Sprawozdanie: Matematyka Dyskretna, Projekt indywidualny

Wykonany przez: Bartłomiej Guś, 297415

1. Część 1

1.1. Zaimplementować algorytm generowania podzbiorów zbioru {1, . . . , n}. Wypisz 10 kolejnych podzbiorów zbioru {1, 2, 3, 4, 5, 6, 7} poczynając od podzbioru {1, 2, 3, 5}

W celu wykonania tego zadania stworzyłem następującą funkcje, która odzwierciedla algorytm generowania podzbiorów zbioru (nazwa pliku: algorytm_generowania_podzbiorow.m):

```
function podzbiory_zbioru = algorytm_generowania_podzbiorow(n)

podzbiory_zbioru = {};

podzbiory_zbioru(1,1) = {[]};

iterator = 1;

zbior_liczb = 1:n;
zbior_liczb_pom = zbior_liczb;
czy_to_koniec = 1;

while 1

    for i = 1:n

      maximum = max(zbior_liczb_pom);
      badany_podzbior = podzbiory_zbioru(iterator,1);
      badany_podzbior = cell2mat(badany_podzbior);
      index = find(badany_podzbior==maximum);

      czy_istnieje_taki_element = isempty(index);

      if czy_istnieje_taki_element == 1
          czy_to_koniec = 0;
          nastepny_podzbior = [badany_podzbior,maximum];
          nastepny_podzbior(nastepny_podzbior>maximum)=[];

          podzbiory_zbioru(iterator+1,1) = {nastepny_podzbior};
          break
      end

      zbior_liczb_pom(zbior_liczb_pom==maximum) = [];
    end

    if czy_to_koniec == 1
        return
    end

    zbior_liczb_pom = zbior_liczb;
    czy_to_koniec = 1;
    iterator = iterator + 1;

end

end
```

Funkcja ta zwraca w postaci macierzy o komórkach typu cell, znalezione podzbiory zbioru {1, . . . , n}.

W celu znalezienia konkretnego podzbioru stworzyłem następującą funkcje, która zwraca indeks szukanego podzbioru w macierzy, która zawiera wszystkie podzbiory (nazwa pliku: znajdz.m):

```
function index = znajdz(podzbiory,szukany_podzbior)

index = 0;
wielkosc_podzbiory = size(podzbiory,1);

for i = 1:wielkosc_podzbiory

    badany_podzbior = podzbiory(i,1);
    badany_podzbior = cell2mat(badany_podzbior);
    czy_to_ten = isequal(badany_podzbior,szukany_podzbior);

    if czy_to_ten == 1
        index = i;
        return
    end

end
```

Poniższy fragment kodu odpowiada za stworzenie macierzy zawierającej wszystkie podzbiory zbioru {1, . . . , n}, znalezienie indeksu poszukiwanego podzbioru czyli {1,2,3,5} oraz wypisanie kolejnych dziesięciu podzbiorów, poczynając od poszukiwanego w Live Script.

Zad.1 cz.1

```
n = 7;

podzbiory = algorytm_generowania_podzbiorow(n);

szukany_podzbior = [1,2,3,5];

index = znajdz(podzbiory,szukany_podzbior);

jak_wiele = index:1:index+9;

dziesiec_kolejnych_podzbiorow = podzbiory(jak_wiele,1);

for i = 1:size(dziesiec_kolejnych_podzbiorow,1)


    do_wyswietlenia_podzbior = dziesiec_kolejnych_podzbiorow(i,1);

    do_wyswietlenia_podzbior = cell2mat(do_wyswietlenia_podzbior);

    X = ['Podzbior ',num2str(i),' : ',num2str(do_wyswietlenia_podzbior)];

    disp(X);


end
```

Wynik:

```
  Podzbiór 1 : 1  2  3  5
  Podzbiór 2 : 1  2  3  5  7
  Podzbiór 3 : 1  2  3  5  6
  Podzbiór 4 : 1  2  3  5  6  7
  Podzbiór 5 : 1  2  3  4
  Podzbiór 6 : 1  2  3  4  7
  Podzbiór 7 : 1  2  3  4  6
  Podzbiór 8 : 1  2  3  4  6  7
  Podzbiór 9 : 1  2  3  4  5
  Podzbiór 10 : 1  2  3  4  5  7
```

1.2. Zaimplementować algorytm generowania k-elementowych podzbiorów zbioru {1, . . . , n}. Wypisz 10 kolejnych 5-elementowych podzbiorów zbioru {1, 2, 3, 4, 5, 6, 7}.

