PERFORMANCE

Inputs

Input_short: file containing 17 lines
Input_large: file containing 1,011,297 lines

Output:

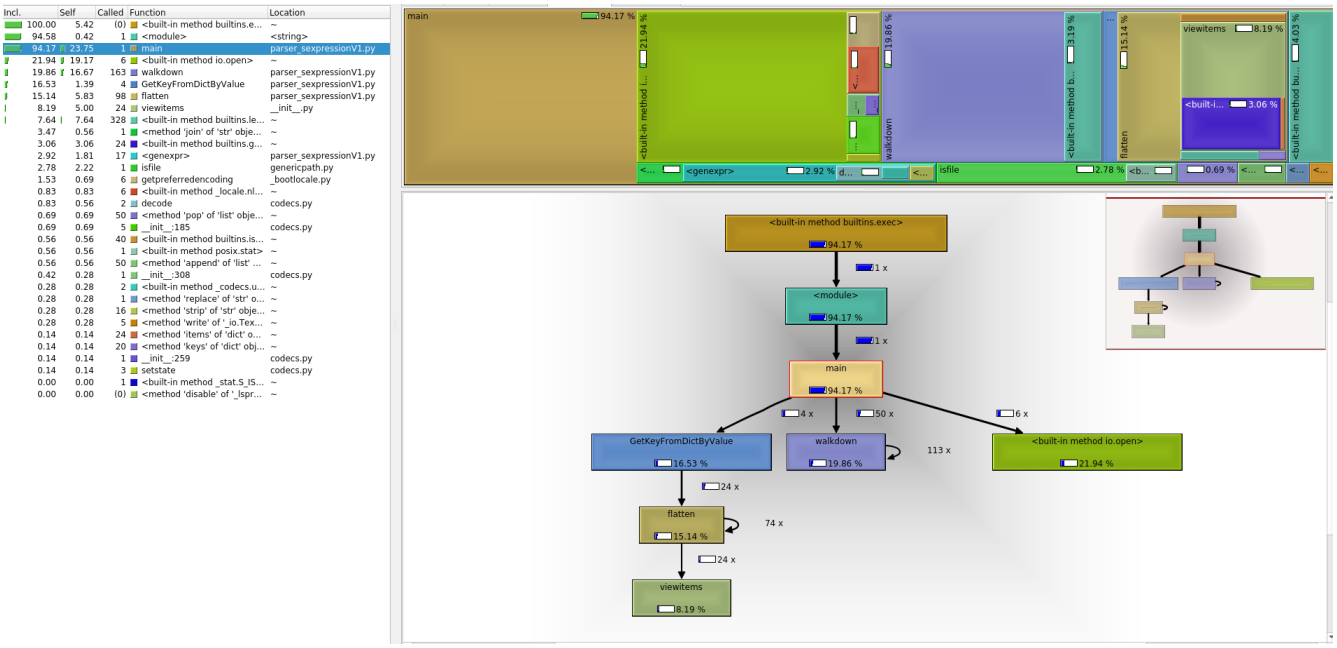
```
-----------------------------------------------------------------------------------------------------
Call: psexpV1
The function took 0.0003859996795654297 seconds to complete file ./input_short
The function took 13.416962146759033 seconds to complete file ./input_large
-----------------------------------------------------------------------------------------------------
Call: psexpV2
The function took 0.00644683837890625 seconds to complete file ./input_short
The function took 639.8281652927399 seconds to complete file ./input_large
-----------------------------------------------------------------------------------------------------
Call: psexpV3
The function took 0.0062634944915771484 seconds to complete file ./input_short
The function took 568.7677512168884 seconds to complete file ./input_large
-----------------------------------------------------------------------------------------------------
Call: psexpV4
The function took 0.0004253387451171875 seconds to complete file ./input_short
The function took 839.4797215461731 seconds to complete file ./input_large
-----------------------------------------------------------------------------------------------------
Call: psexpV5
The function took 0.008693695068359375 seconds to complete file ./input_short
The function took 23.590721368789673 seconds to complete file ./input_large
```

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-

'''
Filename: benchmark.py
Description: timer manager
Version: 0.0
Author: JI
Date: 28/01/2018
Licence: Apache License version 2.0
'''

import parser_sexpressionV1 as psexpV1
import parser_sexpressionV2 as psexpV2
import parser_sexpressionV3 as psexpV3
import parser_sexpressionV4 as psexpV4
import parser_sexpressionV5 as psexpV5
import time, sys
import cProfile

class MyTimer():

    def __init__(self, fname):
        self.start = time.time()
        self.fname = fname

    def __enter__(self):
        return self

    def __exit__(self, exc_type, exc_val, exc_tb):
        end = time.time()
        runtime = end - self.start
        msg = 'The function took {time} seconds to complete file {fname}'
        print(msg.format(time=runtime, fname=self.fname))

def timeit(fun, fileinput, fileoutput):
    cprof = "cprof_" + fun
    cprof = fileoutput.replace(".main()", ".cprof")
    n = 100
    print("-"*n)
    print('Call: {}'.format(fun))
    cProfile.run(fun + '("' + fileinput + '", "' + fileoutput + '")', cprof)

if __name__ == '__main__':

    fileinput1 = "./input_short"
    fileinput2 = "./input_large"
    fileoutput = "./output"

