2018 年度 情報領域演習第三
— 第 13 回 —

注意事項

- 前回に引き続き木構造データを扱うアルゴリズムについて実装を通じて学習する．今回もアルゴリズムを理解しないと解くことが難しいため，いきなり問題文や入出力例を読まずに，必ずその前にある説明をよく読むこと．わからないことがあれば TA や教員に質問しよう．赤黒木の挿入の動きについては，

  `https://www.cs.usfca.edu/~galles/visualization/RedBlack.html`

  が参考になる．
- 演習時間終了時点での提出状況についても評価の対象となるので，時間内にできるだけ多くの問題に挑戦するとよい．また，遅刻や途中退室は記録に残るので注意すること．
- プログラムの形式が指定されている場合には，それに従うこと．従わなくても「成功」と表示されることがあるが，得点にはならない．また，採点の際は `checker`とは異なる入出力例を用いて動作確認するので，「失敗」する反例に対する小手先の対策だけでは得点にならないことがあるので注意せよ．

はじめに

前回同様，CED において以下のコマンド (青字の部分) を実行することで出席を表明できる．ただし，N は所属するクラス名 1，2，3 で置き換え，N の前には空白を入れないこと．

```
[p1610999@blue00 ~]> /ced-home/staff/18jr3/13/checkerN
提出開始: 1 月 xx 日 14 時 40 分 0 秒
提出締切: 1 月 yy 日 14 時 39 分 0 秒
ユーザ: p1610999, 出席状況: 2019-01-xx-14-58
 問題:    結果 | 提出日時          | ハッシュ値                       |
    1:  未提出 |                   |                                  |
    2:  未提出 |                   |                                  |
                        ⋮
```

出席を表明するには，授業の開始から 30 分以内に実行する必要がある．実行しても出席状況が「欠席」である場合は教員に伝えること．なお，上の出力は例であるので，実際の締切りは各自コマンドで確認せよ．

提出も同じコマンドを用いて以下のように実行する．ただし，N はクラス名 (1 から 3)，X は 1 から 5 のいずれかの問題番号であり，`prog.c` は提出する C プログラムのファイル名であり，X の前後には忘れずに空白を入れること．

```
[p1610999@blue00 ~]> /ced-home/staff/18jr3/13/checkerN X prog.c
```

ここで，ファイル名として指定するのは，問題番号 X の問題を解く C プログラムのソースファイルであり，コンパイル済みの実行ファイルではない．ファイル名は特に指定しないが，「英数字からなる文字列 `.c`」などとすることが望ましい．このコマンドを実行することにより，指定されたファイルがコンパイルされ，実行テストプログラムが自動的に起動される．コンパイルに失敗した場合にはエラーとなり，提出されたことにはならないので注意すること．コンパイルが成功した場合には複数回の実行テストが行われ，実行テストにも成功すると「成功」と表示される．実行テストに失敗すると「失敗」と表示されるとともに反例となる入力と出力が表示されるので，失敗した理由を見つけるための参考にするとよい．なお，入出力が大きい場合やファイルとして入力したい場合には共に表示されるパスにあるファイルを使うと便利である．ただし，入力のない問題の場合は失敗しても反例は出力されない．

正しく提出できたか確認するためには以下のように出席の表明と同じコマンドを用いて確認できる．

```
[p1610999@blue00 ~]> /ced-home/staff/18jr3/13/checkerN
提出開始: 1 月 xx 日 14 時 40 分 0 秒
提出締切: 1 月 yy 日 14 時 39 分 0 秒
ユーザ: p1610999, 出席状況: 2019-01-xx-14-47
 問題:    結果 | 提出日時            | ハッシュ値                          |
    1:    成功 | 2019-01-xx-15-33  | 9b762c81932f3980cf03a768e044e65b |
                        ⋮
```

ハッシュ値は，提出した C プログラムのソースファイルの MD5 値である．CED では `md5sum` コマンドを用いて MD5 値を見ることにより，提出したファイルと手元のファイルが同じものであるかを確認することができる．

```
[p1610999@blue00 ~]> md5sum prog.c
9b762c81932f3980cf03a768e044e65b
```

