Multidimensional Binary Search Trees

Martin Rau

December 14, 2021

Abstract

This entry provides a formalization of multidimensional binary trees, also known as k-d trees. It includes a balanced build algorithm as well as the nearest neighbor algorithm and the range search algorithm. It is based on the papers "Multidimensional binary search trees used for associative searching" [1] and "An Algorithm for Finding Best Matches in Logarithmic Expected Time" [2].

Contents

1 Definition of the k-d Tree **2**
- 1.1 Definition of the k-d Tree Invariant and Related Functions . . 2
- 1.2 Lemmas adapted from *HOL−Library.Tree* to k-d Tree 4
- 1.3 Lemmas adapted from *HOL−Library.Tree-Real* to k-d Tree . 5

2 Building a balanced k-d Tree from a List of Points **6**
- 2.1 Auxiliary Lemmas 7
- 2.2 Widest Spread Axis 7
- 2.3 Fast Axis Median 8
- 2.4 Building the Tree 9
- 2.5 Main Theorems 10

3 Range Searching **11**
- 3.1 Rectangle Definition 11
- 3.2 Search Function 11
- 3.3 Auxiliary Lemmas 11
- 3.4 Main Theorem 12

4 Nearest Neighbor Search on the k-d Tree **12**
- 4.1 Auxiliary Lemmas about *sorted-wrt* 12
- 4.2 Neighbors Sorted wrt. Distance 13
- 4.3 The Recursive Nearest Neighbor Algorithm 13
- 4.4 Auxiliary Lemmas 14
- 4.5 The Main Theorems 14
- 4.6 Nearest Neighbors Definition and Theorems 15

1 Definition of the k-d Tree

theory *KD-Tree*
imports
Complex-Main
HOL−Analysis.Finite-Cartesian-Product
HOL−Analysis.Topology-Euclidean-Space
begin

A k-d tree is a space-partitioning data structure for organizing points in a k-dimensional space. In principle the k-d tree is a binary tree. The leafs hold the k-dimensional points and the nodes contain left and right subtrees as well as a discriminator v at a particular axis k. Every node divides the space into two parts by splitting along a hyperplane. Consider a node n with associated discriminator v at axis k. All points in the left subtree must have a value at axis k that is less than or equal to v and all points in the right subtree must have a value at axis k that is greater than v.

Deviations from the papers:

The chosen tree representation is taken from [2] with one minor adjustment. Originally the leafs hold buckets of points of size b. This representation fixes the bucket size to $b = 1$, a single point per Leaf. This is only a minor adjustment since the paper proves that $b = 1$ is the optimal bucket size for minimizing the running time of the nearest neighbor algorithm [2], only simplifies building the optimized k-d trees [2] and has little influence on the search algorithm [1].

type-synonym *'k point* = (*real*, *'k*) *vec*

lemma *dist-point-def*:
fixes p_0 :: (*'k*::*finite*) *point*
shows *dist* p_0 p_1 = *sqrt* ($\sum k \in UNIV.\ (p_0\$k - p_1\$k)^2$)
⟨*proof*⟩

datatype *'k kdt* =
Leaf 'k point
| *Node 'k real 'k kdt 'k kdt*

1.1 Definition of the k-d Tree Invariant and Related Functions

fun *set-kdt* :: *'k kdt* ⇒ (*'k point*) *set* **where**
set-kdt (*Leaf p*) = { *p* }
| *set-kdt* (*Node - - l r*) = *set-kdt l* ∪ *set-kdt r*

definition *spread* :: (*'k*::*finite*) ⇒ *'k point set* ⇒ *real* **where**
spread k P = (*if P* = {} *then 0 else let V* = (λ*p*. *p*\$*k*) ‘ *P in Max V* − *Min V*)

definition *widest-spread-axis* :: (*'k*::*finite*) ⇒ *'k set* ⇒ *'k point set* ⇒ *bool* **where**

widest-spread-axis k K ps $\longleftrightarrow$ $(\forall\, k' \in K.\ \mathit{spread}\ k'\ \mathit{ps} \leq \mathit{spread}\ k\ \mathit{ps})$