    timeit("psexpV1.main", fileinput1, fileoutput)
    with MyTimer(fileinput1):
        psexpV1.main(fileinput1, fileoutput)
    with MyTimer(fileinput2):
        psexpV1.main(fileinput2, fileoutput)

    timeit("psexpV2.main", fileinput1, fileoutput)
    with MyTimer(fileinput1):
        psexpV2.main(fileinput1, fileoutput)
    with MyTimer(fileinput2):
        psexpV2.main(fileinput2, fileoutput)

    timeit("psexpV3.main", fileinput1, fileoutput)
    with MyTimer(fileinput1):
        psexpV3.main(fileinput1, fileoutput)
    with MyTimer(fileinput2):
        psexpV3.main(fileinput2, fileoutput)

    timeit("psexpV4.main", fileinput1, fileoutput)
    with MyTimer(fileinput1):
        psexpV4.main(fileinput1, fileoutput)
```

```
with MyTimer(fileinput2):
    psexpV4.main(fileinput2, fileoutput)

timeit("psexpV5.main", fileinput1, fileoutput)
with MyTimer(fileinput1):
    psexpV5.main(fileinput1, fileoutput)
with MyTimer(fileinput2):
    psexpV5.main(fileinput2, fileoutput)

sys.exit(0)
```

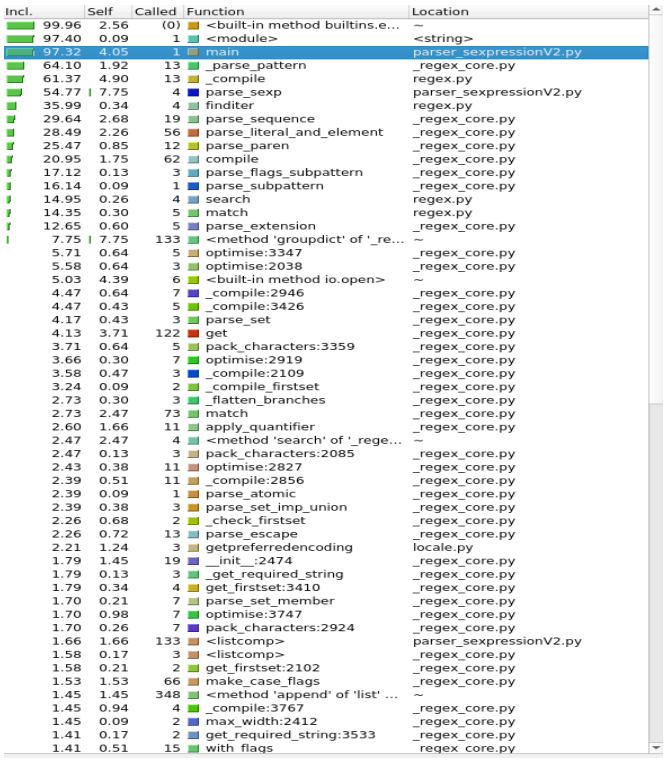
```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import sys
import os
from future.utils import viewitems

'''
Filename: parser_sexpressionV1.py
Description: parse s-expressions reading character by character
Version: 0.1
Author: JI
Date: 28/01/2018
Licence: Apache License version 2.0
Input:  the input consists of a sequence of test cases in the form of integer/tree pairs.
        Each test case consists of an integer followed by one or more spaces followed by
        a binary tree formatted as an S-expression as described above.
        All binary tree S-expressions will be valid, but expressions may be spread over
        several lines and may contain spaces. There will be one or more test cases in an
        input file, and input is terminated by end-of-file.
Output: there should be one line of output for each test case (integer/tree pair) in the
        input file. For each pair I,T (I represents the integer, T represents the tree)
        the output is the string yes if there is a root-to-leaf path in T whose sum is I
        and no if there is no path in T whose sum is I.
format: (integer()()())...
'''


def flatten(dictionary):
    for key, value in viewitems(dictionary):
        if isinstance(value, dict):
            # recurse
            for res in flatten(value):
                yield res
        else:
            yield len(dictionary.keys()), value

def GetKeyFromDictByValue(self, dictionary, value_to_find):
    for key, value in flatten(dictionary):
        if (value == value_to_find) and (key == 1):
            return key

def walkdown(tree, nodes, total=0):
    if len(nodes) == 0: return tree

    node_name = nodes[0]
    i = int(node_name)
    if tree and node_name in tree:
        tree[node_name] = walkdown(tree[node_name], nodes[1:], tree[node_name]['sum'])
    else:
        tree[node_name] = {'sum': int(total) + i}
    return tree

def main(fileinput, fileoutput):

    if not os.path.isfile(fileinput):
        print("Error. I could not find the read file %s" %(fileinput))
        sys.exit(1)

    with open(fileinput, 'r') as f:
        content = ''.join(line.strip() for line in f)

    content = content.replace(" ", "")

    with open(fileoutput, 'w') as f:
        f.write('')

    node_name = ''
    parent_names = []
    ref = []
```

```
    tree = {}
    counter = 0
    k = 0
    while k < len(content):
        ch = content[k]
        if ch == '(':
            if counter == 0:
                ref = int(node_name)
            else:
                parent_names.append(node_name)
                tree = walkdown(tree, parent_names)
            node_name = ''
            counter +=1
        elif ch == ')':
            counter -=1
            if counter == 0:
                node_name = ''
                res = 'yes' if GetKeyFromDictByValue(None, tree, ref) == 1 else 'no'
                with open(fileoutput, 'a') as f:
                    #print res
                    f.write(res + '\n')
                tree = {}
            else:
                node_name = parent_names.pop()
        else:
            node_name = node_name + ch

