L6: Symbolic Processing

CS1101S: Programming Methodology

Martin Henz

September 15, 2021

Support

- Next week is Recess Week

Support

- Next week is Recess Week
- Midterm on Wednesday Week 7, 29/9, 10am

Support

- Next week is Recess Week
- Midterm on Wednesday Week 7, 29/9, 10am
- Consultations (remedial sheet)

Support

- Next week is Recess Week
- Midterm on Wednesday Week 7, 29/9, 10am
- Consultations (remedial sheet)
- Mastery checks

Update on Contests

Game of Tones: Contest opened on Monday, entry submissions due on 23/9 (Can we get to hear interesting sounds?)

Update on Contests

Game of Tones: Contest opened on Monday, entry submissions due on 23/9 (Can we get to hear interesting sounds?)

The Choreographer: voting ended on Monday...

Update on Contests

Game of Tones: Contest opened on Monday, entry submissions due on 23/9 (Can we get to hear interesting sounds?)

The Choreographer: voting ended on Monday... drum roll...

1. Binary Search and Binary Search Trees (2.3.3)
2. Sorting
3. Differentiation

Scenario: Finding the bursts

Tyre manufacturer is filming stress-tests of their products

Scenario: Finding the bursts

Tyre manufacturer is filming stress-tests of their products

- hundreds of videos (3 depicted), each with thousands of frames (10 depicted)

Scenario: Finding the bursts

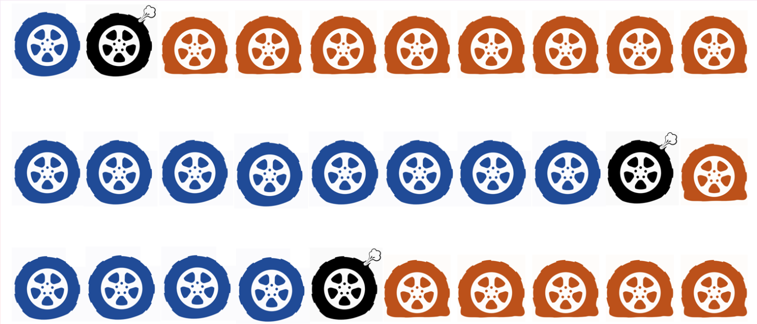
Tyre manufacturer is filming stress-tests of their products

- hundreds of videos (3 depicted), each with thousands of frames (10 depicted)
- in each video, find the frame in which the tyre bursts

Scenario: Finding the bursts

Tyre manufacturer is filming stress-tests of their products

- hundreds of videos (3 depicted), each with thousands of frames (10 depicted)
- in each video, find the frame in which the tyre bursts



Find the bursting frame in video i

```
function linear_search(video, n) {
    const frame = get_frame_from_video(video, n);
    return bursting_tyre(frame)
           ? n
           : linear_search(video, n + 1);
}
linear_search(stress_test, 0);
```

.

Knowledge about tyres

The Truth about Tyres

In the beginning, every tyre is intact.
It remains intact, until the bursting frame,
and then it is broken and remains broken.

Knowledge about tyres

The Truth about Tyres

In the beginning, every tyre is intact.
It remains intact, until the bursting frame,
and then it is broken and remains broken.

How could we exploit this property?

What if we just pick a value k?
What information do we gain
from calling `broken_tyre`?

Find that number with binary search

```
function binary_search(video, n, m) {
  if (m <= n) {
    return m;
  } else {
    const k = math_ceil((n + m) / 2);
    const frame = get_frame_from_video(video, k);
    return broken_tyre(frame)
           ? binary_search(video, n, k - 1)
           : binary_search(video, k, m);
  }
}
```

Observations

Runtime

At each step, we cut the search space in half. If problem size is 2^n, we get to size 1 after n steps.

Observations

Runtime

At each step, we cut the search space in half. If problem size is 2^n, we get to size 1 after n steps. Runtime: $\Theta(\log n)$

Observations

Runtime

At each step, we cut the search space in half. If problem size is 2^n, we get to size 1 after n steps. Runtime: $\Theta(\log n)$

Why does this work?