W celu wykonania tego zadania stworzyłem następującą funkcje, która odzwierciedla algorytm generowania k - elementowych podzbiorów zbioru {1, . . . , n} (nazwa pliku: algorytm_generowania_k_elementowych_podzbiorow.m):

```
function podzbiory_k_elementowe =
algorytm_generowania_k_elementowych_podzbiorow(n,k)

podzbiory_k_elementowe = {};

podzbiory_k_elementowe(1,1) = {[1:k]};
zbior_liczb = 1:n;

iterator = 1;

while 1

    badany_podzbior = podzbiory_k_elementowe(iterator,1);
    badany_podzbior = cell2mat(badany_podzbior);
    znaleziony_index = 0;

    for i = 1:k

        index = find(badany_podzbior==(badany_podzbior(1,i)+1));

        czy_istnieje_taki_element = isempty(index);

        if czy_istnieje_taki_element == 1
            znaleziony_index = i;
            break
        end
    end

    if badany_podzbior(1,znaleziony_index)==zbior_liczb(end)
       return
    elseif badany_podzbior(1,znaleziony_index)<zbior_liczb(end)
        badany_podzbior(1,znaleziony_index) =
badany_podzbior(1,znaleziony_index) + 1;
        badany_podzbior(1,1:znaleziony_index-1) = (1:znaleziony_index-1);
    end

    podzbiory_k_elementowe(iterator+1,1) = {badany_podzbior};

    iterator = iterator + 1;
end


end
```

Funkcja ta zwraca w postaci macierzy o komórkach typu cell, znalezione k – elementowe podzbiory zbioru {1, . . . , n}.

Poniższy fragment odpowiada za obliczenie k elementowych podzbiorów dla n = 7 i k= 5. Oraz wypisanie pierwszych dziesięciu w Live Script.

Zad.2 cz.1

```
n = 7;

k = 5;

podzbiory_k_elementowe = algorytm_generowania_k_elementowych_podzbiorow(n,k);

dziesiec_pierwszych_podzbiorow = podzbiory_k_elementowe(1:10,1);

for i = 1:size(dziesiec_pierwszych_podzbiorow,1)


    do_wyswietlenia_podzbior = dziesiec_pierwszych_podzbiorow(i,1);

    do_wyswietlenia_podzbior = cell2mat(do_wyswietlenia_podzbior);

    X = ['Podzbior ',num2str(i),' ,',num2str(k),' elementowy',' :
',num2str(do_wyswietlenia_podzbior)];

    disp(X);


end
```

Wynik:

```
  Podzbior 1 ,5 elementowy : 1  2  3  4  5
  Podzbior 2 ,5 elementowy : 1  2  3  4  6
  Podzbior 3 ,5 elementowy : 1  2  3  5  6
  Podzbior 4 ,5 elementowy : 1  2  4  5  6
  Podzbior 5 ,5 elementowy : 1  3  4  5  6
  Podzbior 6 ,5 elementowy : 2  3  4  5  6
  Podzbior 7 ,5 elementowy : 1  2  3  4  7
  Podzbior 8 ,5 elementowy : 1  2  3  5  7
  Podzbior 9 ,5 elementowy : 1  2  4  5  7
  Podzbior 10 ,5 elementowy : 1  3  4  5  7
```

1.3. Zaimplementować algorytm generowania permutacji zbioru {1, . . . , n}. Wypisz 10 kolejnych permutacji zbioru {1, 2, 3, 4, 5, 6} poczynając od permutacji (456321).

