli (cykle długości 1 mogą być pominięte, bo mają nieparzystą dł.), (okazuje się, że ten sposób obliczania znaku permutacji działa nawet wówczas, gdy permutacja $p$ jest zapisana w postaci zależnych (nierozłącznych) cykli),

$\mathrm{T}(p)$ to liczba transpozycji w dowolnym przedstawieniu permutacji $p$ w postaci złożenia transpozycji (sąsiadujących lub nie sąsiadujących elementów); przypomnijmy, że jeżeli w pewnym przedstawieniu permutacji $p$ w postaci złożenia transpozycji występuje parzysta (nieparzysta) liczba transpozycji, to w każdym innym przedstawieniu liczba transpozycji również będzie parzysta (nieparzysta); minimalna liczba transpozycji, potrzebna do zapisania permutacji w postaci złożenia transpozycji (nie sąsiadujących elementów), wynosi $n - c(p)$,

$\mathbf{A}$ to macierz kwadratowa $n$ x $n$ z „wymijającymi się" jedynkami, odpowiadająca permutacji $p$, tzn. taka, w której, dla $i = 1, 2, \cdots, n$, na przecięciu $i$-tego wiersza oraz $p(i)$-tej kolumny są jedynki, i zera w pozostałych komórkach. Istnieje druga (transponowana) wersja macierzy $\mathbf{A}$, w której jedynki są na przecięciu $p(i)$-tego wiersza oraz $i$-tej kolumny. Przy takim kodowaniu permutacji, operacji składania permutacji odpowiada mnożenie macierzy kodujących składane permutacje, w tej samej kolejności, w jakiej permutacje są składane.

p) wylosować permutację q należącą do $S_5$ albo $S_6$ i dla tej permutacji wyznaczyć wszystkie możliwe (równoważne) zapisy w postaci złożenia rozłącznych cykli funkcją p_all_cyclic_notations(q). Użyć skryptu z następnej strony. Wyjaśnić, dlaczego otrzymane zapisy są równoważne. Zauważyć, że takie operacje na rozłącznych cyklach, jak: zamiana cykli miejscami, cykliczne przesuwanie liczb w cyklach bądź pomijanie cykli długości 1, nie zmieniają permutacji, tzn. zapis 1-wierszowy rozważanej permutacji nie ulega zmianie.

Aby przekonać się, że wszystkie notacje są równoważne, można użyć instrukcji:
**unikalne_q : perm2cycles(unique(map(cycles2perm, wszystkie_notacje))[1]);**

Jeżeli unikalne_q jest taką samą permutacją jak q, wtedy notacje są równoważne.
Skopiować tę instrukcję do Maximy i wykonać. Wyjaśnić, co ona dokładnie robi.

Inny sposób przekonania się, że wszystkie notacje są równoważne, może być taki:
**makelist( is_equal(q, perm), perm, wszystkie_notacje );**

Jeżeli otrzymana lista zawiera same true, wtedy wszystkie notacje są równoważne.
Skopiować tę instrukcję do Maximy i wykonać. Wyjaśnić, co ona dokładnie robi.

```
(%i130) q : p_random(5);
        wszystkie_notacje : p_all_cyclic_notations(q);
        ile : length(wszystkie_notacje);
        print_list(wszystkie_notacje,6);
(q)     [[1,4,5]]
(wszystkie_notacje) [[[1,4,5]],[[1,4,5],[2]],[[1,4,5],[2],[3]],[[1,4,5],[3]],[[1,4,5],[3],[2]],[[2],[1,4,5]],[[2],[1,4,5],[3]],[[2],[3],[1,4,5]
        ],[[2],[3],[4,5,1]],[[2],[3],[5,1,4]],[[2],[4,5,1]],[[2],[4,5,1],[3]],[[2],[5,1,4]],[[2],[5,1,4],[3]],[[3],[1,4,5]],[[3],[1,4,5],[2
        ]],[[3],[2],[1,4,5]],[[3],[2],[4,5,1]],[[3],[2],[5,1,4]],[[3],[4,5,1]],[[3],[4,5,1],[2]],[[3],[5,1,4]],[[3],[5,1,4],[2]],[[4,5,1]],[
        [4,5,1],[2]],[[4,5,1],[2],[3]],[[4,5,1],[3]],[[4,5,1],[3],[2]],[[5,1,4]],[[5,1,4],[2]],[[5,1,4],[2],[3]],[[5,1,4],[3]],[[5,1,4],[3
        ],[2]]]
(ile)   33
```

(%o130)

| | | | | | |
|---|---|---|---|---|---|
| [[1,4,5]] | [[2],[1,4,5],[3]] | [[2],[5,1,4]] | [[3],[2],[5,1,4]] | [[4,5,1],[2]] | [[5,1,4],[2],[3]] |
| [[1,4,5],[2]] | [[2],[3],[1,4,5]] | [[2],[5,1,4],[3]] | [[3],[4,5,1]] | [[4,5,1],[2],[3]] | [[5,1,4],[3]] |
| [[1,4,5],[2],[3]] | [[2],[3],[4,5,1]] | [[3],[1,4,5]] | [[3],[4,5,1],[2]] | [[4,5,1],[3]] | [[5,1,4],[3],[2]] |
| [[1,4,5],[3]] | [[2],[3],[5,1,4]] | [[3],[1,4,5],[2]] | [[3],[5,1,4]] | [[4,5,1],[3],[2]] | |
| [[1,4,5],[3],[2]] | [[2],[4,5,1]] | [[3],[2],[1,4,5]] | [[3],[5,1,4],[2]] | [[5,1,4]] | |
| [[2],[1,4,5]] | [[2],[4,5,1],[3]] | [[3],[2],[4,5,1]] | [[4,5,1]] | [[5,1,4],[2]] | |

q) dla tej samej permutacji q wyznaczyć liczbę wszystkich możliwych zapisów w postaci złożenia rozłącznych cykli funkcją p_number_of_cyclic_notations(q), bez generowania tych zapisów, np.

```
(%i93)  p_number_of_cyclic_notations(q);
(%o93)  33
```