        k +=1

if __name__ == '__main__':

    fileinput = './input_short'
    fileoutput = './output'

    main(fileinput, fileoutput)
```



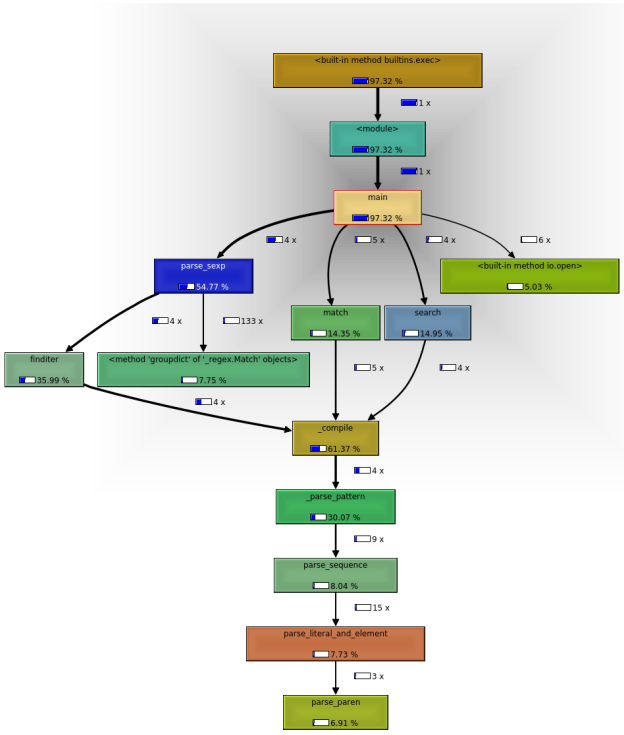
```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import regex
import os, sys

'''
Filename: parser_sexpressionV2.py
Description: parse s-expressions using regular expressions
Version: 0.2
Author: JI
Date: 28/01/2018
Licence: Apache License version 2.0
Input:  the input consists of a sequence of test cases in the form of integer/tree pairs.
        Each test case consists of an integer followed by one or more spaces followed by
        a binary tree formatted as an S-expression as described above.
        All binary tree S-expressions will be valid, but expressions may be spread over
        several lines and may contain spaces. There will be one or more test cases in an
        input file, and input is terminated by end-of-file.
Output: there should be one line of output for each test case (integer/tree pair) in the
        input file. For each pair I,T (I represents the integer, T represents the tree)
        the output is the string yes if there is a root-to-leaf path in T whose sum is I
        and no if there is no path in T whose sum is I.
format: (integer()()).....
'''

term_regex = r'''(?mx)
    \s*(?:
        (?P<brackl>\()|
        (?P<brackr>\))|
        (?P<num>\-?\d+\.\d+|\-?\d+)|
        (?P<sq>"[^"]*")|
        (?P<s>[^(^)\s]+)
       )'''

def parse_sexp(sexp):
    stack = []
    out = []

    for termtypes in regex.finditer(term_regex, sexp):
        term, value = [(t,v) for t,v in termtypes.groupdict().items() if v][0]

        if   term == 'brackl':
            stack.append(out)
            out = []
        elif term == 'brackr':
            assert stack, "Trouble with nesting of brackets"
            tmpout, out = out, stack.pop(-1)
            out.append(tmpout)
        elif term == 'num':
            v = float(value)
            if v.is_integer(): v = int(v)
            out.append(v)
        elif term == 'sq':
            out.append(value[1:-1])
        elif term == 's':
            out.append(value)
        else:
            raise NotImplementedError("Error: %r" % (term, value))
    assert not stack, "Trouble with nesting of brackets"
    return out[0]

def unwrap_list(lst, output = [], node=0):
    if type(lst) == type([]):
        if len(lst) == 3:
            if lst[1]:
                unwrap_list(lst[1], output, node + lst[0])
            if lst[2]:
                unwrap_list(lst[2], output, node + lst[0])
            if not lst[1] and not lst[2]:
```

```
                output.append(node + lst[0])

def main(fileinput, fileoutput):

    if not os.path.isfile(fileinput):
        print("Error. I could not find the read file %s" %(fileinput))
        sys.exit(1)

    with open(fileinput, 'r') as f:
        content = ''.join(line.strip() for line in f)

    content = content.replace(" ", "")

    with open(fileoutput, 'w') as f:
        f.write('')

    patternNum = r'(\d+)'
    patternSexp = '\((?>[^()]+)|(?R))*\)'

    r = regex.match(patternNum, content)
    if r:
        ref = int(r.group())
        r = regex.search(patternSexp, content, flags=0)
        while r:
            result = []
            l = r.span()

            parsed = parse_sexp(content[l[0]:l[1]])

            unwrap_list(parsed, result)
            res = 'yes' if ref in result else 'no'
            with open(fileoutput, 'a') as f:
                #print res
                f.write(res + '\n')
            content = content[l[1]:]
            r = regex.match(patternNum, content)
            if r:
                ref = int(r.group())
                r = regex.search(patternSexp, content, flags=0)

if __name__ == '__main__':