We need to assume that "The Truth about Tyres" holds:
"In the beginning, every tyre is intact.
It remains intact, until the bursting frame,
and then it is broken and remains broken."

Observations

Runtime

At each step, we cut the search space in half. If problem size is 2^n, we get to size 1 after n steps. Runtime: $\Theta(\log n)$

Why does this work?

We need to assume that "The Truth about Tyres" holds:
"In the beginning, every tyre is intact.
It remains intact, until the bursting frame,
and then it is broken and remains broken."

Under this assumption...

...each test cuts the search space in half

Binary search for entries

Binary search for entries

The problem

Check if a entry is included in a (changing) collection of entries.

Binary search for entries

The problem

Check if a entry is included in a (changing) collection of entries.

Property of entries

A *total order* exists: Two entries can be compared.
If they are "equal" then they are "the same". Otherwise, one is either "larger'
' or "smaller" than the other.

Binary search for entries

The problem

Check if a entry is included in a (changing) collection of entries.

Property of entries

A *total order* exists: Two entries can be compared.

If they are "equal" then they are "the same". Otherwise, one is either "larger'
' or "smaller" than the other.

Very easy for Numbers and Strings: <

Binary search for entries

The problem

Check if a entry is included in a (changing) collection of entries.

Property of entries

A *total order* exists: Two entries can be compared.
If they are "equal" then they are "the same". Otherwise, one is either "larger'
' or "smaller" than the other.
Very easy for Numbers and Strings: <
quite easy for most data structures.

Binary search for entries

The problem

Check if a entry is included in a (changing) collection of entries.

Property of entries

A *total order* exists: Two entries can be compared.
If they are "equal" then they are "the same". Otherwise, one is either "larger'
' or "smaller" than the other.
Very easy for Numbers and Strings: <
quite easy for most data structures.

Idea

Invest: turn collection into a
data structure that makes search easy.

Binary Trees

Definition

A binary tree of a certain type is an *abstraction* for binary search.

Binary Trees

Definition

A binary tree of a certain type is an *abstraction* for binary search.

Binary search trees as data structures

A binary search tree is the empty tree,
or it has an entry,
a left branch and a right branch
(both also binary search trees).

Binary Trees

Definition

A binary tree of a certain type is an *abstraction* for binary search.

Binary search trees as data structures

A binary search tree is the empty tree,
or it has an entry,
a left branch and a right branch
(both also binary search trees).

Search tree property

All entries in the left branch
are smaller than the entry, and
all entries in the right branch
are larger than the entry.

Mission "Search and Rescue" (out today)

Mission "Search and Rescue" (out today)

Given

A binary search tree of Strings and a String

Mission "Search and Rescue" (out today)

Given

A binary search tree of Strings and a String

The problem specfication

Check if the String occurs in the tree.

Mission "Search and Rescue" (out today)

Given

A binary search tree of Strings and a String

The problem specfication

Check if the String occurs in the tree.

More precise specification

`find(tree, n)` returns
`true` if `n` occurs in `tree`,
and `false` otherwise.

Mission “Search and Rescue” (out today)

Given

A binary search tree of Strings and a String

The problem specfication

Check if the String occurs in the tree.

More precise specification

`find(tree, n)` returns
`true` if `n` occurs in `tree`,
and `false` otherwise.

Abstraction

Mission "Search and Rescue" (out today)

Given

A binary search tree of Strings and a String

The problem specfication

Check if the String occurs in the tree.

More precise specification

`find(tree, n)` returns
`true` if `n` occurs in `tree`,
and `false` otherwise.

Abstraction

Watch out: The problem uses
the *binary tree abstraction*

The problem of sorting

The problem of sorting

Given

List `xs` of elements from given universe X, and *total order* on X.

The problem of sorting

Given

List `xs` of elements from given universe X, and *total order* on X.

Wanted

A permutation of of `xs` such that each element is greater than or equal to the previous one, with respect to the total order.

The problem of sorting

Given

List `xs` of elements from given universe X, and *total order* on X.

Wanted

A permutation of of `xs` such that each element is greater than or equal to the previous one, with respect to the total order.

Comparisons only

The only allowed operations on the elements are comparisons, such as <, >, <=, >=, === or !==.