fun *invar* :: $('k::\mathit{finite})\ \mathit{kdt} \Rightarrow \mathit{bool}$ **where**
  invar (*Leaf p*) $\longleftrightarrow$ *True*
| *invar* (*Node k v l r*) $\longleftrightarrow$ $(\forall\, p \in \mathit{set\text{-}kdt}\ l.\ p\$k \leq v) \wedge (\forall\, p \in \mathit{set\text{-}kdt}\ r.\ v < p\$k)$
$\wedge$
    widest-spread-axis k UNIV (*set-kdt l* $\cup$ *set-kdt r*) $\wedge$ *invar l* $\wedge$ *invar r*

fun *size-kdt* :: $'k\ \mathit{kdt} \Rightarrow \mathit{nat}$ **where**
  size-kdt (*Leaf* -) = *1*
| *size-kdt* (*Node* - - *l r*) = *size-kdt l* + *size-kdt r*

fun *height* :: $'k\ \mathit{kdt} \Rightarrow \mathit{nat}$ **where**
  height (*Leaf* -) = *0*
| *height* (*Node* - - *l r*) = *max* (*height l*) (*height r*) + *1*

fun *min-height* :: $'k\ \mathit{kdt} \Rightarrow \mathit{nat}$ **where**
  min-height (*Leaf* -) = *0*
| *min-height* (*Node* - - *l r*) = *min* (*min-height l*) (*min-height r*) + *1*

definition *balanced* :: $'k\ \mathit{kdt} \Rightarrow \mathit{bool}$ **where**
  balanced kdt $\longleftrightarrow$ *height kdt* $-$ *min-height kdt* $\leq$ *1*

fun *complete* :: $'k\ \mathit{kdt} \Rightarrow \mathit{bool}$ **where**
  complete (*Leaf* -) = *True*
| *complete* (*Node* - - *l r*) $\longleftrightarrow$ *complete l* $\wedge$ *complete r* $\wedge$ *height l* = *height r*

lemma *invar-l*:
  invar (*Node k v l r*) $\Longrightarrow$ *invar l*
  ⟨*proof*⟩

lemma *invar-r*:
  invar (*Node k v l r*) $\Longrightarrow$ *invar r*
  ⟨*proof*⟩

lemma *invar-l-le-k*:
  invar (*Node k v l r*) $\Longrightarrow$ $\forall\, p \in \mathit{set\text{-}kdt}\ l.\ p\$k \leq v$
  ⟨*proof*⟩

lemma *invar-r-ge-k*:
  invar (*Node k v l r*) $\Longrightarrow$ $\forall\, p \in \mathit{set\text{-}kdt}\ r.\ v < p\$k$
  ⟨*proof*⟩

lemma *invar-set*:
  set-kdt (*Node k v l r*) = *set-kdt l* $\cup$ *set-kdt r*
  ⟨*proof*⟩

1.2 Lemmas adapted from *HOL−Library.Tree* to *k*-d Tree

lemma *size-ge0*[*simp*]:
0 < size-kdt kdt
⟨*proof*⟩

lemma *eq-size-1*[*simp*]:
size-kdt kdt = 1 ⟷ (∃ p. kdt = Leaf p)
⟨*proof*⟩

lemma *eq-1-size*[*simp*]:
1 = size-kdt kdt ⟷ (∃ p. kdt = Leaf p)
⟨*proof*⟩

lemma *neq-Leaf-iff*:
(∄ p. kdt = Leaf p) = (∃ k v l r. kdt = Node k v l r)
⟨*proof*⟩

lemma *eq-height-0*[*simp*]:
height kdt = 0 ⟷ (∃ p. kdt = Leaf p)
⟨*proof*⟩

lemma *eq-0-height*[*simp*]:
0 = height kdt ⟷ (∃ p. kdt = Leaf p)
⟨*proof*⟩

lemma *eq-min-height-0*[*simp*]:
min-height kdt = 0 ⟷ (∃ p. kdt = Leaf p)
⟨*proof*⟩

