DS-210: PROGRAMMING FOR DATA SCIENCE

LECTURE 35

1. ERROR HANDLING IN RUST

2. ALGORITHM DESIGN: DYNAMIC PROGRAMMING

ERROR HANDLING IN RUST

Two basic options:

- terminate when an error occurs: macro `panic!(...)`
- pass information about an error: enum `Result<T,E>`

MACRO `panic!(...)`

- Use for unrecoverable errors
- Terminates the application

MACRO `panic!(...)`

- Use for unrecoverable errors
- Terminates the application

In [2]:
```
fn divide(a:u32, b:u32) -> u32 {
    if b == 0 {
        panic!("I'm sorry, Dave. I'm afraid I can't do that.");
    }
    a/b
}
```

MACRO `panic!(...)`

- Use for unrecoverable errors
- Terminates the application

In [2]:
```
fn divide(a:u32, b:u32) -> u32 {
    if b == 0 {
        panic!("I'm sorry, Dave. I'm afraid I can't do that.");
    }
    a/b
}
```

In [3]:
```
divide(20,7)
```

Out[3]: 2

MACRO `panic!(...)`

- Use for unrecoverable errors
- Terminates the application

In [2]:
```
fn divide(a:u32, b:u32) -> u32 {
    if b == 0 {
        panic!("I'm sorry, Dave. I'm afraid I can't do that.");
    }
    a/b
}
```

In [3]:
```
divide(20,7)
```

Out[3]:
```
2
```

In [4]:
```
divide(20,0)
```

```
thread '<unnamed>' panicked at 'I'm sorry, Dave. I'm af
raid I can't do that.', src/lib.rs:4:9
stack backtrace:
   0: std::panicking::begin_panic
   1: run_user_code_3
   2: evcxr::runtime::Runtime::run_loop
   3: evcxr::runtime::runtime_hook
   4: evcxr_jupyter::main
note: Some details are omitted, run with `RUST_BACKTRAC
E=full` for a verbose backtrace.
Segmentation fault.
   0: evcxr::runtime::Runtime::install_crash_handlers::
segfault_handler
   1: <unknown>
   2: mi_free_generic
   3: alloc::alloc::dealloc
             at /rustc/9d1b2106e23b1abd32fce1f17267604a
5102f57a/library/alloc/src/alloc.rs:105:14
      <alloc::alloc::Global as core::alloc::Allocator
>::deallocate
```

ENUM `Result<T,E>`

```rust
enum Result<T,E> {
    Ok(T),
    Err(E),
}
```

Functions can use it to

- return a result
- or information about an encountered error

ENUM `Result<T,E>`

```
enum Result<T,E> {
    Ok(T),
    Err(E),
}
```

Functions can use it to

- return a result
- or information about an encountered error

In [5]:
```
fn divide(a:u32, b:u32) -> Result<u32, &'static str> {
    if b != 0 {
        Ok(a / b)
    } else {
        Err("Division by zero")
    }
}
```

ENUM `Result<T,E>`

```
enum Result<T,E> {
    Ok(T),
    Err(E),
}
```

Functions can use it to

- return a result
- or information about an encountered error

In [5]:
```
fn divide(a:u32, b:u32) -> Result<u32, &'static str> {
    if b != 0 {
        Ok(a / b)
    } else {
        Err("Division by zero")
    }
}
```

In [6]:
```
divide(20,7)
```

Out[6]:
```
Ok(2)
```

In [7]:
```
divide(20,0)
```

Out[7]:
```
Err("Division by zero")
```

ENUM Result<T,E>

```rust
enum Result<T,E> {
    Ok(T),
    Err(E),
}
```

Functions can use it to

- return a result
- or information about an encountered error

In [5]:
```rust
fn divide(a:u32, b:u32) -> Result<u32, &'static str> {
    if b != 0 {
        Ok(a / b)
    } else {
        Err("Division by zero")
    }
}
```

In [6]:
```rust
divide(20,7)
```

Out[6]:
```
Ok(2)
```

In [7]:
```rust
divide(20,0)
```

Out[7]:
```
Err("Division by zero")
```

- Useful when the error best handled somewhere else
- **Example:** input/output subroutines in the standard library

COMMON PATTERN: PROPAGATING ERRORS

- We are interested in the positive outcome: `t` in `Ok(t)`
- But if an error occurs, we want to propagate it
- This can be handled using `match` statements

In [8]:
```
// compute a/b + c/d
fn calculate(a:u32, b:u32, c:u32, d:u32) -> Result<u32, &'static str> {
    let first = match divide(a,b) {
        Ok(t) => t,
        Err(e) => return Err(e),
    };
    let second = match divide(c,d) {
        Ok(t) => t,
        Err(e) => return Err(e),
    };
    Ok(first + second)
}
```