    fileinput = './input_short'
    fileoutput = './output'

    main(fileinput, fileoutput)
```



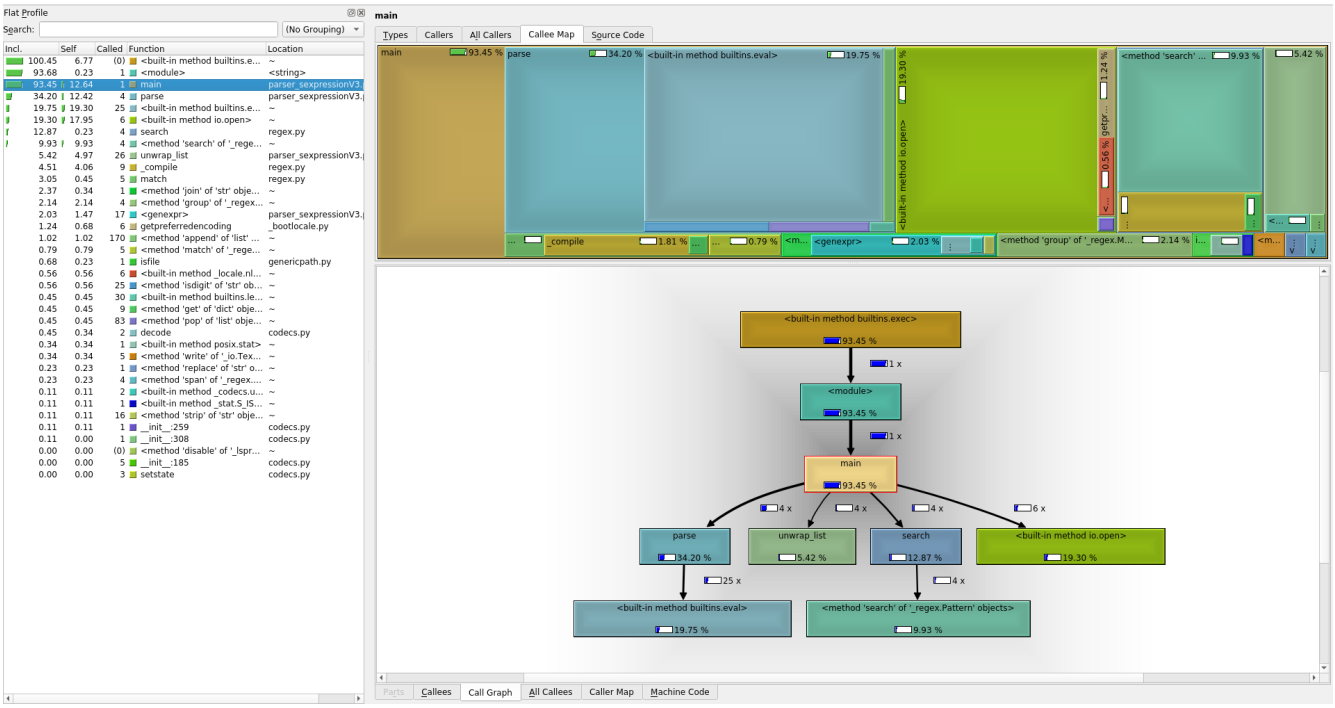
Incl.	Self	Called	Function	Location
99.96	2.56	(0)	<built-in method builtins.e...	~
97.40	0.09	1	<module>	<string>
97.32	4.05	1	main	parser_sexpressionV2.py
64.10	1.92	13	_parse_pattern	_regex_core.py
61.37	4.90	13	_compile	regex.py
54.77	7.75	4	parse_sexp	parser_sexpressionV2.py
35.99	0.34	4	finditer	regex.py
29.64	2.68	19	parse_sequence	_regex_core.py
28.49	2.26	56	parse_literal_and_element	_regex_core.py
25.47	0.85	12	parse_paren	_regex_core.py
20.95	1.75	62	compile	_regex_core.py
17.12	0.13	3	parse_flags_subpattern	_regex_core.py
16.14	0.09	1	parse_subpattern	_regex_core.py
14.95	0.26	4	search	regex.py
14.35	0.30	5	match	regex.py
12.65	0.60	5	parse_extension	_regex_core.py
7.75	7.75	133	<method 'groupdict' of '_re...	~
5.71	0.64	5	optimise:3347	_regex_core.py
5.58	0.64	3	optimise:2038	_regex_core.py
5.03	4.39	6	<built-in method io.open>	~
4.47	0.64	7	_compile:2946	_regex_core.py
4.47	0.43	5	_compile:3426	_regex_core.py
4.17	0.43	3	parse_set	_regex_core.py
4.13	3.71	122	get	_regex_core.py
3.71	0.64	5	pack_characters:3359	_regex_core.py
3.66	0.30	7	optimise:2919	_regex_core.py
3.58	0.47	3	_compile:2109	_regex_core.py
3.24	0.09	2	_compile_firstset	_regex_core.py
2.73	0.30	3	_flatten_branches	_regex_core.py
2.73	2.47	73	match	_regex_core.py
2.60	1.66	11	apply_quantifier	_regex_core.py
2.47	2.47	4	<method 'search' of '_rege...	~
2.47	0.13	3	pack_characters:2085	_regex_core.py
2.43	0.38	11	optimise:2827	_regex_core.py
2.39	0.51	11	_compile:2856	_regex_core.py
2.39	0.09	1	parse_atomic	_regex_core.py
2.39	0.38	3	parse_set_imp_union	_regex_core.py
2.26	0.68	2	_check_firstset	_regex_core.py
2.26	0.72	13	parse_escape	_regex_core.py
2.21	1.24	3	getpreferredencoding	locale.py
1.79	1.45	19	__init__:2474	_regex_core.py
1.79	0.13	3	_get_required_string	_regex_core.py
1.79	0.34	4	get_firstset:3410	_regex_core.py
1.70	0.21	7	parse_set_member	_regex_core.py
1.70	0.98	7	optimise:3747	_regex_core.py
1.70	0.26	7	pack_characters:2924	_regex_core.py
1.66	1.66	133	<listcomp>	parser_sexpressionV2.py
1.58	0.17	3	<listcomp>	_regex_core.py
1.58	0.21	2	get_firstset:2102	_regex_core.py
1.53	1.53	66	make_case_flags	_regex_core.py
1.45	1.45	348	<method 'append' of 'list' ...	~
1.45	0.94	4	_compile:3767	_regex_core.py
1.45	0.09	2	max_width:2412	_regex_core.py
1.41	0.17	2	get_required_string:3533	_regex_core.py
1.41	0.51	15	with_flags	_regex_core.py


parse_sexp 54.77 %
_parse_pattern 17.38 %
_parse_pattern 5.99 %
_parse_pattern 6.70 %
<listcomp> 1.66 %
<method 'append' of... 0.68 %
<method 'search' of '_regex.Pattern' ob... 2.47 %
<method 'groupdict' of '_regex.Match' objects> 7.75 %
<built-in method io.open> 5.03 %


<built-in method builtins.exec>
97.32 %
1 x
<module>
97.32 %
1 x
main
97.32 %
4 x
5 x
4 x
6 x
parse_sexp
54.77 %
<built-in method io.open>
5.03 %
4 x
133 x
match
14.35 %
search
14.95 %
finditer
35.99 %
<method 'groupdict' of '_regex.Match' objects>
7.75 %
5 x
4 x
4 x
_compile
61.37 %
4 x
_parse_pattern
30.07 %
9 x
parse_sequence
8.04 %
15 x
parse_literal_and_element
7.73 %
3 x
parse_paren
6.91 %

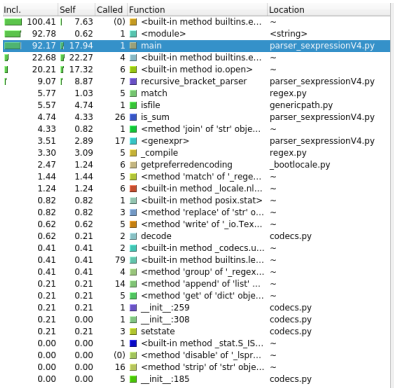
```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import regex
import os, sys
from string import whitespace