Poniższy kod odzwierciedla implementacje algorytmu generowania permutacji zbioru {1, . . . , n} (nazwa pliku: algorytm_generowania_permutacji_zbioru.m):

```
function permutacje_zbioru = algorytm_generowania_permutacji_zbioru(n)

permutacje_zbioru = {};
permutacje_zbioru(1,1) = {[1:n]};
iterator = 1;

while 1

    badany_podzbior = permutacje_zbioru(iterator,1);
    badany_podzbior = cell2mat(badany_podzbior);
    znaleziony_index = 0;

    for i = n:-1:2

        if badany_podzbior(i-1) <  badany_podzbior(i)
            znaleziony_index = i-1;
            break
        end
```

```
    end

    if znaleziony_index == 0
        return
    else
        a_j = badany_podzbior(1,znaleziony_index);
        zbior_pom = badany_podzbior(1,(znaleziony_index+1):size(badany_podzbior,2));
        zbior_pom = zbior_pom(zbior_pom>a_j);
        najmniejsza_wartosc_zbior_pom = min(zbior_pom);
        indeks_najmniejszej_wartosci_zbior_pom =
find(badany_podzbior==najmniejsza_wartosc_zbior_pom);

        for i = 1:size(indeks_najmniejszej_wartosci_zbior_pom,2)

            if indeks_najmniejszej_wartosci_zbior_pom(i) > znaleziony_index
                badany_podzbior(1,znaleziony_index) = najmniejsza_wartosc_zbior_pom;
                badany_podzbior(1,indeks_najmniejszej_wartosci_zbior_pom) = a_j;
                wartosci_od_ajplusjeden_do_an =
badany_podzbior(1,(znaleziony_index+1):size(badany_podzbior,2));
                wartosci_od_ajplusjeden_do_an_flip =
flip(wartosci_od_ajplusjeden_do_an);
                badany_podzbior(1,(znaleziony_index+1):size(badany_podzbior,2)) =
wartosci_od_ajplusjeden_do_an_flip;
                break;
            end

        end

    end

    permutacje_zbioru(iterator+1,1) = {badany_podzbior};

    iterator = iterator + 1;
end

end
```

Funkcja ta zwraca w postaci macierzy o komórkach typu cell, znalezione permutacje zbioru {1, . . . , n}.

Poniższy fragment odpowiada za obliczenie permutacji zbioru dla n = 6. Oraz wypisanie pierwszych dziesięciu poczynając od poszukiwanej 456321 (za pomocą funkcji znajdz, tej samej co w przypadku algorytmu generowania podzbiorów) w Live Script.

Zad.3 cz.1

```
n = 6;

permutacje_zbioru = algorytm_generowania_permutacji_zbioru(n);

szukana_permutacja = [4,5,6,3,2,1];

index = znajdz(permutacje_zbioru,szukana_permutacja);

jak_wiele = index:1:index+9;

dziesiec_kolejnych_permutacji = permutacje_zbioru(jak_wiele,1);

for i = 1:size(dziesiec_kolejnych_permutacji,1)


    do_wyswietlenia_permutacja = dziesiec_kolejnych_permutacji(i,1);

    do_wyswietlenia_permutacja = cell2mat(do_wyswietlenia_permutacja);

    X = ['Permutacja ',num2str(i),' : ',num2str(do_wyswietlenia_permutacja)];

    disp(X);
```

```
end
```

```
  Permutacja 1 : 4  5  6  3  2  1
  Permutacja 2 : 4  6  1  2  3  5
  Permutacja 3 : 4  6  1  2  5  3
  Permutacja 4 : 4  6  1  3  2  5
  Permutacja 5 : 4  6  1  3  5  2
  Permutacja 6 : 4  6  1  5  2  3
  Permutacja 7 : 4  6  1  5  3  2
  Permutacja 8 : 4  6  2  1  3  5
  Permutacja 9 : 4  6  2  1  5  3
  Permutacja 10 : 4  6  2  3  1  5
```

2. Część 2

Uwaga: Badany graf jest grafem spójnym, więc można zastosować do niego algorytmy przeszukiwania w głąb jak i wszerz!

