Trabalho Prático 0 - AEDS III

0.0.0

Generated by Doxygen 1.8.13

Contents

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Data Structure Documentation

3.1 cell Struct Reference

Cell structure.

```
#include <cell.h>
```

Data Fields

- long long min
- long long max
- long long sum

3.1.1 Detailed Description

Cell structure.

This struct stores the max, min and the sum of the interval.

3.1.2 Field Documentation

3.1.2.1 max

```
long long max
```

3.1.2.2 min

```
long long min
```

Minimum and Maximum value of the interval

3.1.2.3 sum

```
long long sum
```

Sum of the interval

The documentation for this struct was generated from the following file:

- src/lib/cell.h

Chapter 4

File Documentation

4.1 src/arvore.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "lib/segtree.h"
#include "lib/cell.h"
```

Functions

- int add (int a)
- int sub (int a)
- int main ()

  Main function.

4.1.1 Function Documentation

4.1.1.1 add()

```
int add (
            int a )
```

4.1.1.2 main()

```
int main ( )
```

Main function.

This function uses the Bryan solution based on a Segment Tree that stores the input array data.

Returns

0

4.1.1.3 sub()

```
int sub (
            int a )
```

4.2 src/lib/cell.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include "cell.h"
```

Functions

- Cell Cell_create ()

  Creates a new cell and initialize it's values with 0.
- void Cell_destroy (Cell ∗cell)

  Destroys the given cell.
- void Cell_print (Cell cell)

  Prints the content of the given cell.
- void Cell_fill (Cell ∗cell, int ∗array, int i, int j)

  Fill the cell data with the given array and intervals.

4.2.1 Function Documentation

4.2.1.1 Cell_create()

```
Cell Cell_create ( )
```

Creates a new cell and initialize it's values with 0.

Complexity: O(1)

Returns

The created cell

4.2.1.2 Cell_destroy()

```
void Cell_destroy (
            Cell * cell )
```

Destroys the given cell.

Complexity: O(1)

Parameters

cell	The cell to be destroyed

4.2.1.3 Cell_fill()

```
void Cell_fill (
            Cell * cell,
            int * array,
            int i,
            int j )
```

Fill the cell data with the given array and intervals.

Complexity: O(size-end)

Parameters

	cell	The cell
	array	The array
in	*i*	Start of the interval
in	*j*	End of the interval

4.2.1.4 Cell_print()

```
void Cell_print (
            Cell cell )
```

Prints the content of the given cell.

It doesn't print a new line.

Complexity: O(1)

Parameters

in	*cell*	The cell

4.3 src/lib/cell.h File Reference

Data Structures

- struct cell

  Cell structure.

Typedefs

- typedef struct cell Cell

Functions

- Cell Cell_create ()

  Creates a new cell and initialize it's values with 0.
- void Cell_destroy (Cell ∗cell)

  Destroys the given cell.
- void Cell_print (Cell cell)

  Prints the content of the given cell.
- void Cell_fill (Cell ∗cell, int ∗array, int i, int j)

  Fill the cell data with the given array and intervals.

4.3.1 Typedef Documentation

4.3.1.1 Cell

```
typedef struct cell Cell
```

4.3.2 Function Documentation

4.3.2.1 Cell_create()

```
Cell Cell_create ( )
```

Creates a new cell and initialize it's values with 0.

Complexity: O(1)

Returns

The created cell

4.3.2.2 Cell_destroy()

```
void Cell_destroy (
            Cell * cell )
```

Destroys the given cell.

Complexity: O(1)

Parameters

cell	The cell to be destroyed

4.3.2.3 Cell_fill()

```
void Cell_fill (
            Cell * cell,
            int * array,
            int i,
            int j )
```

Fill the cell data with the given array and intervals.

Complexity: O(size-end)

Parameters

	cell	The cell
	array	The array
in	*i*	Start of the interval
in	*j*	End of the interval

4.3.2.4 Cell_print()

```
void Cell_print (
            Cell cell )
```

Prints the content of the given cell.

It doesn't print a new line.