lemma *eq-0-min-height*[*simp*]:
0 = min-height kdt ⟷ (∃ p. kdt = Leaf p)
⟨*proof*⟩

lemma *size-height*:
size-kdt kdt ≤ 2 ^ height kdt
⟨*proof*⟩

lemma *min-height-le-height*:
min-height kdt ≤ height kdt
⟨*proof*⟩

lemma *min-height-size*:
2 ^ min-height kdt ≤ size-kdt kdt
⟨*proof*⟩

lemma *complete-iff-height*:
complete kdt ⟷ (min-height kdt = height kdt)
⟨*proof*⟩

lemma *size-if-complete*:
complete kdt $\Longrightarrow$ *size-kdt kdt* = 2 ^ *height kdt*
⟨*proof*⟩

lemma *complete-if-size-height*:
size-kdt kdt = 2 ^ *height kdt* $\Longrightarrow$ *complete kdt*
⟨*proof*⟩

lemma *complete-if-size-min-height*:
size-kdt kdt = 2 ^ *min-height kdt* $\Longrightarrow$ *complete kdt*
⟨*proof*⟩

lemma *complete-iff-size*:
complete kdt $\longleftrightarrow$ *size-kdt kdt* = 2 ^ *height kdt*
⟨*proof*⟩

lemma *size-height-if-incomplete*:
¬ *complete kdt* $\Longrightarrow$ *size-kdt kdt* < 2 ^ *height kdt*
⟨*proof*⟩

lemma *min-height-size-if-incomplete*:
¬ *complete kdt* $\Longrightarrow$ 2 ^ *min-height kdt* < *size-kdt kdt*
⟨*proof*⟩

lemma *balanced-subtreeL*:
balanced (*Node k v l r*) $\Longrightarrow$ *balanced l*
⟨*proof*⟩

lemma *balanced-subtreeR*:
balanced (*Node k v l r*) $\Longrightarrow$ *balanced r*
⟨*proof*⟩

lemma *balanced-optimal*:
assumes *balanced kdt size-kdt kdt* ≤ *size-kdt kdt′*
shows *height kdt* ≤ *height kdt′*
⟨*proof*⟩

1.3 Lemmas adapted from *HOL−Library.Tree-Real* to *k*-d Tree

lemma *size-height-log*:
log 2 (*size-kdt kdt*) ≤ *height kdt*
⟨*proof*⟩

lemma *min-height-size-log*:
min-height kdt ≤ *log 2* (*size-kdt kdt*)
⟨*proof*⟩

lemma *size-log-if-complete*:
complete kdt $\Longrightarrow$ *height kdt* = *log 2* (*size-kdt kdt*)

⟨*proof*⟩

lemma *min-height-size-log-if-incomplete*:
¬ *complete kdt* ⟹ *min-height kdt* < *log 2* (*size-kdt kdt*)
⟨*proof*⟩

lemma *min-height-balanced*:
assumes *balanced kdt*
shows *min-height kdt* = *nat*(*floor*(*log 2* (*size-kdt kdt*)))
⟨*proof*⟩

lemma *height-balanced*:
assumes *balanced kdt*
shows *height kdt* = *nat*(*ceiling*(*log 2* (*size-kdt kdt*)))
⟨*proof*⟩

lemma *balanced-Node-if-wbal1*:
assumes *balanced l balanced r size-kdt l* = *size-kdt r* + *1*
shows *balanced* (*Node k v l r*)
⟨*proof*⟩

lemma *balanced-sym*:
balanced (*Node k v l r*) ⟹ *balanced* (*Node k′ v′ r l*)
⟨*proof*⟩

lemma *balanced-Node-if-wbal2*:
assumes *balanced l balanced r abs*(*int*(*size-kdt l*) − *int*(*size-kdt r*)) ≤ *1*
shows *balanced* (*Node k v l r*)
⟨*proof*⟩

end

2 Building a balanced k-d Tree from a List of Points

theory *Build*
imports
KD-Tree
Median-Of-Medians-Selection.Median-Of-Medians-Selection
begin

Build a balanced k-d Tree by recursively partition the points into two lists. The partitioning criteria will be the median at a particular axis k. The left list will contain all points p with $p\ \$\ k \leq median$. The right list will contain all points with median at axis $median < p\ \$\ k$. The left and right list differ in length by one or none. The axis k will the widest spread axis.

