Ćwiczenia 2

Znajdowanie maksimum, $O(n^2)$ procesorów CRCW, czas $O(1)$

Dane: Tablica $A[1..n]$ zawierająca liczby naturalne
Wynik: Zmienna *wynik* powinna zawierać indeks największego elementu w A (gdy jest ich wiele, to tego który ma mniejszy numer)

---

1. inicjalizacja tablicy $pom[1..n]$ wartościami 1,
2. procesor $p_{i,j}$ $(i \neq j)$, sprawdza warunek $A[i] < A[j] \vee (A[i] = A[j] \wedge i > j)$, jeśli ma on wartość `true` zapisuje w $pom[i] = 0$, ten krok wykonuje się równolegle, zabiera on $O(1)$ czasu,
3. teraz w tablica *pom* zawiera dokładnie jedną wartość 1,
4. procesor $p_{i,i}$ sprawdza czy $pom[i] = 1$, jeśli tak, to zapisuje w *wynik* swój indeks.

Znajdowanie maksimum, $O(n)$ procesorów CRCW, czas $O(\log\log n)$

Dane: Tablica $A[1..n]$ zawierająca liczby naturalne
Wynik: Zmienna *wynik* powinna zawierać indeks największego elementu w A (gdy jest ich wiele, to tego który ma mniejszy numer)

---

1. podział tablicy A na bloki o długości $\sqrt{n}$, każdemu blokowi przypisujemy $\sqrt{n}$ procesorów i rekurencyjnie znajdujemy w nim maksimum,
2. po powrocie z rekurencji obliczone jest $\sqrt{n}$ wartości, z których trzeba wybrać i zwrócić największą, ponieważ dostępnych jest n procesorów, można tu zastosować poprzedni algorytm. Ten krok zabiera $O(1)$ czasu

Złożoność: czas wykonania algorytmu można wyrazić rekurencyjnym wzorem $T(n) = T(\sqrt{n}) + O(1)$, co po rozwiązaniu daje $T(n) = O(\log\log n)$.

Znajdowanie maksimum, $O(n^{1+\epsilon})$ procesorów CRCW, czas $O(1)$ (przy ustalonym ϵ)

Dane: Tablica $A[1..n]$ zawierająca liczby naturalne, ustalone ϵ
Wynik: Zmienna *wynik* powinna zawierać indeks największego elementu w A (gdy jest ich wiele, to tego który ma mniejszy numer)

---

1. podział tablicy A na bloki o rozmiarze n^ϵ (takich bloków będzie $n^{1-\epsilon}$), każdemu blokowi przypisuje się $n^{2\epsilon}$ procesorów (czyli wykorzystujemy dokładnie $n^{1-\epsilon} \cdot n^{2\epsilon} = n^{1+\epsilon}$ procesorów),
2. znajdowanie w obrębie każdego bloku maksimum (w czasie $O(1)$), stosując pierwszy algorytm na znajdowanie maksimum,

3. teraz zostało $n^{1-\epsilon}$ kandydatów na maksimum (tyle ile bloków), należy teraz ponowić kroki 1—2 ograniczając się tylko do tych wartości. Po podziale na bloki o długości n^{ϵ}, otrzymuje się teraz już $n^{1-2\epsilon}$ bloków. W trzeciej fazie liczba bloków spadnie do $n^{1-3\epsilon}$, w czwartek do $n^{1-4\epsilon}$, itd.
4. po $O(\epsilon^{-1})$ (co dla ustalonego ϵ jest równe $O(1)$), fazach zostaje 1 blok. Jako *wynik* zwracana jest maksymalna wartość, w nim zawarta.

Sumy prefiksowe, $O\left(\frac{n}{\log n}\right)$ procesorów EREW, czas $O(\log n)$

Dane: Tablica $A[1..n]$ zawierająca wartości liczby całkowite
Wynik: Tablica $sum[1..n]$ powinna zawierać sumy prefiksowe $sum[i] = \sum_{j=1}^{j\leq i} A[j]$

---

1. podział tablicy A na $\frac{n}{\log n}$ części, o rozmiarze $k = \log n$, każdej części przydzielany jest 1 procesor,
2. w obrębie każdej części przypisany procesor oblicza sumy prefiksowe danego bloku (stosując najzwyklejszy sekwencyjny algorytm). Algorytm zapisuje wyniki w tablicy sum. Ten krok zajmuje $O(\log n)$ czasu,
3. w wydzielonym zbiorze elementów: końcówek bloków (elementy $sum[ik]$ dla $1 \leq i \leq \frac{n}{\log n}$), obliczane są sumy prefiksowe stosując standardowy algorytm liczenia sum prefiksowych (tym razem mamy tyle samo procesorów co elementów: $\frac{n}{\log n}$),
4. elementy $sum[ik]$ zawierają już poprawne wartości, należy jeszcze poprawić resztę, w tym celu dla każdego j z przedziału $1 < j < \log n$, wykonuje się:

   ```
   for x:=1 to n/log(n) pardo sum[x*k+j]=sum[x*k]+sum[x*k+j];
   ```

   powyższy kod poprawia wartości w elementach $sum[xk + j]$ (w $sum[xk]$ zapisana jest suma $A[1..xk]$, w $sum[xk + j]$ suma $A[(xk + 1)..(xk + j)]$). Każdy z takich kroków zajmuje $O(1)$ czasu, a ponieważ, jest ich $O(\log n)$, to taki również jest czas wykonania.

Znajdowanie pierwszej jedynki w tablicy, $O(n)$ procesorów CRCW, czas $O(1)$

Dane: Tablica $A[1..n]$ zawierająca wartości $\{0, 1\}$
Wynik: Zmienna *wynik* powinna zawierać indeks pierwszego elementu z lewej zawierającego wartość 1

---

1. inicjalizacja pomocniczej tablicy $pom[1..\sqrt{n}]$ wartościami 0,
2. podział tablicy A na bloki o rozmiarze $\sqrt{n}$, każdemu blokowi należy przydzielić $\sqrt{n}$ procesorów (czyli jeden procesor jest przydzielony do jednego elementu tablicy A), procesor z bloku i, przydzielony elementowi j, sprawdza wartość $A[j]$, jeśli jest ona równa 1, to zapisuje w $pom[i]$ wartość 1, ten krok zajmuje $O(1)$ czasu,

3. wyszukanie w tablicy *pom* pierwszego bloku zawierającego 1, ponieważ mamy do dyspozycji $O(n)$ procesorów, a tablica *pom* ma rozmiar $O(\sqrt{n})$, stosujemy tu opisany wcześniej algorytm (dla $O(n^2)$ procesorów i $O(n)$ danych). Ten krok zajmuje $O(1)$ czasu.

4. niech k, będzie pierwszym blokiem zawierającym 1, pierwszej jedynki należy, już teraz szukać tylko w obrębie tego bloku, ma on rozmiar $O(\sqrt{n})$, stąd można zastosować sztuczkę z poprzedniego punktu

5. znaleziony indeks zapisywany jest w zmiennej *wynik*