記憶縛り
(Limited Memory)

JOI 春合宿 2015 Day 4

解説： 保坂 和宏

問題概要

- 良い文字列 (括弧の対応 OK)

<>[ ]　　[<>]<>

- 良い文字列ではない

><　　[<]>

問題概要

- S が良い文字列か否か判定せよ
  - ただし 1 文字ずつしか聞けない！
  - 22 ビットしか記憶していられない
- 制約
  - 長さ $1 \leq N \leq 100$
  - 聞ける回数 15,000 回

考察

- そもそも普通に入力が全部読めたら？

普通に読めたら

- 定義に従って区間 DP
  - $dp[i][j]$:= 位置 i から位置 j までの部分文字列が良い文字列であるか？
  - $O(N^3)$ 時間

- 空文字列 (つまり，長さ 0 の文字列) は良い文字列である．
- x が良い文字列ならば，<x> (つまり，x を山括弧 <> で囲った文字列) は良い文字列である．
- x が良い文字列ならば，[x] (つまり，x を角括弧 [] で囲った文字列) は良い文字列である．
- x, y が良い文字列ならば，xy (つまり，x, y をこの順に連結した文字列) は良い文字列である．
- 以上で良い文字列と定義されるもののみが良い文字列である．

普通に読めたら

- スタックの利用
  - 開き括弧が来たらスタックに push
  - 閉じ括弧が来たら
    - スタックが空なら *NO*
    - スタックの top と対応していなかったら *NO*
    - 対応していたら pop して続ける

  最後にスタックが空なら *YES*
- $O(N)$ 時間

普通に読めたら

- スタックの利用

[<>] <>

スタック：

普通に読めたら

- スタックの利用

[<>] <>

スタック：[

普通に読めたら

- スタックの利用

[<>] <>

スタック：[<

普通に読めたら

- スタックの利用

[<>] <>

スタック：[<

普通に読めたら

- スタックの利用

[<>] <>

スタック：[

普通に読めたら

- スタックの利用

[<>] <>

スタック：<

普通に読めたら

- スタックの利用

[<>] <>

スタック：<

小課題 1 ($N \leq 8$)

- 8 文字くらいなら全部覚えられそう

今何文字目？
0 から 8 まで
(4 ビット)

文字列の内容
1 文字 4 ($= 2^2$) 種類
(16 ビット)

小課題 1 ($N \leq 8$)

Memory(N, M)

- M から今何文字目かと文字列の内容を読み取る
  - エラーなら適当に処理する
- 今 N 文字目なら良い文字列か判定する
- 次の文字を Get する
- エンコードして返す

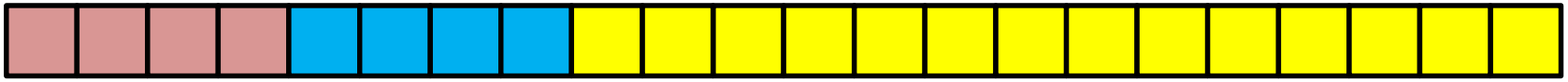
小課題 2 ($N \leq 14$)

- スタックの利用
  - 開き括弧が来たらスタックに push
  - 閉じ括弧が来たら
    - スタックが空なら *NO*
    - スタックの top と対応していなかったら *NO*
    - 対応していたら pop して続ける

  最後にスタックが空なら *YES*
- スタックだけ覚えておけばできる！

小課題 2 ($N \leq 14$)

- スタック 14 文字くらいなら全部覚えられそう



今何文字目？
0 から 14 まで
(4 ビット)

スタックの長さ
0 から 14 まで
(4 ビット)

スタックの内容
1 文字 2 種類
(14 ビット)

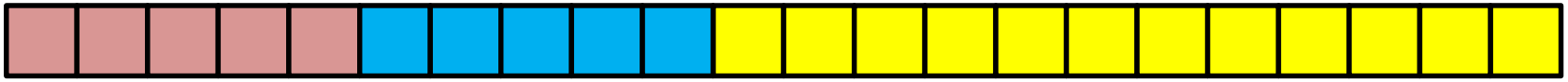
小課題 3 ($N \leq 24$)