2.1 Auxiliary Lemmas

lemma *length-filter-mset-sorted-nth*:
assumes *distinct xs* $n < length\ xs$ *sorted xs*
shows $\{\#\ x \in\#\ mset\ xs.\ x \leq xs\ !\ n\ \#\} = mset\ (take\ (n + 1)\ xs)$
⟨*proof*⟩

lemma *length-filter-sort-nth*:
assumes *distinct xs* $n < length\ xs$
shows $length\ (filter\ (\lambda x.\ x \leq sort\ xs\ !\ n)\ xs) = n + 1$
⟨*proof*⟩

2.2 Widest Spread Axis

definition *calc-spread* :: (*'k*::*finite*) ⇒ *'k point list* ⇒ *real* **where**
calc-spread k ps = (*case ps of* [] ⇒ *0* | *ps* ⇒
let ks = *map* (λ*p*. *p*$*k*) (*tl ps*) *in*
fold max ks ((*hd ps*)$*k*) − *fold min ks* ((*hd ps*)$*k*)
)

fun *widest-spread* :: (*'k*::*finite*) *list* ⇒ *'k point list* ⇒ *'k* × *real* **where**
widest-spread [] - = *undefined*
| *widest-spread* [*k*] *ps* = (*k*, *calc-spread k ps*)
| *widest-spread* (*k* # *ks*) *ps* = (
let (*k'*, *s'*) = *widest-spread ks ps in*
let s = *calc-spread k ps in*
if s ≤ *s' then* (*k'*, *s'*) *else* (*k*, *s*)
)

lemma *calc-spread-spec*:
calc-spread k ps = *spread k* (*set ps*)
⟨*proof*⟩

lemma *widest-spread-calc-spread*:
ks ≠ [] ⟹ (*k*, *s*) = *widest-spread ks ps* ⟹ *s* = *calc-spread k ps*
⟨*proof*⟩

lemma *widest-spread-axis-Un*:
shows *widest-spread-axis k K P* ⟹ *spread k' P* ≤ *spread k P* ⟹ *widest-spread-axis k* (*K* ∪ { *k'* }) *P*
and *widest-spread-axis k K P* ⟹ *spread k P* ≤ *spread k' P* ⟹ *widest-spread-axis k'* (*K* ∪ { *k'* }) *P*
⟨*proof*⟩

lemma *widest-spread-spec*:
(*k*, *s*) = *widest-spread ks ps* ⟹ *widest-spread-axis k* (*set ks*) (*set ps*)
⟨*proof*⟩

2.3 Fast Axis Median

definition *axis-median* :: (*'k*::*finite*) ⇒ *'k point list* ⇒ *real* **where**
axis-median k ps = (*let n* = (*length ps* − *1*) *div 2 in fast-select n* (*map* (λ*p*. *p*$*k*) *ps*))

lemma *length-filter-le-axis-median*:
assumes *0* < *length ps* ∀ *k*. *distinct* (*map* (λ*p*. *p*$*k*) *ps*)
shows *length* (*filter* (λ*p*. *p*$*k* ≤ *axis-median k ps*) *ps*) = (*length ps* − *1*) *div 2* + *1*
⟨*proof*⟩

definition *partition-by-median* :: (*'k*::*finite*) ⇒ *'k point list* ⇒ *'k point list* × *real* × *'k point list* **where**
partition-by-median k ps = (
let m = *axis-median k ps in*
let (*l*, *r*) = *partition* (λ*p*. *p*$*k* ≤ *m*) *ps in*
(*l*, *m*, *r*)
)

lemma *set-partition-by-median*:
(*l*, *m*, *r*) = *partition-by-median k ps* ⟹ *set ps* = *set l* ∪ *set r*
⟨*proof*⟩

