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

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*: $a \leq b \Longrightarrow c + a \leq c + b$
assumes *add-right-mono*: $a \leq b \Longrightarrow a + c \leq b + c$
begin

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

by (*metis add-right-mono max.absorb-iff1 max-def*)

lemma *max-add-distrib-right*: $x + max\ y\ z = max\ (x+y)\ (x+z)$
by (*metis add-left-mono max.absorb1 max.cobounded2 max-def*)

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
  show *?l* $\subseteq$ *?r*
  proof
    fix *as* **assume** *as* $\in$ *?l*
    then show *as* $\in$ *?r* **by** (*cases as*) (*auto simp*: *fronts-def*)
  qed
  show *?r* $\subseteq$ *?l* **unfolding** *fronts-def* **by** *auto*
qed

lemma *front-sums-cons*: *front-sums* (*x#xs*) = (+) *x* ‘ *front-sums xs* $\cup$ {*0*}
proof −
  have *sum-list* ‘ ((#) *x*) ‘ *fronts xs* = (+) *x* ‘ *front-sums xs* **unfolding** *front-sums-def*
by *force*
  then show *?thesis* **by** (*simp add*: *front-sums-def fronts-cons*)
qed

lemma *finite-fronts*: *finite* (*fronts xs*)
  by (*induction xs*) (*simp add*: *fronts-def*, *simp add*: *fronts-cons*)

lemma *finite-front-sums*: *finite* (*front-sums xs*)
  using *front-sums-def finite-fronts* **by** *simp*

lemma *front-sums-not-empty*: *front-sums xs* $\neq$ {}
  unfolding *front-sums-def fronts-def* **using** *image-iff* **by** *fastforce*

lemma *max-front-sum*: *Max* (*front-sums* (*x#xs*)) = *max 0* (*x* + *Max* (*front-sums xs*))
using *finite-front-sums front-sums-not-empty*

by (*auto simp add*: *front-sums-cons hom-Max-commute max-add-distrib-right*)

lemma *mss-rec-naive-aux-front-sums*: *mss-rec-naive-aux xs* = *Max* (*front-sums xs*)
by (*induction xs*) (*simp add*: *front-sums-def fronts-def*, *auto simp*: *max-front-sum*)

lemma *front-sums*: *front-sums xs* = {*s*. ∃ *as bs*. *xs* = *as* @ *bs* ∧ *s* = *sum-list as*}
unfolding *front-sums-def fronts-def* **by** *auto*

lemma *mss-rec-naive-aux*: *mss-rec-naive-aux xs* = *Max* {*s*. ∃ *as bs*. *xs* = *as* @ *bs* ∧ *s* = *sum-list as*}
using *front-sums mss-rec-naive-aux-front-sums* **by** *simp*

definition *mids* :: *'a list* ⇒ *'a list set* **where**
  mids xs ≡ {*bs*. ∃ *as cs*. *xs* = *as* @ *bs* @ *cs*}

definition *mid-sums xs* ≡ *sum-list* ` *mids xs*

lemma *fronts-mids*: *bs* ∈ *fronts xs* ⟹ *bs* ∈ *mids xs*
unfolding *fronts-def mids-def* **by** *auto*

lemma *mids-mids-cons*: *bs* ∈ *mids xs* ⟹ *bs* ∈ *mids* (*x*#*xs*)
proof−
  fix *bs* **assume** *bs* ∈ *mids xs*
  then obtain *as cs* **where** *xs* = *as* @ *bs* @ *cs* **unfolding** *mids-def* **by** *blast*
  then have *x* # *xs* = (*x*#*as*) @ *bs* @ *cs* **by** *simp*
  then show *bs* ∈ *mids* (*x*#*xs*) **unfolding** *mids-def* **by** *blast*
qed

lemma *mids-cons*: *mids* (*x*#*xs*) = *fronts* (*x*#*xs*) ∪ *mids xs* (**is** *?l* = *?r*)
proof
  show *?l* ⊆ *?r*
  proof
    fix *bs* **assume** *bs* ∈ *?l*
    then obtain *as cs* **where** *as-bs-cs*: (*x*#*xs*) = *as* @ *bs* @ *cs* **unfolding** *mids-def* **by** *blast*
    then show *bs* ∈ *?r*
    proof (*cases as*)
      case *Nil*
      then have *bs* ∈ *fronts* (*x*#*xs*) **by** (*simp add*: *fronts-def as-bs-cs*)
      then show *?thesis* **by** *simp*
    next
      case (*Cons a as'*)
      then have *xs* = *as'* @ *bs* @ *cs* **using** *as-bs-cs* **by** *simp*
      then show *?thesis* **unfolding** *mids-def* **by** *auto*
    qed
  qed
  show *?r* ⊆ *?l* **using** *fronts-mids mids-mids-cons* **by** *auto*
qed

