Lab1 - 括弧匹配实验

---

1. 实验要求

给定一个由括号构成的串，若该串是合法匹配的，返回串中所有匹配的括号对中左右括号距离的最大值；否则返回NONE。左右括号的距离定义为串中二者之间字符的数量，即 $\max\{j-i+1|(si,sj)\text{是串}s\text{中一对匹配的括号}\}$。要求分别使用枚举法和分治法求解。

2. 实验思路

2.1 分治法求解思路

- parenDist与pd为主要函数，算法思路描述如下（pd函数中描述的为主要思路）

parenDist

- **params**

```
    paren seq
        -> int option
```

- **steps**
  1. divide the input paren seq into a tree.
  2. calculate using the function `pd`.(see details in the function pd)
  3. return first part of the result of `pd`, or NONE when it equals to SOME 0

pd

- **params&returns**

```
    paren seq
        -> (int option * int * int * int * int * int)
```

- @s0 : input paren seq
- @max(maximum length) : maximum length of matched paren seq so far.
- @closed(maximum closed length) : the length of the longest `closed` paren seq.
    1. consists of one continuous matched paren seq, can not be separately matched , e.g. the @closed of "()()(())(" is 4 instead of 8
    2. has no right paren #")" next to its right, no left paren #"(" next to its left.
- @lo(left open number) : number of unclosed left paren #"(".
- @ro(right open number) : number of unclosed right paren #")".
- @ld(left distance) : maximum distance for an unclosed right paren #")" to the left boundary.
- @rd(right distance) : maximum distance for an unclosed left paren #"(" to the right boundary.
- **steps**
    1. split the paren seq using `showt` recursely, for each recursion, calculate both subtree in parallel, and save the params needed from both side of subtrees.
    2. notice two point:
        1. a tree's length strictly equals to its `rd+ld+closed`.
        2. a tree can only have one `closed` part, or the part between two closed part should also be closed by a #"(" and a #")" according to the definition of @closed.
    3. judge whether left subtree's lo (`llo`) or right subtree's ro (`rro`) is larger, save the difference into `curopen = llo-rro`.
        1. if `llo` is larger:
            1. obviously, the new @closed equals to left subtree's @closed (`closed=lclosed`), because it has an unclosed #"(" next to its right, excluding all the right part to be `closed`.
            2. max is simply the record of whether the @max of both subtrees, the @closed of the cmerged one, or the recently merged and matched part, maybe unconfirmed to the second restrict of `closed`, is larger. That is `max = maxium(lmax, rmax, closed, 2 * minimum(lrd, rld)).`
            3. the new @lo equals to all the remained llo after the merge, together with the `rlo` that do nothing during the merge, the result is `lo=curopen+rlo`.
            4. because there is no ro on the right part (all closed by `llo` when merged), obviously the new @ro equals to the left subtree's @ro (`ro=lro`).
            5. having no relation to the merge, the new @ld equals to the left subtree's @ld (`ld=lld`).
            6. the new @rd equals to the sum of left subtree's @rd (`lrd`) and the whole right tree, that is `rd=lrd+rld+rrd+rclosed` according to step `2.1`.
        2. if `rro` is larger:

    1. simply change all the 'l' and 'r' in step `3`, notice that `curopen` should be changed to `~curopen`
3. if `llo` equals to `rro`:
    1. @closed should be `max(lclosed, rclosed, lrd+rld)` because in this case, the recently merged part (`lrd+rld`) is also `closed`.
    2. @max as shown in former cases.
    3. @ro equals `lro`, similar to step `3.1.4`.
    4. @lo equals to `rlo`, combining step `3.3.3` and `3.2.1`.
    5. @ld equals to `lld`, similar to step `3.1.5`.
    6. @rd equals to `rrd`, combining step `3.3.5` and `3.2.1`.
4. return `(@max, @closed, @lo, @ro, @ld, @rd)`

3. 回答问题

3.1 关于枚举法求解

- *Task 5.2 (5%)*. What is the work and span of your brute-force solution? You should assume subseq has $\mathcal{O}(1)$ work and span.
- **Answer 5.2**
    - $\mathcal{O}(2^n)$.

3.2 关于分治法求解

- *Task 5.4 (20%)*. The specification in Task 5.3 stated that the work of your solution must follow a recurrence that was parametric in the work it takes to view a sequence as a tree. Naturally, this depends on the implementation of SEQUENCE.
    1. Solve the work recurrence with the assumption that $W_{showt} \in \Theta(lgn)$ where n is the length of the input sequence.
    2. Solve the work recurrence with the assumption that $W_{showt} \in \Theta(n)$ where n is the length of the input sequence.
    3. In two or three sentences, describe a data structure to implement the sequence α seq that allows showt to have $\Theta(lgn)$ work.
    4. In two or three sentences, describe a data structure to implement the sequence α seq that allows showt to have $\Theta(n)$ work.
- **Answer 5.4**
    1. $W(n) = \mathcal{O}(n)$.
        1. To solve this, first we need to know that $W(n) = 2W(\frac{n}{2}) + \lg n$ represent a leaves domainated complexity. As following. $\frac{2\lg\frac{n}{2}}{\lg n} = \frac{2\lg n - 2\lg 2}{\lg n} = 2 - \frac{2\lg 2}{\lg n}$ $\because \frac{2\lg 2}{\lg n} < 1, n \to \infty \therefore \frac{2\lg\frac{n}{2}}{\lg n} > 1, n \to \infty$
        2. Then it's easy to solve the answer by brick method, shown as following. $W(n) \approx \mathcal{O}(2^{\lg n - 1}\lg\frac{n}{2^{\lg n - 1}}) \sim \mathcal{O}(n\lg\frac{n}{\frac{n}{2}}) \sim \mathcal{O}(n) \therefore W(n) = \mathcal{O}(n)$
    2. $W(n) = \mathcal{O}(n\log n)$.
        - For this one, we can easily get the answer via the master theorem.