lemma *filter-partition-by-median*:
assumes (*l*, *m*, *r*) = *partition-by-median k ps*
shows ∀ *p* ∈ *set l*. *p*$*k* ≤ *m*
and ∀ *p* ∈ *set r*. ¬*p*$*k* ≤ *m*
⟨*proof*⟩

lemma *sum-length-partition-by-median*:
assumes (*l*, *m*, *r*) = *partition-by-median k ps*
shows *length ps* = *length l* + *length r*
⟨*proof*⟩

lemma *length-l-partition-by-median*:
assumes *0* < *length ps* ∀ *k*. *distinct* (*map* (λ*p*. *p*$*k*) *ps*) (*l*, *m*, *r*) = *partition-by-median k ps*
shows *length l* = (*length ps* − *1*) *div 2* + *1*
⟨*proof*⟩

corollary *lengths-partition-by-median-1*:
assumes *0* < *length ps* ∀ *k*. *distinct* (*map* (λ*p*. *p*$*k*) *ps*) (*l*, *m*, *r*) = *partition-by-median k ps*
shows *length l* − *length r* ≤ *1*
and *length r* ≤ *length l*
and *0* < *length l*
and *length r* < *length ps*
⟨*proof*⟩

corollary *lengths-partition-by-median-2*:
assumes *1 < length ps* $\forall k$. *distinct* (*map* (λp. *p$k*) *ps*) (*l*, *m*, *r*) = *partition-by-median k ps*
shows *0 < length r*
and *length l < length ps*
⟨*proof*⟩

lemmas *length-partition-by-median* =
sum-length-partition-by-median length-l-partition-by-median
lengths-partition-by-median-1 lengths-partition-by-median-2

2.4 Building the Tree

function (*domintros*, *sequential*) *build* :: ('*k*::*finite*) *list* ⇒ '*k point list* ⇒ '*k kdt*
where
build - [] = *undefined*
| *build* - [*p*] = *Leaf p*
| *build ks ps* = (
let (*k*, -) = *widest-spread ks ps in*
let (*l*, *m*, *r*) = *partition-by-median k ps in*
Node k m (*build ks l*) (*build ks r*)
)
⟨*proof*⟩

lemma *build-domintros3*:
assumes (*k*, *s*) = *widest-spread ks* (*x* # *y* # *zs*) (*l*, *m*, *r*) = *partition-by-median k* (*x* # *y* # *zs*)
assumes *build-dom* (*ks*, *l*) *build-dom* (*ks*, *r*)
shows *build-dom* (*ks*, *x* # *y* # *zs*)
⟨*proof*⟩

lemma *build-termination*:
assumes $\forall k$. *distinct* (*map* (λp. *p$k*) *ps*)
shows *build-dom* (*ks*, *ps*)
⟨*proof*⟩

lemma *build-psimp-1*:
ps = [*p*] ⟹ *build k ps* = *Leaf p*
⟨*proof*⟩

lemma *build-psimp-2*:
assumes (*k*, *s*) = *widest-spread ks* (*x* # *y* # *zs*) (*l*, *m*, *r*) = *partition-by-median k* (*x* # *y* # *zs*)
assumes *build-dom* (*ks*, *l*) *build-dom* (*ks*, *r*)
shows *build ks* (*x* # *y* # *zs*) = *Node k m* (*build ks l*) (*build ks r*)
⟨*proof*⟩

lemma *length-xs-gt-1*:
1 < length xs ⟹ $\exists x\ y\ ys$. *xs* = *x* # *y* # *ys*

⟨*proof*⟩

lemma *build-psimp-3*:
assumes *1 < length ps* (*k*, *s*) = *widest-spread ks ps* (*l*, *m*, *r*) = *partition-by-median k ps*
assumes *build-dom* (*ks*, *l*) *build-dom* (*ks*, *r*)
shows *build ks ps* = *Node k m* (*build ks l*) (*build ks r*)
⟨*proof*⟩

lemmas *build-psimps*[*simp*] = *build-psimp-1 build-psimp-3*

2.5 Main Theorems

theorem *set-build*:
0 < length ps ⟹ ∀ *k*. *distinct* (*map* (λ*p*. *p*$*k*) *ps*) ⟹ *set ps* = *set-kdt* (*build ks ps*)
⟨*proof*⟩