なお，一度「成功」となった問題に対し，実行テストに失敗するプログラムを送信してしまうと，結果が「失敗」になってしまうので注意すること．

1 平衡木の続き

前回，偏りのない二分探索木を作成する方法として AVL 木を C 言語で実装したが，AVL 木は条件が厳しいため削除や挿入が起こった際の調節にやや時間がかかる．今回はその問題を解決する赤黒木を実装する．AVL 木と同様に挿入や削除に必要な時間計算量は $O(\log n)$ であるが，赤黒木は AVL 木より条件が緩いため，より高速な挿入や削除が可能となる．ただし，今回は赤黒木に対する削除は実装せず，挿入を中心に実装してみよう．

前回に引き続き，今回も以下の学生の成績に関する情報を構造体 `student` を扱う．

```
struct student { int id; char name[32]; int score; };
```

各問題で扱われる二分探索木の節点には，構造体 `student` の情報が学籍番号 (メンバ `id`) の大小により適切に格納されているものとする．また，赤黒木において節点に必要な情報は前回の二分探索木とは異なるので，節点のための構造体については後述のものを扱う．

1.1 木に関する用語の復習

木の高さ 根から葉にたどり着くまでに通る節点の数 (葉を除く) の最大値を**木の高さ**という．たとえば，葉の高さは 0 であり，左右の子が葉である節点を根とする木の高さは 1 である．

行きがけ順 親は子より先に見るが，子同士なら左の子を先に見る．

通りがけ順 左の子は親より先に見るが，親は右の子より先に見る．

帰りがけ順 子は親より先に見るが，子同士なら左にある子を先に見る．

二分木 どの節点の子の数も 2 か 0 である木を**二分木**という [1]．

部分木 木 T に対し，T の中の 1 つの節点を根とし，その子孫からなる木を T **の部分木**という．

左右の部分木 二分木の (葉を除く) 節点において，左の子の節点を根とする木を**左の部分木**，右の子の節点を根とする木を**右の部分木**という．

[1] 子の数が 1 の場合も許す流儀もある．

完全二分木　根から葉にたどり着くまでに通る節点の数が一定である二分木を**完全二分木**という．別の言い方をするなら「(葉を除く) どの節点についても，左右の部分木の高さが等しい」ということであり，全く偏りがない二分木である．

二分探索木　大小比較が可能なデータ (キー) が各節点に格納されている二分木について，

> どの節点についても，その節点のキーは左の部分木に含まれるキーよりも大きく，その節点のキーは右の部分木に含まれるキーより小さいか等しい

が言えるとき，この二分木を**二分探索木**という．

1.2　赤黒木の定義

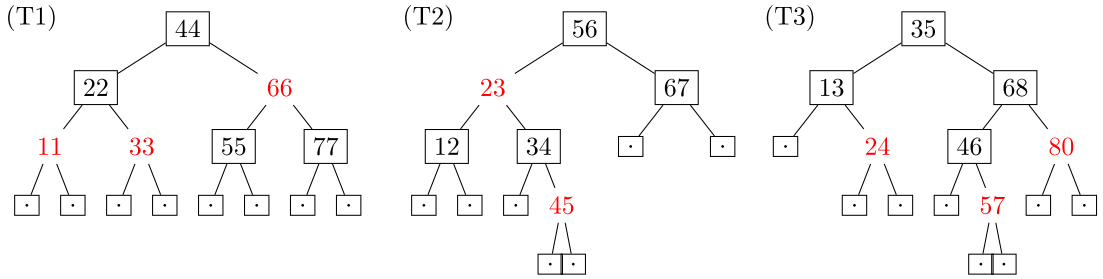
赤黒木は AVL 木と同じく偏りのない二分探索木の一つである．赤黒木は，各節点に赤か黒かどちらかの色がついている二分探索木で，節点の色は以下の条件を満たさなければならない．

条件 1　葉は必ず黒 [2]

条件 2　赤の節点の子は必ず黒 (黒の節点の子は赤でも黒でもよい)

条件 3　根から葉にたどり着くまでに通る黒い節点の数がすべて同じ

たとえば，以下の二分探索木において，四角で囲まれた節点は 黒，その他の節点は赤であるとすると，(T1) と (T2) は赤黒木であるが，(T3) は赤黒木ではない．