How to sort list with n elements?

How to sort list with n elements?

Our strategy

Wishful thinking!

How to sort list with n elements?

Our strategy

Wishful thinking!
Imagine we can sort lists with m elements, where $m < n$.

How to sort list with n elements?

Our strategy

Wishful thinking!

Imagine we can sort lists with m elements, where $m < n$.

Approach A

$m = n - 1$

How to sort list with n elements?

Our strategy

Wishful thinking!
Imagine we can sort lists with m elements, where $m < n$.

Approach A

$m = n - 1$

Approach B

$m = n/2$

Algorithm A1

Algorithm A1

Idea

Sort the tail of the given list using wishful thinking! *Insert* the head in the right place.

Algorithm A1

Idea

Sort the tail of the given list using wishful thinking! *Insert* the head in the right place.

In Source

```
function insertion_sort(xs) {
   return is_null(xs)
          ? xs
          : insert(head(xs),
                   insertion_sort(
                       tail(xs)));
}
```

A1: Insertion Sort

```
function insert(x, xs) {
    return is_null(xs)
           ? list(x)
           : x <= head(xs)
           ? pair(x,xs)
           : pair(head(xs), insert(x, tail(xs)));
}
function insertion_sort(xs) {
    return is_null(xs)
           ? xs
           : insert(head(xs),
                        insertion_sort(
                            tail(xs)));
}
```

Order of Growth?

```
function insert(x, xs) {
    return is_null(xs)
           ? list(x)
           : x <= head(xs)
           ? pair(x,xs)
           : pair(head(xs), insert(x, tail(xs)));
}
function insertion_sort(xs) {
    return is_null(xs)
           ? xs
           : insert(head(xs),
                    insertion_sort(tail(xs)));
}
```

Algorithm A2

Algorithm A2

Idea

Find the smallest element x
and remove it from the list.

Algorithm A2

Idea

Find the smallest element x
and remove it from the list.
Sort the remaining list,
and put x in front.

A2: Selection Sort in Source

```
function selection_sort(xs) {
    if (is_null(xs)) {
        return xs;
    } else {
        const x = smallest(xs); // P6A
        return pair(x,
                    selection_sort(
                        remove(x, xs)));
    }
}
```

Order of growth?

Recall: How to sort list with n elements?

Our strategy

Wishful thinking! Imagine we can sort lists with m elements, where $m < n$.

Approach A

$m = n - 1$

Recall: How to sort list with n elements?

Our strategy

Wishful thinking! Imagine we can sort lists with m elements, where $m < n$.

Approach A

$m = n - 1$

Approach B

$m = n/2$

Recall: How to sort list with n elements?

Our strategy

Wishful thinking! Imagine we can sort lists with m elements, where $m < n$.

Approach A

$m = n - 1$

Approach B

$m = n/2$

Idea of Algorithm B1

Split the list **in half**,
sort each half using wishful thinking,
merge the sorted lists together

B1: Merge Sort

```
function merge_sort(xs) {
    if (is_null(xs) ||
        is_null(tail(xs))) {
        return xs;
    } else {
        const mid = middle(length(xs));
        return merge(merge_sort(take(xs, mid)),
                     merge_sort(drop(xs, mid)));
    }
}                   // take, drop: see P6A
```

B1: Merge Sort (function merge)

```
function merge(xs, ys) {
    if (is_null(xs)) {
        return ys;
    } else if (is_null(ys)) {
        return xs;
    } else {
        const x = head(xs);
        const y = head(ys);
        return (x < y)
               ? pair(x, merge(tail(xs), ys))
               : pair(y, merge(xs, tail(ys)));
    }
}
```

Helper functions for Merge Sort

```
// take half, rounded down
function middle(n) {
    // ???
}
// put the first n elements of xs into a list
function take(xs, n) {
    // ???
}
// drop first n elements from list, return rest
function drop(xs, n) {
    // ???
}
```

Complexity of Merge Sort?

Question

What is the order of growth of the runtime for sorting an n-element list using Merge Sort?

Complexity of Merge Sort?

Question

What is the order of growth of the runtime for sorting an n-element list using Merge Sort?

