Преобразование в польскую нотацию

Функтор

1) Написать объект, который запоминает арифметические операции, проделанные над ним и может их повторить. 2) Написать объект, который трансформирует операции сравнения над собой в WHERE для SQL.

```
x = Var()
f = 3 * x + x ** 2
f(1) == 4

y = SQL("table.field")
str(y > 12 && y != 17)  == "table.field > 12 AND table.field
```

Гномья сортировка

В массиве сравниваются соседние элементы. Если они неупорядоченны - они меняются местами и делается шаг назад. Если они упорядоченны, то шаг вперед. Если дошли до конца, то сортировка оконченна.

Кодирование Шеннона — Фано

- Сообщение бъется на элементы
- Изначально коды для всех элементов пустые
- Элементы множества выписывают в порядке убывания вероятностей.
- Множество делится на две части, суммарные вероятности символов которых максимально близки друг другу.
- К коду первой половины элементов дописывается "0", второй "1"
- Алгоритм повторяется для обоих частей

Кодирование и декодирование файла по Хаффману.

На диске есть файл с именем "input.txt". Его нужно прочитать, закодировать символы используя не адаптивный алгоритм Хаффмана и записать результат в output.bin. В решении должно быть две функции hf_encode(string) str->str, и hf_decode(string) str->str. Первая кодирует, вторая декодирует. Входными элементами для алгоритма являются отдельные байты файла.

Интерпретатор minilisp

Программа на mini-lisp имеет вид (oper param1 param2 para3 .... paramn), здесь oper это имя функции - любой набор символов, кроме пробелов. param2 - целое, строка в кавычках (без кавычек внутри) или другая программа на mini-lisp. Допустимые oper - '+' (складывает все операнды), '-' (вычитает из первого все операнды), print (печатает все операнды через пробел). Сложение имеет такой же смысл, как и в питоне. Вычитание для строк не определено. Нужно написать функция eval_minilisp, которая исполнить программу переданную параметром. По умолчанию из main вызывать eval_minilisp("example.lst")

- eval_minilisp ('(+ 1 2 3)') должна вернуть 6
- eval_minilisp ('(print (+ "a" "bc")) ') => должны напечатать 'abc'

Интерпретатор minilisp с промежуточным деревом

- Сделать задание "Интерпретатор minilisp" но промежуточно необходимо преобразовать дерево в форму, удобную для промежуточной обработки.
- Сделать систему разбора расширяемой снаружи новыми командами.

интерпретатор подмножества языка forth

Программа на Forth состоит из набора команд(слов), некоторые из которых имеют параметры. Для хранения данных используется стек - команды получают свои операнды с вершины стека и туда же сохраняют результаты. В подмножестве 5 команд:

- put значение - помещает значение в стек. Значение может быть числом или строкой. Строка заключается в кавычки, внутри строки кавычек быть не может.
- pop - убирает значение из стека
- add - убирает из стека 2 значения, складывает их и помещает результат в стек
- sub - убирает из стека 2 значения, вычитает их и помещает результат в стек
- print - вынимает из стека 1 значение и печатает его.

```
1       put 3
2       put "asdaadasdas"
```

Каждая команда начинается с новой строки. Строки, начинающиеся с '#' - комментарии. Ваша программа должна содержать функцию eval_forth(), принимающую строку на языке forth и исполняющую ее. По умолчанию из main вызывать eval_forth("example.frt") Пример, если в example.rft будет:

```
1       put 1
2       put 3
3       add
4       print
```

То программа должна напечатать '4'.

Сложение имеет такой же смысл, как и в питоне. Вычитание для строк не определено Программа должна содержать функцию eval_forth(), принимающую строку на языке forth и исполняющую ее. По умолчанию из main вызывать eval_forth("example.frt")

Умножение больших чисел

Реализовать алгоритм Карацубы для умножения больших чисел.

$$AB * CD == (A + B * 2^m) * (C + D * 2^m)$$
$$== A * C + 2^{2*m} * B * D + 2^m * (B * C + A * D)$$
$$== A * C + 2^{2*m} * B * D + 2^m * ((A + B)(C + D) - A * C - B * D)$$

Острова

Задан двумерный массив из 0,1. Островом называется связная группа единиц, т.е. такие что от любой из них можно дойти до любой другой, перемещаюсь за шаг на одну клетку вверх, вниз, вправо, влево или по диагонали и не попадая на клетки с '0'. Посчитать количество островов.