Complexity: O(1)

Parameters

in	*cell*	The cell

4.4 src/lib/matrix.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include "cell.h"
#include "matrix.h"
```

Functions

- Cell ∗∗ Matrix_create (int n)

    Builds the matrix with the given array.
- void Matrix_fill (Cell ∗∗matrix, int ∗array, int n)

    Fills the matrix with the given data.
- void Matrix_destroy (Cell ∗∗matrix, int n)

    Destroys the matrix.
- void Matrix_print (Cell ∗∗matrix, int n)

    Prints the matrix.

4.4.1 Function Documentation

4.4.1.1 Matrix_create()

```
Cell** Matrix_create (
            int n )
```

Builds the matrix with the given array.

Complexity: $O(n^2)$

Parameters

in	*n*	The length of the input array

Returns

A pointer to the created matrix

4.4.1.2 Matrix_destroy()

```
void Matrix_destroy (
            Cell ** matrix,
            int n )
```

Destroys the matrix.

Complexity: O(n)

Parameters

	matrix	The matrix
in	*n*	The length of the input array

4.4.1.3 Matrix_fill()

```
void Matrix_fill (
            Cell ** matrix,
            int * array,
            int n )
```

Fills the matrix with the given data.

Complexity: $O(n^3)$, because the maximum interval given to Cell_Fill has size n, and is executed n^2 times.

Parameters

	matrix	The matrix
	array	The array
in	*n*	The length of the input array

4.4.1.4 Matrix_print()

```
void Matrix_print (
```

```
Cell ** matrix,
int n )
```

Prints the matrix.

Complexity: O(n²)

Parameters

	matrix	The matrix
in	*n*	The length of the input array

4.5 src/lib/matrix.h File Reference

```
#include "cell.h"
```

Functions

- Cell ∗∗ Matrix_create (int n)

    Builds the matrix with the given array.
- void Matrix_fill (Cell ∗∗matrix, int ∗array, int n)

    Fills the matrix with the given data.
- void Matrix_destroy (Cell ∗∗matrix, int n)

    Destroys the matrix.
- void Matrix_print (Cell ∗∗matrix, int n)

    Prints the matrix.

4.5.1 Function Documentation

4.5.1.1 Matrix_create()

```
Cell** Matrix_create (
        int n )
```

Builds the matrix with the given array.

Complexity: O(n²)

Parameters

in	*n*	The length of the input array

Returns

A pointer to the created matrix

4.5.1.2 Matrix_destroy()

```
void Matrix_destroy (
            Cell ** matrix,
            int n )
```

Destroys the matrix.

Complexity: O(n)

Parameters

	matrix	The matrix
in	*n*	The length of the input array

4.5.1.3 Matrix_fill()

```
void Matrix_fill (
            Cell ** matrix,
            int * array,
            int n )
```

Fills the matrix with the given data.

Complexity: O(n^3), because the maximum interval given to Cell_Fill has size n, and is executed n^2 times.

Parameters

	matrix	The matrix
	array	The array
in	*n*	The length of the input array

4.5.1.4 Matrix_print()

```
void Matrix_print (
            Cell ** matrix,
            int n )
```

Prints the matrix.

Complexity: O(n^2)

Parameters

	matrix	The matrix
in	*n*	The length of the input array

4.6 src/lib/segtree.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <limits.h>
#include "segtree.h"
#include "cell.h"
```

Functions

- void SegTree_construct (Cell ∗segtree, int pos, int ∗array, int start, int end)

  Recursively constructs the Segment Tree.
- int SegTree_size (int n)

  Calculates the Segment Tree's number of nodes with the given the input array size.
- Cell ∗ SegTree_create (int ∗array, int n)

  Creates a new Segment Tree.
- void SegTree_destroy (Cell ∗segtree)

  Destroys the given Segment Tree.
- Cell SegTree_rangeQuery (Cell ∗segtree, int pos, int start, int end, int currStart, int currEnd)

  Perform a range query on the Segment Tree recursivelly.
- Cell SegTree_query (Cell ∗segtree, int n, int start, int end)

  Interface for the recursive range query.
- void SegTree_rangeUpdate (Cell ∗segtree, int pos, int start, int end, int currStart, int currEnd, int(∗transform)(int n))

  Recursively updates the Segment Tree.
- void SegTree_update (Cell ∗segtree, int n, int start, int end, int(∗transform)(int n))

  Interface for the recursive segtree update.

4.6.1 Function Documentation

4.6.1.1 SegTree_construct()

```
void SegTree_construct (
            Cell * segtree,
            int pos,
            int * array,
            int start,
            int end )
```

Recursively constructs the Segment Tree.

Complexity:

With n being the Segment Tree's number of nodes, the complexity is O(n), because we need to visit every node of the tree to calculate its data.