lemma *mid-sums-cons*: *mid-sums* $(x\#xs)$ = *front-sums* $(x\#xs)$ ∪ *mid-sums xs*
unfolding *mid-sums-def* **by** (*auto simp*: *mids-cons front-sums-def*)

lemma *finite-mids*: *finite* (*mids xs*)
by (*induction xs*) (*simp add*: *mids-def*, *simp add*: *mids-cons finite-fronts*)

lemma *finite-mid-sums*: *finite* (*mid-sums xs*)
by (*simp add*: *mid-sums-def finite-mids*)

lemma *mid-sums-not-empty*: *mid-sums xs* ≠ {}
unfolding *mid-sums-def mids-def* **by** *blast*

lemma *max-mid-sums-cons*: *Max* (*mid-sums* $(x\#xs)$) = *max* (*Max* (*front-sums* $(x\#xs)$)) (*Max* (*mid-sums xs*))
by (*auto simp*: *mid-sums-cons Max-Un finite-front-sums finite-mid-sums front-sums-not-empty mid-sums-not-empty*)

lemma *mss-rec-naive-max-mid-sum*: *mss-rec-naive xs* = *Max* (*mid-sums xs*)
by (*induction xs*) (*simp add*: *mid-sums-def mids-def*, *auto simp*: *max-mid-sums-cons mss-rec-naive-aux front-sums*)

lemma *mid-sums*: *mid-sums xs* = {*s*. ∃ *as bs cs*. *xs* = *as* @ *bs* @ *cs* ∧ *s* = *sum-list bs*}
by (*auto simp*: *mid-sums-def mids-def*)

theorem *mss-rec-naive*: *mss-rec-naive xs* = *Max* {*s*. ∃ *as bs cs*. *xs* = *as* @ *bs* @ *cs* ∧ *s* = *sum-list bs*}
unfolding *mss-rec-naive-max-mid-sum mid-sums* **by** *simp*

1.2 Kadane's Algorithms

fun *kadane* :: $'a$ *list* ⇒ $'a$ ⇒ $'a$ ⇒ $'a$ **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* ≡ *kadane xs 0 0*

lemma *Max-front-sums-geq-0*: *Max* (*front-sums xs*) ≥ *0*
proof−
have [] ∈ *fronts xs* **unfolding** *fronts-def* **by** *blast*
then have *0* ∈ *front-sums xs* **unfolding** *front-sums-def* **by** *force*
then show *?thesis* **using** *finite-front-sums Max-ge* **by** *simp*
qed

lemma *Max-mid-sums-geq-0*: *Max* (*mid-sums xs*) ≥ *0*
proof−

have *0* $\in$ *mid-sums xs* **unfolding** *mid-sums-def mids-def* **by** *force*
then show *?thesis* **using** *finite-mid-sums Max-ge* **by** *simp*
qed

lemma *kadane*: $m \geq cur \Longrightarrow m \geq 0 \Longrightarrow$ *kadane xs cur m* = *max m* (*max* (*cur* + *Max* (*front-sums xs*)) (*Max* (*mid-sums xs*)))
proof (*induction xs cur m rule*: *kadane.induct*)
case (*1 cur m*)
then show *?case* **unfolding** *front-sums-def fronts-def mid-sums-def mids-def* **by** *auto*
next
case (*2 x xs cur m*)
then show *?case*
apply (*auto simp*: *max-front-sum max-mid-sums-cons Let-def*)
by (*smt* (*verit*, *ccfv-threshold*) *Max-front-sums-geq-0 add-assoc add-0-right max.assoc max.coboundedI1 max.left-commute max.orderE max-add-distrib-left max-add-distrib-right*)
qed

lemma *Max-front-sums-leq-Max-mid-sums*: *Max* (*front-sums xs*) $\leq$ *Max* (*mid-sums xs*)
proof−
have *front-sums xs* $\subseteq$ *mid-sums xs* **unfolding** *front-sums-def mid-sums-def* **using** *fronts-mids subset-iff* **by** *blast*
then show *?thesis* **using** *front-sums-not-empty finite-mid-sums Max-mono* **by** *blast*
qed

lemma *mss-kadane-mid-sums*: *mss-kadane xs* = *Max* (*mid-sums xs*)
unfolding *mss-kadane-def* **using** *kadane Max-mid-sums-geq-0 Max-front-sums-leq-Max-mid-sums* **by** *auto*

theorem *mss-kadane*: *mss-kadane xs* = *Max* {*s*. $\exists$ *as bs cs*. *xs* = *as* @ *bs* @ *cs* $\wedge$ *s* = *sum-list bs*}
using *mss-kadane-mid-sums mid-sums* **by** *auto*

end

end

References

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