Sprawozdanie 4

Algorytmy z powracaniem

Andrei Staravoitau, nr 150218,

Informatyka I6, II semestr

1. **Algorytm z powracaniem** (ang. backtracking algorithm) to ogólna technika algorytmiczna,

wykorzystująca rekurencję, która polega na rozwiązywaniu problemu obliczeniowego w

następujący sposób:

   1) Algorytm buduje rozwiązanie problemu stopniowo, po każdym kroku sprawdzając czy

aktualna kombinacja (kandydat na rozwiązanie) jest dopuszczalna i czy jest

poszukiwanym rozwiązaniem problemu.

   2) Jeśli dana kombinacja nie jest dopuszczalna lub nie jest szukanym rozwiązaniem

algorytm powraca do stanu, w którym może wygenerować innego kandydata

i stopniowo buduje dalej.

   3) Jeśli dana kombinacja jest rozwiązaniem problemu algorytm kończy działanie (jeśli

szukał tylko jednego rozwiązania) lub – powracając – generuje kolejne kombinacje

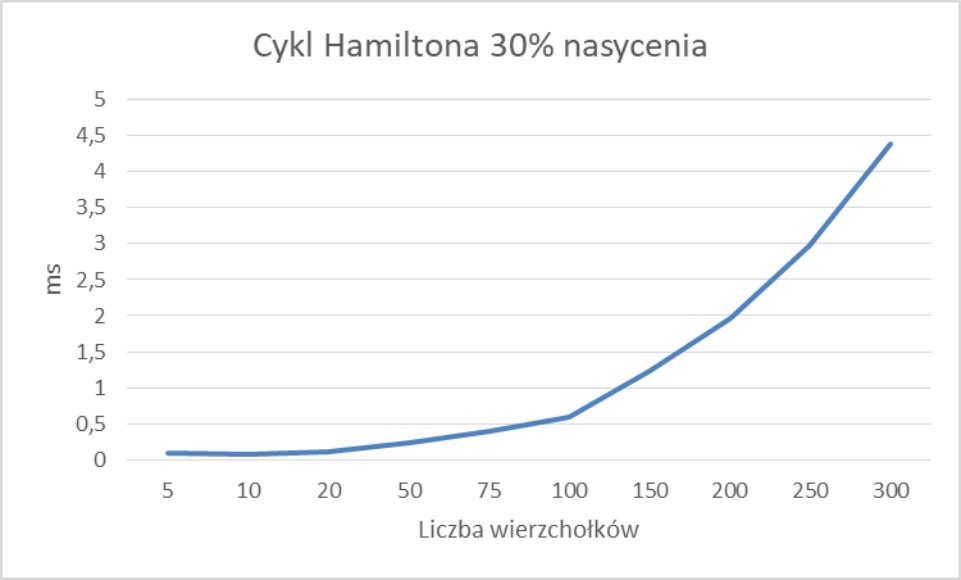
(jeśli szukał wielu rozwiązań).

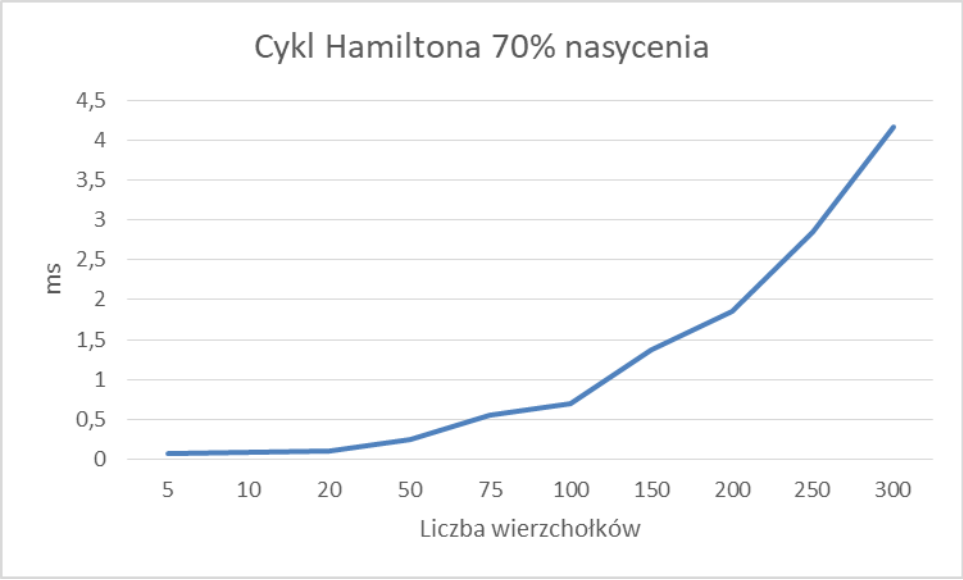
2. **Ścieżka Hamiltona** w grafie G=(V,E) to ścieżka prosta, która przechodzi przez wszystkie wierzchołki tego grafu (tzn. przez każdy wierzchołek grafu przechodzi dokładnie jeden raz).
3. **Cykl Hamiltona** w grafie to zamknięta ścieżka Hamiltona (zaczyna się i kończy w tym samym wierzchołku).
   Problem złożoności czasowej znajdowania rozwiązania problemu grafu hamiltonowskiego wiąże się z brakiem twierdzenia takiego jak twierdzenie Eulera dla grafów Eulera. Owo twierdzenie pozwala w czasie liniowym (tj. zależnym liniowo od, w tym przypadku, liczby wierzchołków) znaleźć odpowiedź na pytanie, czy graf jest eulerowski. W przypadku grafów Hamiltona twierdzenie takie prawdopodobnie nie istnieje. Problem poszukiwania cyklu Hamiltona rozwiązałem za pomocą algorytmu Robertsa-Floresa.
   Problem jest z klasy NP-zupełne, Algorytmy pseudowielomianowe lub wykładnicze mają złożoność obliczeniową: $O(n*x)$, $O(x^n)$