まず，どの二分探索木も**条件 1** と**条件 2** は満たしていることはすぐに確認できる．二分探索木 (T1) は完全二分木であるが，色は非対称につけられているものの，根から葉にたどり着くまでに通る黒い節点 (根や葉も含む) の数はすべて 3 であり**条件 3** が成り立つため，赤黒木である．二分探索木 (T2) は，根から遠い葉 (45 の子) と近い葉 (67 の子) があって一見偏りがあるように見えるが，根から葉にたどり着くまでに通る黒い節点の数は近くても遠くてもすべて 3 であり**条件 3** が成り立つため，赤黒木である．二分探索木 (T3) は，根から葉にたどり着くまでに通る黒い節点の数が 4 の場合 (46 を通る場合) と 3 の場合 (それ以外) があり**条件 3** が成り立たないため，赤黒木ではない．一般に，ある二分探索木が赤黒木の条件を満たすとき，その中の節点を根とする部分木も必ず赤黒木の条件を満たす．これも赤黒木の重要な性質の 1 つである．

赤黒木の条件を満たす二分探索木が偏りがないと言える理由　(この段落は課題とは直接関係がないので次の段落に進んでもよいが，興味があれば読むとよいかもしれない．) 赤黒木の条件が成り立っているときに偏りのない二分探索木になっていることは，次の事実からわかる．データの個数 (葉でない節点の個数) を n としよう．木の高さを h とすると，高さの定義から，根から葉にたどり着くまでに h 個の節点を通るような道が存在する．**条件 2** より，この道において赤い節点を連続して通ることはないので，この道にある赤い節点の数は $h/2$ 以下である．したがって，この道の黒い節点の数は $h/2$ 以上であり，**条件 3** より，葉にたどり着くまでのすべての道において黒い節点の個数は $h/2$ 以上であると言える．よって，この木全体に含まれる葉でない節点の個数 n は，高さ $h/2$ の完全二分木に含まれる葉でない節点の個数以上であるはずであり，$n \geq 2^{h/2} - 1$，すなわち，$h \leq 2\log_2(n+1)$ が成り立つ (つまり，データの数 n が 1023 であっても，それらを格納する赤黒木の高さ h は 20 以下)．よって，赤黒木は偏りのない二分木と言える．

[2] 根も黒であることを必要とする流儀もある．

この演習では，赤黒木の各節点は以下の構造体を用いて表し，葉は NULL とする．

```
struct rb_node { datatype data; struct rb_node *left, *right; int black; };
```

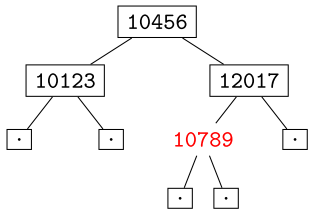
赤黒木の各節点には赤か黒かいずれかの色がついているため，通常の二分探索木のための定義と比べて，メンバ black が追加されている．メンバ black は，その節点が赤であれば 0，黒であれば 1 である．各節点の色はこのメンバに記録しておいて必要なときに参照させればよいが，葉 (NULL) である場合は同じようにメンバに参照しようとするとセグメントエラーを起こしてしまう．このため，次のような関数 is_red を用意しておくと非常に便利である．

```
int is_red(struct rb_node *t) {
  return (t != NULL && !t->black);
}
```

この関数は,「アドレス t の指す節点が赤い節点であれば 1 を返し，(葉も含めて) 黒い節点であれば 0 を返す関数」である．C 言語で e_1 && e_2 は，e_1 が 0 であれば e_2 を見ずに 0 を返すので，t が NULL のときには t->black を見にいくことがないため，セグメントエラーを起こさずに済む．**条件 1** より，葉の色は黒であり，以下の様々な操作でも葉でない黒い節点と同じように扱われることが多いので，この関数を使って節点の色を判定すればプログラムの見通しがスッキリするし，何よりセグメントエラーを誤って起こす心配が少なくなるだろう．

今回扱う赤黒木の各節点に格納するデータも構造体 student なので，typedef struct student datatype; と合わせて宣言する必要がある．また，前回と同様に各問題の入力は行きがけ順による木の表現を用いるものするが，以下の例のように各節点の情報には色を表す文字も与えられている (右は対応する赤黒木を図示したもの)：

```
[b]10456,David Beckham,77
[b]10123,Ichiro Suzuki,51
.
.
[b]12017,Osamu Dazai,50
[r]10789,Cristiano Ronaldo,7
.
.
.
```