Parameters

	segtree	The Segment Tree array
in	*pos*	The current position on the segtree
in	*array*	The input array
in	*start*	The start of the interval (0-indexed, inclusive)
in	*end*	The end of the interval (0-indexed, inclusive)

4.6.1.2 SegTree_create()

```
Cell* SegTree_create (
            int * array,
            int n )
```

Creates a new Segment Tree.

Complexity:

The complexity for creating a new Segment Tree is related to the function SegTree_construct, which is O(N), being N the number of nodes of the tree.

The number of nodes N, in the best case is 2·n-1, with n being the input array length. In worst case N will not pass from 4·n-1. The full explanation for this is on the SegTree_size function description.

In all cases, the complexity for creating a new Segment Tree is O(n).

Parameters

in	*array*	The input array
in	*n*	The input array length

Returns

A pointer to the created Segment Tree

4.6.1.3 SegTree_destroy()

```
void SegTree_destroy (
            Cell * segtree )
```

Destroys the given Segment Tree.

Complexity: O(1)

Parameters

segtree	The segtree

4.6.1.4 SegTree_query()

```
Cell SegTree_query (
            Cell * segtree,
            int n,
            int start,
            int end )
```

Interface for the recursive range query.

Complexity: The complexity here is related to the complexity of the SegTree_rangeQuery function, which is O(log N), being N the number of nodes of the tree.

The number of nodes N, in the best case is 2·n-1, with n being the input array length. In worst case N will not pass from 4·n-1. The full explanation for this is on the SegTree_size function description.

With 2·n-1 or 4·n-1, the complexity of the function is O(log n).

Parameters

	segtree	The segtree
in	*n*	The input array length
in	*start*	The start of the interval (1-indexed, inclusive)
in	*end*	The end of the interval (1-indexed, inclusive)

Returns

The results of the query

4.6.1.5 SegTree_rangeQuery()

```
Cell SegTree_rangeQuery (
            Cell * segtree,
            int pos,
            int start,
            int end,
            int currStart,
            int currEnd )
```

Perform a range query on the Segment Tree recursivelly.

Complexity:

In the best case, the range of the query will be from 0 to (L - 1), being L the length of the input array. In this case, the complexity for retrieving data from the segtree is O(1).

In the worst case, the range of the query will have length of 1 (query of 1 element). In this case, the alghoritm will dive into the tree til reach the correspondant leaf. The complexity for retrieving the leaf data will be of O(H), with H begin the height of the tree.

As the Segment Tree is a Full Binary Tree, the height of the tree will correspond log N, being N the number of nodes. So the complexity for retrieving data from a leaf is O(log N).

Parameters

	segtree	The segtree
in	*pos*	The position
in	*start*	The start of the interval (0-indexed, inclusive)
in	*end*	The end of the interval (0-indexed, inclusive)
in	*currStart*	The current start of the interval (0-indexed, inclusive)
in	*currEnd*	The current end of the interval (0-indexed, inclusive)

Returns

The results of the query

4.6.1.6 SegTree_rangeUpdate()

```
void SegTree_rangeUpdate (
            Cell * segtree,
            int pos,
            int start,
            int end,
            int currStart,
            int currEnd,
            int(*)(int n) transform )
```

Recursively updates the Segment Tree.

Complexity:

As the updates are realized on the leafs of the tree, and then the nodes that represent the intervals have to be updated, the cost on the update will be related to the amount of nodes that has to be updated plus the height of the tree. The cost to reach a leaf is log H, being H the height of the tree. Let LF the number of leafs to be updated, that is the same as the size of the interval to be updated (end-start-1).

As the Segment Tree is a Full Binary Tree, the height of the tree will correspond log N, being N the number of nodes. So the complexity for retrieving data from a leaf is O(log N).

The complexity of a update is O(LF·log N).

Parameters

	segtree	The segtree
in	*pos*	The current position on the segtree array
in	*start*	The start of the interval (0-indexed, inclusive)
in	*end*	The end of the interval (0-indexed, inclusive)
in	*currStart*	The current start of the interval (0-indexed, inclusive)
in	*currEnd*	The current end of the interval (0-indexed, inclusive)

4.6.1.7 SegTree_size()

```
int SegTree_size (
            int n )
```

Calculates the Segment Tree's number of nodes with the given the input array size.

The number of nodes n required for a segtree is calculated with the length of the input array. In the best case, the length L is a power of 2, so the number of nodes is n=2·L-1. If L is not a power of 2, we have to find the next power of 2 after L. In this case, the number of nodes will not pass from n=2·2^((log L) + 1) - 1, which can be simplified to n=4·L-1.