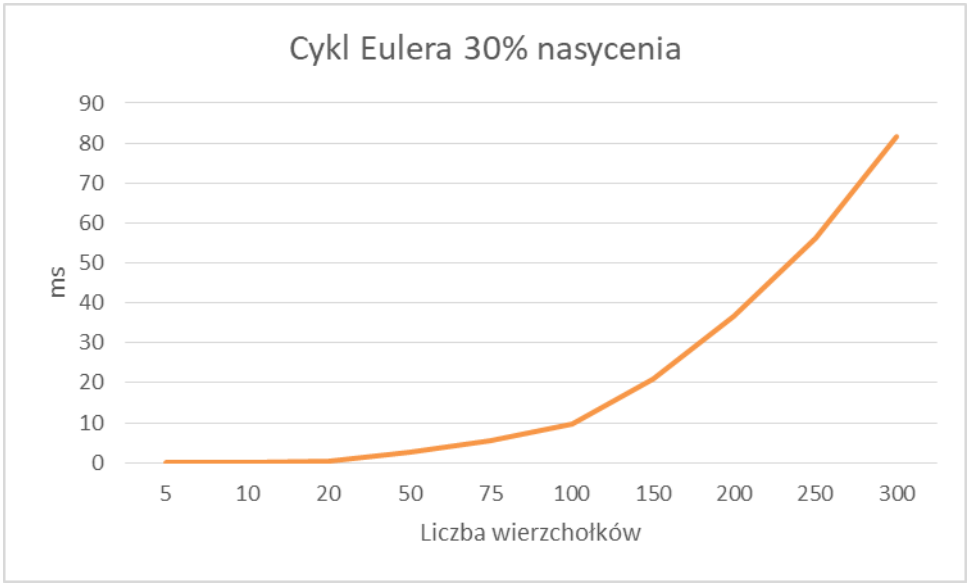
4. **Ścieżka Eulera** w grafie G=(V,E) to ścieżka prosta, która przechodzi przez wszystkie krawędzie tego grafu; przez każdą krawędź dokładnie jeden raz.

