Zadanie 4

Udowodnij, że algorytm mnożenia liczb "po rosyjsku" jest poprawny. Jaka jest jego złożoność czasowa i pamięciowa przy:

- jednorodnym kryterium kosztów,
- logarytmicznym kryterium kosztów?

Rozwiązanie:

Dowód. Zdefiniujmy a_i, b_i:

- $a_1 = a, a_k = 1, a_{i+1} = \lfloor a_i/2 \rfloor \Rightarrow a = \sum_{i=0}^{k} 2^i a_i$, gdzie $a_i \in \{0, 1\}$, czyli zapis a w postaci binarnej.
- $b_1 = b, b_{i+1} = 2b_i \Rightarrow b_i = 2^i b$.

Wtedy $\sum_{i=0,odd(a_i)}^{k} b_i = \sum_{i=0}^{k} a_i 2^i b = b \sum_{i=0}^{k} 2^i a_i = ab$. □

Rozpatrzmy jeszcze złożoności w kryteriach jednorodnym i logarytmicznym.

- Kryterium jednorodne - koszt każdej operacji maszyny RAM jest jednostkowy. Wykonując mnożenie $a \cdot b$, dodajemy do siebie tyle liczb, ile jest w zapisie binarnym liczby b. Złożoność algorytmu wynosi $O(\lceil log_2(b) \rceil)$.
- Kryterium logarytmiczne - koszt każdej operacji maszyny RAM jest równy sumie długości binarnej operandów. Jak wyżej, dodajemy do siebie tyle liczb, jest ile jest w zapisie binarnym liczby b. Natomiast długość operandów wynosi co najwyżej $\lceil log_2(a \cdot b) \rceil$. W takim razie złożoność wynosi $O(\lceil log_2(b) \rceil \cdot \lceil log_2(a \cdot b) \rceil)$.

Zadanie 5

Pokaż w jaki sposób algorytm macierzy obliczania n-tej liczby Fibonacciego można uogólnić na inne ciągi, w których kolejne elementy definiowane są kombinacją liniową skończonej liczby elementów wcześniejszych. Następnie uogólnij swoje rozwiązanie na przypadek, w którym n-ty element ciągu definiowany jest jako suma kombinacji liniowej skończonej liczby elementów wcześniejszych oraz wielomianu zmiennej n.

Rozwiązanie: Uogólniamy algorytm macierzowy obliczania n-tej liczby Fibonacciego na inne ciągi, w których kolejne elementy są kombinacją liniową skończonej liczby elementów wcześniejszych. Inaczej, szukamy macierzy A, że:

$$A \cdot \begin{bmatrix} a_{k+l-1} \\ \vdots \\ a_k \end{bmatrix} = \begin{bmatrix} a_{k+l} \\ \vdots \\ a_{k+1} \end{bmatrix}$$

gdzie $a_{k+l} = \alpha_0 a_k + \cdots + \alpha_l a_{k+l-1}$. Za pomocą operacji elementarnych sprawdzamy, że:

$$A = \begin{bmatrix} a_l & a_{l-1} & \cdots & a_0 \\ 1 & 0 & \cdots & 0 \\ 0 & 1 & \ddots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix}$$

Teraz, chcemy uogólnić to rozwiązanie na przypadek, gdzie $a_n = \alpha_l a_{n-1} + \cdots + \alpha_0 a_{n-k} + W(n)$. Wiemy, że wielomian ten jest postaci $W(n) = b_0 n^0 + b_1 n^1 + \cdots + b_s n^s$. Wobec tego szukamy takiej macierzy B, że:

$$B \cdot \begin{bmatrix} a_{n-1} \\ a_{n-2} \\ \vdots \\ a_{n-k} \\ 1 \\ n \\ n^2 \\ \vdots \\ n^s \end{bmatrix} = \begin{bmatrix} a_n \\ a_{n-1} \\ \vdots \\ a_{n-k+1} \\ 1 \\ n+1 \\ (n+1)^2 \\ \vdots \\ (n+1)^s \end{bmatrix}$$

Wiemy, że $(n+1)^k = \sum_{i=0}^{k} \binom{n}{k} n^i$. W takim razie z operacji na macierzach można wywnioskować, że:

$$\begin{bmatrix} \binom{0}{0} & 0 & \cdots & 0 \\ \binom{1}{0} & \binom{1}{1} & \cdots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ \binom{s}{0} & \binom{s}{1} & \cdots & \binom{s}{s} \end{bmatrix} \cdot \begin{bmatrix} 1 \\ n \\ \vdots \\ n^s \end{bmatrix} = \begin{bmatrix} 1 \\ n+1 \\ \vdots \\ (n+1)^s \end{bmatrix}$$

Macierz z dwumianami Newtona będzie częścią macierzy B, którą szukamy. Chcemy, by każdy z elementów $\{n^0, \cdots, n^s\}$ był pomnożony przez odpowiadającą mu stałą $\{b_0, \cdots, b_s\}$, oraz, podobnie jak w przypadku bez wielomianu, wcześniejsze wyrazy ciągu były mnożone przez odpowiednie stałe.