theorem *invar-build*:
0 < length ps ⟹ ∀ *k*. *distinct* (*map* (λ*p*. *p*$*k*) *ps*) ⟹ *set ks* = *UNIV* ⟹ *invar* (*build ks ps*)
⟨*proof*⟩

theorem *size-build*:
0 < length ps ⟹ ∀ *k*. *distinct* (*map* (λ*p*. *p*$*k*) *ps*) ⟹ *size-kdt* (*build ks ps*) = *length ps*
⟨*proof*⟩

theorem *balanced-build*:
0 < length ps ⟹ ∀ *k*. *distinct* (*map* (λ*p*. *p*$*k*) *ps*) ⟹ *balanced* (*build ks ps*)
⟨*proof*⟩

lemma *complete-if-balanced-size-2powh*:
assumes *balanced kdt size-kdt kdt* = *2* ^ *h*
shows *complete kdt*
⟨*proof*⟩

theorem *complete-build*:
length ps = *2* ^ *h* ⟹ ∀ *k*. *distinct* (*map* (λ*p*. *p*$*k*) *ps*) ⟹ *complete* (*build k ps*)
⟨*proof*⟩

corollary *height-build*:
assumes *length ps* = *2* ^ *h* ∀ *k*. *distinct* (*map* (λ*p*. *p*$*k*) *ps*)
shows *h* = *height* (*build k ps*)
⟨*proof*⟩

end

3 Range Searching

theory *Range-Search*
imports
  KD-Tree
begin

Given two k-dimensional points p_0 and p_1 which bound the search space, the search should return only the points which satisfy the following criteria:
For every point p in the resulting set:
For every axis k:
$p_0\ \$\ k \leq p\ \$\ k \wedge p\ \$\ k \leq p_1\ \$\ k$

For a *2*-d tree a query corresponds to selecting all the points in the rectangle that has p_0 and p_1 as its defining edges.

3.1 Rectangle Definition

lemma *cbox-point-def*:
  fixes p_0 :: ('k::*finite*) *point*
  shows *cbox* p_0 p_1 = { p. $\forall k$. $p_0\$k \leq p\$k \wedge p\$k \leq p_1\$k$ }
⟨*proof*⟩

3.2 Search Function

fun *search* :: ('k::*finite*) *point* ⇒ 'k *point* ⇒ 'k *kdt* ⇒ 'k *point set* **where**
  search p_0 p_1 (*Leaf* p) = (*if* $p \in$ *cbox* p_0 p_1 *then* { p } *else* {})
| *search* p_0 p_1 (*Node* k v l r) = (
    if $v < p_0\$k$ *then*
      search p_0 p_1 r
    else if $p_1\$k < v$ *then*
      search p_0 p_1 l
    else
      search p_0 p_1 $l \cup$ *search* p_0 p_1 r
)

3.3 Auxiliary Lemmas

lemma *l-empty*:
  assumes *invar* (*Node* k v l r) $v < p_0\$k$
  shows *set-kdt* $l \cap$ *cbox* p_0 p_1 = {}
⟨*proof*⟩

lemma *r-empty*:
  assumes *invar* (*Node* k v l r) $p_1\$k < v$
  shows *set-kdt* $r \cap$ *cbox* p_0 p_1 = {}
⟨*proof*⟩

3.4 Main Theorem

theorem *search-cbox*:
  assumes *invar kdt*
  shows *search* p_0 p_1 *kdt* = *set-kdt kdt* ∩ *cbox* p_0 p_1
  ⟨*proof*⟩

end

4 Nearest Neighbor Search on the k-d Tree

theory *Nearest-Neighbors*
imports
  KD-Tree
begin

Verifying nearest neighbor search on the k-d tree. Given a k-d tree and a point p, which might not be in the tree, find the points ps that are closest to p using the Euclidean metric.