'''
Filename: parser_sexpressionV3.py
Description: parse s-expressions using regular expressions
Version: 0.3
Author: JI
Date: 28/01/2018
Licence: Apache License version 2.0
Input:  the input consists of a sequence of test cases in the form of integer/tree pairs.
        Each test case consists of an integer followed by one or more spaces followed by
        a binary tree formatted as an S-expression as described above.
        All binary tree S-expressions will be valid, but expressions may be spread over
        several lines and may contain spaces. There will be one or more test cases in an
        input file, and input is terminated by end-of-file.
Output: there should be one line of output for each test case (integer/tree pair) in the
        input file. For each pair I,T (I represents the integer, T represents the tree)
        the output is the string yes if there is a root-to-leaf path in T whose sum is I
        and no if there is no path in T whose sum is I.
format: (integer()()...
'''


atom_end = set('()"\'') | set(whitespace)

def parse(sexp):
    stack, i, length = [[]], 0, len(sexp)
    while i < length:
        c = sexp[i]

        reading = type(stack[-1])
        if reading == list:
            if   c == '(': stack.append([])
            elif c == ')':
                stack[-2].append(stack.pop())
                if stack[-1][0] == ('quote',): stack[-2].append(stack.pop())
            elif c == '"': stack.append('')
            elif c == "'": stack.append([('quote',)])
            elif c in whitespace: pass
            else: stack.append((c,))
        elif reading == str:
            if   c == '"':
                stack[-2].append(stack.pop())
                if stack[-1][0] == ('quote',): stack[-2].append(stack.pop())
            elif c == '\\':
                i += 1
                stack[-1] += sexp[i]
            else: stack[-1] += c
        elif reading == tuple:
            if c in atom_end:
                atom = stack.pop()
                if atom[0][0].isdigit(): stack[-1].append(eval(atom[0]))
                else: stack[-1].append(atom)
                if stack[-1][0] == ('quote',): stack[-2].append(stack.pop())
                continue
            else: stack[-1] = ((stack[-1][0] + c),)
        i += 1
    return stack.pop()

