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 $f_n = A_k f_{n-k} + A_{k-1} f_{n-k+1} + \ldots A_1 f_{n-1}$. 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 $F = (f_{n-k} \dots f_{n-1}, n^m, n^{m-1}, \dots, 1)$ 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

Rozdział 4

Lista 4

Rozdział 5

Lista 5