

Projekt 1  
Matematyka dyskretna  
Symulator gry worldie  
Wykonał:  
Ivan Kaliankovich

Spis treści

Algorytm generowania podzbiorów zbioru .....	2
Algorytm generowania k-elementowych podzbiorów zbioru .....	2
Algorytm generowania permutacji zbioru .....	4
Zadanie 4.....	4

## Algorytm generowania podzbiorów zbioru

Umieszczam poniżej kod fragment kodu zawierający algorytm

```
5  
6 function [A] = podzbior(n,X)  
7     % Inicjalizacja komórki A  
8     A = cell(1,2^n-1);  
9     % Pierwszy podzbiór to zbiór X  
10    A{1} = X;  
11    % Pętla generująca kolejne podzbiory  
12    for i = 2:2^n-1  
13        % Znajdź największy element nie należący do A  
14        a = max(setdiff(1:n,A{i-1}));  
15        % Jeśli nie ma takiego elementu, to koniec  
16        if isempty(a)  
17            break  
18        end  
19        % Dodaj a do zbioru A i usuń elementy większe od a  
20        A{i} = sort([A{i-1} a]);  
21        A{i}(find(A{i}==a)+1:end) = [];  
22    end  
23 end
```

W pliku .m wołam algorytm i wypisuję jego zawartość w konsoli

```
1 A = podzbior(7,[1 2 3 5]);  
2 for i = 1:10  
3     disp(A{i})  
4 end
```

W tym przypadku 7 to jest największy element zbioru głównego, podaję również pierwszy zbiór, od którego generowane są pozostałe zbiory.

## Algorytm generowania k-elementowych podzbiorów zbioru

Kod źródłowy algorytmu:

```

6 function kelem_podzbior(n,p,k)
7 if k>n
8     error('ilość elementów w podzbiorze nie może być większa od liczby elementów w zbiorze głównym')
9 end
10 %macierz do zapisywania podzbiorów
11 podzbior = zeros(p,k);
12 %zbior glowny
13 glowny = 1:n;
14 %pierwszy podzbiór
15 A=1:k;
16 podzbior(1, :) = A ;
17 %generowanie kolejnych podzbiorów
18 for j=2:p
19     imin=k+1;
20     for i=1:k
21         if ~ismember(A(i)+1,A)
22             if i<imin
23                 imin=i;
24             end
25         end
26         if (A(i)==glowny(n)&&A(1)==(n-k+1))
27             disp(podzbior);
28             error('koniec zapisu zapisano juz ostatni podzbiór')
29         end
30     end
31     %zapisanie wartości nowego podzbioru
32     A(1:imin-1)=glowny(1:imin-1);
33     A(imin)=A(imin)+1;
34     podzbior(j, :) = A;
35 end
36 %wypisuje wartości macierzy p x k z podzbiorami
37 disp(podzbior);
38 end

```

Warunki początkowe: \_\_\_\_\_

```

1 p=10;
2 k=5;
3 n=7;
4 kelem_podzbior(n,p,k);

```

## Algorytm generowania permutacji zbioru

```
7 function permutacje(n, p, X)
8     for k = 1:p
9         disp(['Permutacja ', num2str(k), ':']);
10        disp(X);
11
12        % Znajdź największe j, takie że aj < aj+1
13        j = n - 1;
14        while j >= 1 && X(j) > X(j + 1)
15            j = j - 1;
16        end
17
18        % Jeśli nie znaleziono j, to to jest ostatnia permutacja
19        if j == 0
20            break;
21        end
22
23        % Znajdź najmniejsze k, takie że ak > aj i k > j
24        k = n;
25        while X(k) <= X(j)
26            k = k - 1;
27        end
28
29        % Zamień aj z ak
30        X([j, k]) = X([k, j]);
31
32        % Odwróć porządek elementów aj+1, ..., an
33        X(j+1:end) = flip(X(j+1:end));
34    end
35 end
```

Warunki początkowe:

```
1 n = 6; % Największy element zbioru
2 p = 10; % Ilość generowanych permutacji
3 X = [4, 5, 6, 3, 2, 1]; % Początkowa permutacja
4
5 permutacje(n, p, X);
6
```

## Zadanie 4

Główną motywacją do zrobienia zadania był fakt, że czasem z kolegami lubimy grać w różnego rodzaju gry ze słowami. Wybrałem zgadywanie słów z przedstawionych liter jako najprostszy z przypadków.

Za podstawę rozwiązywania zadania wybrałem nieco zmodyfikowany algorytm permutacji z zadania 3 powyżej.

```

1 function X = permutacje(n, p, X)
2     for k = 1:p
3         % Znajdź największe j, takie że aj < aj+1
4         j = n - 1;
5         while j >= 1 && X(j) > X(j + 1)
6             j = j - 1;
7         end
8
9         % Jeśli nie znaleziono j, to to jest ostatnia permutacja
10        if j == 0
11            break;
12        end
13
14        % Znajdź najmniejsze k, takie że ak > aj i k > j
15        k = n;
16        while X(k) <= X(j)
17            k = k - 1;
18        end
19
20        % Zamień aj z ak
21        X([j, k]) = X([k, j]);
22
23        % Odwróć porządek elementów aj+1, ..., an
24        X(j+1:end) = flip(X(j+1:end));
25    end
26 end

```


Teraz funkcja permutację zwraca ostateczny zbiór po p iteracjach algorytmu oraz nic nie wypisuje.

