一日一テーマやるやつ

一日で終わらなければ次の日も

えび

2018 年 7 月 31 日

1　2018 年 7 月 9 日

1.1　謎木（van Emde Boas tree）

整数の集合の演算を高速に行うデータ構造．通常の演算（insertion/erasure/membership）に加え，ある値 x を超える最小の値を求める successor，その逆の predecessor を得る演算と，最小値・最大値を求める演算も行える．$[0, U-1]$ の区間の整数を全体集合とする．bound された整数の集合を扱うデータ構造であり，multiset や map として使おうとするのは険しい．

全体を陽に持つとメモリがやばいので動的に確保するが，概念としては木構造になっている．根ノードは $\sqrt{U}$ 個の子を持ち，その子は $\sqrt{\sqrt{U}}$ 個の子を持つ（以下同様）．端数が出ると厄介なので，$U = 2^{(2^n)}$ としておく．各ノードは区間の一部分を管理していて，$[0, u-1]$ を管理しているノードの i 番目（0-indexed）の子ノードは $[i\sqrt{u}, (i+1)\sqrt{u}-1]$ を管理している．実装の際には，各子ノードは $[0, \sqrt{u}-1]$ を管理する集合とし，i 番目の子ノードが値 j を持っていれば元のノードは $i\sqrt{u}+j$ を持っていると判断するようにする．

各ノードが持っているデータは次の三つである．

- そのノードが持つ集合の最小値 m・最大値 M
- 子ノードたちへのポインタ
- summary と呼ばれる集合（後述）

$[0, u-1]$ を管理するノード v の持つ summary s は $[0, \sqrt{u}-1]$ の区間を管理するノード（すなわち v の各子ノード）と同じ構造を持っていて，$i \in s$ であることと，v の i 番目の子ノードの持つ集合が空でないことが同値となるように保たせる．

ノード v が管理する区間に含まれる要素数によって，データの持ち方は異なる．要素数が 2 以下のときは m と M のみを用いて表現し，s や子ノードたちは空とする．

v が持つ集合の要素を昇順に $\{a_1, a_2, \ldots, a_n\}$ とする．$n = 0$ のときは $m > M$ など，それ以外の状態では取り得ない状態にしておく．$n = 1$ のときは $m = M = a_1$ とし，$n = 2$ のときは $m = a_1$，$M = a_2$ とす

る．$n \geq 3$ のときは $m = a_1$，$M = a_n$ として，各子ノードを用いて $\{a_2, \ldots, a_{n-1}\}$ を表現する（子ノードには a_1 および a_n は含めない）．

これのお気持ちになるとあとは簡単で，これを保つように変更したり，これを元にして探索すればよい．successor については少々頭が必要で，以下のような処理をする．クエリを x とする．

- $x \geq M$ や $M < m$ であれば該当する要素は存在しない．
  - x を返してエラーを示すとか，例外を投げるとかをする．
- $x < m$ なら m を返す．
- 子ノードが空なら M を返す．
- x が入るべき子ノードの持つ最大値が x を超えていれば，その子ノードに該当する要素が含まれるので，そのノードを探索する．
  - 該当する子ノードの添字 i は $\lfloor x/\sqrt{U} \rfloor$ となる．
- s の最大値が i 以下であれば，M を返す．
- s における i の successor j を求め，j 番目の子ノードの最小値を返す．

$U = 256$ 程度であれば，64-bit 整数 4 つを bitset として扱うことで集合を表現できるため，それを利用した．64-bit 整数 1 つで持って各階層のノード数を $\{64, 4096, 16777216, \ldots\}$ とした場合，2^{64} と噛み合わなくなるのが嫌だったのでこのようにしてある．$U = 16$ のときに特殊化する実装と $U = 256$ のときに特殊化する実装を比較したところ，後者の方が高速であった．

AOJ の Lesson の `ITP2_7_C` で verify．大きい値側から求めて逆向きに出力することで predecessor についても verify 済み．min や max は陽には verify していないが，まぁ合っている気がする（えー）．