def unwrap_list(lst, output = [], node=0):
    if type(lst) == type([]):
        if len(lst) == 3:
            if lst[1]:
                unwrap_list(lst[1], output, node + lst[0])
            if lst[2]:
```

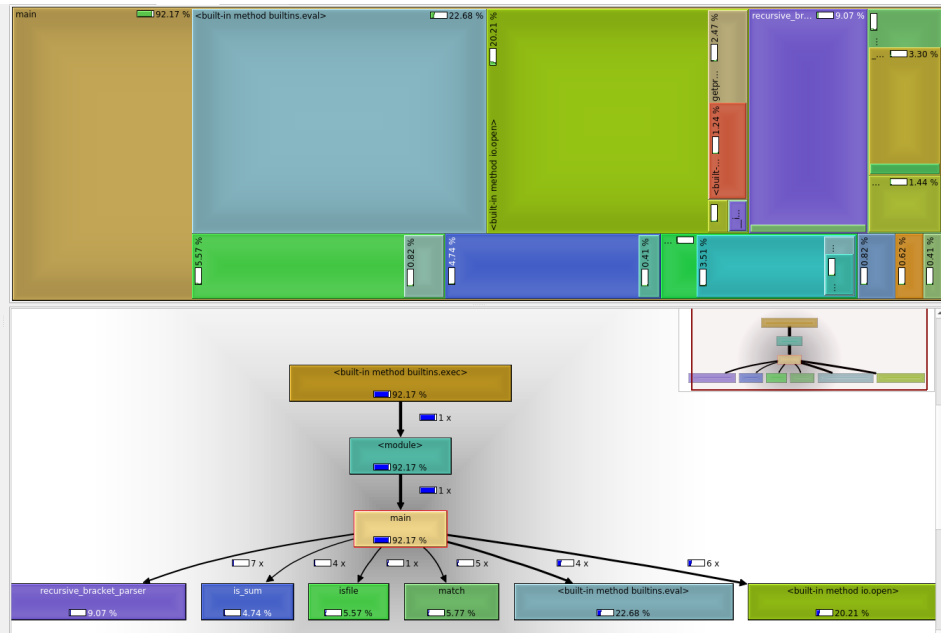
```
                unwrap_list(lst[2], output, node + lst[0])
            if not lst[1] and not lst[2]:
                output.append(node + lst[0])


def main(fileinput, fileoutput):

    if not os.path.isfile(fileinput):
        print("Error. I could not find the read file %s" %(fileinput))
        sys.exit(1)

    with open(fileinput, 'r') as f:
        content = ''.join(line.strip() for line in f)

    content = content.replace(" ", "")

    with open(fileoutput, 'w') as f:
        f.write('')

    patternNum = r'(\d+)'
    patternSexp = '\((?>[^()]+)|(?R))*\)'

    r = regex.match(patternNum, content)
    if r:
        ref = int(r.group())
        r = regex.search(patternSexp, content, flags=0)
        while r:
            result = []
            l = r.span()

            parsed = parse(content[l[0]:l[1]])

            unwrap_list(parsed[0], result)

            res = 'yes' if ref in result else 'no'
            with open(fileoutput, 'a') as f:
                #print res
                f.write(res + '\n')
            content = content[l[1]:]
            r = regex.match(patternNum, content)
            if r:
                ref = int(r.group())
                r = regex.search(patternSexp, content, flags=0)

if __name__ == '__main__':

    fileinput = './input_short'
    fileoutput = './output'