- スタックの利用
  - 開き括弧が来たらスタックに push
  - 閉じ括弧が来たら
    - スタックが空なら *NO*
    - スタックの top と対応していなかったら *NO*
    - 対応していたら pop して続ける

  最後にスタックが空なら *YES*
- スタックが大きくなりすぎたら NO 確定
  - 高々 $N/2$ 文字

小課題 3 ($N \leq 24$)

- スタック 12 文字くらいなら全部覚えられそう



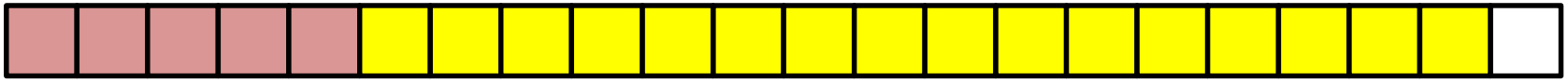
今何文字目？
0 から 24 まで
(5 ビット)

スタックの長さ
0 から 24 まで
(5 ビット)

スタックの内容
1 文字 2 種類
(12 ビット)

小課題 4 ($N \leq 30$)

- スタック 15 文字くらいなら全部覚えられそう



今何文字目？
0 から 30 まで
(5 ビット)

スタックの内容
長さ 0 から 15 まで・文字 1 文字 2 種類
(16 ビット)

小課題 4 ($N \leq 30$)

- スタック 15 文字くらいなら全部覚えられそう？？？
- 0 文字以上 15 文字以下の < と [の列は $2^0 + 2^1 + \cdots + 2^{15} = 2^{16} - 1$ 通りしかない
  - 適切に番号を振れば 16 ビットで覚えられる

小課題 4 ($N \leq 30$)

← 空文字列

<

[

<<

<[

[<

[[

<<<

<<[

<[<

...

小課題 4 ($N \leq 30$)

← 空文字列

0

1

00

01

10

11

000

001

010

...

小課題 4 ($N \leq 30$)

000001

000010

000011

000100

000101

000110

000111

001000

001001

001010

...

小課題 5 (<, > のみ)

- スタックの中身が < だけ→長さで決まる

今何文字目？
0 から 100 まで
(7 ビット)

スタックの長さ
0 から 100 まで
(7 ビット)

満点解法

- スタックを全部覚えるのはとても無理
  - 良い文字列なら最大 50 文字

満点解法

- スタックを**一部**覚えればいいのでは？
  - スタックの 1 文字目だけ覚える
  - スタックの 2 文字目だけ覚える
  - スタックの 3 文字目だけ覚える

    ……

  というのを別々にやればよい

満点解法

- スタックの 1 文字目だけ覚える

[<>] <>

スタック：

満点解法

- スタックの 1 文字目だけ覚える

[<>] <>

スタック： [

満点解法

- スタックの 1 文字目だけ覚える

[<>] <>

スタック : [?

満点解法

- スタックの 1 文字目だけ覚える

[<>] <>

スタック : [?

満点解法

- スタックの 1 文字目だけ覚える

[<>] <>

スタック：[

満点解法

- スタックの 1 文字目だけ覚える

[<>] <>

スタック：<

満点解法

- スタックの 1 文字目だけ覚える

[<>] <>

スタック：<

満点解法

- スタックの 2 文字目だけ覚える

[<>] <>

スタック：

満点解法

- スタックの 2 文字目だけ覚える

[<>] <>

スタック：？

満点解法

- スタックの 2 文字目だけ覚える

[<>] <>

スタック： ?<

満点解法

- スタックの 2 文字目だけ覚える

[<>] <>

スタック： ? <

満点解法

- スタックの 2 文字目だけ覚える

[<>] <>

スタック：？

満点解法

- スタックの 2 文字目だけ覚える

[<>] <>

スタック： ?