Being n the input array length, let np the power of 2 that is equal or higher than n. The size of the Segment Tree array will be 2·np-1.

The strategy adopted here is to calculate the log2 of n and get the ceil of the result. This way, we can power 2 for the result and get the next power of 2. Then, we use 2·(2^np)-1 to get the the size of the Segment Tree array.

Complexity: (1)

Parameters

in	*n*	The input array size

Returns

The size of the Segment Tree array

4.6.1.8 SegTree_update()

```
void SegTree_update (
            Cell * segtree,
            int n,
            int start,
            int end,
            int(*)(int n) transform )
```

Interface for the recursive segtree update.

Complexity: The complexity here is related to the complexity of the SegTree_rangeUpdate function, which is O(L↩F·log N), being N the number of nodes of the tree, and LF is end - start - 1.

The number of nodes N, in the best case is 2·n-1, with n being the input array length. In worst case N will not pass from 4·n-1. The full explanation for this is on the SegTree_size function description.

With 2·n-1 or 4·n-1, the complexity of the function is O(LF·log n).

Parameters

	segtree	The segtree
in	*n*	The length of the input array
in	*start*	The start of the interval (1-indexed, inclusive)
in	*end*	The end of the interval (1-indexed, inclusive)
in	*transform*	The transform function

4.7 src/lib/segtree.h File Reference

```
#include "cell.h"
```

Functions

- void SegTree_construct (Cell ∗segtree, int pos, int ∗array, int start, int end)
  Recursively constructs the Segment Tree.
- int SegTree_size (int n)
  Calculates the Segment Tree's number of nodes with the given the input array size.
- Cell ∗ SegTree_create (int ∗array, int n)
  Creates a new Segment Tree.
- void SegTree_destroy (Cell ∗segtree)
  Destroys the given Segment Tree.
- Cell SegTree_rangeQuery (Cell ∗segtree, int pos, int start, int end, int currStart, int currEnd)
  Perform a range query on the Segment Tree recursivelly.
- Cell SegTree_query (Cell ∗segtree, int n, int start, int end)
  Interface for the recursive range query.
- void SegTree_rangeUpdate (Cell ∗segtree, int pos, int start, int end, int currStart, int currEnd, int(∗transform)(int n))
  Recursively updates the Segment Tree.
- void SegTree_update (Cell ∗segtree, int n, int start, int end, int(∗transform)(int n))
  Interface for the recursive segtree update.

4.7.1 Function Documentation

4.7.1.1 SegTree_construct()

```
void SegTree_construct (
            Cell * segtree,
            int pos,
            int * array,
            int start,
            int end )
```

Recursively constructs the Segment Tree.

Complexity:

With n being the Segment Tree's number of nodes, the complexity is O(n), because we need to visit every node of the tree to calculate its data.

Parameters

	segtree	The Segment Tree array
in	*pos*	The current position on the segtree
in	*array*	The input array
in	*start*	The start of the interval (0-indexed, inclusive)
in	*end*	The end of the interval (0-indexed, inclusive)

4.7.1.2 SegTree_create()

```
Cell* SegTree_create (
            int * array,
            int n )
```

Creates a new Segment Tree.

Complexity:

The complexity for creating a new Segment Tree is related to the function SegTree_construct, which is O(N), being N the number of nodes of the tree.

The number of nodes N, in the best case is 2·n-1, with n being the input array length. In worst case N will not pass from 4·n-1. The full explanation for this is on the SegTree_size function description.

In all cases, the complexity for creating a new Segment Tree is O(n).

Parameters

in	*array*	The input array
in	*n*	The input array length

Returns

A pointer to the created Segment Tree

4.7.1.3 SegTree_destroy()

```
void SegTree_destroy (
            Cell * segtree )
```

Destroys the given Segment Tree.

Complexity: O(1)

Parameters

segtree	The segtree

4.7.1.4 SegTree_query()

```
Cell SegTree_query (
            Cell * segtree,
            int n,
            int start,
            int end )
```

Interface for the recursive range query.

Complexity: The complexity here is related to the complexity of the SegTree_rangeQuery function, which is O(log N), being N the number of nodes of the tree.

The number of nodes N, in the best case is 2·n-1, with n being the input array length. In worst case N will not pass from 4·n-1. The full explanation for this is on the SegTree_size function description.

With 2·n-1 or 4·n-1, the complexity of the function is O(log n).