Wyznaczyć tę liczbę na podstawie analizy permutacji q zapisanej w postaci rozłącznych cykli.

r) wyznaczyć kilka początkowych wyrazów ciągu $a_n$, zdefiniowanego jako liczba wszystkich równoważnych sposobów zapisu permutacji identycznościowej $e_n$ w postaci złożenia rozłącznych cykli, np. tak

```
(%i26)  makelist( p_number_of_cyclic_notations(e(n)), n, 1, 10);
(%o26)  [1,3,11,49,261,1631,11743,95901,876809,8877691]
```

Wprowadzić te wyrazy na stronie https://oeis.org/ i dowiedzieć się więcej o tym ciągu.

s) dla dwóch losowych permutacji należących to tego samego Sn pokazać, że ich złożeniu odpowiada mnożenie macierzy (z wymijającymi się jedynkami) kodujących te permutacje zgodnie z regułą, że każda jedynka znajduje się na przecięciu f(i)-tego wiersza oraz i-tej kolumny wtedy i tylko wtedy, gdy w permutacji f i przechodzi na f(i), dla każdego i od 1 do n. Operatorem mnożenia macierzy w Maximie jest kropka. Przykładowe rozwiązanie:

```
(%i11)  per1 : p_random(5)$   per2 : p_random(5)$
        zlozenie : per1 ## per2$
        print( per1, "·", per2, "=", zlozenie )$
        print( cycles2perm(per1), "·", cycles2perm(per2), "=", cycles2perm(zlozenie) )$
        macierz1 : transpose( perm2matrix(per1) )$
        macierz2 : transpose( perm2matrix(per2) )$
        iloczyn : macierz1.macierz2$
        print( macierz1,"·",macierz2,"=",iloczyn )$
[[1,2,5,3]] · [[2,4,3,5]] = [[1,2,4],[5]]
[2,5,1,4,3] · [1,4,5,3,2] = [2,4,3,1,5]
```

$$\begin{pmatrix}0&0&1&0&0\\1&0&0&0&0\\0&0&0&0&1\\0&0&0&1&0\\0&1&0&0&0\end{pmatrix}\cdot\begin{pmatrix}1&0&0&0&0\\0&0&0&0&1\\0&0&0&1&0\\0&1&0&0&0\\0&0&1&0&0\end{pmatrix}=\begin{pmatrix}0&0&0&1&0\\1&0&0&0&0\\0&0&1&0&0\\0&1&0&0&0\\0&0&0&0&1\end{pmatrix}$$

t) Dla każdego n od 2 do 6 wykazać, że dla każdej permutacji należącej do Sn permutacja odwrotna posiada tyle samo inwersji, co permutacja oryginalna. Napisać odpowiedni skrypt. Analitycznie udowodnić, że to twierdzenie jest prawdziwe dla każdego n.

u) Dla każdego n od 2 do 6 wykazać, że dla każdej permutacji należącej do Sn permutacja odwrotna ma taki sam znak, co permutacja oryginalna. Napisać odpowiedni skrypt.

**Zad. 2.** Obliczyć permutację wynikową g, tzn. zapisać ją w najprostszej postaci: 1-wierszowej oraz cyklowej (cykle muszą być rozłączne). Permutacja g jest zdefiniowana wyrażeniem:

$g = \langle 6,2,3,1,5,4\rangle \; ( (2,5,3)(1,4) \langle 5,4,3,1,6,2\rangle^{-141} ((4,1,3)(5,6,3))^{93} )^{-1} \langle 4,5,1,3,2,6\rangle$

Podany zapis jest zapisem stosowanym podczas obliczeń na kartce papieru. Aby obliczyć to za pomocą komputera, należy całe wyrażenie przepisać do Maximy w formie trochę zmienionej, odpowiedniej do obliczeń komputerowych. Operatorem składania permutacji na komputerze jest ##, a operatorem potęgowania jest @. Na kartce papieru nawiasy <> oznaczają permutację zapisaną w postaci 1-wierszowej, a nawiasy () oznaczają zapis cyklowy. W Maximie do zapisu permutacji używa się zawsze nawiasów [], a nawiasy () służą do wymuszenia kolejności operacji. Należy pamiętać, że składanie permutacji nie jest przemienne. W Maximie całe podane wyrażenie należy wprowadzić bez jakiegokolwiek wstępnego przekształcania (upraszczania), w tym przykładzie wygląda ono tak:

```
(%i3)   g : [6,2,3,1,5,4] ## ( [[2,5,3],[1,4]] ## [5,4,3,1,6,2]@-141 ##([[4,1,3]]##[[5,6,3]])@93)@-1##[4,5,1,3,2,6];
(g)     [[2,3,5,4,6]]
(%i4)   cycles2perm(g);
(%o4)   [1,3,5,6,4,2]
```

Biblioteka kombinatoryczno-grafowa posiada dosyć inteligentną funkcję składania permutacji, która tylko w sytuacjach oczywistej sprzeczności zwraca błąd o niemożliwości wykonania złożenia. Jeżeli Maxima ma trudności z wykonaniem obliczeń, to albo błędnie zapisano wyrażenie, albo brakuje punktów stałych w permutacjach zapisanych cyklowo, albo gdzieś brakuje operatora składania permutacji, albo brakuje nawiasów (), albo - niestety - wystąpił inny poważny błąd.