ちなみに実測で `std::set` に勝てていません（最悪ケース時）．ケースによっては勝っているものもありますが，競プロにおいては最も時間がかかるものが指標になってしまうため．i 番目の子ノードを管理するのに `std::map` を使っているんだけど，そこを工夫できると嬉しい？

2　2018 年 7 月 10 日 – 2018 年 7 月 31 日

2.1　Dynamic Connectivity (online)

Union-find に delete を追加したもの．“**This is a *much* harder problem!**”[*1]

まず森に関する D.C. を解く方法を考え，それを用いて一般のグラフに関する D.C. を解く．

2.1.1　2018 年 7 月 18 日

以下では，上記の森に対する D.C. を解く方法を示す．

まず，赤黒木を用意する．これは辺 (u, v) を要素として持ち，要素を in-order で読んだ辺の列が Euler tour

[*1] だいなみく・こねくちびちー・ぷろぶれむ

を表す．処理の都合上，自己辺 (u, u) を各頂点に張っておく．赤黒木自体の理解が大変なため，すでに一週間ほど使っていてつらいが，2-3-4 木に助けてもらいながらがんばった．

ここでの赤黒木は `split` と `merge` をできる必要がある．また，値の大小関係で順を管理するのではないことに注意が必要である．`split` の際には，基準となる点から根に向かって辿り，その方向に従って左右の木にくっつけていく方法を採用した．

はじめに `reroot`(u) と呼ばれるサブルーチンを定義しておく．これは Euler tour の始点を変更する処理で，以下のようにする．

- (u, u) で `split` し，木 L と木 R に分ける
- 木 R が木 L の左に来るように `merge` する
  - 木 R と木 L の間には (u, u) が入るようにする

`link`(u, v) をする際，事前条件として u と v が既につながっていないことが求められる [*2]．頂点 u，v を表す赤黒木のノード（すなわち (u, u) と (v, v)）が属する赤黒木をそれぞれ T_u，T_v とおく．

- T_u と T_v をそれぞれ u と v で `reroot`
- T_u と T_v を (u, v) をはさんで `merge`
- T_v の末尾に (v, u) を `insert`
  - 実装によっては T_v と木 $\{(v, u)\}$ を `merge` と考えてもよい

`are-connected`(u, v) に答える際は，T_u と T_v の根が同じかどうかを見ればよい．

`cut`(u, v) の事前条件は u と v が連結していることだが，これは当たり前といえば当たり前（切れているのに切ってほしいなら no-op でよい）．

- `reroot`(u) をして (u, v) の方が (v, u) より先に来るようにする
  - 逆でもよいが，どちらが先に来るかをはっきりさせておくと楽になる
    - これは嘘で，`reroot`(u) では (u, v) が (v, u) より先に来ることが保証できないよね
    - これも嘘で，全体が木であることを考えるとそれを保証できる
- (u, v)，(v, u) でそれぞれ `split` し，得られる木を T_J，T_K，T_L とする．
- T_J と T_L を `merge` した木が新たな T_u，T_K が新たな T_v となる．

以上で，森に関する D.C. は解決．

2.1.2　2018 年 7 月 31 日

一般のグラフ G での D.C. では，辺集合を二つに分けて考える．G 上で全域森（当然一意でないこともあるが，適当に決める）をなす辺集合を *tree edge*，その補集合を *regular edge*（または *non-tree edge*）とする．

*2 これは結構つらくて，この条件下で verify できる D.C. の問題を探すのに苦労する．そもそも存在するの？

また，各辺には *specificity*（または *level*）と呼ばれる非負整数値が定められていて，specificity が k 以上の tree edge で作られる森を $\mathcal{F}_k$ とする．この森を表現するために，上で紹介したデータ構造を用いる [*3]．

`are-connected`(u, v) は簡単で，$\mathcal{F}_0$ を見ればよい．G 上での連結性は $\mathcal{F}_0$ 上での連結性と等価なため．

`link`(u, v) の手順は以下の通り．新たに辺を追加する際の specificity の初期値は 0 である．

- u と v がすでに連結なら，regular edge に (u, v) を追加して終了
- そうでない場合
  - u の属する木 T_u と v の属する木 T_v を連結する
  - tree edge に (u, v) を追加する