    main(fileinput, fileoutput)
```


Flat Profile
Search:
(No Grouping)
Incl. Self Called Function Location
100.45 6.77 (0) <built-in method builtins.e... ~
93.68 0.23 1 <module> <string>
93.45 12.64 1 main parser_sexpressionV3.
34.20 12.42 4 parse parser_sexpressionV3.
19.75 19.30 25 <built-in method builtins.e... ~
19.30 17.95 6 <built-in method io.open> ~
12.87 0.23 4 search regex.py
9.93 9.93 4 <method 'search' of '_rege... ~
5.42 4.97 26 unwrap_list parser_sexpressionV3.
4.51 4.06 9 _compile regex.py
3.05 0.45 5 match regex.py
2.37 0.34 1 <method 'join' of 'str' obje... ~
2.14 2.14 4 <method 'group' of '_regex... ~
2.03 1.47 17 <genexpr> parser_sexpressionV3.
1.24 0.68 6 getpreferredencoding _bootlocale.py
1.02 1.02 170 <method 'append' of 'list' ... ~
0.79 0.79 5 <method 'match' of '_rege... ~
0.68 0.23 1 isfile genericpath.py
0.56 0.56 6 <built-in method _locale.nl... ~
0.56 0.56 25 <method 'isdigit' of 'str' ob... ~
0.45 0.45 30 <built-in method builtins.le... ~
0.45 0.45 9 <method 'get' of 'dict' obje... ~
0.45 0.45 83 <method 'pop' of 'list' obje... ~
0.45 0.34 2 decode codecs.py
0.34 0.34 1 <built-in method posix.stat> ~
0.34 0.34 5 <method 'write' of '_io.Tex... ~
0.23 0.23 1 <method 'replace' of 'str' o... ~
0.23 0.23 4 <method 'span' of '_regex.... ~
0.11 0.11 2 <built-in method _codecs.u... ~
0.11 0.11 1 <built-in method _stat.S_IS... ~
0.11 0.11 16 <method 'strip' of 'str' obje... ~
0.11 0.11 1 __init__:259 codecs.py
0.11 0.00 1 __init__:308 codecs.py
0.00 0.00 (0) <method 'disable' of '_lspr... ~
0.00 0.00 5 __init__:185 codecs.py
0.00 0.00 3 setstate codecs.py
main
Types Callers All Callers Callee Map Source Code
main 93.45 %
parse 34.20 %
<built-in method builtins.eval> 19.75 %
<built-in method io.open> 19.30 %
<method 'search' ... 9.93 %
5.42 %
_compile 1.81 %
0.79 %
<genexpr> 2.03 %
<method 'group' of '_regex.M... 2.14 %
<built-in method builtins.exec>
93.45 %
1 x
<module>
93.45 %
1 x
main
93.45 %
4 x
4 x
4 x
6 x
parse
34.20 %
unwrap_list
5.42 %
search
12.87 %
<built-in method io.open>
19.30 %
25 x
4 x
<built-in method builtins.eval>
19.75 %
<method 'search' of '_regex.Pattern' objects>
9.93 %
Parts Callees Call Graph All Callees Caller Map Machine Code

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import regex
import os, sys

'''
Filename: parser_sexpressionV4.py
Description: parse s-expressions converting the structures to arrays
Version: 0.4
Author: JI
Date: 28/01/2018
Licence: Apache License version 2.0
Input:  the input consists of a sequence of test cases in the form of integer/tree pairs.
        Each test case consists of an integer followed by one or more spaces followed by
        a binary tree formatted as an S-expression as described above.
        All binary tree S-expressions will be valid, but expressions may be spread over
        several lines and may contain spaces. There will be one or more test cases in an
        input file, and input is terminated by end-of-file.
Output: there should be one line of output for each test case (integer/tree pair) in the
        input file. For each pair I,T (I represents the integer, T represents the tree)
        the output is the string yes if there is a root-to-leaf path in T whose sum is I
        and no if there is no path in T whose sum is I.
format: (integer()()())...
'''

def recursive_bracket_parser(s, i=0, j=0):
    res = []
    counter = 0

    if i < 0: i = 0
    if j == 0  or len(s) < j: j = len(s)
    if i > j: i = j

    while i < j:
        if s[i] == '[':
            counter +=1
            if counter == 1: res.append(i)
        elif s[i] == ']':
            counter -=1
            if counter == 0:
                res.append(i+1)
                return res
        i += 1

    return []


def is_sum(tree, num):
    if (len(tree) == 0):   # 'in' a leaf
        return False
    if (len(tree[1]) == 0 & len(tree[2]) == 0):   # leaf
        return num == tree[0]
    return (is_sum(tree[1], num-tree[0]) | is_sum(tree[2], num-tree[0]))


def main(fileinput, fileoutput):

    if not os.path.isfile(fileinput):
        print("Error. I could not find the read file %s" %(fileinput))
        sys.exit(1)

    with open(fileinput, 'r') as f:
        content = ''.join(line.strip() for line in f)

    content = content.replace(" ", "")
    content = content.replace('(',',[')
    content = content.replace(')',']')

    with open(fileoutput, 'w') as f:
```

```
            f.write('')

        patternNum = r'(\d+)'
        r = regex.match(patternNum, content)
        if r:
            ref = int(r.group())

            s = recursive_bracket_parser(content)

            while s:

                ltree = eval(content[s[0]:s[1]])

                res = 'yes' if is_sum(ltree,ref) else 'no'
                with open(fileoutput, 'a') as f:
                    #print res
                    f.write(res + '\n')

                content = content[s[1]:]

                s = []
                r = regex.match(patternNum, content)

                if r:
                    ref = int(r.group())
                    s = recursive_bracket_parser(content)
                    s = recursive_bracket_parser(content)


if __name__ == '__main__':

    fileinput = './input_short'
    fileoutput = './output'

    main(fileinput, fileoutput)
```





```
_#!/usr/bin/env python
# -*- coding: utf-8 -*-

import sys, os
import itertools
from future.utils import viewitems
from threading import Lock
from multiprocessing.dummy import Pool as ThreadPool

'''
Filename: parser_sexpressionV5.py
Description: multi-threading parse s-expressions after reading each line
Version: 0.5
Author: JI
Date: 28/01/2018
Licence: Apache License version 2.0
Input:  the input consists of a sequence of test cases in the form of integer/tree pairs.
        Each test case consists of an integer followed by one or more spaces followed by
        a binary tree formatted as an S-expression as described above.
        All binary tree S-expressions will be valid, but expressions may be spread over
        several lines and may contain spaces. There will be one or more test cases in an
        input file, and input is terminated by end-of-file.
Output: there should be one line of output for each test case (integer/tree pair) in the
        input file. For each pair I,T (I represents the integer, T represents the tree)
        the output is the string yes if there is a root-to-leaf path in T whose sum is I
        and no if there is no path in T whose sum is I.
format: (integer()()...
'''


lock = Lock()
NUMPROC = 12

def write2file(fpath, text):

    lock.acquire()  # thread blocks at this line until it can obtain lock
    f = open(fpath,'a')
    f.write(text + '\n')
    f.flush()
    f.close()
    lock.release()

def log2console(text):
    lock.acquire()  # thread blocks at this line until it can obtain lock
    print('{}'.format(text))
    sys.stdout.flush()
    lock.release()

def flatten(dictionary):
    for key, value in viewitems(dictionary):
        if isinstance(value, dict):
            # recurse
            for res in flatten(value):
                yield res
        else:
            yield len(dictionary.keys()), value

def GetKeyFromDictByValue(self, dictionary, value_to_find):
    for key, value in flatten(dictionary):
        if (value == value_to_find) and (key == 1):
            return key

def walkdown(tree, nodes, total=0):
    if len(nodes) == 0: return tree

    node_name = nodes[0]
    i = int(node_name)
    if tree and node_name in tree:
        tree[node_name] = walkdown(tree[node_name], nodes[1:], tree[node_name]['sum'])
    else:
```

```
        tree[node_name] = {'sum': int(total) + i}
    return tree

def parse_sexpression(args):

    content = args[0]
    wfile   = args[1]

    node_name = ''
    parent_names = []
    ref = []
    tree = {}
    counter = 0
    k = 0
    while k < len(content):
        ch = content[k]
        if ch == '(':
            if counter == 0:
                ref = int(node_name)
            else:
                parent_names.append(node_name)
                tree = walkdown(tree, parent_names)
            node_name = ''
            counter +=1
        elif ch == ')':
            counter -=1
            if counter == 0:
                node_name = ''
                res = 'yes' if GetKeyFromDictByValue(None, tree, ref) == 1 else 'no'
                write2file(wfile, res + ': ' + content)
                #log2console(res + ': ' + content)
                tree = {}
            else:
                node_name = parent_names.pop()
        else:
            node_name = node_name + ch

        k +=1

def main(fileinput, fileoutput):

    with open(fileoutput, 'w') as f:
        f.write('')

    if not os.path.isfile(fileinput):
        print("Error. I could not find the read file %s" %(fileinput))
        sys.exit(1)

    content = []
    sexpr = ''
    with open(fileinput, 'r') as f:
        for line in f:
            line.strip()
            line = line.replace(" ", "")
            line = line.rstrip("\n\r")
            if line:
                sexpr = sexpr + line
                if sexpr.count("(") == sexpr.count(")"):
                    content.append(sexpr)
                    sexpr = ''

    pool = ThreadPool(NUMPROC)
    pool.map(parse_sexpression, zip(content, itertools.repeat(fileoutput)))

if __name__ == '__main__':

    fileinput = './input_short'
    fileoutput = './output'

    main(fileinput, fileoutput)
```