**Zad. 3.** Rozwiązywanie równania permutacyjnego z jedną niewiadomą, występującą tylko jeden raz, i tylko w potędze +1 albo -1 (trudniejsze równania używają niewiadomej wiele razy i wyższych potęg, i zasadniczo nie da się ich rozwiązać analitycznie).

a) Rozwiązać w $S_5$ równanie permutacyjne z niewiadomą **f**:

$(4,2)\,(1,3,2)\,\{\,(5,2,4)(1,3)\,\mathbf{f}^{-1}\,(5,2,4,1,3)\,\langle 5,2,4,1,3\rangle\,\}^{-1}\,(5,2,4) = \{\,\langle 4,2,3,5,1\rangle^{-63}\,((5,3)\,(1,2,3,4))^{-1}\,\}^{2019}$

Aby rozwiązać w Maximie podane równanie, należy użyć funkcji p_solve2(rownanie, n). W tym przypadku n wynosi 5. Do Maximy trzeba przepisać to równanie w formacie komputerowym, bez wykonywania jakichkolwiek wstępnych przekształceń. Jeżeli równanie zostało przepisane poprawnie, ale Maxima nie potrafi go rozwiązać (zwraca informację o błędzie, że nie można składać permutacji z różnych Sn), wówczas zapis komputerowy permutacji zdefiniowanych cyklowo należy uzupełnić o pominięte punkty stałe. Chodzi o to, żeby wszystkie permutacje zostały przez Maximę wykryte jako należące do tego samego Sn. Podane równanie ma dokładnie jedno rozwiązanie, to znaczy tylko jedna permutacja w $S_5$ je spełnia. Należy pamiętać, że Maxima listę permutacji, które spełniają zadane równanie, zwraca w zapisie 1-wierszowym. Tak więc, aby porównać rozwiązanie komputerowe z ręcznym, należy permutację f, wyznaczoną ręcznie, przekształcić do postaci 1-wierszowej. W trakcie rozwiązywania tego zadania na papierze trzeba pamiętać, że właściwości

funkcji potęgowej znane z algebry nie działają w przypadku cykli nierozłącznych (działają, ale tylko dla cykli rozłącznych) oraz że składanie lewostronne i prawostronne to dwie różne operacje.

Dla przejrzystości można lewą i prawą stronę równania w Maximie zdefiniować jako dwie oddzielne funkcje tej samej zmiennej f. Operacja kill(f) przygotowuje (czyści) zmienną f.

```
(%i16) kill(f)$
       lewa(f) := [[4,2]] ## [[1,3,2]] ## ( [[5,2,4],[1,3]] ## f@-1 ## [[5,2,4,1,3]] ## [5,2,4,1,3] )@-1 ## [[5,2,4]];
       prawa(f) := ( [4,2,3,5,1]@-63 ## ( [[5,3]] ## [[1,2,3,4]] )@-1 )@2019;
       rownanie(f) := ( lewa(f) == prawa(f) );
       p_solve2(rownanie, 5);

(%o13) lewa(f):=[[4,2]] ## ([[1,3,2]] ## (([[5,2,4],[1,3]] ## (f @ (-1) ## ([[5,2,4,1,3]] ## [5,2,4,1,3]))) @ (-1) ## [[5,2,4]]))
(%o14) prawa(f):=([4,2,3,5,1] @ (-63) ## ([[5,3]] ## [[1,2,3,4]]) @ (-1)) @ 2019
(%o15) rownanie(f):=lewa(f) == prawa(f)
(%o16) [[1,5,3,4,2]]
```

Tak więc rozwiązaniem **f** tego równania jest tylko jedna permutacja, która w zapisie 1-wierszowym ma postać <1, 5, 3, 4, 2>, natomiast w zapisie cyklowym wygląda ona tak: (1)(2, 5)(3)(4). Warto dokonać sprawdzenia wyznaczonego rozwiązania poprzez obliczenie lewej i prawej strony równania.

b) permutacja jest samoodwrotna (jest inwolucją), jeżeli spełnia równanie $f^{-1} = f$. Składając prawostronnie obie strony tego równania z permutacją f, otrzymujemy $f^{-1} \cdot f = f \cdot f$, czyli $e = f^2$, a zatem $f^2 = e$, gdzie e jest permutacją identycznościową odpowiedniego rozmiaru, tzn. należy do tego samego Sn, co permutacja f. Rozwiązać równanie $f^2 = e$ w Maximie dla n od 1 do 6 i wykazać, że rozwiązaniem tego równania są rzeczywiście inwolucje oraz wyznaczyć ciąg liczb inwolucji w Sn (w funkcji n) i przekonać się, że to jest ten sam ciąg co http://oeis.org/A000085. Przypomnijmy: permutacja jest inwolucją, jeżeli w zapisie w postaci złożenia rozłącznych cykli posiada tylko cykle długości 1 lub 2. Funkcja is_involution() sprawdza, czy permutacja jest inwolucją. Funkcja p_all_involutions(n) zwraca listę wszystkich inwolucji w Sn. Rozwiązać analitycznie równanie $f^2 = e$ w $S_4$, w $S_5$ i w $S_6$ poprzez rozważenie odpowiednich rzędów (a zatem i typów) permutacji w Sn. Zadanie nie jest łatwe.

c) (dla ambitnych) korzystając z macierzowego zapisu permutacji (w postaci macierzy kwadratowej z wymijającymi się jedynkami) z jedynkami na przecięciu i-tej kolumny i f(i)-tego wiersza, zapisać równanie permutacyjne rozważane w punkcie a) albo b) w postaci równoważnego równania macierzowego z wieloma niewiadomymi. Porównując macierze z lewej i z prawej strony znaku równości zapisać odpowiednią liczbę równań algebraicznych. Rozwiązać powstały układ równań analitycznie i w Maximie. Czy tym sposobem można rozwiązać dowolne równanie permutacyjne? Najlepiej rozpocząć od rozwiązania na kartce papieru równania $f^2 = e$ w $S_3$. Będzie 9 niewiadomych, które mogą przyjmować wartości tylko 0 albo 1, i tylko 3 niewiadome mogą być równe 1. W $S_4$ będzie aż 16 niewiadomych, z których dokładnie 4 będą jedynkami, a pozostałych 12 będzie zerami. Jeżeli rozwiązaniem równania permutacyjnego jest więcej niż jedna permutacja, to powstały układ równań musi być rozwiązywany w zależności od jednego lub więcej parametrów.

**Zad. 4.** Funkcją p_random(n) dla n = 6 wylosować dowolną permutację w $S_6$. Następnie określić, czy jest ona: nieporządkiem, inwolucją, transpozycją, czy jest 1-cyklowa, parzysta, nieparzysta. Sprawdzić wymienione cechy za pomocą odpowiednich funkcji w Maximie.

