# Rozdział 1

# Lista 1

## 1.1.

## 1.2.

## 1.3.

## 1.4.

$$a_1 = a,\ a_k = 1,\ a_{i+1} = \lfloor \frac{a_i}{2} \rfloor$$

$$b_a = b,\ b_{i+1} = 2b_i \Rightarrow b_i = 2^i b$$

Niech $a = \sum_{i=1}^{k} 2^{i-1}\bar{a}_i,\ \bar{a}_i \in \{0, 1\}$, czyli zapis w postaci binarnej. Wtedy

$$a_n = \sum_{i=1}^{n} 2^{i-1}\bar{a}_{i+(k-n)}$$

Dowód:

$$\sum_{i=1,\ nieparzyste\ a_i}^{k} b_i = \sum_{i=1}^{k} \bar{a}_i 2^i b = b\sum_{i=1}^{k} 2^i \bar{a}_i = ab$$

Kryterium jednorodne - koszt każdej operacji jest jednostkowy. Zatem ile jedynek w zapisie binarnym liczby $b$, taką mamy złożoność $\Rightarrow O(\lceil (log_2 b) \rceil)$
Kryterium lagarytmiczne - koszt operacji zależy od odługości operandów. Dodawań mamy tyle samo $\Rightarrow O(\lceil log_2 b \rceil \cdot \lceil log_2 ab \rceil)$

## 1.5.

Niech $f_n = A_k f_{n-k} + A_{k-1} f_{n-k+1} + \dots A_1 f_{n-1}$. Jeżeli $f_n$ zależy od $k$ poprzednich lementów to tworzymy następującą macierz $k \times k$(x-kolumny, y-wiersze, indeksowanie od 0)

$$M_{x,y} = 1,\ dla\ x = y + 1$$

$$M_{x,k} = A_{n-k+x}$$

Reszta elementów to 0. Przykład dla $k = 3$

$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ A_{n-3} & A_{n-2} & A_{n-1} \end{bmatrix}$$

Żeby policzyć $f_n$ tworzymy wektor $F = (f_{n-k} \dots f_{n-1})$ i obliczamy $M \cdot F^T$.

Tworzymy macierz rozmiaru $(k+m) \times (k+m)$, $k$ - ilość poprzednich wyrazów ciągu, $m$ - stopień wielomianu. Dzielimy ją na cztery prostokąty. Lewy górny taki jak w poprzedniej części. Prawy górny wypełniamy zerami, oprócz ostatniego wiersza, który wypełniamy współczynnikami wielomianu. Lewy dolny wypełniamy zerami. Żeby wypełnić prawy dolny zastanówmy się najpierw jak uzyskać $(n+1)^i$. Oczywiście z dwumianu newtona. prawy dolny prostokąt będzie miał postać

$$\begin{bmatrix} \binom{m}{m} & \dots & \dots & \binom{m}{0} \\ 0 & \binom{m-1}{m-1} & \dots & \binom{m-1}{0} \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 1 \end{bmatrix}$$

Znowu tworzymy sobie wektor $F = (f_{n-k} \cdots f_{n-1}, n^m, n^{m-1}, \dots, 1)$ i obliczamy $M \cdot F^T$.

## 1.6.

## 1.7.

Mamy daną listę $L$. Dzielimy ją na podlisty długości $\sqrt(n)$. Tworzymy dodatkową listę $K$ o długości $\sqrt(n)$, zawierającą wskaźniki na pierwsze elementy utworzonych wcześniej podlist. Przy wstawianiu elementu przeglądamy najpierw Listę $K$, a następnie listę $L$ od miejsca, na które wskazywał wskaźnik z $K$. Maksymalnie przejrzymy $2\sqrt(n)$ elementów. Po wstawieniu elementu musimy ouaktualnić listę wskaźników $K$, czego koszt to znowu $\sqrt(n)$

## 1.8.

Wejście: Skierowany acykliczny graf.
Wyjście: Długość najdłuższej ścieżki.
LengthTo - tablica $|V(G)|$ elementów początkowo równych 0.
TopOrder(G) - posortowane topologicznie wierzchołki.

```
for each vertex V in topOrder(G) do
    for each edge (V, W) in E(G) do
        if LengthTo[W] <= LengthTo[V] + weight(G,(V,W)) then
           LengthTo[W]  = LengthTo[V] + weight(G,(V,W))

return max(LengthTo[V] for V in V(G))
```

Sortowanie topologiczne działa w czasie $O(E+V)$, więc całość działa w czasie $O(E+V+E+V) = O(E+V)$. Żeby wypisać drogę musimy tylko zapamiętywać, dla których wierzchołków spełniony był IF.