Gra zaczyna się od użytkownika, który jest proszony o wpisanie dowolnego słowa do konsoli. Po wpisaniu słowa wyskakują okienko dla użytkownika 2 ze słowem o zamienionej kolejności liter w stosunku do słowa wprowadzonego przez użytkownika 1. Po zobaczeniu słowa, użytkownik 2 ma nieskończoną ilość prób do zgadnięcia oryginalnego słowa. W dowolnym momencie użytkownik może wpisać słowo kluczowe „pomoc”, wtedy pod permutowanym słowem pojawi nowe, pomocnicze, w którym pierwsze dwie litery są prawidłowe, natomiast reszta liter jest jeszcze raz permutowana.



ttmamkyeaa  
matktaemay

Tak wygląda interfejs w tym przypadku, początkowym słowem jest matematyka.



ttmamkyeaa  
matktaemay  
Zgadłeś!

W przypadku zgadnięcia słowa pojawia się komunikat „Zgadłeś”

Poniżej przedstawiam kod programu:

```
1 N_PERMUTACJI = 40;
2
3 disp("Witam pierwszego użytkownika");
4 word = input('Proszę wprowadzić słowo: ', "s");
5
6 % permutacja podanego słowa
7 word_length = length(word);
8 permut_array = 1:word_length;
9 permut = permutacje(word_length, N_PERMUTACJI, permut_array);
10 disp(permut);
11 new_word = word(permut);
12
13 % narysowanie nowego słowa
14 x = 0.3;
15 y = 0.4;
16 new_word_print = sprintf(new_word);
17 annotation('textbox', [x, y, 0.2, 0.2], 'String', new_word_print, ...
18     'FitBoxToText', 'on', 'Color', 'blue', 'EdgeColor', 'none', 'FontSize', 30);
19
20 % gra w zgadywanie
21 disp("Zapraszam drugiego użytkownika do ułożenia słowa z liter napisanych na niebiesko");
22 guess = char(zeros(word_length));
23 while strcmp(guess, word) == 0
24     guess = input('Wpisz słowo \nJesli chcesz podpowiedz, wpisz pomocy \n', "s");
25     if guess == "pomocy"
26         break
27     end
28
29     if length(guess) ~= word_length
30         guess = char(zeros(word_length));
31         disp("Zła liczba znaków");
32     end
33 end
34
35 if guess == "pomocy"
36     % tworzenie słowa pomocniczego
37     permut_array = 1:word_length-2;
38     permut = permutacje(word_length-2, N_PERMUTACJI, permut_array);
39     modified_word = word(3:end);
40     new_new_word = strcat(word(1:2), modified_word(permut));
41     annotation('textbox', [x, y-0.1, 0.2, 0.2], 'String', new_new_word(1:2), ...
42         'FitBoxToText', 'on', 'Color', 'red', 'EdgeColor', 'none', 'FontSize', 30);
43     annotation('textbox', [x+0.1, y-0.1, 0.2, 0.2], 'String', new_new_word(3:end), ...
44         'FitBoxToText', 'on', 'Color', 'blue', 'EdgeColor', 'none', 'FontSize', 30);
45     disp("Dodałem nowe znaki, pierwsze dwie literki są na swoim miejscu");
46
47     guess = char(zeros(word_length));
48     while strcmp(guess, word) == 0
49         guess = input('Wpisz słowo \nTeraz dwie pierwsze literki są na swoim miejscu \n', "s");
50         if length(guess) ~= word_length
51             guess = char(zeros(word_length));
52             disp("Zła liczba znaków");
53         end
54     end
55 end
56
57 annotation('textbox', [x, y-0.2, 0.2, 0.2], 'String', 'Zgadłeś!', ...
58     'FitBoxToText', 'on', 'Color', 'red', 'EdgeColor', 'none', 'FontSize', 30);
59
```

Znaczącą wadą tego programu jest algorytm permutacji, który wymaga podania liczby iteracji permutacji oraz fakt, że algorytm jest bardzo zależny od macierzy wejściowej.

W trakcie testów przekonałem się, że do takiego rodzaju zagadnienia bardziej pasującym jest rozwiązanie korzystające ze zwykłej funkcji `randperm()` tak jak poniżej

```
1 function wynik = randomizer_test(n, p, k)
2     wynik = zeros(p, k); % Wstawienie pustej macierzy
3
4     % Generowanie podzbiorów
5     for i = 1:p
6         A = randperm(n, k); % Losowy wybór k elementów spośród n
7         wynik(i, :) = A;    % Dodanie podzbioru
8     end
9 end
```

W tym przypadku słowo jest trudniejsze do zgadnięcia.

Algorytm przedstawiony przeze mnie bardziej by się nadawał w przypadkach, kiedy wymagana jak największa ilość podzbiorów utworzonych ze zbioru początkowego. Na przykład program tworzący słowa złożonych z liter innego słowa wejściowego. Natomiast taki program wymagałby dużej bazy danych wszystkich słów.