**Zad. 5.** Wylosować permutację h w $S_6$. Zapisać ją w postaci 1-wierszowej i wyznaczyć zbiór inwersji, wektor inwersyjny oraz numer tej permutacji na liście w porządku leksykograficznym. Sprawdzić każdą z tych „rzeczy" w Maximie. Stosując poznany na wykładzie algorytm generowania permutacji w porządku leksykograficznym, dla zapisu 1-wierszowego, wyznaczyć ręcznie 5 kolejnych permutacji następujących po h i sprawdzić swoje rozwiązanie w programie Maxima, np. funkcją p_next().

**Zad. 6.** Dla podanego numeru w porządku leksykograficznym (np. 107) wyznaczyć, metodą dzieleń z resztą, permutację w $S_5$ i w $S_6$. Co wspólnego mają otrzymane permutacje, gdy porównujemy ich zapisy 1-wierszowe?

**Zad. 7.** Wyprowadzić ręcznie, na kartce papieru, przedstawienie permutacji p = (4,1,5)(2,3) w postaci złożenia minimalnej liczby transpozycji sąsiednich elementów na dwa sposoby:

1. sprowadzając na swoje naturalne pozycje kolejno liczby 1, 2, 3, 4, 5,

2. sprowadzając na swoje naturalne pozycje liczby w kolejności 5, 4, 3, 2, 1.

Trzeba pamiętać, że to sprowadzanie liczb wykonuje się dla zapisu 1-wierszowego permutacji p. Sprawdzić oba rozwiązania analityczne za pomocą funkcji p_transpos_neighb_left(p) oraz p_transpos_neighb_right(p). Zauważyć, że w obu przypadkach liczba wynikowych transpozycji jest taka sama. Wyznaczyć, poprzez wypisanie wszystkich inwersji, liczbę inwersji w permutacji p i przekonać się, że jest to ta sama liczba. Otrzymane obie listy transpozycji złożyć (każdą oddzielnie) ręcznie i komputerowo, i upewnić się, że ich złożenie daje oryginalną permutację p. Listę permutacji można złożyć w Maximie za pomocą funkcji p_list_composition(lista) lub wielu operatorów ##.

Napisać krótki skrypt, który pokaże, że prawostronne złożenie dowolnej permutacji f, zapisanej 1-wierszowo, z dowolną transpozycją (i, j) daje permutację, która w zapisie 1-wierszowym wygląda podobnie jak f, z tym, że ma jedynie zamienione liczby na pozycjach i oraz j. Wykazać również, że lewostronne złożenie transpozycji (i, j) oraz f zamienia w f miejscami liczby i oraz j.

Za pomocą funkcji p_to_transpositions_neighb_comp(p, [m]), bez używania opcjonalnego argumentu m, wyznaczyć wszystkie możliwe sposoby przedstawienia permutacji p w postaci złożenia minimalnej liczby transpozycji sąsiednich elementów. Ile jest tych sposobów? Za pomocą funkcji print_list() zapisać tę listę ładnie wierszami. Można to osiągnąć wykonując poniższy ciąg instrukcji:

```
p : [[4,1,5],[2,3]]$
lista_list_transpozycji_sasiednich : p_to_transpositions_neighb_comp(p)$
ladna_lista : makelist(makelist(cycles2string(perm2mincycles(p)),p,lista), lista, lista_list_transpozycji_sasiednich)$
length(lista_list_transpozycji_sasiednich);
print_list(ladna_lista);
```

Czy da się jakoś obliczyć liczbę wszystkich sposobów przedstawienia permutacji p w postaci złożenia minimalnej liczby transpozycji sąsiednich elementów? Od czego zależy ta liczba i jak? Czy istnieje jakiś wzór na tę liczbę? Zauważ, że niektóre przedstawienia różnią się tylko kolejnością transpozycji, gdyż dwie sąsiadujące w złożeniu rozłączne transpozycje można zamienić miejscami.

(dla ambitnych) Dla każdej permutacji należącej do $S_4$ wyznaczyć liczbę wszystkich sposobów przedstawienia jej w postaci złożenia minimalnej liczby transpozycji sąsiednich elementów. Sporządzić wykres liczby sposobów w funkcji numeru permutacji. Które permutacje, oprócz identycznościowej, można przedstawić na największą liczbę sposobów. Jaką wspólną cechę mają te permutacje? Wszystkie permutacje można wygenerować (w porządku leksykograficznym) funkcją p_all(4). Zrobić to samo zadanie również dla $S_5$. Uwaga: obliczenia mogą trochę potrwać. Spróbować najpierw oszacować całkowity czas obliczeń na podstawie czasu zużytego dla jednej permutacji.

Odnaleźć na ładnej liście dwa ciągi (dwie listy) transpozycji sąsiednich elementów wyznaczone w pierwszej części tego zadania, tzn. powstałe w wyniku sprowadzania liczb na swoje naturalne pozycje, w kolejności 1, 2, 3, 4, 5 oraz 5, 4, 3, 2, 1. Muszą gdzieś te dwie listy być.

Wybrać (odpisać), mniej więcej ze środka otrzymanej listy, jedno z przedstawień permutacji p w postaci złożenia minimalnej liczby transpozycji sąsiednich elementów. Uważać na znaczenie nawiasów przy odczytywaniu listy transpozycji, w przypadku odczytywania jej z listy lista_list_transpozycji_sasiednich. Każde przedstawienie na tej liście zaczyna się nawiasami [[[ i kończy nawiasami ]]]. Dla ładnej listy każde przedstawienie jest zapisane w osobnej linii. Wykonać na papierze i komputerowo złożenie transpozycji (od prawej do lewej!) z wybranej listy, w celu otrzymania permutacji p! Dla złożenia komputerowego może okazać się konieczne dopisanie w niektórych transpozycjach punktu stałego równego n (tu 5).