4.1 Auxiliary Lemmas about *sorted-wrt*

lemma
  assumes *sorted-wrt f xs*
  shows *sorted-wrt-take*: *sorted-wrt f* (*take n xs*)
  and *sorted-wrt-drop*: *sorted-wrt f* (*drop n xs*)
⟨*proof*⟩

definition *sorted-wrt-dist* :: ('*k*::*finite*) *point* ⇒ '*k point list* ⇒ *bool* **where**
  sorted-wrt-dist p ≡ *sorted-wrt* (λp_0 p_1. *dist* p_0 *p* ≤ *dist* p_1 *p*)

lemma *sorted-wrt-dist-insort-key*:
  sorted-wrt-dist p ps ⟹ *sorted-wrt-dist p* (*insort-key* (λ*q*. *dist q p*) *q ps*)
  ⟨*proof*⟩

lemma *sorted-wrt-dist-take-drop*:
  assumes *sorted-wrt-dist p ps*
  shows ∀ p_0 ∈ *set* (*take n ps*). ∀ p_1 ∈ *set* (*drop n ps*). *dist* p_0 *p* ≤ *dist* p_1 *p*
  ⟨*proof*⟩

lemma *sorted-wrt-dist-last-take-mono*:
  assumes *sorted-wrt-dist p ps n* ≤ *length ps 0* < *n*
  shows *dist* (*last* (*take n ps*)) *p* ≤ *dist* (*last ps*) *p*
  ⟨*proof*⟩

lemma *sorted-wrt-dist-last-insort-key-eq*:
  assumes *sorted-wrt-dist p ps insort-key* (λ*q*. *dist q p*) *q ps* ≠ *ps* @ [*q*]
  shows *last* (*insort-key* (λ*q*. *dist q p*) *q ps*) = *last ps*
  ⟨*proof*⟩

lemma *sorted-wrt-dist-last*:
assumes *sorted-wrt-dist p ps*
shows $\forall\, q \in set\ ps.\ dist\ q\ p \leq dist\ (last\ ps)\ p$
⟨*proof*⟩

4.2 Neighbors Sorted wrt. Distance

definition *upd-nbors* :: *nat* ⇒ ('*k*::*finite*) *point* ⇒ '*k point* ⇒ '*k point list* ⇒ '*k point list* **where**
upd-nbors n p q ps = *take n* (*insort-key* (λ*q*. *dist q p*) *q ps*)

lemma *sorted-wrt-dist-nbors*:
assumes *sorted-wrt-dist p ps*
shows *sorted-wrt-dist p* (*upd-nbors n p q ps*)
⟨*proof*⟩

lemma *sorted-wrt-dist-nbors-diff*:
assumes *sorted-wrt-dist p ps*
shows $\forall\, r \in set\ ps \cup \{q\} - set\ (upd\text{-}nbors\ n\ p\ q\ ps).\ \forall\, s \in set\ (upd\text{-}nbors\ n\ p\ q\ ps).\ dist\ s\ p \leq dist\ r\ p$
⟨*proof*⟩

lemma *sorted-wrt-dist-last-upd-nbors-mono*:
assumes *sorted-wrt-dist p ps n* ≤ *length ps 0* < *n*
shows *dist* (*last* (*upd-nbors n p q ps*)) *p* ≤ *dist* (*last ps*) *p*
⟨*proof*⟩

4.3 The Recursive Nearest Neighbor Algorithm

fun *nearest-nbors* :: *nat* ⇒ ('*k*::*finite*) *point list* ⇒ '*k point* ⇒ '*k kdt* ⇒ '*k point list* **where**
nearest-nbors n ps p (*Leaf q*) = *upd-nbors n p q ps*
| *nearest-nbors n ps p* (*Node k v l r*) = (
if p$*k* ≤ *v then*
let candidates = *nearest-nbors n ps p l in*
if length candidates = *n* ∧ *dist p* (*last candidates*) ≤ *dist v* (*p*$*k*) *then*
candidates
else
nearest-nbors n candidates p r
else
let candidates = *nearest-nbors n ps p r in*
if length candidates = *n* ∧ *dist p* (*last candidates*) ≤ *dist v* (*p*$*k*) *then*
candidates
else
nearest-nbors n candidates p l
)

