K: Sort Compressed Strings

原案 : climpet
問題文 : tsutaj
データセット : riantkb
解答 : hos, riantkb, smiken, tsutaj
解説 : riantkb

概要

- 「圧縮された文字列」が N (≦ 50) 個あるので、展開したときの辞書順でソートしてください
  - 同じ文字列に展開される場合は先に入力された方が前に来なければならない（安定ソート）
  - |S_i| ≦ 2000
  - （各文字列の展開後の長さ）≦10^{10}
  - 圧縮された文字列の例：
    - 3(AB)C → ABABABC
    - 2(2(2A)) → AAAAAAAA

解法

- ローリングハッシュを使って「展開された後の文字列の先頭 K 文字が一致しているかどうか」の判定を行うことを考える

- もしそれができれば、二分探索をすると 2 つの文字列が最初にどこで異なるかが分かり、その異なった最初の文字を取ってくれば
  2 つの文字列の比較ができる

解法

- 先頭 K 文字のローリングハッシュはローリングハッシュの値を持ちながら構文解析をすればできる

- 基本的には「いま展開後何文字となるところまで読んだか」を持ちながら構文解析をし、それが K 以下であるならばローリングハッシュに追加する、ということを行っていく
  - M(seq) のような形で途中までのものが必要となるが、
    seq の展開後の長さが分かればx 回繰り返したあと先頭y 文字が必要、
    とわかるので再帰していけばよい

解法

- M 回繰り返しのハッシュの求め方については、
  hsh * (1 + x + x^2 + ... + x^{M-1})
  みたいなのが求まればよく、繰り返し自乗法などで求まる
  - (x^M - 1) / (x - 1) という形にする場合は、逆元を求めるのに余計に O(log MOD) かけると TLE する可能性が高いので注意
    - ハッシュと一緒に base^{-length} を持っておくとその log が落ちる
  - また、方針によっては定数倍が重いことがあり、メモ化などでの高速化が必要な場合がある
    - 構文木の各ノードについてハッシュの結果を持っておいて、
      その部分木全て使う場合はその持っておいた結果を使用する、など
    - また、各文字列について先頭 K 文字の結果を一度計算したらメモする、というのもかなり有効
    - 繰り返しのときの再帰で実装をミスるとその部分が指数になったりもするので注意

- 計算量は O(N max(|S|) log N log L)（L: 展開後の文字列の長さの max）
  - TL は 4 sec ですが、ジャッジ解は 0.1 sec ほどで通っています

ジャッジ解

- hos (C++): 259 lines, 8.0 kB
- riantkb (C++): 243 lines, 7.3 kB
- smiken (C++): 268 lines, 4.5 kB
- tsutaj (C++): 292 lines, 8.8 kB

統計情報

- Acceptances
  - 0 teams
- First Acceptance
  - N/A