2.1. Przeszukiwanie w głąb

Poniższy kod odzwierciedla implementacje przeszukiwania w głąb (nazwa pliku: algorytm_przeszukiwania_grafu_w_glab.m):

```
function [kolejne_kroki_stosu,zbior_E,zbior_E_kolejne_kroki] =
algorytm_przeszukiwania_grafu_w_glab(wierzcholek_poczatkowy,polaczenia)

stos = [wierzcholek_poczatkowy];

odwiedzone_wierzcholki = [wierzcholek_poczatkowy];

kolejne_kroki_stosu = {stos};
zbior_E = [];
zbior_E_kolejne_kroki = {{}};

numer_iteracji = 1;

disp(['Iteracja: ', num2str(numer_iteracji)]);
disp('Stos w obecnej iteracji:');
disp(stos);
disp('Zbior E:');
disp(zbior_E);
disp(' ');

while isempty(stos) == 0

    aktualnie_rozpatrywany_wierzcholek = stos(1,1);

    kolejny_wierzcholek_do_odwiedzenia =
sprawdz_czy_istnieje_nastepny_do_odwiedzenia_stos...
    (aktualnie_rozpatrywany_wierzcholek,polaczenia,odwiedzone_wierzcholki);

    czy_istnieje_kolejny_wierzcholek_do_odwiedzenia =
isempty(kolejny_wierzcholek_do_odwiedzenia);

    if czy_istnieje_kolejny_wierzcholek_do_odwiedzenia == 0

        odwiedzone_wierzcholki =
[odwiedzone_wierzcholki;kolejny_wierzcholek_do_odwiedzenia];
        obecne_E = [stos(1,1),kolejny_wierzcholek_do_odwiedzenia];

        zbior_E = [zbior_E; obecne_E];

        stos = [kolejny_wierzcholek_do_odwiedzenia;stos];

        kolejne_kroki_stosu{end+1,1} = stos;
```

```
    else

        stos = stos(2:end);
        kolejne_kroki_stosu{end+1,1} = stos;

    end

    numer_iteracji = numer_iteracji + 1;

    zbior_E_kolejne_kroki{end+1,1} = zbior_E;

    disp(['Iteracja: ', num2str(numer_iteracji)]);
    disp('Stos w obecnej iteracji:');
    disp(stos);
    disp('Zbior E:');
    disp(zbior_E);
    disp(' ');

end

end
```

Poniższy fragment opisuje stworzoną funkcję do wyszukiwania kolejnego wierzchołka do stosu (nazwa pliku: sprawdz_czy_istnieje_nastepny_do_odwiedzenia_stos.m):

```
function kolejny_wierzcholek_do_odwiedzenia =
sprawdz_czy_istnieje_nastepny_do_odwiedzenia_stos...
    (aktualnie_rozpatrywany_wierzcholek,polaczenia,odwiedzone_wierzcholki)

kolejny_wierzcholek_do_odwiedzenia = [];

for i = 1:size(polaczenia,1)

    badane_polaczenie = polaczenia{i,1};
    badane_polaczenie_wierzcholek = badane_polaczenie{1,1};
    badane_polaczenie_wektor_polaczen = badane_polaczenie{1,2};

    if badane_polaczenie_wierzcholek == aktualnie_rozpatrywany_wierzcholek

        for j = 1:size(badane_polaczenie_wektor_polaczen,2)

            czy_byl_juz_odwiedzany =
isempty(find(odwiedzone_wierzcholki==badane_polaczenie_wektor_polaczen(1,j)));

            if czy_byl_juz_odwiedzany == 1

                kolejny_wierzcholek_do_odwiedzenia =
badane_polaczenie_wektor_polaczen(1,j);
                return

            end

        end

    end

end


end
```

Poniżej wywołanie w LiveScript, na początku jest tworzona macierz z połączeniami i określany wierzchołek początkowy :

Zad.1 cz.2 - DFS

```
%Badany graf jest grafem spójnym, więc można zastosować do niego algorytmy
%przeszukiwania w głąb.
polaczenia = {{"a",["b","d","h"]};{"b",["a","h"]};{"c",["h"]};{"d",["a","h"]};...

{"e",["f","h"]};{"f",["g","h","i"]};{"g",["f","i"]};{"h",["a","b","c","d","e","f"]};...
    {"i",["f","g"]}};

wierzcholek_poczatkowy = "a";

[kolejne_kroki_stosu_w_glab,zbior_E_DFS,zbior_E_kolejne_kroki_DFS] =
algorytm_przeszukiwania_grafu_w_glab(wierzcholek_poczatkowy...
    ,polaczenia);
```