Incl.	Self	Called	Function	Location
157.47	0.15	5	_find_and_load	<frozen importlib._boots
157.32	0.07	11	_call_with_frames_removed	<frozen importlib._boots
99.73	0.03	1	<module>	<string>
99.70	0.40	1	main	parser_sexpressionV5.py
89.90	0.04	1	Pool	__init__.py
70.75	0.10	5	_find_and_load_unlocked	<frozen importlib._boots
70.29	0.18	5	_load_unlocked	<frozen importlib._boots
70.03	0.05	3	exec_module:672	<frozen importlib._boots
63.75	0.23	3	get_code	<frozen importlib._boots
60.11	0.05	3	source_to_code	<frozen importlib._boots
60.06	60.06	3	<built-in method builtins.c...	~
48.82	0.30	3	<built-in method builtins.e...	~
48.81	0.13	1	<module>	pool.py
48.16	0.13	18	_handle_fromlist	<frozen importlib._boots
47.80	0.01	1	<built-in method builtins.__...	~
37.81	0.16	1	<module>	util.py
18.91	0.01	1	__init__:788	pool.py
18.90	0.16	1	__init__:153	pool.py
18.08	0.40	16	wait:533	threading.py
17.52	0.70	16	wait:263	threading.py
15.94	15.94	65	<method 'acquire' of '_thre...	~
15.23	0.39	1	_repopulate_pool	pool.py
13.68	0.27	15	start	threading.py
11.74	0.18	12	start	__init__.py
8.97	0.02	1	map	pool.py
8.55	0.03	1	get	pool.py
8.52	0.00	1	wait	pool.py
3.85	3.85	15	<built-in method _thread.s...	~
2.94	0.46	12	Process	pool.py
2.36	0.16	1	<module>	subprocess.py
2.13	0.32	12	__init__	__init__.py
2.12	0.18	5	_find_spec	<frozen importlib._boots
1.86	0.11	3	_code_to_bytecode	<frozen importlib._boots
1.78	1.56	17	<built-in method builtins.__...	~
1.71	0.02	4	find_spec:1149	<frozen importlib._boots
1.69	0.12	4	_get_spec:1117	<frozen importlib._boots
1.67	0.52	15	__init__:757	threading.py
1.59	1.59	3	<built-in method marshal.d...	~
1.51	0.38	7	find_spec:1233	<frozen importlib._boots
1.29	0.07	5	module_from_spec	<frozen importlib._boots
0.83	0.35	16	__init__:498	threading.py
0.83	0.04	3	_cache_bytecode	<frozen importlib._boots
0.77	0.77	12	__setitem__	weakref.py
0.63	0.15	3	set_data	<frozen importlib._boots
0.63	0.63	45	<built-in method _thread.al...	~
0.58	0.58	25	__init__:215	threading.py
0.56	0.02	1	create_module:919	<frozen importlib._boots
0.54	0.13	5	_init_module_attrs	<frozen importlib._boots
0.53	0.53	1	<built-in method _imp.crea...	~
0.53	0.08	20	_path_stat	<frozen importlib._boots
0.52	0.19	17	_is_owned	threading.py
0.47	0.47	21	<built-in method posix.stat>	~
0.46	0.31	6	get_data	<frozen importlib._boots
0.40	0.07	1	_map_async	pool.py