Permutację p, zapisaną 1-wierszowo, złożyć z ostatnią (po prawej) transpozycją w wybranym w poprzednim akapicie przedstawieniu i wynik złożenia zapisać 1-wierszowo. Otrzymany wynik złożyć z przedostatnią transpozycją itd. Na każdym kroku zapisywać wynik złożenia w postaci 1-wierszowej. Po uwzględnieniu wszystkich transpozycji, w ostatnim zapisie powinna wyjść permutacja identycznościowa. Czy można teraz wskazać kolejność, w jakiej liczby 1, 2, 3, 4, 5 wędrowały na swoje pozycje? Czy raczej w tym przypadku taka reguła nie występuje? Prześledzić, jak zmienia się pozycja każdej z liczb 1, 2, 3, 4 i 5 w permutacjach 1-wierszowych, rozważanych w tym akapicie.

W podanym na poprzedniej stronie ciągu instrukcji zmienić wywołanie funkcji p_to_transpositions_neighb_comp (p, [m]), na wywołanie z opcjonalnym parametrem m (bez używania nawiasów []). Najpierw dać m równe liczbie inwersji w permutacji p. Następnie użyć liczby o 2 mniejszej, a na końcu liczby o 2 większej. Wyjaśnić otrzymane wyniki.

Skopiować podany wcześniej kod do nowej komórki w Maximie i zastąpić funkcję p_to_transpositions_neighb_comp() funkcją p_to_transpositions_comp(). Zauważyć, że liczba transpozycji na każdej liście jest teraz mniejsza, bo transpozycje nie są sąsiednich elementów. Liczba list jest także mniejsza, ponieważ prawie zawsze liczba sposobów przedstawienia konkretnej permutacji, w postaci złożenia minimalnej liczby transpozycji sąsiednich elementów, jest większa od liczby sposobów, gdy dopuszczalne są transpozycje niesąsiednich. Wybrać jedną z list i przedstawić każdą transpozycję niesąsiednich elementów w postaci złożenia minimalnej liczby transpozycji sąsiednich elementów. Jak wygląda teraz cały ciąg transpozycji sąsiednich elementów? Ile transpozycji jest w tym ciągu? O ile więcej jest transpozycji niż w przypadku ich minimalnej liczby, gdy użyto funkcji p_to_transpositions_neighb_comp()? Dlaczego ta różnica jest liczbą parzystą?

(dla ambitnych) Liczby 1, 2, 3, 4 i 5 można sprowadzić na swoje naturalne pozycje na 5! = 120 sposobów. Aby przekonać się, że nie każda kolejność sprowadzania liczb na swoje naturalne pozycje prowadzi do przedstawienia permutacji p w postaci złożenia transpozycji sąsiednich elementów, wymusić jakąś własną kolejność sprowadzania liczb poprzez użycie funkcji p_transpos_neighb _by_order(p, f) zadając dowolną permutację f (należącą do tego samego Sn, co p). Wybrać (wymyślić lub zgadnąć) taką permutację f, aby w wyniku złożenia otrzymanych transpozycji nie wyszła permutacja p. Jak wygląda przekształcona permutacja p po sprowadzeniu liczb według zadanej kolejności f? Powinna być bliska permutacji identycznościowej, ale nie równa jej. Określić brakujące transpozycje sąsiednich elementów, które doprowadzą w końcu permutację p do permutacji identycznościowej. Dokonać teraz złożenia wszystkich transpozycji, wraz z tymi dodatkowymi, i przekonać się, że wyjdzie permutacja p. O ile transpozycji więcej trzeba było użyć niż w przypadku

funkcji p_transpos_neighb_left(p) oraz p_transpos_neighb_right(p)? Czy liczba tych dodatkowych transpozycji była parzysta? Dlaczego?

(dla ambitnych c.d.) Aby wykonać w sposób systematyczny badanie wpływu kolejności sprowadzania liczb na pozycje naturalne na liczbę transpozycji sąsiednich elementów, można użyć skryptu podanego na kolejnej stronie. Jego szczegółową analizę pozostawiamy czytelnikowi.

Z tego skryptu wynika, że istnieje 35 przedstawień permutacji p w postaci złożenia minimalnej liczby (7) transpozycji sąsiednich elementów. 15 z tych przedstawień można opisać sprowadzaniem liczb 1, 2, 3, 4, 5 na ich naturalne pozycje, w różnych kolejnościach. W pozostałych 20 przedstawieniach liczby wracają na swoje pozycje nie po kolei, ale raczej najpierw trochę jedna, potem trochę druga itd., w pewnym sensie wracają równolegle, a raczej równocześnie.

```
n : 5$   p : [ [4,1,5], [2,3] ];     cycles2perm(p);    Sn : p_all(n)$
listy_transpozycji_sasiednich : makelist( p_transpos_neighb_by_order(p, f) , f, Sn )$
length(unique(listy_transpozycji_sasiednich));
zlozenia : map(p_list_composition, listy_transpozycji_sasiednich)$
ile_transpozycji : map(length, listy_transpozycji_sasiednich);
min_indeksy : sublist_indices(ile_transpozycji, lambda([ile], is(ile=lmin(ile_transpozycji))));
ile_min : length(min_indeksy);
ile_min_unik : length(unique( makelist(listy_transpozycji_sasiednich[i], i, listify(min_indeksy)) ));
dobre_indeksy : sublist_indices(zlozenia, lambda([q], is(q==p)));
zle_indeksy : setdifference(setify(makelist(i,i,1,n!)), setify(dobre_indeksy));
ile_zlych : cardinality(zle_indeksy);
ile_dobrych : n! − ile_zlych;
zle_perm_f : makelist(Sn[i], i, listify(zle_indeksy));
przedstawienia_minimalne : p_to_transpositions_neighb_comp(p)$
ile_min_wszystkich : length(przedstawienia_minimalne);
map(length, przedstawienia);
roznica : ile_min_wszystkich − ile_min_unik;
```

```
(p)          [[4,1,5],[2,3]]
(%o853)      [5,3,2,1,4]
(%o856)      55
(ile_transpozycji) [7,7,9,9,7,7,8,7,9,9,7,7,9,9,9,9,9,10,7,7,7,7,7,7,9,8,10,9,7,7,8,7,9,9,7,7,10,9,9,9,9,10,7,7,7,7,7
,7,8,7,8,8,7,7,8,7,9,9,7,7,9,9,9,9,9,9,10,7,7,8,8,7,8,9,9,9,9,9,9,10,9,9,9,9,9,10,9,9,9,9,9,10,9,9,9,10,10,11,7
,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7]
(min_indeksy) [1,2,5,6,8,11,12,19,20,21,22,23,24,29,30,32,35,36,43,44,45,46,47,48,50,53,54,56,59,60,67,68,71,
97,98,99,100,101,102,103,104,105,106,107,108,109,110,111,112,113,114,115,116,117,118,119,120]
(ile_min)    57
(ile_min_unik) 15
(dobre_indeksy) [1,2,3,4,5,6,8,9,10,11,12,13,14,15,16,17,19,20,21,22,23,24,28,29,30,32,33,34,35,36,38,39,40,41,
43,44,45,46,47,48,50,53,54,56,57,58,59,60,61,62,63,64,65,67,68,71,73,74,75,76,77,78,80,81,82,83,85,86,87,
88,89,91,92,93,97,98,99,100,101,102,103,104,105,106,107,108,109,110,111,112,113,114,115,116,117,118,119
,120]
(ile_zlych)  22
(ile_dobrych) 98
(zle_perm_f) [[1,3,2,4,5],[1,4,5,3,2],[2,1,3,4,5],[2,1,3,5,4],[2,1,4,3,5],[2,3,1,4,5],[2,4,1,3,5],[2,4,5,3,1],
[3,1,2,4,5],[3,1,4,2,5],[3,1,4,5,2],[3,2,1,4,5],[3,4,5,2,1],[3,5,2,1,4],[3,5,2,4,1],[3,5,4,2,1],[4,2,1,3,5
],[4,2,5,3,1],[4,3,5,2,1],[4,5,2,3,1],[4,5,3,1,2],[4,5,3,2,1]]
(ile_min_wszystkich) 35
(%o869)      [7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7]
(roznica)    20
```