W celu łatwiejszej czytelności wyników następne kroki umieściłem w arkuszu kalkulacyjnym:

	Stos dla drzewa DFS																	
Nr kroku	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
	a	b	h	c	h	d	h	e	f	g	i	g	f	e	h	b	a	pusty
		a	b	h	b	h	b	h	e	f	g	f	e	h	b	a		
			a	b	a	b	a	b	h	e	f	e	h	b	a			
				a		a		a	b	h	e	h	b	a				
									a	b	h	b	a					
										a	b	a						
											a							

	Stan zbioru krawędzi dla DFS																	
Nr kroku	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
	pusty	[a b]	[a b]	[a b]	[a b]	[a b]	[a b]	[a b]	[a b]	[a b]	[a b]	[a b]	[a b]	[a b]	[a b]	[a b]	[a b]	[a b]
			[b h]	[b h]	[b h]	[b h]	[b h]	[b h]	[b h]	[b h]	[b h]	[b h]	[b h]	[b h]	[b h]	[b h]	[b h]	[b h]
				[h c]	[h c]	[h c]	[h c]	[h c]	[h c]	[h c]	[h c]	[h c]	[h c]	[h c]	[h c]	[h c]	[h c]	[h c]
						[h d]	[h d]	[h d]	[h d]	[h d]	[h d]	[h d]	[h d]	[h d]	[h d]	[h d]	[h d]	[h d]
								[h e]	[h e]	[h e]	[h e]	[h e]	[h e]	[h e]	[h e]	[h e]	[h e]	[h e]
									[e f]	[e f]	[e f]	[e f]	[e f]	[e f]	[e f]	[e f]	[e f]	[e f]
										[f g]	[f g]	[f g]	[f g]	[f g]	[f g]	[f g]	[f g]	[f g]
											[g i]	[g i]	[g i]	[g i]	[g i]	[g i]	[g i]	[g i]

2.2. Przeszukiwanie wszerz

Poniższy kod odzwierciedla implementacje przeszukiwania wszerz (nazwa pliku: algorytm_przeszukiwania_grafu_wszerz.m):

```
function [kolejne_kroki_kolejki,zbior_E,zbior_E_kolejne_kroki] =
algorytm_przeszukiwania_grafu_wszerz(wierzcholek_poczatkowy,polaczenia)

kolejka = [wierzcholek_poczatkowy];

odwiedzone_wierzcholki = [wierzcholek_poczatkowy];

kolejne_kroki_kolejki = {kolejka};
zbior_E = [];

zbior_E_kolejne_kroki = {{}};

numer_iteracji = 1;

disp(['Iteracja: ', num2str(numer_iteracji)]);
```

```
disp('Kolejka w obecnej iteracji:');
disp(kolejka);
disp('Zbior E:');
disp(zbior_E);
disp(' ');

while isempty(kolejka) == 0

    aktualnie_rozpatrywany_wierzcholek = kolejka(1,1);

    kolejne_wierzcholki_do_odwiedzenia =
sprawdz_czy_istnieje_nastepny_do_odwiedzenia_kolejka...
    (aktualnie_rozpatrywany_wierzcholek,polaczenia,odwiedzone_wierzcholki);

    czy_istnieje_kolejny_wierzcholek_do_odwiedzenia =
isempty(kolejne_wierzcholki_do_odwiedzenia);

    if czy_istnieje_kolejny_wierzcholek_do_odwiedzenia == 0

        odwiedzone_wierzcholki =
[odwiedzone_wierzcholki;kolejne_wierzcholki_do_odwiedzenia];

        obecne_E = [];

        for i = 1:size(kolejne_wierzcholki_do_odwiedzenia,1)

            obecne_E_do_dodania =
[kolejka(1,1),kolejne_wierzcholki_do_odwiedzenia(i,1)];
            obecne_E = [obecne_E;obecne_E_do_dodania];

        end

        zbior_E = [zbior_E; obecne_E];

        kolejka = [kolejka;kolejne_wierzcholki_do_odwiedzenia];
        kolejne_kroki_kolejki{end+1,1} = kolejka;

    else

        kolejka = kolejka(2:end);
        kolejne_kroki_kolejki{end+1,1} = kolejka;

    end

    numer_iteracji = numer_iteracji + 1;

    zbior_E_kolejne_kroki{end+1,1} = zbior_E;

    disp(['Iteracja: ', num2str(numer_iteracji)]);
    disp('Kolejka w obecnej iteracji:');
    disp(kolejka);
    disp('Zbior E:');
    disp(zbior_E);
    disp(' ');


end


end
```