満点解法

- スタックの 2 文字目だけ覚える

[<>] <>

スタック：？

満点解法

- スタックを**一部**覚えればいいのでは？
  - スタックの 1 文字目だけ覚える
  - スタックの 2 文字目だけ覚える
  - スタックの 3 文字目だけ覚える

    ……

  というのを別々にやればよい
- 「文字列 S を読む」を N 周
  - N^2 ステップくらい (15,000 回制限 OK)

満点解法

- k 周目はスタックの k 文字目だけ覚える

今何周目？
0 から 100 まで
(7 ビット)

今何文字目？
0 から 100 まで
(7 ビット)

スタックの長さ
0 から 100 まで
(7 ビット)

スタックの一部
1 文字 2 種類
(1 ビット)

満点解法

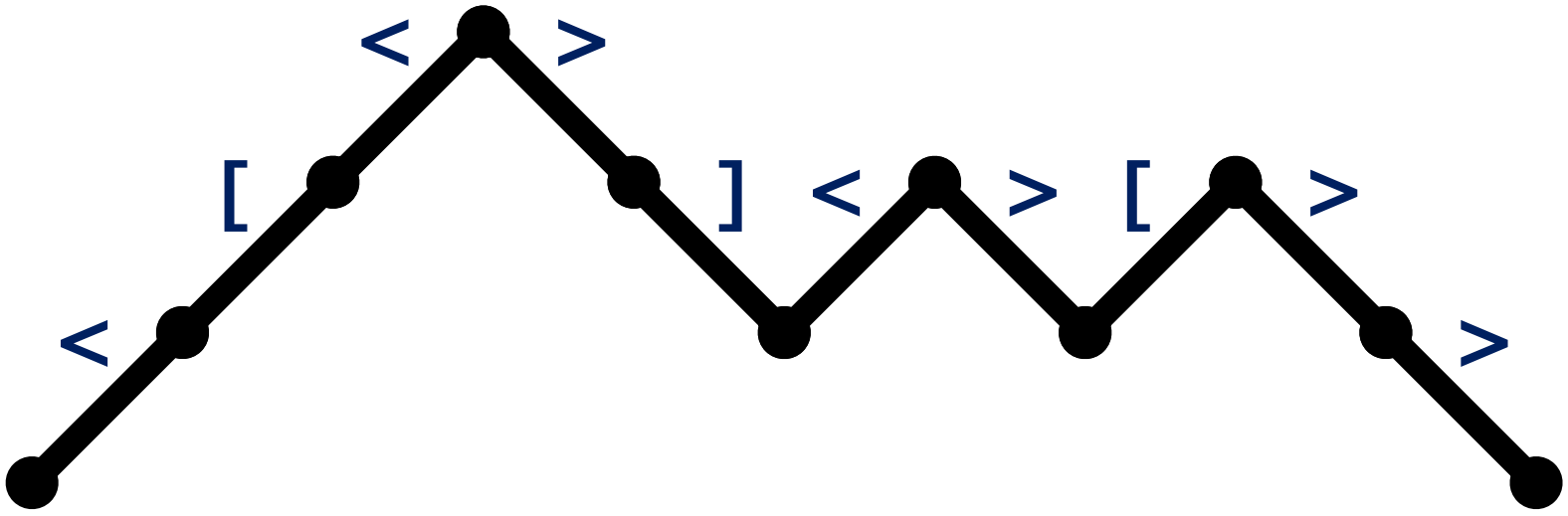
- 4 ビットくらい節約することもできる
  - スタックの長さは $N/2$ 以下としてよい
    - よって $N/2$ 周までによい
  - (スタックの長さ) ≤ (現在位置)
  - (スタックの長さ) ≡ (現在位置) $\pmod 2$

別解

- $k = 1, 2, \ldots, N$ に対して
  - S の k 文字目が開き括弧なら
    - その開き括弧の種類を覚える
    - k 文字目から順番に見て行って，ネストの深さが対応する閉じ括弧を探す→括弧の種類が違っていたら *NO*
  - S の k 文字目が閉じ括弧なら
    - 左に向かって同様に
- 必要な記憶量は同じくらい