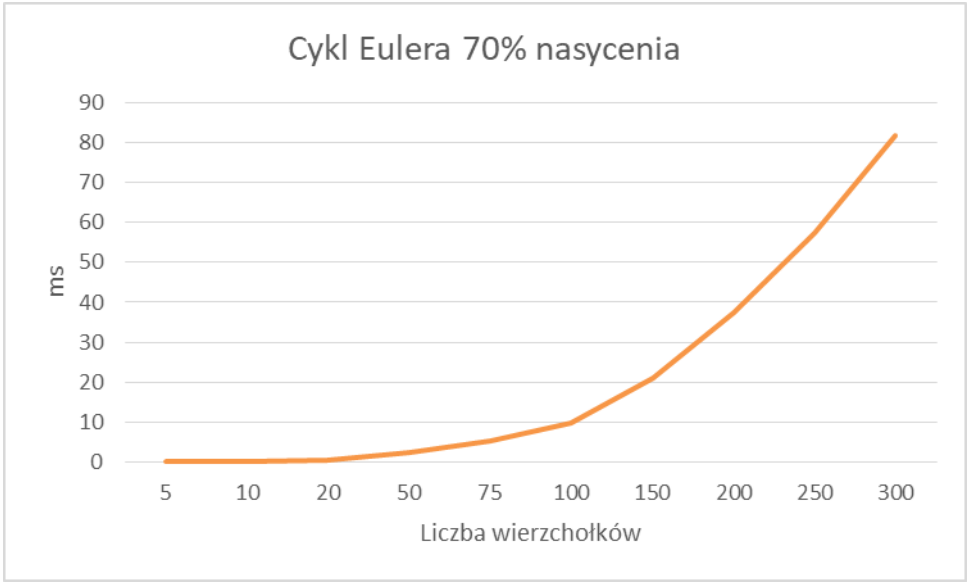
5. **Cykl Eulera** w grafie to zamknięta ścieżka Eulera (zaczyna się i kończy w tym samym wierzchołku).
6. Przez mnie została wybrana prezentacja grafu przez macierz sąsiedztwa, dlatego że ją można lekko wykorzystać w obuch algorytmach przeszukiwania cykli oraz, moim zdaniem, daje jak najwięcej informacji o grafie.