Answer

Come to Reflection R6!

Algorithm B2

Question: Is there another way of "splitting" and "combining"?

Algorithm B2

Question: Is there another way of “splitting” and “combining”?

Answer: Yes! See Mission
“Sorting Things Out”

Representing functions: Directly

Our first approach is to represent functions *directly* in Source.

Example

```
function my_f(x) {
    return x * x + 1;
}
function eval_numeric(f, x) {
    return f(x);
}
eval_numeric(my_f, 7);
// returns 50
```

Describing the graph of functions

```
// make graph curve for function f;
// graph covers the range for x from x1 to x2

function function_to_graph(f, x1, x2) {
  function graph(t) {
    // for t from 0 to 1,
    // x ranges from x1 to x2
    const x = x1 + t * (x2 - x1);
    return make_point(x, f(x));
  }
  return graph;
}
```

Plotting the graph of functions

```
// plot the graph of function f from x1 to x2
function plot_graph(f, x1, x2) {
    return (draw_connected_full_view(200))
             (function_to_graph(f,x1,x2));
}
```

Numerical Differentiation

```
// numerical differentiation; simplest method
const dx = 0.0001;
function deriv_numeric(f) {
    return x => (f(x + dx) - f(x)) / dx;
}
```

Symbolic differentiation

The rules

$$
\begin{aligned}
\frac{dc}{dx} &= 0 \qquad \text{for constant } c \\
\frac{dx}{dx} &= 1 \\
\frac{d(u+v)}{dx} &= \frac{du}{dx} + \frac{dv}{dx} \\
\frac{d(uv)}{dx} &= u\frac{dv}{dx} + v\frac{du}{dx}
\end{aligned}
$$

Symbolic differentiation: Example

Remember from high school:

$$
\begin{aligned}
f(x) &= x^2 + x + 4 \\
f'(x) &= 2x + 1
\end{aligned}
$$

Symbolic differentiation: Example

Remember from high school:

$$\begin{aligned} f(x) &= x^2 + x + 4 \\ f'(x) &= 2x + 1 \end{aligned}$$

We want to compute *the formula* for f', not just a function that behaves like f'!

Symbolic differentiation: Example

Remember from high school:

$$\begin{aligned} f(x) &= x^2 + x + 4 \\ f'(x) &= 2x + 1 \end{aligned}$$

We want to compute *the formula* for f', not just a function that behaves like f'!

First step: represent formulas as data structures

Symbolic differentiation: Example

Remember from high school:

$$\begin{aligned} f(x) &= x^2 + x + 4 \\ f'(x) &= 2x + 1 \end{aligned}$$

We want to compute *the formula* for f', not just a function that behaves like f'!

First step: represent formulas as data structures

```
// exp represents x * x + x + 4
const exp = make_sum(make_product("x", "x"),
                     make_sum("x", make_number(4)));
```

Symbolic evaluation

Example expression

```
// exp represents x * x + x + 4
const exp = make_sum(make_product("x", "x"),
                     make_sum("x", make_number(4)));
```

Symbolic evaluation

Example expression

```
// exp represents x * x + x + 4
const exp = make_sum(make_product("x", "x"),
                     make_sum("x", make_number(4)));
```

Symbolic evaluation

```
eval_symbolic(exp, "x", 3);
// should return 16
```

Implementing eval_symbolic

```
function eval_symbolic(exp, name, val) {
 return is_number(exp)
    ? value(exp)
    : is_variable(exp)
    ? (is_same_var(exp, name) ? val : NaN)
    : is_sum(exp)
    ? eval_symbolic(addend(exp), name, val) +
      eval_symbolic(augend(exp), name, val)
    : is_prod(exp)
    ? eval_symbolic(multiplier(exp), name, val) *
      eval_symbolic(multiplicand(exp), name, val)
    : "unknown exp type: " + exp;
}
eval_symbolic(square, "x", 4);
```

Symbolic differentiation

Example expression

```
const exp = make_sum(make_product("x", "x"),
                     make_sum("x",make_number(4)));
```

Symbolic differentiation

Example expression

```
const exp = make_sum(make_product("x", "x"),
                     make_sum("x",make_number(4)));
```

