

Maximum Segment Sum

Nils Cremer

October 27, 2022

Abstract

In this work we consider the *maximum segment sum* problem [1], that is to compute, given a list of numbers, the largest of the sums of the contiguous segments of that list. We assume that the elements of the list are not necessarily numbers but just elements of some linearly ordered group. Both an implementation for a naive algorithm ($\mathcal{O}(n^2)$) as well as for Kadane's algorithm [1] ($\mathcal{O}(n)$) are given and their correctness proven.

Contents

1	Maximum Segment Sum	1
1.1	Naive Solution	2
1.2	Kadane's Algorithms	3

1 Maximum Segment Sum

```
theory Maximum-Segment-Sum  
  imports Main  
begin
```

The *maximum segment sum* problem is to compute, given a list of numbers, the largest of the sums of the contiguous segments of that list. It is also known as the *maximum sum subarray* problem and has been considered many times in the literature; the Wikipedia article [Maximum subarray problem](#) is a good starting point.

We assume that the elements of the list are not necessarily numbers but just elements of some linearly ordered group.

```
class linordered-group-add = linorder + group-add +  
assumes add-left-mono:  $a \leq b \implies c + a \leq c + b$   
assumes add-right-mono:  $a \leq b \implies a + c \leq b + c$   
begin
```

```
lemma max-add-distrib-left:  $\max y z + x = \max (y+x) (z+x)$ 
```

<proof>

lemma *max-add-distrib-right*: $x + \max y z = \max (x+y) (x+z)$
<proof>

1.1 Naive Solution

fun *mss-rec-naive-aux* :: 'a list \Rightarrow 'a **where**
 mss-rec-naive-aux [] = 0
| *mss-rec-naive-aux* (x#xs) = max 0 (x + *mss-rec-naive-aux* xs)

fun *mss-rec-naive* :: 'a list \Rightarrow 'a **where**
 mss-rec-naive [] = 0
| *mss-rec-naive* (x#xs) = max (*mss-rec-naive-aux* (x#xs)) (*mss-rec-naive* xs)

definition *fronts* :: 'a list \Rightarrow 'a list set **where**
 fronts xs = {as. \exists bs. xs = as @ bs}

definition *front-sums* xs \equiv sum-list ' *fronts* xs

lemma *fronts-cons*: *fronts* (x#xs) = ((#) x) ' *fronts* xs \cup {[]} (**is** ?l = ?r)
<proof>

lemma *front-sums-cons*: *front-sums* (x#xs) = (+) x ' *front-sums* xs \cup {0}
<proof>

lemma *finite-fronts*: finite (*fronts* xs)
<proof>

lemma *finite-front-sums*: finite (*front-sums* xs)
<proof>

lemma *front-sums-not-empty*: *front-sums* xs \neq {}
<proof>

lemma *max-front-sum*: Max (*front-sums* (x#xs)) = max 0 (x + Max (*front-sums* xs))
<proof>

lemma *mss-rec-naive-aux-front-sums*: *mss-rec-naive-aux* xs = Max (*front-sums* xs)
<proof>

lemma *front-sums*: *front-sums* xs = {s. \exists as bs. xs = as @ bs \wedge s = sum-list as}
<proof>

lemma *mss-rec-naive-aux*: *mss-rec-naive-aux* xs = Max {s. \exists as bs. xs = as @ bs \wedge s = sum-list as}
<proof>

definition $mids :: 'a list \Rightarrow 'a list set$ **where**

$mids\ xs \equiv \{bs. \exists as\ cs. xs = as @ bs @ cs\}$

definition $mid-sums\ xs \equiv sum-list\ `mids\ xs$

lemma $fronts-mids: bs \in fronts\ xs \implies bs \in mids\ xs$

$\langle proof \rangle$

lemma $mids-mids-cons: bs \in mids\ xs \implies bs \in mids\ (x\#\ xs)$

$\langle proof \rangle$

lemma $mids-cons: mids\ (x\#\ xs) = fronts\ (x\#\ xs) \cup mids\ xs$ (**is** $?l = ?r$)

$\langle proof \rangle$

lemma $mid-sums-cons: mid-sums\ (x\#\ xs) = front-sums\ (x\#\ xs) \cup mid-sums\ xs$

$\langle proof \rangle$

lemma $finite-mids: finite\ (mids\ xs)$

$\langle proof \rangle$

lemma $finite-mid-sums: finite\ (mid-sums\ xs)$

$\langle proof \rangle$

lemma $mid-sums-not-empty: mid-sums\ xs \neq \{\}$

$\langle proof \rangle$

lemma $max-mid-sums-cons: Max\ (mid-sums\ (x\#\ xs)) = max\ (Max\ (front-sums\ (x\#\ xs)))\ (Max\ (mid-sums\ xs))$

$\langle proof \rangle$

lemma $mss-rec-naive-max-mid-sum: mss-rec-naive\ xs = Max\ (mid-sums\ xs)$

$\langle proof \rangle$

lemma $mid-sums: mid-sums\ xs = \{s. \exists as\ bs\ cs. xs = as @ bs @ cs \wedge s = sum-list\ bs\}$

$\langle proof \rangle$

theorem $mss-rec-naive: mss-rec-naive\ xs = Max\ \{s. \exists as\ bs\ cs. xs = as @ bs @ cs \wedge s = sum-list\ bs\}$

$\langle proof \rangle$

1.2 Kadane's Algorithms

fun $kadane :: 'a list \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a$ **where**

$kadane\ []\ cur\ m = m$

| $kadane\ (x\#\ xs)\ cur\ m =$

$(let\ cur' = max\ (cur + x)\ x\ in$

$kadane\ xs\ cur'\ (max\ m\ cur'))$

definition $mss-kadane\ xs \equiv kadane\ xs\ 0\ 0$

lemma $Max-front-sums-geq-0: Max\ (front-sums\ xs) \geq 0$
<proof>

lemma $Max-mid-sums-geq-0: Max\ (mid-sums\ xs) \geq 0$
<proof>

lemma $kadane: m \geq cur \implies m \geq 0 \implies kadane\ xs\ cur\ m = max\ m\ (max\ (cur + Max\ (front-sums\ xs))\ (Max\ (mid-sums\ xs)))$
<proof>

lemma $Max-front-sums-leq-Max-mid-sums: Max\ (front-sums\ xs) \leq Max\ (mid-sums\ xs)$
<proof>

lemma $mss-kadane-mid-sums: mss-kadane\ xs = Max\ (mid-sums\ xs)$
<proof>

theorem $mss-kadane: mss-kadane\ xs = Max\ \{s. \exists\ as\ bs\ cs. xs = as\ @\ bs\ @\ cs \wedge s = sum-list\ bs\}$
<proof>

end

end

References

- [1] Wikipedia. Maximum subarray problem, 2022. [https://en.wikipedia.org/wiki/Maximum_subarray_problem; accessed 25-September-2022].