Wykresy znajdowania istniejących cykli w grafach



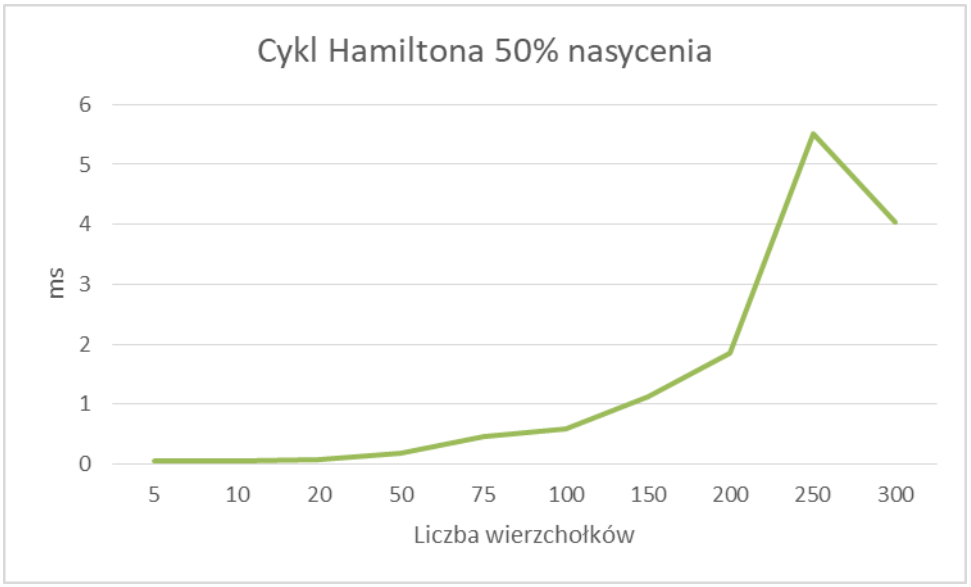


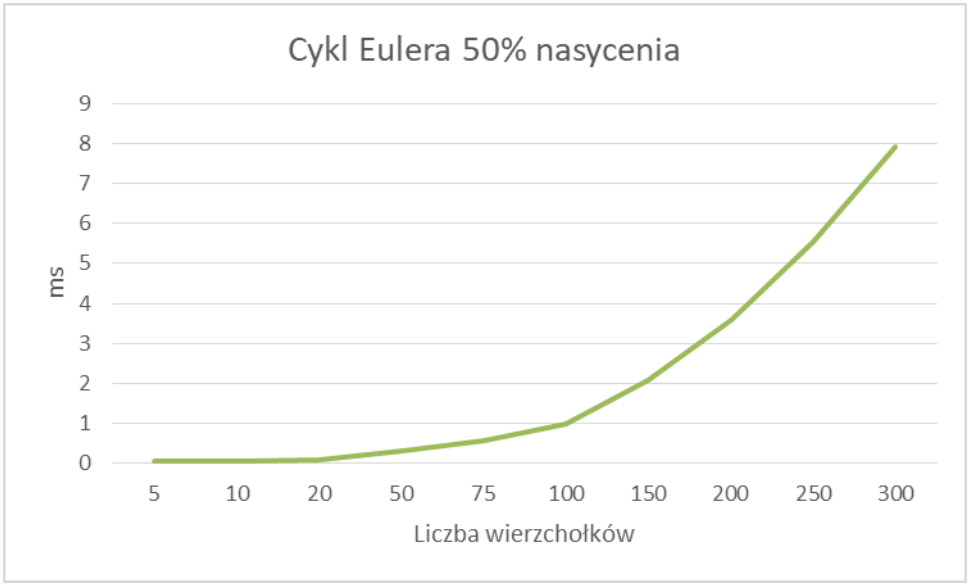




Jak widać na przedstawionych wykresach, nasycenie krawędziami grafów nie ma znacznego wpływu na czas przeszukiwania cykli w grafach.

Wykresy znajdowania nieistniejących cykli w grafach





Jak widać z przedstawionych wyżej wykresów, brak cykli w grafie powoduje znaczne zmniejszenie czasu znalezienia cyklu Eulera, chociaż nie ma dużego znaczenia pod czas szukania cyklu Hamiltona.

Code

```
from timeit import default_timer as timer

import copy

'''

ham i eul

0 1 0 0 0 1

1 0 0 0 1 0

0 0 0 1 1 0
```

0 0 1 0 0 1

0 1 1 0 0 0

1 0 0 1 0 0

ham

0 0 0 1 0 1

0 0 0 1 1 0

0 0 0 0 1 1

1 1 0 0 0 0

0 1 1 0 0 1

1 0 1 0 1 0

eul

0 1 0 1 1 1

1 0 0 0 1 0

0 0 0 0 1 1

1 0 0 0 1 0

1 1 1 1 0 0

1 0 1 0 0 0

nic

0 1 0 1 1 0

1 0 0 0 1 0

0 0 0 0 1 1

1 0 0 0 1 0

1 1 1 1 0 0

0 0 1 0 0 0

'''

macierz_sasiedstwa = []