**Zad. 8.** Stosując funkcję p_filter(warunki, n) zbudować niezbyt skomplikowaną funkcję booleowską do odfiltrowania permutacji w $S_5$ lub w $S_6$, posiadających zadane właściwości. Warunki sformułować tak, aby spełniało je kilkadziesiąt lub kilkaset permutacji, ale nie za dużo (nie wszystkie w Sn). Następnie wyznaczyć liczbę tych permutacji na kartce papieru, korzystając z diagramu Venna dla dwóch lub więcej zbiorów i ze wzoru na liczbę permutacji danego typu oraz wykorzystując definicje takich permutacji, jak: inwolucje, nieporządki, permutacje parzyste, definicję rzędu permutacji itp. Nie zawsze obliczenia będą łatwe. Na przykład, jeżeli interesują nas wszystkie permutacje, które w $S_5$ są nieporządkami albo nie są parzyste (= albo są nieparzyste), to należy użyć następujących instrukcji

```
(%i39)  warunki(p) := is_derangement(p)  xor  not is_even(p);
        wybrane_permutacje : p_filter(warunki, 5);
        length(wybrane_permutacje);
(%o37)  warunki(p) :=is_derangement(p)  xor notis_even(p)
(wybrane_permutacje) [[1,2,3,5,4],[1,2,4,3,5],[1,2,5,4,3],[1,3,2,4,5],[1,3,4,5,2],[1,3,5,2,4],[1,4,2,5,3],[1,4,3,2,5],[1,4,
5,3,2],[1,5,2,3,4],[1,5,3,4,2],[1,5,4,2,3],[2,1,3,4,5],[2,3,4,1,5],[2,3,4,5,1],[2,3,5,1,4],[2,3,5,4,1],[2,4,1,3
,5],[2,4,1,5,3],[2,4,3,5,1],[2,4,5,3,1],[2,5,1,3,4],[2,5,1,4,3],[2,5,3,1,4],[2,5,4,1,3],[3,1,4,2,5],[3,1,4,5,2]
,[3,1,5,2,4],[3,1,5,4,2],[3,2,1,4,5],[3,2,4,5,1],[3,2,5,1,4],[3,4,2,1,5],[3,4,2,5,1],[3,4,5,1,2],[3,5,2,1,4],[3
,5,2,4,1],[3,5,4,2,1],[4,1,2,3,5],[4,1,2,5,3],[4,1,3,5,2],[4,1,5,3,2],[4,2,1,5,3],[4,2,3,1,5],[4,2,5,3,1],[4,3,
1,2,5],[4,3,1,5,2],[4,3,5,2,1],[4,5,1,2,3],[4,5,2,3,1],[4,5,3,2,1],[5,1,2,3,4],[5,1,2,4,3],[5,1,3,2,4],[5,1,4,2
,3],[5,2,1,3,4],[5,2,3,4,1],[5,2,4,1,3],[5,3,1,2,4],[5,3,1,4,2],[5,3,4,1,2],[5,4,1,3,2],[5,4,2,1,3],[5,4,3,1,2]
]
(%o39)  64
```

Jak widać, jest 64 takich permutacji wśród 120 w $S_5$. Wybrać 2 dowolne permutacje z wygenerowanej listy i sprawdzić, czy spełniają one zadany warunek logiczny. W celu ręcznego wyznaczenia liczby permutacji spełniających zadane warunki, należy sporządzić tabelę, w której wierszach znajdą się wszystkie typy permutacji w $S_5$, a w kolumnach odpowiednie warunki logiczne, tzn. osobno pierwszy, drugi, ewentualnie trzeci, i potem cały. Należy wstawić symbol ✓ albo ✗ w odpowiednich komórkach tej tabeli i na końcu zsumować liczby permutacji dla typów spełniających warunki zadania.