Parameters

	segtree	The segtree
in	*n*	The input array length
in	*start*	The start of the interval (1-indexed, inclusive)
in	*end*	The end of the interval (1-indexed, inclusive)

Returns

The results of the query

4.7.1.5 SegTree_rangeQuery()

```
Cell SegTree_rangeQuery (
            Cell * segtree,
            int pos,
            int start,
            int end,
            int currStart,
            int currEnd )
```

Perform a range query on the Segment Tree recursivelly.

Complexity:

In the best case, the range of the query will be from 0 to (L - 1), being L the length of the input array. In this case, the complexity for retrieving data from the segtree is O(1).

In the worst case, the range of the query will have length of 1 (query of 1 element). In this case, the alghoritm will dive into the tree til reach the correspondant leaf. The complexity for retrieving the leaf data will be of O(H), with H begin the height of the tree.

As the Segment Tree is a Full Binary Tree, the height of the tree will correspond log N, being N the number of nodes. So the complexity for retrieving data from a leaf is O(log N).

Parameters

	segtree	The segtree
in	*pos*	The position
in	*start*	The start of the interval (0-indexed, inclusive)
in	*end*	The end of the interval (0-indexed, inclusive)
in	*currStart*	The current start of the interval (0-indexed, inclusive)
in	*currEnd*	The current end of the interval (0-indexed, inclusive)

Returns

The results of the query

4.7.1.6 SegTree_rangeUpdate()

```
void SegTree_rangeUpdate (
            Cell * segtree,
            int pos,
            int start,
            int end,
            int currStart,
            int currEnd,
            int(*)(int n) transform )
```

Recursively updates the Segment Tree.

Complexity:

As the updates are realized on the leafs of the tree, and then the nodes that represent the intervals have to be updated, the cost on the update will be related to the amount of nodes that has to be updated plus the height of the tree. The cost to reach a leaf is log H, being H the height of the tree. Let LF the number of leafs to be updated, that is the same as the size of the interval to be updated (end-start-1).

As the Segment Tree is a Full Binary Tree, the height of the tree will correspond log N, being N the number of nodes. So the complexity for retrieving data from a leaf is O(log N).

The complexity of a update is O(LF·log N).

Parameters

	segtree	The segtree
in	*pos*	The current position on the segtree array
in	*start*	The start of the interval (0-indexed, inclusive)
in	*end*	The end of the interval (0-indexed, inclusive)
in	*currStart*	The current start of the interval (0-indexed, inclusive)
in	*currEnd*	The current end of the interval (0-indexed, inclusive)

4.7.1.7 SegTree_size()

```
int SegTree_size (
            int n )
```

Calculates the Segment Tree's number of nodes with the given the input array size.

The number of nodes n required for a segtree is calculated with the length of the input array. In the best case, the length L is a power of 2, so the number of nodes is n=2·L-1. If L is not a power of 2, we have to find the next power of 2 after L. In this case, the number of nodes will not pass from n=2·2^((log L) + 1) - 1, which can be simplified to n=4·L-1.

Being n the input array length, let np the power of 2 that is equal or higher than n. The size of the Segment Tree array will be 2·np-1.

The strategy adopted here is to calculate the log2 of n and get the ceil of the result. This way, we can power 2 for the result and get the next power of 2. Then, we use 2·(2^np)-1 to get the the size of the Segment Tree array.

Complexity: (1)

Parameters

in	*n*	The input array size

Returns

The size of the Segment Tree array

4.7.1.8 SegTree_update()

```
void SegTree_update (
            Cell * segtree,
            int n,
            int start,
            int end,
            int(*)(int n) transform )
```

Interface for the recursive segtree update.

Complexity: The complexity here is related to the complexity of the SegTree_rangeUpdate function, which is O(L↩
F·log N), being N the number of nodes of the tree, and LF is end - start - 1.

The number of nodes N, in the best case is 2·n-1, with n being the input array length. In worst case N will not pass from 4·n-1. The full explanation for this is on the SegTree_size function description.

With 2·n-1 or 4·n-1, the complexity of the function is O(LF·log n).

Parameters

	segtree	The segtree
in	*n*	The length of the input array
in	*start*	The start of the interval (1-indexed, inclusive)
in	*end*	The end of the interval (1-indexed, inclusive)
in	*transform*	The transform function

4.8 src/matriz.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "lib/matrix.h"
#include "lib/cell.h"
```

Functions

- int main ()
  Main function.

4.8.1 Function Documentation

4.8.1.1 main()

```
int main ( )
```

Main function.

This function uses the Nubby solution based on a matrix that stores the data for each possible interval.

Complexity: $O(n^3)$

Returns

0

Index