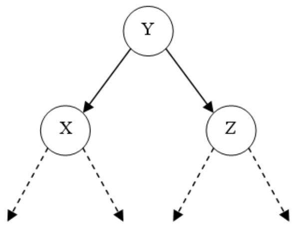
ASD - ćwiczenia IX

Kopce binarne

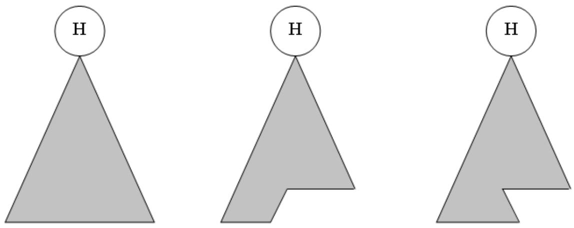
- własność porządku kopca



  gdzie dla każdej trójki wierzchołków kopca (X, Y, Z) porządek etykiet *elem* jest następujący

  - $X.elem \leq Y.elem$ oraz $Z.elem \leq Y.elem$ w przypadku kopca **typu max**,
  - $X.elem \geq Y.elem$ oraz $Z.elem \geq Y.elem$ w przypadku kopca **typu min**,

- własność lewostronnego wypełnienia kopca



- definicja wskaźnikowa struktury typu węzeł kopca w pseudokodzie

```
typedef struct HeapNode Heap;

struct HeapNode {
  element elem;
  struct HeapNode left, right, parent, next;
};
```

- podstawowe operacje dla kopca binarnego:

  - $EMPTY\ :\ \mathcal{H} \rightarrow \{TURE, FALSE\}$, sprawdzenie czy struktura jest pusta,
  - $INSERT\ :\ \mathcal{H} \times E \rightarrow \mathcal{H}$, wstawienie elementu do struktury,
  - $MIN\ :\ \mathcal{H} \rightarrow E$, „obejrzenie“ minimalnego elementu przechowywanego w strukturze typu min,
  - $DELMIN\ :\ \mathcal{H} \rightarrow \mathcal{H}$, usunięcie minimalnego elementu ze struktury typu min,
  - $MAX\ :\ \mathcal{H} \rightarrow E$, „obejrzenie“ maksymalnego elementu przechowywanego w strukturze typu min,

- $DELMAX : \mathcal{H} \to \mathcal{H}$, usunięcie maksymalnego elementu ze struktury typu min,

  gdzie $\mathcal{H}$ jest przestrzenią kopców binarnych, E zbiorem etykiet wierzchołków kopca binarnego,

- złożoność czasowa podstawowych operacji n-elementowego kopca binarnego:
  - $A(\texttt{EMPTY()}, n) = O(1)$, $W(\texttt{EMPTY()}, n) = O(1)$,
  - $A(\texttt{INSERT()}, n) = O(\log(n))$, $W(\texttt{INSERT()}, n) = O(\log(n))$,
  - $A(\texttt{MIN()}, n) = O(1)$, $W(\texttt{MIN()}, n) = O(1)$,
  - $A(\texttt{DELMIN()}, n) = O(\log(n))$, $W(\texttt{DELMIN()}, n) = O(\log(n))$,
  - $A(\texttt{MAX()}, n) = O(1)$, $W(\texttt{MAX()}, n) = O(1)$,
  - $A(\texttt{DELMAX()}, n) = O(\log(n))$, $W(\texttt{DELMAX()}, n) = O(\log(n))$.
- dodatkowe operacje dla kopca binarnego: typu min oraz typu max:
  - $DELETE : \mathcal{H} \times E \to \mathcal{H}$, usunięcie elementu ze struktury,
  - $MEMBER : \mathcal{H} \times E \to \{TURE, FALSE\}$, sprawdzenie, czy dany element jest przechowywany w strukturze,
- złożoność czasowa dodatkowych operacji n-elementowego kopca binarnego:
  - $A(\texttt{DELETE()}, n) = O(n)$, $W(\texttt{DELETE()}, n) = O(n)$,
  - $A(\texttt{MEMBER()}, n) = O(n)$, $W(\texttt{MEMBER()}, n) = O(n)$,

Zadania

1. Dany jest kopiec H o n parami różnych elementach, gdzie $n > 2$, którego korzeń zawiera element maksymalny.

   (a) Jaki jest koszt znalezienia k-tego co do wielkości elementu w kopcu H, jeżeli wolno stosować tylko operacje $INSERT$, $DELMAX$, MAX, $EMPTY$ (charakterystyczne dla kolejek priorytetowych)?

   (b) Zaproponuj funkcję

   ```
   element FIND_SECOND(Heap H)
   ```

   o złożoności $O(1)$, która poda drugi co do wielkości element kopca H. Po zakończeniu procedury zawartość kopca powinna być taka sama jak przed jego rozpoczęciem.

   (c) Zaproponuj możliwie efektywną funkcję

   ```
   element FIND_MIN(Heap H)
   ```

   która zwróci minimalny element kopca H. Po zakończeniu funkcji, zawartość kopca powinna być taka sama jak przed jego rozpoczęciem.