Przykładowe warunki, które można użyć do filtrowania:

0. permutacja jest nieporządkiem albo nie jest parzysta, (64, 355)
1. permutacja jest inwolucją lub nie jest 1-cyklowa, (26, 76)
2. permutacja nie jest transpozycją i nie posiada cyklu długości 3, (70, 505)
3. permutacja ma 5 razy mniej punktów stałych niż inwersji, (16, 27)
4. permutacja jest parzysta albo jej rząd jest parzysty lub jest nieparzysta, (105, 585)
5. permutacja posiada co najmniej 2 cykle i każdy ma inną długość; w tym zadaniu żadnego cyklu długości 1 nie pomijamy w zapisie cyklowym permutacji, (50, 354)
6. liczba punktów stałych w permutacji i rząd permutacji odwrotnej są tej samej parzystości, (21 , 559)
7. permutacja jest inwolucją albo transpozycją albo jest nieparzysta, (66 (!), 376 (!))
8. permutacja posiada parzystą liczbę cykli parzystej długości lub parzystą liczbę inwersji, (60, 360).

Dla ułatwienia (dla kontroli), w nawiasach podano wynikowe liczby permutacji w $S_5$ i w $S_6$.

Funkcje Maximy, które mogą się przydać, to: oddp, evenp, is, lambda, some, unique, makelist, sort, block, length, return. Dopuszczalne operatory logiczne to: not, and, or, xor. Potrzebne będą oczywiście także funkcje z biblioteki kombinatoryczno-grafowej.

**Zad. 9.** Używając funkcji zaczynających się od p_all… wygenerować różne podzbiory zbioru permutacji Sn. Dla funkcji p_all(n) przyglądnąć się kolejności, w jakiej permutacje pojawiają się na wynikowej liście. Będzie to porządek leksykograficzny. Dla przejrzystości funkcją print_list() zapisać permutacje jedna pod drugą. Funkcje p_all_plain_changes1(n) oraz p_all_plain_changes2(n) generują wszystkie permutacje z Sn w kolejności minimalnych zmian. Przeanalizować, jak te zmiany propagują przy przechodzeniu od jednej permutacji do drugiej. Dla funkcji p_all_even(n) oraz p_all_odd(n) sprawdzić, czy rzeczywiście wszystkie permutacje na danej liście są parzyste albo nieparzyste. Aby dla każdej permutacji na liście wykonać pewną funkcję i otrzymać listę wynikową, należy użyć funkcji makelist albo map. Na przykład instrukcje makelist(p_sign(p), p, p_all(4)) oraz map(p_sign, p_all(4)) wyznaczą znaki wszystkich permutacji w $S_4$. Zamiast p_all(4) można wstawić zmienną, w której wcześniej będą wygenerowane wszystkie permutacje z $S_4$, np. S4 : p_all(4); makelist(p_sign(p), p, S4);

**Zad. 10.** Wygenerować komputerowo wszystkie permutacje tego samego typu, wybranego samodzielnie albo wskazanego przez prowadzącego. Typ powinien należeć do $S_5$ albo do $S_6$. W celu wygenerowania rozważanych permutacji, można skorzystać z poniższych instrukcji. W tym przykładzie są generowane wszystkie permutacje typu $[1^2 2^2]$, który należy do $S_6$.

```
(%i5)   typ : [ [1,2], [2,2] ]$
        permutacje_tego_samego_typu : p_all_of_type(typ)$
        cyklowo : map(cycles2string, map(perm2maxcycles, permutacje_tego_samego_typu));
        liczba : [length(cyklowo), p_type_size(typ)];
(cyklowo) [(1)(2)(3,4)(5,6),(1)(2)(3,5)(4,6),(1)(2)(3,6)(4,5),(1)(2,3)(4)(5,6),(1)(2,3)(4,5)(6),(1)(2,3)(4,6)(5),(1)(2,4)(3)(5,6),
  (1)(2,4)(3,5)(6),(1)(2,4)(3,6)(5),(1)(2,5)(3)(4,6),(1)(2,5)(3,4)(6),(1)(2,5)(3,6)(4),(1)(2,6)(3)(4,5),(1)(2,6)(3,4)(5),(1)(2,6)(3,5)(4),
  (1,2)(3)(4)(5,6),(1,2)(3)(4,5)(6),(1,2)(3)(4,6)(5),(1,2)(3,4)(5)(6),(1,2)(3,5)(4)(6),(1,2)(3,6)(4)(5),(1,3)(2)(4)(5,6),(1,3)(2)(4,5)(6),
  (1,3)(2)(4,6)(5),(1,3)(2,4)(5)(6),(1,3)(2,5)(4)(6),(1,3)(2,6)(4)(5),(1,4)(2)(3)(5,6),(1,4)(2)(3,5)(6),(1,4)(2)(3,6)(5),(1,4)(2,3)(5)(6),
  (1,4)(2,5)(3)(6),(1,4)(2,6)(3)(5),(1,5)(2)(3)(4,6),(1,5)(2)(3,4)(6),(1,5)(2)(3,6)(4),(1,5)(2,3)(4)(6),(1,5)(2,4)(3)(6),(1,5)(2,6)(3)(4),
  (1,6)(2)(3)(4,5),(1,6)(2)(3,4)(5),(1,6)(2)(3,5)(4),(1,6)(2,3)(4)(5),(1,6)(2,4)(3)(5),(1,6)(2,5)(3)(4)]
  (liczba)  [45,45]
```

Otrzymaną liczbę permutacji porównać z obliczeniami „ręcznymi” liczebności zadanego typu permutacji, korzystając ze wzoru poznanego na wykładach. W tym przykładzie obliczenia będą wyglądać tak: $||[1^2 2^2]|| = \frac{6!}{1^2\,2^2\,2!\,2!} = \frac{720}{1\cdot4\cdot2\cdot2} = \frac{720}{8\cdot2} = \frac{90}{2} = 45$. Wyznaczyć, dla porównania, liczbę permutacji zadanego typu poprzez kombinatoryczne obliczenie liczby sposobów zaetykietowania wierzchołków grafu skierowanego liczbami od 1 do n, składającego się z odpowiedniej liczby cykli odpowiedniej długości (w tym przypadku grafu skierowanego składającego się z dwóch cykli długości 1 i dwóch cykli długości 2). W tym zadaniu obliczenia wyglądają tak:

$$||[1^2 2^2]|| = \frac{C_6^1\cdot(1-1)!\cdot C_5^1\cdot(1-1)!\cdot C_4^2\cdot(2-1)!\cdot C_2^2\cdot(2-1)!}{2!\cdot2!} = \frac{C_6^1\cdot C_5^1\cdot C_4^2\cdot C_2^2\cdot(0)!\cdot(0)!\cdot(1)!\cdot(1)!}{2!\cdot2!} = \frac{6!\cdot(0)!\cdot(0)!\cdot(1)!\cdot(1)!}{(1!\cdot1!\cdot2!\cdot2!)\cdot2!\cdot2!} = 45$$