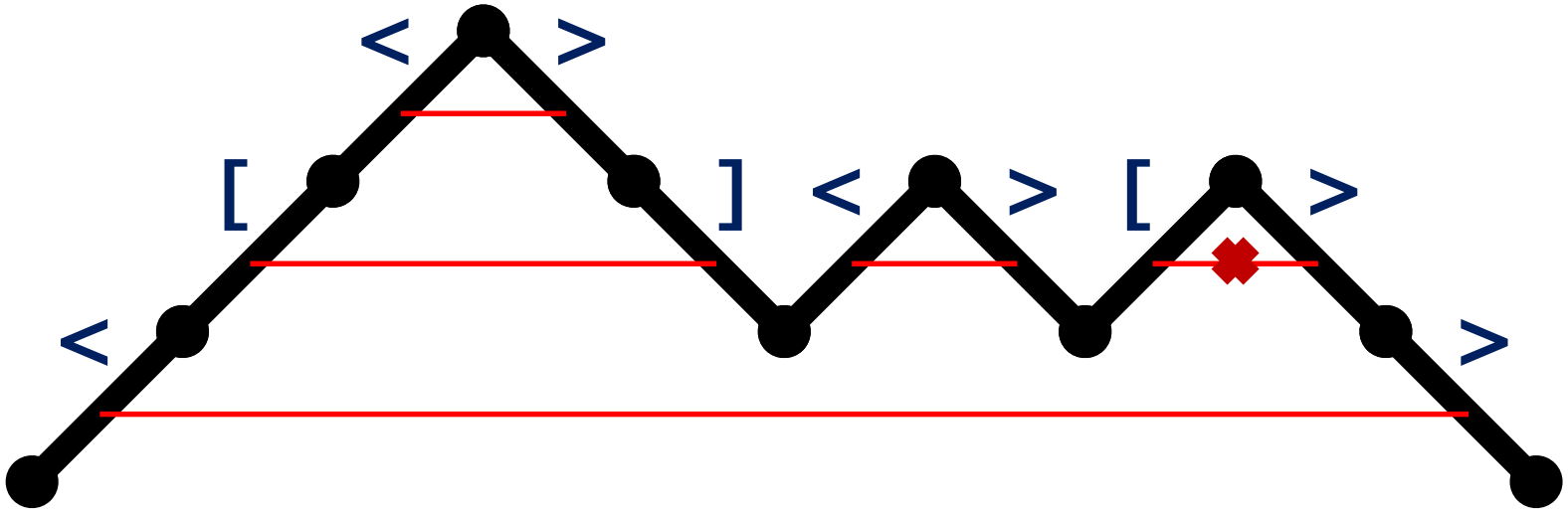
満点解法たち

<[<>]<>[>>



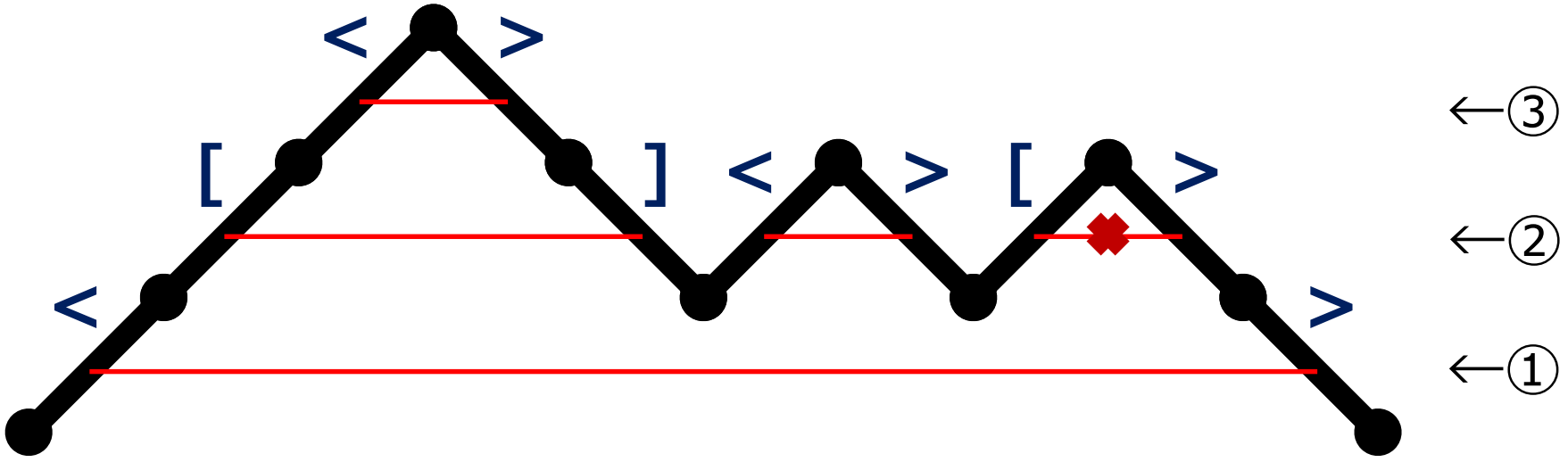
満点解法たち

<[<>]<>[>>



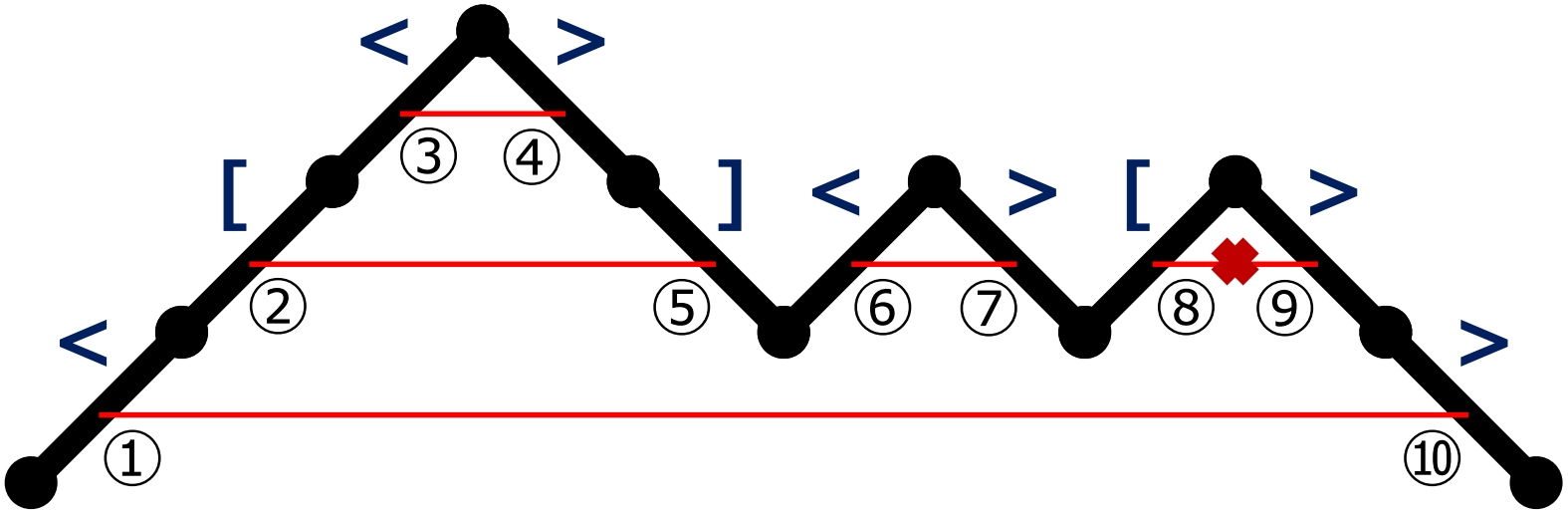
満点解法たち

<[<>]<>[>>



満点解法たち

<[<>]<>[>>



デバッグについて

- いろいろテストしないと正しく実装するのは難しいです
- 今回は「採点プログラムのサンプル」(a.k.a. grader) が 2 種類用意されていました
  - やりとりのデバッグ用 (grader-simple)
  - より実際の採点に近いほう (grader-strict)

デバッグについて

- 提供される grader が利用できないと不利
  - practice は重要
- ただし提供される grader 以外の機能がほしい場合もあるかも？
  - grader を改造する
  - 自分で main を書く