2. Zaprojektuj możliwie efektywną funkcję

   ```
   Heap JOIN(Heap H1, Heap H2),
   ```

   która utworzy strukturę danych typu `Heap`, będącą rezultatem połączenia dwóch wejściowych kopców $H1$ oraz $H2$ typu max. Oszacuj złożoność czasową i pamięciową metody.

3. Zakładamy, że wierzchołek pewnego drzewa przeszukiwań binarnych T jest reprezentowany przez strukturę danych typu `Tree`, a wierzchołek pewnego kopca H przez strukturę danych typu `Heap`. Zaprojektuj procedury

```
Heap TRANSFORM_MIN(Tree T)
```

oraz

```
Heap TRANSFORM_MAX(Tree T)
```

tworzące odpowiednio kopiec niemalejący i kopiec nierosnący z drzewa wejściowego T. Wynikiem działania obu procedur ma być korzeń kopca nierosnącego bądź niemalejącego.

   (a) Oszacuj złożoność swoich rozwiązań względem liczby wierzchołków drzewa przeszukiwań binarnych.

   (b) Czy twoje procedury są wrażliwe na kształt drzewa T (tj. czy ich czas działania zależy od stopnia zrównoważenia struktury).

   Przy konstruowaniu rozwiązań można korzystać z dowolnych liniowych struktur danych.

4. Rozważmy drzewo genealogiczne T, będące elementem przestrzeń drzew genealogicznych $\mathcal{T}$, rodu państwa Algorytmicznych, który w chwili obecnej składa się z pewnej skończonej liczby rodzin $F[1], F[2], \ldots$. Każda z nich (tj. rodzina $F[i]$) stanowi zbiór pewnej liczby osób (rodziców oraz dzieci) $F[i][1], F[i][2], \ldots$, o parami różnym wieku. Każdy ze zbiorów $F[i]$ może zmieniać swój stan ze względu na przyjście na świat nowego dziecka bądź śmierć najstarszej osoby w rodzinie. Dodatkowo, w chwili kiedy dziecko $F[i][j]$ z danej rodziny $F[i]$ wchodzi w związek małżeński (oczywiście z osobą spoza rodu państwa Algorytmicznych), wtedy formalnie jest ono usuwane z rodziny $F[i]$ i tworzy własną rodzinę $F[k]$ dodawaną do rozważanego drzewa rodowego. Zakładamy dalej, że każdy członek rodu państwa Algorytmicznych reprezentowany jest przez strukturę `Member` postaci

```
struct Member {
  int index, age;
};
```

   gdzie zmienne `index` i `age` definiują kolejno indeks rozważanej osoby w rodzinie $F[i]$ oraz jej wiek. *Nestorem rodziny* $F[i]$ nazywamy osobę najstarszą wiekiem spośród osób $F[i][1], F[i][2], \ldots$. *Nestorem rodu* nazywamy nestora nestorów rodzin $F[1], F[2], \ldots$. Zaprojektuj strukturę danych typu `GenTree`, która będzie w możliwie efektywny sposób implementowała drzewo rodowe T państwa Algorytmicznych oraz pozwoli na wykonanie następujących operacji:

   - $NEW\_CHILD : \mathcal{T} \times \mathbb{N} \to \mathcal{T}$, gdzie wywołanie funkcji ma postać `NEW_CHILD(T,i)` – operacja wstawiania nowego dziecka do wybranej rodziny $F[i]$, złożoność $O(\log(n)) + O(\log(m))$, dla n oraz m będących kolejno aktualną liczbą rodzin rodu Algorytmicznych, oraz licznością rodziny, w której pojawia się nowe dziecko,
   - $DIE : \mathcal{T} \times \mathbb{N} \to \mathcal{T}$, gdzie wywołanie funkcji ma postać `DIE(T,i)` – operacja usunięcia najstarszej osoby z wybranej rodziny $F[i]$, złożoność $O(\log(n)) + O(\log(m))$, dla n oraz m będących kolejno aktualną liczbą rodzin rodu Algorytmicznych, oraz licznością rodziny, z której usuwamy najstarszą osobę,