Операции над множествами через сортировку

Написать следующие функции над массивами. Все они должны исполняться за $O(n * log(n))$, где n - количество элементов в большем массиве. В результирующем массиве не должно быть дубликатов.

- union - пересечение двух множеств. Все элементы, которые есть хотя-бы в одном из множеств.
- difference - все элементы, которые есть в одном из множеств, но отсутствуют во втором.

FP

Написать функции:

- my_map, принимает функцию и список, возвращяет список полученный в итоге применения переданной функции к каждому элементу из списка-параметра. map(func, lst) == [func(lst [0]), func(lst [1]), ..., func(lst [N])]
- my_filter(func, lst1) -> lst2. lst2 содержит только те элементы из lst1, для которых func возвращает True
- my_reduce(my_fold) описание reduce.
- рекурсивные варианты всех этих функций
- Функцию bind, которая принимает функцию func и список параметров params1 и возвращает функцию, при вызове которой со списком параметров params2 вызывается func с объединенным списком параметров params1 + params2. То-же, но с поддержекой именованных аргументов. bind(func, 1, 2, "3")(2, 4) == func(1, 2, "3", 2, 4)
- my_map_gen, my_filter_gen, my_reduce_gen, которые принимают генераторы и возвращают генераторы

super

Написать my_super, аналогичный по поведению встроенному

Задание - func_info

Написать функцию func_info, которая принимает функцию и печатает ее

- Имя
- Количество параметров
- Документацию
- Значения параметров по умолчанию
- Поля искать через ipython/google/python doc

Задание - композиция функций

Написать функцию haskell_dot, которая принимает неограниченное количество функций и возвращает новую функцию, поторая при вызове последовательно применяет все сохраненные функции к параметру.

haskell_dot(f1, f2, f3, ...) -> fC
fC(x) = f1(f2(f3(...(x))))

Сериализация

Написать функции, умеющуе сериализовать и десериализовать следующие типы данных: list, int, str, dict и все их комбинации. Например такое значение {'a':1, 'b':[1,2,3,['3']], 4:7}. Длинные целые поддерживать не надо. Сериализаций называется превращение значения в строку, представление которой не зависит от аппаратных особенностей компьютера. serialize должна возвращать строку, вызов deserialize от которой вернет значение, равное значению, переданному в serialize. Нельзя использовать eval/pickle/marshal и прочие готовые решения. Строки внутри передаваемого значения могут содержать любые символы (например '
x00', '{', '}', '[', ']', '(', ')', etc). Не стоит для преобразования полагаться на встроенное преобразование объектов в строку с помошью функций str/repr. Необходимые модули: struct.

RPC

Используя функцию из задачи "Сериализация" или pickle.dumps и сокеты реализовать сервер и клиент для удаленного вызова функций. Удаленный вызов означает, что на одном компьютере(клиенте) вызывается специальная процедура-прокси которая передает параметры по сети на сервер. На сервере необходимая процедура вызывается с этими параметрами. Результат передаетмя назад на клиент и процедура-прокси возвращает его. Таким образом код, испольщующий прокси не замечает факта общения по сети.

Алгоритм Кнута-Морриса-Пратта

-

ИИ для шахмат

- Без взятия на проходе и рокировки
- Классы для фигур и доски
- функция оценки позиции
- перебор
- генетический алгоритм для подгона перебора

Морской бой

ORM

Задача ORM - преобразовывать python выражения в соответствующие запросы sql.

```
1 t1 = orm.Table("users")
2 t2 = orm.Table("addresses")
3
4 expr = (t1.name == 'foo').and(t2.uid == t1.id)
5
6 print str(expr)
7
8 print str(orm.select(t2.address).where(expr))
```

Выход из лабиринта

Тетрис

ООП, сделать GUI - заготовку