デバッグについて

(解答コード)

```
#ifndef EVAL
char Get(int I) {
  return ‘<‘;
}
int main(void) {
  printf(“%d¥n”, Memory(100, 4194303))
  return 0;
}
#endif
```

おまけ

- 空間計算量 $O(\log N)$ で括弧がとれているか判定しよう，という問題だった

- ではメモリ $O(1)$ でできるか？

おまけ

- 実は次のことが知られている

  空間計算量 $o(1)$ で判定できる

  ⇳

  正規言語である

- **正規言語** (regular language)
  - 定義の例：DFA で判定できる
  - 参考：
    http://www.kmonos.net/wlog/115.html#_2300101221

おまけ

- 「括弧の対応がとれた文字列の集合」は正規言語ではない
  - ことが知られている (cf. pumping lemma)
  - 括弧の種類が 1 種類であっても正規でない

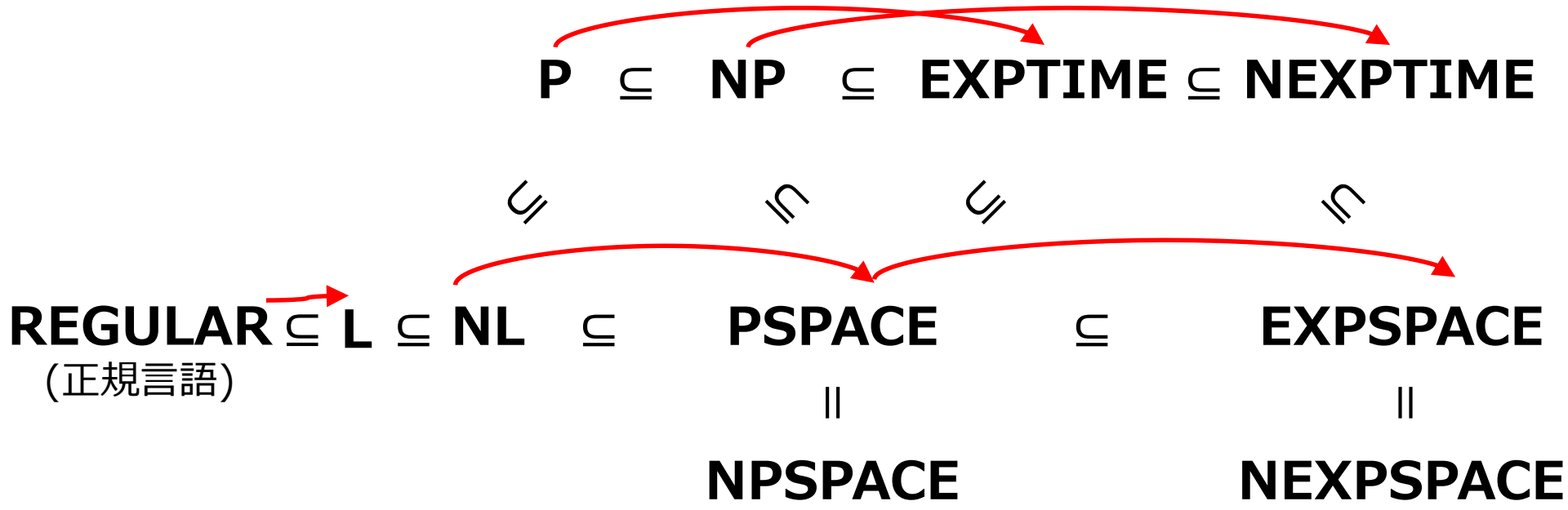
- よって空間計算量 $O(1)$ は無理
  - 実は正規言語でなければ $O(\log \log N)$ すら無理なことが知られているらしい

おまけ

- $O(\log N)$ 空間で判定できる言語 (判定問題) のクラスは **L** と書かれる
  - **NL** というのもあります (非決定性 $O(\log N)$ 空間)
    - **P** とか **NP** とかくらい大事に違いない

おまけ

- 知られている包含関係
  - 赤矢印が真の包含 $\subsetneq$



得点分布