Poniższy fragment opisuje stworzoną funkcję do wyszukiwania kolejnych wierzchołków do kolejki (nazwa pliku: sprawdz_czy_istnieje_nastepny_do_odwiedzenia_kolejka.m):

```
function kolejne_wierzcholki_do_odwiedzenia =
sprawdz_czy_istnieje_nastepny_do_odwiedzenia_kolejka...
    (aktualnie_rozpatrywany_wierzcholek,polaczenia,odwiedzone_wierzcholki)
```

```
kolejne_wierzcholki_do_odwiedzenia = [];

for i = 1:size(polaczenia,1)

    badane_polaczenie = polaczenia{i,1};
    badane_polaczenie_wierzcholek = badane_polaczenie{1,1};
    badane_polaczenie_wektor_polaczen = badane_polaczenie{1,2};

    if badane_polaczenie_wierzcholek == aktualnie_rozpatrywany_wierzcholek

        for j = 1:size(badane_polaczenie_wektor_polaczen,2)

            czy_byl_juz_odwiedzany =
isempty(find(odwiedzone_wierzcholki==badane_polaczenie_wektor_polaczen(1,j)));

            if czy_byl_juz_odwiedzany == 1

                kolejne_wierzcholki_do_odwiedzenia =
[kolejne_wierzcholki_do_odwiedzenia;badane_polaczenie_wektor_polaczen(1,j)];

            end

        end

    end

end


end
```

Poniżej wywołanie w Live Script, na początku jest tworzona macierz z połączeniami i określany wierzchołek początkowy:

Zad.1 cz.2 - BFS

```
%Badany graf jest grafem spójnym, więc można zastosować do niego algorytmy
%przeszukiwania wszerz.

polaczenia = {{"a",["b","d","h"]};{"b",["a","h"]};{"c",["h"]};{"d",["a","h"]};...

{"e",["f","h"]};{"f",["g","h","i"]};{"g",["f","i"]};{"h",["a","b","c","d","e","f"]};...
    {"i",["f","g"]}};


wierzcholek_poczatkowy = "a";


[kolejne_kroki_kolejki_wszerz,zbior_E_BFS,zbior_E_kolejne_kroki_BFS] =
algorytm_przeszukiwania_grafu_wszerz(wierzcholek_poczatkowy...
    ,polaczenia);
```

W celu łatwiejszej czytelności wyników następne kroki umieściłem w arkuszu kalkulacyjnym:

	Kolejka dla drzewa BFS												
Nr kroku	1	2	3	4	5	6	7	8	9	10	11	12	13
	a	a	b	d	h	h	c	e	f	f	g	i	pusty
		b	d	h		c	e	f		g	i		
		d	h			e	f			i			
		h				f							
	Stan zbioru krawędzi dla drzewa BFS												
Nr kroku	1	2	3	4	5	6	7	8	9	10	11	12	13
	pusty	[a b]	[a b]	[a b]	[a b]	[a b]	[a b]	[a b]	[a b]	[a b]	[a b]	[a b]	[a b]
		[a d]	[a d]	[a d]	[a d]	[a d]	[a d]	[a d]	[a d]	[a d]	[a d]	[a d]	[a d]
		[a h]	[a h]	[a h]	[a h]	[a h]	[a h]	[a h]	[a h]	[a h]	[a h]	[a h]	[a h]
						[h c]	[h c]	[h c]	[h c]	[h c]	[h c]	[h c]	[h c]
						[h e]	[h e]	[h e]	[h e]	[h e]	[h e]	[h e]	[h e]
						[h f]	[h f]	[h f]	[h f]	[h f]	[h f]	[h f]	[h f]
										[f g]	[f g]	[f g]	[f g]
										[f i]	[f i]	[f i]	[f i]