Typ do tego zadania można wybrać spośród wszystkich typów należących do $S_5$ albo do $S_6$. Bardziej cierpliwi mogą wybrać typ należący do $S_7$. Zaleca się zapisanie pliku przed uruchomieniem obliczeń, gdyż opcja przerwania obliczeń (Ctrl+G) nie zawsze przynosi oczekiwany skutek. W sytuacji krytycznej, gdy Ctrl+G nie działa,a chcemy przerwać długotrwałe obliczenia, należy wybrać opcję

Maxima → Restart Maxima, po której użyciu konieczne jest ponowne załadowanie biblioteki kombinatoryczno-grafowej (i każdej innej używanej) oraz ponowna inicjalizacja wszystkich zmiennych.

```
(%i3) print_list( p_all_types(7) );
[[1,7]]
[[1,5],[2,1]]
[[1,3],[2,2]]
[[1,1],[2,3]]
[[1,4],[3,1]]
[[1,2],[2,1],[3,1]]
[[2,2],[3,1]]
[[1,1],[3,2]]
[[1,3],[4,1]]
[[1,1],[2,1],[4,1]]
[[3,1],[4,1]]
[[1,2],[5,1]]
[[2,1],[5,1]]
[[1,1],[6,1]]
[[7,1]]
(%o3) done

(%i4) print_list( p_all_types(6) );
[[1,6]]
[[1,4],[2,1]]
[[1,2],[2,2]]
[[2,3]]
[[1,3],[3,1]]
[[1,1],[2,1],[3,1]]
[[3,2]]
[[1,2],[4,1]]
[[2,1],[4,1]]
[[1,1],[5,1]]
[[6,1]]
(%o4) done

(%i5) print_list( p_all_types(5) );
[[1,5]]
[[1,3],[2,1]]
[[1,1],[2,2]]
[[1,2],[3,1]]
[[2,1],[3,1]]
[[1,1],[4,1]]
[[5,1]]
(%o5) done
```

**Zad. 11.** Permutacja identycznościowa w $S_n$ ma oczywiście postać $< 1, 2, 3, \dots, n >$ i zero inwersji. Dla podanej niżej permutacji $p$ zapisanej 1-wierszowo wyznaczyć analitycznie liczbę inwersji w funkcji $m$ i $n$ (albo tylko $n$), a następnie zapisać wzór na jej znak, korzystając z definicji: $\mathrm{sgn}(p) = (-1)^{\mathrm{I}(p)}$, gdzie $\mathrm{I}(p)$ to liczba inwersji w permutacji $p$. Wzór na $\mathrm{sgn}(p)$, po podstawieniu za $\mathrm{I}(p)$ zależności od $m$ i $n$, przekształcić do najprostszej postaci. Wykonać w Maximie obliczenia znaku z tego wzoru dla różnych par wartości $m$ i $n$, najlepiej dla wielu, i zapisać w odpowiedniej macierzy, która będzie miała n wierszy i m kolumn. Dla porównania wyznaczyć znak permutacji $p$ bezpośrednio z jej zapisu 1-wierszowego funkcją p_sign(), po uwzględnieniu konkretnych wartości $m$ i $n$, również w formie macierzy. Powinno wyjść to samo obydwoma sposobami. Rozwiązać to zadanie również dla permutacji $q$ należącej do $S_{2n}$.

(Permutacja identycznościowa w $S_n$ ma postać $< 1, 2, 3, \dots, n >$ i zero inwersji. Ile inwersji ma permutacja zapisana niżej?)

a) $p =< n, n-1, n-2, \dots, 3, 2, 1 >$
b) $p =< 1, 2, \dots, m-1, m, n, n-1, \dots, m+2, m+1 >$, założyć, że $m < n$
c) $p =< n, n-1, \dots, m+2, m+1, 1, 2, \dots, m-1, \ m >$, założyć, że $m < n$
d) $p =< m+1, m+2, \dots, n-1, n, 1, 2, \dots, m-1, m >$, założyć, że $m < n$
e) $p =< m, m-1, \dots, 2, 1, m+1, \ m+2, \dots, n-1, n >$, założyć, że $m < n$
f) $p =< m, m-1, \dots, 2, 1, n, n-1, \dots, m+2, m+1 >$, założyć, że $m < n$
g) $p =< m+1, m+\ 2, \dots, n-1, n, m, m-1, \dots, 2, 1 >$, założyć, że $m < n$

(Permutacja identycznościowa w $S_{2n}$ ma postać $< 1, 2, 3, \dots, 2n >$ i zero inwersji. Ile inwersji ma permutacja zapisana niżej?)

a) $q =< 1, 3, 5, \dots, 2n-1, 2, 4, 6, \dots, 2n >$
b) $q =< 1, 3, 5, \dots, 2n-1, 2n, 2n-2, 2n-4, \dots, 2 >$
c) $q =< 2, 4, 6, \dots, 2n, 1, 3, 5, \dots, 2n-1 >$

d) $q =< 2n, 2n-2, 2n-4, \ldots, 2, 1, 3, 5, \ldots, 2n-1 >$
e) $q =< 2n-1, 2n-3, 2n-5, \ldots, 3, 1, 2n, 2n-2, 2n-4, \ldots, 2 >$
f) $q =< 1, 2n, 2, 2n-1, 3, 2n-2, \ldots, n, n+1 >$
g) $q =< 2n, 2n-1, 2n-3, 2n-2, 2n-4, 2n-5, \ldots, 2, 1$ dla $n$ nieparz $(1, 2$ dla $n$ parz.$) >$