# Rozdział 2

# Lista 2

# Rozdział 3

# Lista 3

## 3.1.

Zadanie to polega na skonstruowaniu szybkiego algorytmu obliczania największego wspólnego dzielnika dwóch dodatnich liczb całkowitych $a$ i $b$. Przed podaniem algorytmu musimy udowodnić podane właściwości:

$$gcd(a,b) = \begin{cases} 2 \cdot gcd(\frac{a}{2}, \frac{b}{2}) & \text{a,b są parzyste;} \\ gcd(a, \frac{b}{2}) & \text{a jest nieparzyste, b jest parzyste;} \\ gcd(\frac{a-b}{2}, b) & \text{a,b są nieparzyste.} \end{cases}$$

### 3.1.1. a)

Jeżeli $a$ i $b$ są parzyste, 2 na pewno jest ich wspólnym dizelnikiem. Jeżeli $a$ jest nieparzyste i $b$ jest parzyste, wiemy że $b$ dzieli się przez 2, a $a$ nie. Więc $gcd(a,b)$ pozostaje takie same dla $a$ i $b/2$. Ostatnia własność wynika z faktu, że dla nieparzystych $a$ i $b$, $(a-b)$ będzie parzyste.Ponieważ $gcd(a-b,b) = gcd(a,b)$ oraz $(a-b)$ jest teraz parzyste, możemy zastosować drugą własność.

### 3.1.2. b) algorytm rekurencyjny

```
procedure gcd(a, b)
Input: Two n-bit integers a,b
Output: GCD of a and b
if a = b:
        return a
else if (a is even and b is even):
        return 2*gcd(a/2, b/2)
else if (a is odd and b is even):
        return gcd(a, b/2)
else if (a is odd and b is odd and a > b):
        return gcd((a-b)/2, b)
else if (a is odd and b is odd and a < b):
        return gcd(a,(b-a)/2)
```

### 3.1.3. c) złożoność

Załóżmy, że $a$ i $b$ są n-bitowymi liczbami. Rozmiar $a$ i $b$ wynosi $2n$ bitów. Wszystkie z czterech ifów, oprócz przypadku gdzie $a$ jest nieparzyste i $b$ jest parzyste, zmniejsza rozmiar $a$ i $b$ do $2n-2$ bitów, gdzie wcześniej wymieniony przypadek zmniejsza ilość bitów do $2n-1$. Każda z operacji wykonuje się w czasie stałym ponieważ dzielimy lub mnożymy przez 2. Dla dwóch przypadków z odejmowaniem, mamy odejmowanie dwóch n-bitowych liczb (złożoność wynosi $c \cdot n$ gdzie n jest wielkością operandu). Zatem najgorszy przypadek czwartego ifa algorytmu przedstawimy jako:

$$\begin{aligned}
T(2n) &= T(n-1) + cn \\
T(2n-1) &= T(2n-2) + cn \\
T(2n-2) &= T(2n-3) + c(n-1) \text{ oba operandy mają długość } n-1 \\
T(2n-3) &= T(2n-4) + c(n-1) \\
&... \\
T(2) &= T(1) + c
\end{aligned}$$

Podstawieniami możemy zapisać $T(2n)$ jako:

$$T(2n) = 2c \cdot \sum_{i=1}^{n} i$$

co daje nam $O(n^2)$ co w porównaniu do $O(n^3)$ czasu działania algorytmu euklidesa jest szybsze.

## 3.2.

PDF

## 3.3.

## 3.4.

## 3.5.

### 3.5.1. a)

W wektorze pamiętamy pierwszą kolumnę oraz pierwszy wiersz bez pierwszego wyrazu. Wektor ten ma rozmiar $2n-1$, więc dodawanie dwóch takich wektorów mamy w $O(n)$.

### 3.5.2. b)

Macierz Toeplitza ma następującą postać blokową:

$$T = \begin{bmatrix} A & B \\ C & A \end{bmatrix}$$

Zadanie polega na pomnożeniu macierzy T przez wektor blokowy $T = \begin{bmatrix} x \\ y \end{bmatrix}$ w czasie mniejszym niż $n^2$. Korzystając z dwóch (wzajemnie dualnych) obserwacji:

$$Ax = A(x + y - y) = A(x + y) - Ay$$

$$Ay = A(x + y - x) = A(x + y) - Ax$$

mamy:

$$
\begin{aligned}
T = \begin{bmatrix} A & B \\ C & A \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} Ax + By \\ Cx + Ay \end{bmatrix} = \\
= \begin{bmatrix} A(x+y) - Ay + By \\ Cx + Ay \end{bmatrix} = \begin{bmatrix} A(x+y) + (B-A)y \\ Cx + Ay \end{bmatrix} = \\
= \begin{bmatrix} A(x+y) + (B-A)y \\ Cx + A(x+y) - Ax \end{bmatrix} = \begin{bmatrix} A(x+y) + (B-A)y \\ A(x+y) + (C-A)x \end{bmatrix}
\end{aligned}
$$

I zauważamy, że złożoność czasowa to $T(n) = 3T(n/2) + O(n)$ gdzie $O(n)$ zamyka sumę liniowych czasów wszystkich dodawań.

Rozwiązując typowe równanie rekurencyjne mamy $T(n) = O(n^{log_2 3}) < O(n^2)$.

## 3.6.

## 3.7.

# Rozdział 4

# Lista 4

# Rozdział 5

# Lista 5