葉ではない節点の表す行の先頭の角括弧で囲まれた文字が r なら赤い節点，b なら黒い節点を表し，続く文字列は学籍番号 (5 桁の整数値) と名前と得点 (0 から 100 までの整数値) をカンマで区切ったものである．葉は常に黒なので単に . だけの行で表している．今回も，問題を簡単にするために標準入力から与えられた文字列を元に赤黒木を作る関数 get_rbtree を用意した (赤黒木の条件は満たしているとは限らない)．

```
struct rb_node* get_rbtree() {
  struct rb_node *t;
  char c;
  /* ドットだけなら葉 (NULL) を返す */
  if(fgets(buf,sizeof(buf),stdin)==NULL || buf[0]=='.')
    return NULL;
  else {
    /* ドットでなければ節点を表す構造体のアドレス t を用意 */
    t = (struct rb_node*)malloc(sizeof(struct rb_node));
    /* 色を表す文字を c に，学籍番号，名前，得点を t->data に格納 */
    sscanf(buf,"[%c]%d,%[^,],%d",&c,&t->data.id,t->data.name,&t->data.score);
    /* 色の文字が b なら 1，r なら 0 */
    t->black = (c=='b');
    /* 左の子を t->left に，右の子を t->right に格納 */
    t->left = get_rbtree();  t->right = get_rbtree();
    /* t を返す */
    return t;
  }
```

```
}
```

また，構造体 rb_node を用いて表された赤黒木 (条件を満たしているとは限らない) から上述の入力と同じ形式に戻す関数 print_rbtree の定義は以下の通りである．

```
void print_rbtree(struct rb_node *t) {
  if(t==NULL) printf(".\n");
  else {
    printf("[%c]%d,%s,%d\n",t->black?'b':'r',t->data.id,t->data.name,t->data.score);
    print_rbtree(t->left); print_rbtree(t->right);
  }
}
```

1.3 赤黒木の回転

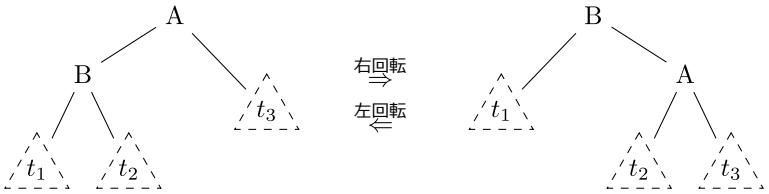
赤黒木に対する挿入は，AVL 木のときと同様に「通常の二分探索木に対する挿入を行ってから回転によって調節する」という手順で達成される．前回同様にまず赤黒木に対する回転を行う関数を定義してみよう．回転によって，赤黒木の**条件 2** や**条件 3** が崩れる可能性があるが，この関数では特に気にする必要はない (あとで挿入を行うときには，赤黒木の条件に注意しながら回転関数を用いる)．

問題 1

この問題の目的は，赤黒木の回転を行う 2 つの関数 rotate_right，rotate_left を定義することである．

```
struct rb_node* rotate_right(struct rb_node *t);
struct rb_node* rotate_left(struct rb_node *t);
```

関数 rotate_right は，「t の指す節点を根とする二分探索木に対して右回転を行い，その根の節点のアドレスを返す関数」であり，関数 rotate_left は同様に左回転を行う関数である．前回見たように，木の回転操作には**右回転**と**左回転**がある．



t_1，t_2，t_3 は何らかの部分木である [3]．先述の通り，これによって赤黒木の条件を満たさなくなってしまうこともあるが，ここでは特に色を修正しなくてもよい．

作成すべきプログラムの形式は後に示す main 関数を含むものとし，関数 rotate_right と rotate_left の定義を適切に埋めることによりプログラムを完成させよ．標準入力から与えられる入力は複数行に渡る文字列で，入力の最初の行は R か L のいずれか 1 文字であり，続く行は先述の形式で赤黒木 (条件を満たしていないこともある) を表したものである．標準出力には，与えられた赤黒木を R なら右回転，L なら左回転を行った後の赤黒木 (こちらも条件を満たしていないことがある) を入力と同じ形式で出力する．ただし，回転できない場合は元の木をそのまま出力するものとする．

[3] これらの部分木は葉であることもありうる．

入力例

```
R
[b]10456,David Beckham,77
[b]10123,Ichiro Suzuki,51
.
.
[b]12017,Osamu Dazai,50
[r]10789,Cristiano Ronaldo,7
.
.
.
```

出力例

```
[b]10123,Ichiro Suzuki,51
.
[b]10456,David Beckham,77
.
[b]12017,Osamu Dazai,50
[r]10789,Cristiano Ronaldo,7
.
.
.
```

