南開大學
Nankai University

一个简单的展示

a simple pre

崔家贺
南开大学　软件学院

2022 年 12 月 14 日

目录

intro

- 张老师说过 lisp 很厉害

intro

- 张老师说过 lisp 很厉害
- 张老师说过他整个博士工作就是用 lisp 做的

intro

- 张老师说过 lisp 很厉害
- 张老师说过他整个博士工作就是用 lisp 做的
- 所以我用 lisp 写了一遍搜索八数码

intro

- 张老师说过 lisp 很厉害
- 张老师说过他整个博士工作就是用 lisp 做的
- 所以我用 lisp 写了一遍搜索八数码
- 具体的来说是 lisp 的 jvm 方言 clojure

intro

- 张老师说过 lisp 很厉害
- 张老师说过他整个博士工作就是用 lisp 做的
- 所以我用 lisp 写了一遍搜索八数码
- 具体的来说是 lisp 的 jvm 方言 clojure
- 介绍一下 lisp

intro

lisp 很**好**

intro

lisp 很**好**

- 基于过程和纯函数

intro

lisp 很**好**

- 基于过程和纯函数
- 一种新的编程思想

intro

lisp 很**好**

- 基于过程和纯函数
- 一种新的编程思想
- 强大的宏机制

纯函数

- 无副作用

纯函数

- 无副作用
- 可以随意并行化和优化

纯函数

- 无副作用
- 可以随意并行化和优化
- 很多并行化计算都是是基于算子的

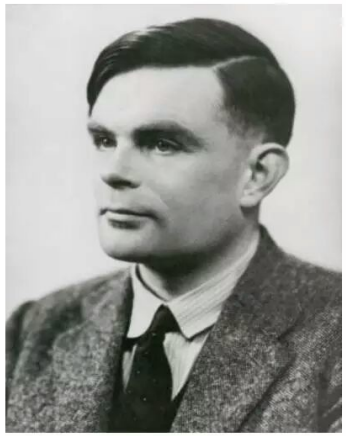
纯函数

- 无副作用
- 可以随意并行化和优化
- 很多并行化计算都是是基于算子的
- 编译器优化有一步就是把带状态的流程转成纯函数和延续点的形式

函数式思想

两种范式

阿隆佐·丘奇
λ 算子
函数式编程



阿伦·图灵
图灵机
命令式编程

图 1: lisp vs c

宏机制

- rust 的过程宏就是一个借鉴

宏机制

- rust 的过程宏就是一个借鉴
- 能够操作 ast

宏机制

- rust 的过程宏就是一个借鉴
- 能够操作 ast
- 可以实现很多好玩的东西

宏机制

- rust 的过程宏就是一个借鉴
- 能够操作 ast
- 可以实现很多好玩的东西
- 例如不对目标函数做任何修改实现输出中间变量

宏机制

例子来自官方 trace 库

```
=> (use 'clojure.tools.trace)

=> (trace (* 2 3)) ;; To trace a value
TRACE: 6
6

=> (trace "tag" (* 2 3)) ;; To trace a value and assign a trace tag
TRACE tag: 6
6

=> (deftrace fubar [x v] (+ x v)) ;; To trace a function call and its return value
=> (fubar 2 3)
TRACE t1107: (fubar 2 3)
TRACE t1107: => 5
5

=> (do (+ 1 3) (* 5 6) (/ 1 0))
ArithmeticException Divide by zero  clojure.lang.Numbers.divide (Numbers.java:156)
=> (trace-forms (+ 1 3) (* 5 6) (/ 1 0)) ;; To identify which form is failing
ArithmeticException Divide by zero
  Form failed: (/ 1 0)
  clojure.lang.Numbers.divide (Numbers.java:156)

(trace-ns myown.namespace) ;; To dynamically trace/untrace all fns in a name space (untrace-ns myown

(trace-vars myown.namespace/fubar) ;; To dynamically trace/untrace specific fns (untrace-vars myown.
```

Long Live the Lisp

lisp 的特性使得他在解决某些问题的时候非常简单
比如：

Long Live the Lisp

lisp 的特性使得他在解决某些问题的时候非常简单
比如：

- 快速排序
- Y 组合子
- 包括这个搜索问题

快速排序

```
(defn quick_sort [n]
  (cond
    (= nil n) []
    (<= (count n) 1) n
    :else (let [pivot (first n)
                others (rest n)]
      (concat
       (quick_sort (filter #(> pivot %) others))
        [pivot]
        (quick_sort (filter #(<= pivot %) others))))))
```

