נושאים מתכדמים بمגוון נוחות - הרציל

תרגיל 1

(1) \[ H = \sum_{i=0}^{n} q_i \cdot \log_2 \frac{1}{q_i} \]

(2) \[ \sum_{i=0}^{n} \frac{1}{p_i} \cdot \log_2 \frac{1}{p_i} \leq \sum_{i=0}^{n} q_i \cdot \log_2 \frac{1}{q_i} \]

(3) השום \[ \sum_{i=0}^{n} q_i \cdot a_i \]

(4) \[ H = \sum_{i=0}^{n} q_i \cdot \log_2 \frac{1}{q_i} \leq \sum_{i=0}^{n} q_i \cdot \log_2 \frac{1}{q_i} = \sum_{i=0}^{n} q_i \cdot \log_2 3^{q_i} = \log_2 3 \sum_{i=0}^{n} q_i a_i \]

(5) \[ \frac{H}{\log_2 3} \leq \sum_{i=0}^{n} q_i a_i \]

שווה המשש.
Given a function $O(m(1 + H))$ (where $H$ is the harmonic number of $m$) and a function $f(n) = O(n)$, we consider the properties of $Splay$-trees, as well as the dynamic properties of other data structures.

**Amortized Analysis:**

The amortized analysis of $Splay$-trees involves understanding the costs of operations in a sequence of operations. We aim to bound the cost of any sequence of $m$ operations by $O(m(1 + H))$.

1. **Access Lemma:**
   - The cost of an access operation is $O(1) + \log_t n$.
   - The total cost of accessing all nodes is $O(n\log n)$.

2. **Splay Operation:**
   - The cost of a splay operation is $O(\log n)$.
   - The total cost of splaying all nodes is $O(n\log n)$.

3. **Amortized Analysis:**
   - The cost of any sequence of $m$ operations is $O(m(1 + H))$.

**Dynamic Properties:**

- The $Splay$-tree maintains the order of elements.
- The $Splay$-tree supports efficient insertions and deletions.

**Static Optimality Theorem:**

The static optimality theorem states that the total cost of all operations is $O(m(1 + H))$, where $m$ is the number of operations.

**Equations:**

- $O(m(1 + H))$ represents the total cost of $m$ operations.
- $\log_t n$ represents the logarithm base $t$ of $n$.
- $n\log n$ represents the total cost of accessing all nodes.

**Summary:**

The amortized analysis of $Splay$-trees provides a tight bound on the cost of operations, making it an efficient data structure for dynamic data management.
\[
\phi_0 - \phi_m = \sum_{i=1}^{n} \log(s_0(x_i)) - \sum_{i=1}^{n} \log(s_m(x_i)) \leq \sum_{i=1}^{n} \left( \log(1) - \log \left( \frac{q_i}{m} \right) \right) = \sum_{i=1}^{n} \log \left( \frac{m}{q_i} \right) \\
\leq \sum_{i=1}^{n} \log m = n \cdot \log m = O(m)
\]

At the beginning (משמוצע), one of the items is exactly 

All the items go down:

\[
\sum_{i=1}^{m} T(\text{Lookup}_i) = \sum_{i=1}^{m} \text{Amort}(O_i) + \phi_0 - \phi_m \leq O(m(1 + H)) + O(m) = O(m(1 + H))
\]

2. 

 случа: \( O(\log n) \) ($\text{Lookup}$)

Notation:

We assume that the sets are maintained with the \textit{Access Lemma} of Splay trees. The lemma states that the cost of an \textit{amortized} query is \( O(\log n) \).

Let's consider the following function for every item \( i \):

\[
3c(r(t) - r(v)) + 1 - (\ell - 1)(c - 1) = 3c \cdot \log \left( \frac{s(t)}{s(v)} \right) + 1 - (\ell - 1)(c - 1)
\]

As the cost of \( \text{Splay} \) is \( O(\log n) \), we can assume that the cost of \( \text{Splay} \) is \( O(1) \).

The cost of \( \text{Splay} \) is \( O(1) \).

\[
\text{Cost}(\text{Insert} - v) = \text{Cost}(\text{Splay} - p) + \ell + O(1)
\]

\[
= 3c \cdot \log(n) + 1 - (\ell - 1)(c - 1) + \ell + O(1) = O(\log n)
\]
Theorem 3.1: For an n-vertex graph G and a node v, we have

$$O(\log n) \cdot \text{Event},$$

where Event is the event that v is a leaf in the reverse tree.

Proof:

Let $G$ be a graph and $v$ a node in $G$. We define an algorithm that decides whether $v$ is a leaf in the reverse tree of $G$.

1. **Initialization:**
   - Start with $v$ as the root of a new reverse tree.
   - Let the reverse tree be empty.

2. **Iteration:**
   - For each neighbor $u$ of $v$ in $G$,
     - if $u$ is not a leaf in the reverse tree, then add $u$ as a leaf in the reverse tree.
   - Set $v$ as the root of the reverse tree.
   - Repeat until all nodes are processed.

3. **Termination:**
   - The algorithm terminates when all nodes have been processed.

The algorithm runs in $O(\log n)$ time, where $n$ is the number of vertices in $G$.

**Claim:**

The algorithm correctly decides whether $v$ is a leaf in the reverse tree of $G$.

**Proof of Claim:**

Let $v$ be a node in $G$. The algorithm starts with $v$ as the root of a new reverse tree and iterates through all its neighbors in $G$. If any neighbor is not a leaf in the reverse tree, then it is added as a leaf in the reverse tree. This process continues until all nodes are processed.

- If $v$ is a leaf in the reverse tree, then it is correctly added as a leaf in the reverse tree.
- If $v$ is not a leaf in the reverse tree, then it is not added as a leaf in the reverse tree.

Therefore, the algorithm correctly decides whether $v$ is a leaf in the reverse tree of $G$.

**Complexity:**

The algorithm runs in $O(\log n)$ time, which is optimal for this problem.
ה意見 של הלוחים המרכז במשקוקולות לכל זרימת ומפתולות \( O(\log n) \) - בורר \( \text{Weighted Sum} \) ב \( O(\log n) \) מבלי ל�いか בזרמי ריצה.