Symbolic differentiation

```
deriv_symbolic(exp, "x");  // should return
// make_sum(make_product("x",make_number(2)),
//          make_number(1))
eval_symbolic(deriv_symbolic(exp,"x"),"x",3);
// should return 7
```

Symbolic representation of functions

- `is_number(e)`: Is e a number?
- `value(e)`: Value of number expression e
- `make_number(n)`: Make number expression with value `n`
- `is_variable(e)`: Is e a variable?
- `is_same_var(v1, v2)`: Are v1 and v2 same variable?
- `is_sum(e)`: Is e a sum?
- `addend(e), augend(e)`: Addend/augend of sum e
- `make_sum(a1, a2)`: Construct the sum of a1 and a2
- `is_product(e)`: Is e a product?
- `multiplier(e)`: Multiplier of the product e
- `multiplicand(e)`: Multiplicand of the product e
- `make_product(m1,m2)`: Construct product of `m1` and `m2`

Definition of deriv_symbolic

```
function deriv_symbolic(exp, x) {
  return is_number(exp)
    ? make_number(0)
    : is_variable(exp)
    ? make_number(is_same_var(exp, x) ? 1 : 0)
    : is_sum(exp)
    ? make_sum(deriv_symbolic(addend(exp), x),
                deriv_symbolic(augend(exp), x))
    : is_product(exp)
    ? make_sum(make_prod(multiplier(exp),
          deriv_symbolic(multiplicand(exp), x)),
        make_prod(multiplicand(exp),
          deriv_symbolic(multiplier(exp), x)))
    : "unknown exp type: " + exp;
}
```

Revisiting example

Example expression

```
const exp = make_sum(make_product("x", "x"),
                     make_sum("x", make_number(4)));
```

Revisiting example

Example expression

```
const exp = make_sum(make_product("x", "x"),
                     make_sum("x", make_number(4)));
```

Symbolic differentiation

```
deriv_symbolic(exp, "x");  // should return
// make_sum(make_product("x",make_number(2)),
            make_number(1))
```

Revisiting example

Example expression

```
const exp = make_sum(make_product("x", "x"),
                     make_sum("x", make_number(4)));
```

Symbolic differentiation

```
deriv_symbolic(exp, "x");  // should return
// make_sum(make_product("x",make_number(2)),
            make_number(1))

// but instead returns complicated expression
// equivalent to: x * 1 + x * 1 + 1 + 0
```

Simplifying formulas: make_sum

```
function make_sum(a1,a2) {
  return is_number(a1) && value(a1) === 0
      ? a2
      : is_number(a2) && value(a2) === 0
        ? a1
        : is_number(a1) && is_number(a2)
          ? make_number(value(a1) + value(a2))
          : list("+", a1, a2);
}
```

Simplifying formulas: make_product

```
function make_product(m1, m2) {
   return (is_number(m1) && value(m1) === 0)
          ||
          (is_number(m2) && value(m2) === 0)
          ? make_number(0)
          : is_number(m1) && value(m1) === 1
            ? m2
          : is_number(m2) && value(m2) === 1
            ? m1
            : is_number(m1) && is_number(m2)
              ? make_number(m1*m2)
              : list("*",m1,m2);
   }
}
```

Key ideas today

Key ideas today

- Binary search

Key ideas today

- Binary search and binary search trees

Key ideas today

- Binary search and binary search trees
- Sorting: A1, A2: $m = n - 1$, B1: $m = n/2$

Key ideas today

- Binary search and binary search trees
- Sorting: A1, A2: $m = n - 1$, B1: $m = n/2$
- Symbolic differentiation: Ideas become data!

Special Announcement

Special Announcement

- Joel Low: Principal Software Engineer at Grab (see LinkedIn)

Special Announcement

- Joel Low: Principal Software Engineer at Grab (see LinkedIn)
- Joel is former Avenger and will give a guest lecture during the CS1101S Brief *this Friday*

Special Announcement

- Joel Low: Principal Software Engineer at Grab (see LinkedIn)
- Joel is former Avenger and will give a guest lecture during the CS1101S Brief *this Friday*
- Title of lecture: Engineering a software career