COMMON PATTERN: PROPAGATING ERRORS

- We are interested in the positive outcome: `t` in `Ok(t)`
- But if an error occurs, we want to propagate it
- This can be handled using `match` statements

In [8]:
```
// compute a/b + c/d
fn calculate(a:u32, b:u32, c:u32, d:u32) -> Result<u32, &'static str> {
    let first = match divide(a,b) {
        Ok(t) => t,
        Err(e) => return Err(e),
    };
    let second = match divide(c,d) {
        Ok(t) => t,
        Err(e) => return Err(e),
    };
    Ok(first + second)
}
```

In [9]:
```
calculate(16,4,18,3)
```

Out[9]:
```
Ok(10)
```

In [10]:
```
calculate(16,0,18,3)
```

Out[10]:
```
Err("Division by zero")
```

THE QUESTION MARK SHORTCUT

- Place `?` after an expression that returns `Result<T,E>`
- This will:
    - give the content of `Ok(t)`
    - or return `Err(e)` from the encompassing function

THE QUESTION MARK SHORTCUT

- Place `?` after an expression that returns `Result<T,E>`
- This will:
    - give the content of `Ok(t)`
    - or return `Err(e)` from the encompassing function

In [11]:

```
// compute a/b + c/d
fn calculate(a:u32, b:u32, c:u32, d:u32) -> Result<u32, &'static str> {
    Ok(divide(a,b)? + divide(c,d)?)
}
```

THE QUESTION MARK SHORTCUT

- Place `?` after an expression that returns `Result<T,E>`
- This will:
    - give the content of `Ok(t)`
    - or return `Err(e)` from the encompassing function

In [11]:
```
// compute a/b + c/d
fn calculate(a:u32, b:u32, c:u32, d:u32) -> Result<u32, &'static str> {
    Ok(divide(a,b)? + divide(c,d)?)
}
```

In [12]:
```
calculate(16,4,18,3)
```

Out[12]:
```
Ok(10)
```

In [13]:
```
calculate(16,0,18,3)
```

Out[13]:
```
Err("Division by zero")
```

1. ERROR HANDLING IN RUST

2. ALGORITHM DESIGN: DYNAMIC PROGRAMMING

BIG PICTURE: REST OF THIS LECTURE AND NEXT

Review a few approaches to algorithm design:

- dynamic programming
- greedy approach
- divide and conquer

HOMEWORK 9: BEST DECISION TREE FOR A CLASSIFICATION PROBLEM

Input: set of n labelled points (x_i, z_i), where $x_i \in \mathbb{R}$ and $z_i \in \{0, 1\}$

Goal: find decision tree with L leaves and highest accuracy on the input set

HOMEWORK 9 RESTRICTION: $L = 2$

How to solve it?

HOMEWORK 9 RESTRICTION: $L = 2$

How to solve it?

Two–leaf decision tree: if $x < T$, output α, else output $(1 - \alpha)$

HOMEWORK 9 RESTRICTION: $L = 2$

How to solve it?

Two–leaf decision tree: if $x < T$, output α, else output $(1 - \alpha)$

Two parameters: T and α

- suffices to try $T = x_i$ for all x_i's and $\alpha \in \{0, 1\}$
- at most $2n$ options

HOMEWORK 9 RESTRICTION: $L = 2$

How to solve it?

Two–leaf decision tree: if $x < T$, output α, else output $(1 - \alpha)$

Two parameters: T and α

- suffices to try $T = x_i$ for all x_i's and $\alpha \in \{0, 1\}$
- at most $2n$ options

Algorithms:

- **Simple:** evaluate accuracy for each T and $\alpha \;\Rightarrow\; O(n^2)$ time
- **More sophisticated:** sort points, move the threshold for each α updating accuracies $\;\Rightarrow\; O(n \log n)$ time

GENERAL L

GENERAL L

How do decision trees with at most L leaves partition the line?

GENERAL L

How do decision trees with at most L leaves partition the line?

- at most L line segments: prediction fixed to 0 or 1 for each
- $\binom{n}{L-1} = O\left(n^{L-1}\right)$ thresholds configurations to consider
- test each: $O\left(n^{L}\right)$–time algorithm

GENERAL L

How do decision trees with at most L leaves partition the line?

