DAA,DSA-Attempted self study

14th Dec 2023

Egyptian fractions

Definition. A rational number which in reduced form has a numerator of 1 is called as an egyptian fraction.

FACT Ancient Egyptians who also worked with base 10 number systems, actually had in thier syntax only these type of fractions and all others were built from these basic fractions.

Question: Can egyptians actually write all fractions less than one in this way?

Answer: Yes as the following algortithm to obtain such an expansion shows.

Egyptifier($p/q \in \mathbb{Q}$)

Ensure: $p < q$ **and** $q \neq 0$

nume $\leftarrow$p
ret_list $\leftarrow$[ ]   \\This list will have all the denominators of the component egyptian fractions.
while nume $\neq$ 0 **do**
    n $\leftarrow \lceil q/p \rceil$
    ret_list.append(n)
    nume $\leftarrow$pn-q
end while
Output ret_list

Claim. This algorithm works!

Proof. The idea is to use the monovariant *nume*. On each iteration, by the choice of n, nume monotonously decreases while being non-negative. This is so because,

$$n = \lceil q/p \rceil \iff n-1 < q/p \leq n \iff 1/n \leq p/q < 1/(n-1) \iff 0 \leq (pn-q)/q \text{ and } pn-q < p$$

∎

15th Dec 2023

Interval Scheduling

Input Certain events given as a list of 2-tuples (s, f) denoting the start and finish timings of the event.

Output A sub-list of events from the original list ,that are non overlapping and have the maximum possible number of events.

This can naturally be usefull in several settings of practical interests like picking the maximum number of rides in a theme park.

The idea is the following. Keep picking the event that ends at the earliest, and does not clash with the ones already chosen. This greedy approach is implemeted in the following way. We use the heap data structure with the following terminology.

1. **MinHeapify** with some order function does heapify where the comaprision is made with the given function.
2. **Pop** removes that element from the heap.
3. **Peek** Returns the first element from the heap.

Scheduler(L:list of event tuples)

```
S ←MinHeapify(L) with start time ordering.
F ←MinHeapify(L) with end time ordering.
Ret_list ←[ ]
while F not empty do
    e ←Peek(F)
    s ←Peek(S)
    while s[0] ≤ e[1] do
        S.Pop(s)
        F.Pop(s)
        s ←Peek(S)
    end while
    Ret_list.append(e)
end while
Output Ret_list
```

Claim. This algorithm does return a list with maximum number of non-clashing(here after referred as compatible) events.

Proof. Say $\mathcal{O}$ is a list of events with m elements such that it has the maximal number of compatible events. Let R be the returned list. We shall prove that $|R| = |\mathcal{O}|$.
The idea is that the greedy algorithm always stays ahed of this optimal list.

Claim. $R[i][1] \leq \mathcal{O}[i][1]$

proof. We prove by induction. For the first step its clear by design. We assume the statement holds for $j-1$. But if so, then in the jth step, $\mathcal{O}[j]$ begins after $\mathcal{R}[j-1]$ ends. Since the algorithm chooses the event with least finish time that does not clash with the existing ones, the above statement follows. ∎

Consequently, If the list $\mathcal{O}$ has got some m events, then R has atleast those many events. ∎