sciezka = []

```
istnieje = False

def cyklHamiltona_niesk(wierzch):
    sciezka.append(wierzch)
    zwiedzony[wierzch] = True
    for i in range(n):
        if macierz_sasiedstwa[wierzch][i] == 1 and not zwiedzony[i]:
            return cyklHamiltona_niesk(i)

    if len(sciezka) == n:
        if macierz_sasiedstwa[sciezka[0]][sciezka[-1]] == 1:
            print("Istnieje cykl, ",sciezka)
            istnieje = True
        else:
            print("Nie ma cykli")

    zwiedzony[wierzch] = False
    sciezka.pop()

    return "Nie ma (więcej) ścieżek Hamiltona"

def cyklEulera_niesk(check):
    for i in range(n):
        count = 0
        for j in range(n):
            if macierz_sasiedstwa[i][j] == 1:
                count += 1
        if count % 2 != 0:
            check += 1
```

```
    if check == 0:
        macCheck = copy.deepcopy(macierz_sasiedstwa)
        sciezkaEulera_niesk(0,macCheck)
    else:
        print("Nie ma cyklu Eulera")

def sciezkaEulera_niesk(wierzch, macCheck):
    for i in range(n):
        if macCheck[wierzch][i] == 1:
            macCheck[wierzch][i] = 0
            macCheck[i][wierzch] = 0
            sciezkaEulera_niesk(i, macCheck)
    sciezka.append(wierzch)

def creategraf_ham(liczbawierzch):
    for i in range(liczbawierzch):
        macierz_sasiedstwa.append([])
        for j in range(liczbawierzch):
            if (j == i+1 or j == i-1) or (i==0 and j == liczbawierzch-1) or (i==liczbawierzch-1 and j == 0):
                macierz_sasiedstwa[i].append(1)
            else:
                macierz_sasiedstwa[i].append(0)
    macierz_indeksów = []
    for x in range(liczbawierzch):
        for j in range(liczbawierzch):
            if macierz_sasiedstwa[x][j]==0 and x!=j:
                macierz_indeksów.append([x,j])
    i=0
    for a in range(len(macierz_indeksów)):
        for b in range(len(macierz_indeksów)):
```

```
    if ([macierz_indeksów[a][0], b] in macierz_indeksów and
        [macierz_indeksów[a][1], b] in macierz_indeksów and
        [b, macierz_indeksów[a][1]] in macierz_indeksów and
        [b, macierz_indeksów[a][0]] in macierz_indeksów and
        [macierz_indeksów[a][0], macierz_indeksów[a][1]] in macierz_indeksów and
        [macierz_indeksów[a][1], macierz_indeksów[a][0]] in macierz_indeksów):
      macierz_sasiedstwa[macierz_indeksów[a][0]][b] = 1
      macierz_sasiedstwa[b][macierz_indeksów[a][0]] = 1

      macierz_sasiedstwa[macierz_indeksów[a][1]][b] = 1
      macierz_sasiedstwa[b][macierz_indeksów[a][1]] = 1

      macierz_sasiedstwa[macierz_indeksów[a][0]][macierz_indeksów[a][1]] = 1
      macierz_sasiedstwa[macierz_indeksów[a][1]][macierz_indeksów[a][0]] = 1



      for i in range(len(macierz_indeksów)):
        if (macierz_indeksów[i] == [b,macierz_indeksów[a][0]] or
            macierz_indeksów[i]==[b, macierz_indeksów[a][1]] or
            macierz_indeksów[i]==[macierz_indeksów[a][1],b] or
            macierz_indeksów[i]==[macierz_indeksów[a][0],b]):
          macierz_indeksów[i][0]=-1
          macierz_indeksów[i][1]=-1
      macierz_indeksów[a][1] = -1
      macierz_indeksów[a][0] = -1



      i+=3
    if i>=nasycenie: break
a+=1
```

```
    i=0
    return


print("Liczba wierzchołków (0 - exit): ")
n=int(input())
while n!=0:
    zwiedzony = [False] * n
    a=0

    print("1 - wpisać ręcznie (macierz sąsiedztwa)\n2 - losowy graf\n0 - exit")
    u=int(input())
    while u!=0:
        macierz_sasiedstwa = []
        if u == 1:
            print("Wiersze macierzy sąsiedztwa:")
            for i in range(n):
                macierz_sasiedstwa.append(list(map(int, input().split())))
            print("1 - Cycl Hamiltona\n2 - Cykl Eulera\n3 - macierz sąsiedztwa\n0 - exit")
            v = int(input())
            while v != 0:
                if v == 1:
                    sciezka = []
                    zwiedzony = [False] * n
                    print(cyklHamiltona_niesk(0))
                if v == 2:
                    macCheck = []
                    sciezka = []
                if v == 3:
                    for i in range(n):
```

```
            for j in range(n):
                print(macierz_sasiedstwa[i][j], end=' ')
            print()
        cyklEulera_niesk(0)
        if sciezka:
            print(sciezka)
    print("1 - Cycl Hamiltona\n2 - Cykl Eulera\n3 - macierz sąsiedztwa\n0 - exit")
    v = int(input())

elif u == 2:
    print("1. Graf z cyklem Hamiltona\n2. Graf bez cyklu")
    g = int(input())
    if g==1:
        print("1. 30% nasycenia krawędziami\n2. 70% nasycenia krawędziami")
        q = int(input())
        if q==1:
            nasycenie = 0.3*(n*(n-1)/2)
        elif q == 2:
            nasycenie = 0.7*(n*(n-1)/2)

        createGraf_ham(n)
        print(macierz_sasiedstwa)
    elif g==2:
        nasycenie = 0.5*(n*(n-1)/2)
        createGraf_ham(n)
        for i in range(n):
            macierz_sasiedstwa[i][n-1]=0
            macierz_sasiedstwa[n-1][i]=0
        print(macierz_sasiedstwa)
    print("1 - Cycl Hamiltona\n2 - Cykl Eulera\n3 - macierz sąsiedztwa\n0 - exit")
```

```
        v = int(input())
        while v!=0:
            if v == 1:
                sciezka = []
                zwiedzony = [False] * n
                start = timer()
                print(cyklHamiltona_niesk(0))
                stop = timer()
                print("%s ms" % ((stop - start) * (1000)))
            if v == 2:
                macCheck = []
                sciezka = []
                start = timer()
                cyklEulera_niesk(0)
                stop = timer()
                print("%s ms" % ((stop - start) * (1000)))
                if sciezka:
                    print(sciezka)
            if v == 3:
                for i in range(n):
                    for j in range(n):
                        print(macierz_sasiedstwa[i][j], end = ' ')
                    print()
            print("1 - Cycl Hamiltona\n2 - Cykl Eulera\n3 - macierz sąsiedztwa\n0 - exit")
            v = int(input())

    print("1 - wpisać ręcznie (macierz sąsiedztwa)\n2 - losowy graf\n0 - exit")
    u=int(input())
print("Liczba wierzchołków (0 - exit): ")
n = int(input())
```