$\because W(n) = aW(\frac{n}{b}) + cn^d, a = 2, b = 2, c = 1, d = 1$
$\therefore W(n) = \mathcal{O}(n^d \log n) = \mathcal{O}(n \log n)$

3. A BST using the order of appearance as keys, with the total length saved. It's easy to know that searching for the half length element costs $\Theta(\log n)$. Then `take` and `drop` can both be done using `split` with the half length element, which also costs $\Theta(\log n)$, as is shown in the class. $W_{showt} = [\Theta(\log n) + \Theta(\log n)] \sim \Theta(\log n)$
4. A two-way linked list is suitable, for $\Theta(n)$ is needed to calculate it's length, and $\Theta(\frac{n}{2})$ for both `take` and `drop`. $W_{showt} = [\Theta(\frac{n}{2}) + \Theta(\frac{n}{2}) + \Theta(n)] \sim \Theta(n)$

3.3 关于渐进复杂度分析

- *Task 6.1 (5%)*. Rearrange the list of functions below so that it is ordered with respect to $\mathcal{O}$—that is, for every index i, all of the functions with index less than i are in $big - \mathcal{O}$ of the function at index i. You can just state the ordering; you don't need to prove anything.
  1. $f(x) = n^{logn^2}$
  2. $f(n) = 2n^{1.5}$
  3. $f(n) = n^n!$
  4. $f(n) = 43^n$
  5. $f(n) = lglglglgn$
  6. $f(n) = 36n^{52} + 15n^{18} + n^2$
  7. $f(n) = n^{n!}$
- **Answer 6.1**
  - **5261437**
- *Task 6.2 (15%)*. Carefully prove each of the following statements, or provide a counterexample and prove that it is in fact a counterexample. You should refer to the definition of $big - \mathcal{O}$. Remember that verbose proofs are not necessarily careful proofs.
  1. $\mathcal{O}$ is a transitive relation on functions. That is to say, for any functions f, g, h, if $f \in \mathcal{O}(g)$ and $g \in \mathcal{O}(h)$, then $f \in \mathcal{O}(h)$.
  2. $\mathcal{O}$ is a symmetric relation on functions. That is to say, for any functions f and g , if $f \in \mathcal{O}(g)$, then $g \in \mathcal{O}(f)$.
  3. $\mathcal{O}$ is an anti-symmetric relation on functions. That is to say, for any functions f and g , if $f \in \mathcal{O}(g)$ and $g \in \mathcal{O}(f)$, then $f = g$.
- **Answer 6.2**
  1. **True.** $f \in \mathcal{O}(g)$ shows $\lim\limits_{n\to\infty} \frac{f(n)}{g(n)} \neq \infty$; $g \in \mathcal{O}(h)$ shows $\lim\limits_{n\to\infty} \frac{g(n)}{h(n)} \neq \infty$. So that we have $\lim\limits_{n\to\infty} \frac{f(n)}{g(n)} \frac{g(n)}{h(n)} \neq \infty$, therefore $\lim\limits_{n\to\infty} \frac{f(n)}{h(n)} \neq \infty$, then $f \in \mathcal{O}(h)$.
  2. **False.** Counterexample: $f(n) \in \mathcal{O}(\lg n)$ and $g(n) = n$. According to 1, because we have $\mathcal{O}(\lg n) \in \mathcal{O}(n)$ (proved using limitation), then $f(n) \in \mathcal{O}(g(n))$, showing this is an example where $f \in \mathcal{O}(g)$. However, $\because \lim\limits_{n\to\infty} \frac{g(n)}{\lg n} = \lim\limits_{n\to\infty} \frac{n}{\lg n} = \infty$, $\lim\limits_{n\to\infty} \frac{\lg n}{f(n)} \neq 0$; $\therefore \lim\limits_{n\to\infty} \frac{g(n)}{f(n)} = \infty$, at variance to $g \in \mathcal{O}(f)$, which needs $\lim\limits_{n\to\infty} \frac{g(n)}{f(n)} \neq \infty$.
  3. **False.** Counterexample: $f(n) = 2n, g(n) = n$. For this example, obviously $\lim\limits_{n\to\infty} \frac{g(n)}{f(n)} \neq \infty$, $\lim\limits_{n\to\infty} \frac{f(n)}{g(n)} \neq \infty$, then we get $f \in \mathcal{O}(g)$ and $g \in \mathcal{O}(f)$, but $f \neq g$.