- at most L line segments: prediction fixed to 0 or 1 for each
- $\binom{n}{L-1} = O\left(n^{L-1}\right)$ thresholds configurations to consider
- test each: $O\left(n^{L}\right)$–time algorithm

OUR GOAL: MUCH FASTER ALGORITHM

DEFINE SUBPROBLEMS

Simplifying assumption: $x_1 < x_2 < \ldots < x_n$

DEFINE SUBPROBLEMS

Simplifying assumption: $x_1 < x_2 < \ldots < x_n$

$M[l, k] =$ the minimum number of mistakes, when classifying the first k points, using at most l ranges

- $l \in \{1, \ldots, L\}$
- $k \in \{1, \ldots, n\}$

DEFINE SUBPROBLEMS

Simplifying assumption: $x_1 < x_2 < \ldots < x_n$

$M[l, k] =$ the minimum number of mistakes, when classifying the first k points, using at most l ranges

- $l \in \{1, \ldots, L\}$
- $k \in \{1, \ldots, n\}$

$M[L, n]$ will give the best accuracy

HOW TO COMPUTE $M[l, k]$?

$M[l, k] =$ the minimum number of mistakes, when classifying the first k points, using at most l ranges

- $l \in \{1, \ldots, L\}$
- $k \in \{1, \ldots, n\}$

HOW TO COMPUTE $M[l, k]$?

$M[l, k] =$ the minimum number of mistakes, when classifying the first k points, using at most l ranges

- $l \in \{1, \ldots, L\}$
- $k \in \{1, \ldots, n\}$

ONE LABEL PREDICTIONS ON $\{x_k : i \leq k \leq j\}$

- Define $S[i, j] =$ number of mispredictions for one label classifiers on this set
- $S[i, j]$ minimum of the numbers of 0 and 1 labels on this set

HOW TO COMPUTE $M[l, k]$?

$M[l, k] =$ the minimum number of mistakes, when classifying the first k points, using at most l ranges

- $l \in \{1, \dots, L\}$
- $k \in \{1, \dots, n\}$

ONE LABEL PREDICTIONS ON $\{x_k : i \leq k \leq j\}$

- Define $S[i, j] =$ number of mispredictions for one label classifiers on this set
- $S[i, j]$ minimum of the numbers of 0 and 1 labels on this set

COMPUTE $M[1, k]$ FOR ALL k

- $M[1, k] \leftarrow S[1, k]$
- $O(n)$ time overall

HOW TO COMPUTE $M[l, k]$?

$M[l, k] =$ the minimum number of mistakes, when classifying the first k points, using at most l ranges

- $l \in \{1, \dots, L\}$
- $k \in \{1, \dots, n\}$

$S[i, j] =$ the minimum number of mistakes, when classifying points $\{x_k : i \leq k \leq j\}$ with one range

HOW TO COMPUTE $M[l, k]$?

$M[l, k] =$ the minimum number of mistakes, when classifying the first k points, using at most l ranges

- $l \in \{1, \dots, L\}$
- $k \in \{1, \dots, n\}$

$S[i, j] =$ the minimum number of mistakes, when classifying points $\{x_k : i \leq k \leq j\}$ with one range

COMPUTE $M[l, k]$ FOR $l \geq 2$ AND ALL k

$$M[l, k] \leftarrow \min_{i=\{1,\dots,k\}} (M[l-1, i] + S[i+1, k])$$

TIME COMPLEXITY?

TIME COMPLEXITY?

- Computing $S[i, j]$ for all i and j: $O(n^2)$

TIME COMPLEXITY?

- Computing $S[i, j]$ for all i and j: $O(n^2)$
- Computing $M[l+1, i]$ for all i from $M[l, i]$: $O(n^2)$

TIME COMPLEXITY?

- Computing $S[i, j]$ for all i and j: $O(n^2)$
- Computing $M[l + 1, i]$ for all i from $M[l, i]$: $O(n^2)$
- Total running time: $O(L) \cdot O(n^2) = O(Ln^2)$
- Much better than the more straightforward $O(n^L)$

RECONSTRUCTING THE SOLUTION

- This gives us $M[L, n] =$ the minimum number of mistakes overall
- How to get the best solution, not just the best cost?

RECONSTRUCTING THE SOLUTION

- This gives us $M[L, n] =$ the minimum number of mistakes overall
- How to get the best solution, not just the best cost?

Iteratively:

- Start from $M[L, n]$
- Find i the best $M[L-1, i] + S[i+1, n]$
- Label $\{x_{i+1}, \ldots, x_n\}$ with the better of 0 and 1
- Continue with $M[L-1, i]$
- ...

DYNAMIC PROGRAMMING IN GENERAL

- Define a small number of subproblems that are
  - sufficient to solve the general problem
  - helpful to solve each other

DYNAMIC PROGRAMMING IN GENERAL

- Define a small number of subproblems that are
  - sufficient to solve the general problem
  - helpful to solve each other

The most classic example: edit distance

- minimum number of edits to turn one string into another
- edits: deletions, insertions, substitutions
- correcting spelling mistakes: how far are two words?

Can you solve it?