图 3: qsort

Y 组合子

```
(def Y 1 reference
  (fn [f]
    ((fn [x] (x x))
     (fn [x] (f (fn [y] ((x x) y)))))))
```

图 4: lispccomb

Y 组合子

当然 c++ 也不是不能写

```cpp
namespace details {
template <typename... Ts>
struct FHelper {
    template <typename R>
    struct RT {
        using StdFunc = std::function<R(std::function<R(Ts...)>, Ts...)>;
    };
};
template <typename T>
struct YHelper {
    static_assert(std::is_function_v<T>, "T must be a function");
    using Func = Ubpa::FuncTraits<T>;
    using R = typename Func::Return;
    using ArgList = typename Func::ArgList;
    using Sig = typename Func::Signature;
    using StdFunc = std::function<Sig>;
    using StdYf = typename Ubpa::ToOtherList_t<ArgList, FHelper>::template RT<R>::StdFunc;
    static constexpr StdFunc Comb(StdYf f) {
        return [&f](auto... args) {
            return f(Comb(f), args...);
        };
    }
};
} // namespace details
template <typename T>
inline constexpr auto Y = details::YHelper<T>::Comb;
```

图 5: cppccomb

看上去比 lisp 复杂许多（

Y 组合子

如果是 haskell（~~有个同学会讲 hs 系的语言，听他讲就行了~~

```
y f = f (y f)
```

图 6: hsccomb

~~看上去比 lisp 还要简单~~（

终于要讲八数码了

不放图了，去场外看代码

终于要讲八数码了

不放图了，去场外看代码 需要解决的问题：bfs 的时候没有找到合适的数据结构所以比较慢
其实可以在 java 里写一个合适的数据结构导入 clojure 里，但是感觉也不是上策
希望张老师可以给讲讲

~~前边没听懂咋办~~

bilibili 搜索 SICP(唯一编程神课)
~~既然张老师是 lisp 高手，我们是不是也能开个 SICP 的课~~
都说编程像构建魔法，这个课就是最高阶的法术
这并不是什么屠龙技，而是相当于内功的东西
纯函数和组合性这些东西是降低耦合的好工具
相当一部分的库也多使用函数式思想，例如 pytorch 的 tensor dispatch 部分和很多的持久化数据结构

为什么没写可视化

可视化虽然看上去好看，实际上并没有什么内在的东西（仅指这个搜索的问题）
而且动不动几万层，15 数码甚至是几十万层的搜索也不好用可视化来展示
我认为拖几个组件去写界面还不如学一下 SICP 课程，了解一些新东西
我感觉学这个东西还挺值得的，毕竟有个梗：“每个学 lisp 的人都应该自己造一个 lisp 方言”

目录

c++ 实现

实现了以下几个，每个都有 bfs 和 dfs 版

- 朴素搜索
- 只带随机化的朴素搜索
- 使用估价函数 1
- 使用估价函数 1+ 随机化
- 使用估价函数 2
- 使用估价函数 2+ 随机化

估价函数

$$h_1(st) = \sum_i |x[st_i]\text{-}x[target_i]| + |y[st_i]\text{-}y[target_i]|$$

意义是曼哈顿距离之和

$$h_2(st) = \sum_i max(|x[st_i]\text{-}x[target_i]|, |y[st_i]\text{-}y[target_i]|)$$

意义是切比雪夫距离之和

随机化

```cpp
auto get_h(CompressT st) -> i32 {
    static typename ST::DecodeT t;
    t = ST::decode(st);
    i32 ans = 0;
    if constexpr (hType == 0) { // h1
        for(int i = 0; i < N * N; ++i) {
            ans += (abs(i / N - tard[t[i]] / N) + abs(i % N - tard[t[i]] % N));
        }
    } else { // h2
        for(int i = 0; i < N * N; ++i) {
            ans += 2 * max(abs(i / N - tard[t[i]] / N), abs(i % N - tard[t[i]] % N));
        }
    }
    if constexpr (isRandom) {
        ans = ans * 10 + rng() % 11;
    }
    return ans;
}
```

图 7: random

状态压缩

$$st[i+3:i] = arr[i]$$

把一个排列序列存进一个 64 位数里，使用康托展开更好，不过我懒得写了

代码

去场外看代码

目录

致谢

致谢

谢谢大家。