W takim razie macierz B, której szukamy, jest macierzą klatkową postaci:

$$B = \begin{bmatrix} A_1 & A_2 \\ A_3 & A_4 \end{bmatrix}$$

gdzie każda z klatek A_1, A_2, A_3, A_4 wygląda następująco:

$$A_1 = \begin{bmatrix} a_l & a_{l-1} & \cdots & a_0 \\ 1 & 0 & \cdots & 0 \\ 0 & 1 & \ddots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix}$$

$$A_2 = \begin{bmatrix} b_0 & b_1 & \cdots & b_s \\ 0 & \cdots & \cdots & 0 \\ \vdots & \ddots & \iddots & \vdots \\ 0 & \vdots & \vdots & 0 \end{bmatrix}$$

$$A_3 = \begin{bmatrix} 0 & \cdots & \cdots & 0 \\ \vdots & \ddots & \iddots & \vdots \\ \vdots & \iddots & \ddots & \vdots \\ 0 & \cdots & \cdots & 0 \end{bmatrix}$$

$$A_4 = \begin{bmatrix} \binom{0}{0} & 0 & \cdots & 0 \\ \binom{1}{0} & \binom{1}{1} & \cdots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ \binom{s}{0} & \binom{s}{1} & \cdots & \binom{s}{s} \end{bmatrix}$$

Zadanie 6

Rozważ poniższy algorytm, który dla danego (wielo)zbioru A liczb całkowitych wyznacza pewną wartość.

while $|A| > 1$ **do**
  $a \leftarrow$ losowa liczba z A;
  $A \leftarrow A \setminus \{a\}$;
  $b \leftarrow$ losowa liczba z A;
  $A \leftarrow A \setminus \{b\}$;
  $A \leftarrow A \cup \{a - b\}$;
end
output (x mod 2), gdzie x jest elementem z A;

Twoim zadaniem jest napisanie programu (w pseudokodzie), możliwie najoszczędniejszego pamięciowo, który wylicza tę samą wartość.

Rozwiązanie: Proponuję następujący program.

$n \leftarrow |A|$;
$res \leftarrow 0$;
for $i \leftarrow 1$ **to** n **do**
  $res \leftarrow (res + a_i) \bmod 2$;
end
output res;

Złożoność pamięciowa wynosi $O(1)$, bo tylko jeden bit jest wymagany do całości programu. Korzystamy z faktu, że kolejność wybierania liczb nie ma żadnego znaczenia, oraz z faktu, że każda z liczb ma resztę z dzielenia przez 2. Gdy weźmiemy dwie liczby, które mają taką samą resztę z dzielenia i je odejmiemy od siebie, to zwracamy liczbę, która ma resztę z dzielenia równą 0, w przeciwnym wypadku oczywiście 1. Wobec tego można uprościć program do policzenia liczby liczb nieparzystych występujących w zbiorze, i wykonywanie dzielenia modulo naszego wyniku na bieżąco w celu zaoszczędzenia pamięci.

Zadanie 7

Ułóż algorytm, który dla drzewa $T = (V, E)$ oraz listy par wierzchołków v_i, u_i $(i = 1, 2, \ldots, m)$ sprawdza, czy v_i leży na ścieżce z u_i do korzenia. Przyjmij, że drzewo jest zadane jako lista $(n-1)$ krawędzi (p_i, v_i) takich, że p_i jest ojcem v_i w drzewie.

Rozwiązanie: Proponuję następujący algorytm.

```
for i ← 0 to m do
    if v_i = u_i then
        output (v_i, u_i): true;
    else
        while ∃p_i∃(p_i, v_i) ∈ V do
            if p_i = u_i then
                output (v_i, u_i): true;
                continue;
            else
                v_i ← p_i;
            end
        end
        output (v_i, u_i): false;
    end
end
```

Wiemy, że każdy wierzchołek v_i ma dokładnie jedną ścieżkę do korzenia, a dokładniej ścieżkę $(a_0, e_0, a_1, e_1, \cdots a_n)$, w której $a_0 = v_i$ i wierzchołek a_{i+1} jest rodzicem wierzchołka a_i. Można więc łatwo sprawdzić występowanie wierzchołka u_i w ścieżce z v_i do korzenia sprawdzając, czy którykolwiek z wierzchołków a_i jest równy wierzchołkowi u_i. Złożoność tego algorytmu to $O(m \cdot (n-1)^2)$.

Istnieje również algorytm wykorzystujący algorytm przeszukiwania grafu DFS/BFS. Do samodzielnego opracowania lub zobaczenia u kogoś innego.