- $NEW\_FAMILY : \mathcal{T} \times \mathbb{N} \times \mathbb{N} \rightarrow \mathcal{T}$, gdzie wywołanie funkcji ma postać `NEW_FAMILY(T,i,j)` – operacja utworzenia nowej rodziny w drzewie rodu państwa Algorytmicznych, którą zakłada osoba indeksem j z rodziny $F[i]$ (tj. osoba $F[i][j]$), złożoność $O(\log(n)) + O(m)$, dla n oraz m będących kolejno aktualną liczbą rodzin rodu Algorytmicznych, oraz licznością rodziny, z której pochodzi osoba wchodząca w związek małżeński,
- $FAMILY\_NESTOR : \mathcal{T} \times \mathbb{N} \rightarrow \mathbb{N}$, gdzie wywołanie funkcji ma postać `FAMILY_NESTOR(T,i)` – operacja „pobrania" wieku nestora rodziny $F[i]$, złożoność $O(\log(n))$, dla n będącego liczbą rodzin rodu Algorytmicznych,
- $NESTOR : \mathcal{T} \rightarrow \mathbb{N}$, gdzie wywołanie funkcji ma postać `NESTOR(T)` – operacja „pobrania" wieku nestora rodu państwa Algorytmicznych, złożoność $O(1)$,
- $INCREMENT : \mathcal{T} \rightarrow \mathbb{N}$, gdzie wywołanie funkcji ma postać `INCREMENT(T)` – operacja zwiększenia wieku każdego z członków rodu o jeden rok oraz określenia aktualnej liczby k członków rodu, złożoność $O(k)$,
- $SIZE : \mathcal{T} \times \mathbb{N} \rightarrow \mathbb{N}$, gdzie wywołanie funkcji ma postać `SIZE(T,i)` – operacja określenia liczby osób aktualnie tworzących rodzinę $F[i]$, złożoność $O(n)$, dla n będącego liczbą rodzin rodu Algorytmicznych.

Przy rozwiązaniu można korzystać z dowolnych standardowych struktur danych (tj. list, stosów, kolejek, drzew, kopców) wraz z przynależnymi im operacjami.

Zadanie o „dobrze ułożonych" macierzach (do domu)

Rozważmy macierz kwadratową M, nie koniecznie w pełni wypełnioną, rozmiaru n kolumn na n wierszy, której elementy są liczbami naturalnymi. Powiemy, że macierz ta jest *dobrze ułożona*, jeżeli każda kolumna macierzy czytana od góry do dołu, oraz każdy wiersz macierzy czytany od strony lewej do prawej, stanowi wektor liczb posortowanych w kolejności niemalejącej. W przypadku kiedy dany element macierzy $M[c][r]$ jest nieokreślony, jego miejsce zajmuje symbol „–", np.:

- macierz dobrze ułożona w pełni wypełniona rozmiaru 4×4:

$$M = \begin{bmatrix} 1 & 2 & 7 & 10 \\ 3 & 5 & 8 & 11 \\ 5 & 6 & 9 & 12 \\ 9 & 13 & 14 & 19 \end{bmatrix},$$

- macierz dobrze ułożona nie w pełni wypełniona rozmiaru 4×4:

$$M = \begin{bmatrix} 1 & 2 & 7 & 10 \\ 3 & 5 & 8 & 11 \\ 4 & 6 & - & - \\ 9 & - & - & - \end{bmatrix},$$

Potraktujmy dalej macierz M, rozmiaru $n \times n$, jako multizbiór elementów ze zbioru liczb naturalnych, mocy co najwyżej n^2. Definiujemy następujące funkcje, których elementem dziedziny jest przestrzeń macierzy dobrze ułożonych $\mathcal{M}$:

- $INSERT : (\mathcal{M} \times \mathbb{N} \times \mathbb{N}) \rightarrow \mathcal{M}$, gdzie $INSERT(M, n, k) = M'$, oraz:
  - $|M| = n^2 \Rightarrow M' = M$,
  - $|M| < n^2 \Rightarrow M' = M \cup \{k\}$,
- $MIN : (\mathcal{M} \times \mathbb{N}) \rightarrow \mathbb{N}$, gdzie $MIN(M, n) = k$, oraz:

- $|M| > 0 \Rightarrow k = \min\{i = 1, 2, \ldots, |M| \,:\, M[i]\}$,
- $|M| = 0 \Rightarrow k =$ wartość nieokreślona ,

- $DELMIN \,:\, (\mathcal{M} \times \mathbb{N}) \rightarrow \mathcal{M}$, gdzie $DELMIN(M, n) = M'$, oraz:
  - $|M| > 0 \Rightarrow M' = M \setminus \{k\}$, dla $k = MIN(M, n)$,
  - $|M| = 0 \Rightarrow M' = M$,
- $XOR \,:\, (\mathcal{M} \times \mathbb{N} \times \mathcal{M} \times \mathbb{N}) \rightarrow \mathcal{M}$, gdzie $XOR(M1, n1, M2, n2) = M'$, oraz:
  - $M' = M1 \oplus M2$,
  - M' jest macierzą rozmiaru $(n1 + n2) \times (n1 + n2)$.

Zaprojektuj funkcje

`int INSERT(int M[][], int n, int k)`,

`int MIN(int M[][], int n)`,

`int [] DELMIN(int M[][], int n)`,

`int [][] XOR(int M1[][], int n1, int M2[][], int n2)`,

implementujące w możliwie efektywny sposób kolejne operacje $INSERT$, MIN, $DELMIN$, XOR, zdefiniowane dla przestrzeni macierzy dobrze ułożonych $\mathcal{M}$. Zadbaj o następujące złożoności czasowe rozwiązania:

- $W(\texttt{INSERT}(), n) = O(n)$,
- $W(\texttt{MIN}(), n) = O(1)$,
- $W(\texttt{DELMIN}(), n) = O(n)$,
- $W(\texttt{XOR}(), n1, n2) = O\left((n1 + n2)^2\right)$.