4.4 Auxiliary Lemmas

lemma *cutoff-r*:
assumes *invar* (*Node k v l r*)
assumes *p*$*k* ≤ *v dist p c* ≤ *dist* (*p*$*k*) *v*
shows ∀ *q* ∈ *set-kdt r*. *dist p c* ≤ *dist p q*
⟨*proof*⟩

lemma *cutoff-l*:
assumes *invar* (*Node k v l r*)
assumes *v* ≤ *p*$*k dist p c* ≤ *dist v* (*p*$*k*)
shows ∀ *q* ∈ *set-kdt l*. *dist p c* ≤ *dist p q*
⟨*proof*⟩

4.5 The Main Theorems

lemma *set-nns*:
set (*nearest-nbors n ps p kdt*) ⊆ *set-kdt kdt* ∪ *set ps*
⟨*proof*⟩

lemma *length-nns*:
length (*nearest-nbors n ps p kdt*) = *min n* (*size-kdt kdt* + *length ps*)
⟨*proof*⟩

lemma *length-nns-gt-0*:
0 < *n* ⟹ 0 < *length* (*nearest-nbors n ps p kdt*)
⟨*proof*⟩

lemma *length-nns-n*:
assumes (*set-kdt kdt* ∪ *set ps*) − *set* (*nearest-nbors n ps p kdt*) ≠ {}
shows *length* (*nearest-nbors n ps p kdt*) = *n*
⟨*proof*⟩

lemma *sorted-nns*:
sorted-wrt-dist p ps ⟹ *sorted-wrt-dist p* (*nearest-nbors n ps p kdt*)
⟨*proof*⟩

lemma *distinct-nns*:
assumes *invar kdt distinct ps set ps* ∩ *set-kdt kdt* = {}
shows *distinct* (*nearest-nbors n ps p kdt*)
⟨*proof*⟩

lemma *last-nns-mono*:
assumes *invar kdt sorted-wrt-dist p ps n* ≤ *length ps* 0 < *n*
shows *dist* (*last* (*nearest-nbors n ps p kdt*)) *p* ≤ *dist* (*last ps*) *p*
⟨*proof*⟩

theorem *dist-nns*:
assumes *invar kdt sorted-wrt-dist p ps set ps* ∩ *set-kdt kdt* = {} *distinct ps* 0 < *n*

shows $\forall\, q \in$ *set-kdt kdt* $\cup$ *set ps* $-$ *set* (*nearest-nbors n ps p kdt*). *dist* (*last* (*nearest-nbors n ps p kdt*)) *p* $\leq$ *dist q p*
⟨*proof*⟩

4.6 Nearest Neighbors Definition and Theorems

definition *nearest-neighbors* :: *nat* $\Rightarrow$ (*'k*::*finite*) *point* $\Rightarrow$ *'k kdt* $\Rightarrow$ *'k point list*
where
nearest-neighbors n p kdt = *nearest-nbors n* [] *p kdt*

theorem *length-nearest-neighbors*:
length (*nearest-neighbors n p kdt*) = *min n* (*size-kdt kdt*)
⟨*proof*⟩

theorem *sorted-wrt-dist-nearest-neighbors*:
sorted-wrt-dist p (*nearest-neighbors n p kdt*)
⟨*proof*⟩

theorem *set-nearest-neighbors*:
set (*nearest-neighbors n p kdt*) $\subseteq$ *set-kdt kdt*
⟨*proof*⟩

theorem *distinct-nearest-neighbors*:
assumes *invar kdt*
shows *distinct* (*nearest-neighbors n p kdt*)
⟨*proof*⟩

theorem *dist-nearest-neighbors*:
assumes *invar kdt nns* = *nearest-neighbors n p kdt*
shows $\forall\, q \in$ (*set-kdt kdt* $-$ *set nns*). $\forall\, r \in$ *set nns*. *dist r p* $\leq$ *dist q p*
⟨*proof*⟩

end

References

[1] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, 1975.

[2] J. H. Friedman, J. L. Bentley, and R. A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Trans. Math. Softw.*, 3(3):209–226, 1977.