`cut`(u, v) の手順は以下の通り．(u, v) が regular edge なら，それを取り除いて終了．そうでなければ，ループ変数 k の初期値を (u, v) の specifity とし，$k \geq 0$ の間，以下を繰り返す．

- (u, v) を取り除き，u，v の属する specificity が k の木をそれぞれ T_u，T_v とする
  - T_u の頂点数の方が小さくなるように適宜入れ替える（cf. マージテク）
- T_u の各辺の specificity を 1 増やす
- T_u から出ている regular edge を順に見る
  - それが T_u と T_v を結んでいれば，それらの木を連結させて終了
  - そうでなければ，その辺の specificity を 1 増やす
- 連結させられなければ，k を 1 減らす

最後まで連結させられなければ，実際に u と v の間のパスが消えたことになる．各ステップを高速に処理するため，以下の対応づけを管理しておくとよい．

- $(u, v) \mapsto$ specificity of (u, v)
- $u \mapsto \{$tree edges incident from $u\}$
- $u \mapsto \{$regular edges incident from $u\}$

これは Holm, de Lichtenberg および Throup による online なアルゴリズムで，log がたくさんついていて重い．Codeforces の Gym/100551-A の制約（頂点数，クエリ数ともに最大 3×10^5）には敵わなかった [*4]．

これとは別で，offline のアルゴリズムで $O(\log n)$ のものがあるので，今後それを勉強しよう．

[*3] 森に関する D.C. を処理できるデータ構造ならなんでもいいはず．
[*4] Time limit exceeded on test 15，実質 AC では？

3　2018 年 7 月 17 日

3.1　CYK[*5] 法

与えられた文 S が文脈自由文法 $\mathcal{G} = [T, N, \sigma, P]$ でに含まれるかを $O(|P| \cdot |S|^3)$ で判定するアルゴリズム．生成規則の適用の仕方の復元も可能で，複数存在する場合はその旨を報告できる．

区間 DP みたいなことをする．DP テーブルの添字は「長さ」「開始位置」「非終端記号（の ID）」で，要素は適用の方法の表す整数（true/false でもよい）．すなわち，$\mathrm{dp}[i][j][k]$ は，S の位置 j から長さ i の部分文字列を k 番目の非終端記号から導出することが可能かどうか（あるいはその種類数）を表す．ここでは 1-indexed で記すが，もちろん 0-indexed でもできる．予め生成規則を Chomsky 標準形に直しておく必要がある．

まず，S に含まれる各文字について，その位置とそれを直接導出する生成規則を求め，それに対応する要素（長さは 1）の値を 1 にする．長さ i を 2 から順に増やしていき，その区間 $[j, j+i]$ を二分した区間 $[j, j+k-1]$ および $[j+k, j+i]$ に対応する部分文字列を見る（順に S_A，S_B，S_C とする）．S_B と S_C がそれぞれ非終端記号 b と c から導出可能で，かつ $a \to b\ c$ とするような生成規則が存在した場合，S_A は非終端記号 a から導出可能であることが分かる．すなわち $\mathrm{dp}[i][j][a]$ を $\mathrm{dp}[k][j][b]$ と $\mathrm{dp}[i-k][j+k][c]$ から更新する．復元したい場合は (i, j, a) から (k, b, c) に対応させる情報を持っておくとよい．

$\mathrm{dp}[|S|][1][\sigma]$ が最終的な答えである．

4　その他

それ以前に学んだもので，まとめておきたいものたち．そのうち書く．

- 部分永続配列
  - 部分永続 Union-find
- Weighted Union-find
- Weighted Quick-find
- tsurai パーザ
- I/O 高速化
  - これいる？

学びたいものたちも挙げておく．

- Starry-sky tree
  - 区間更新できるセグ木と併せて押さえたい
- Wavelet 行列
- Eer-tree
- Suffix array

[*5] Cocke–Younger–Kasami.

- Suffix automaton
- Aho-Corasick 法
- **K**MP 法
- Z algorithm
- 桁 DP を一般化したやつ