入力例

```
L
[b]10456,David Beckham,77
[b]10123,Ichiro Suzuki,51
.
.
[b]12017,Osamu Dazai,50
[r]10789,Cristiano Ronaldo,7
.
.
.
```

出力例

```
[b]12017,Osamu Dazai,50
[b]10456,David Beckham,77
[b]10123,Ichiro Suzuki,51
.
.
[r]10789,Cristiano Ronaldo,7
.
.
.
```

入力例

```
R
[b]10456,David Beckham,77
.
[r]12017,Osamu Dazai,50
[b]10789,Cristiano Ronaldo,7
.
.
.
```

出力例

```
[b]10456,David Beckham,77
.
[r]12017,Osamu Dazai,50
[b]10789,Cristiano Ronaldo,7
.
.
.
↑回転できないので入力と同じ
```

作成すべきプログラムは以下の形式とする．※印を含むコメント部分を適切に書き換えればよいが，必要に応じて補助関数を定義してもよい．

```c
#include<stdio.h>
#include<stdlib.h>
char buf[128];

struct student { int id; char name[32]; int score; };
typedef struct student datatype; /* ← 格納するデータは構造体 student */
struct rb_node { datatype data; struct rb_node *left, *right; int black; };

struct rb_node* get_rbtree() {
  /* ※ここは先述の通り埋めればよい */
}

struct rb_node* rotate_right(struct rb_node *t) {
  /* ※ここを適切なプログラムで埋める */
}

struct rb_node* rotate_left(struct rb_node *t) {
  /* ※ここも適切なプログラムで埋める */
}
```

```
20
21 void print_rbtree(struct rb_node *t) {
22   /* ※ここは先述の通り埋めればよい */
23 }
24
25 int main() {
26   struct rb_node *t;
27   char c;
28   scanf("%c␣",&c);
29   t = get_rbtree();
30   if(c=='R') t = rotate_right(t);     /* R なら右回転 */
31   else if(c=='L') t = rotate_left(t); /* L なら左回転 */
32   print_rbtree(t);
33   return 0;
34 }
```

1.4 赤黒木の判定

ここから先の問題は，赤黒木の条件を満たしたまま回転や挿入を行う必要があるため，まず，与えられた二分探索木が赤黒木であるかどうか判定する関数を実装してみよう．ただし，この問題は後回しにしても，以降の問題には差し支えはない．

問題 2

この問題の目的は，与えられた二分探索木が赤黒木であるかどうか判定する関数 `is_rbtree` を定義することである．

```
int is_rbtree(struct rb_node *t);
```

関数 `is_rbtree` は，「`t` の指す節点を根とする二分探索木に対して，赤黒木であれば 1 を返し，そうでなければ 0 を返す関数」である．ただし，`t` の指す節点を根とする二分木は二分探索木であり，構造体 `rb_node` で表される二分木では葉に色の情報がないため，赤黒木の**条件 1** は自動的に満たされるものとしてよい．

作成すべきプログラムの形式は後に示す `main` 関数を含むものとし，関数 `is_rbtree` の定義を適切に埋めることによりプログラムを完成させよ．標準入力から与えられる入力は複数行に渡る文字列で，先述の形式で色情報を含む二分探索木を表したものである．標準出力には，与えられた二分探索木が赤黒木なら `Yes.` を出力し，そうでなければ `No.` を出力する．

入力例

```
[b]10567,Yuto Nagatomo,55
[r]10234,Makoto Hasebe,17
[b]10123,Ichiro Suzuki,51
.
.
[b]10345,Shohei Ohtani,17
.
[r]10456,David Beckham,77
.
.
[b]10678,Shinji Kagawa,23
.
.
```

出力例

```
Yes.
```

入力例

```
[b]10567,Yuto Nagatomo,55
[r]10234,Makoto Hasebe,17
[r]10123,Ichiro Suzuki,51
.
.
.
[r]10456,David Beckham,77
.
.
```

出力例

```
No.
条件 2 に違反しているため
```

入力例

```
[b]10345,Shohei Ohtani,17
[b]10123,Ichiro Suzuki,51
.
[r]10234,Makoto Hasebe,17
.
.
[b]10678,Shinji Kagawa,23
[b]10456,David Beckham,77
.
[r]10567,Yuto Nagatomo,55
.
.
[r]10890,Yukio Mishima,100
.
.
```

出力例

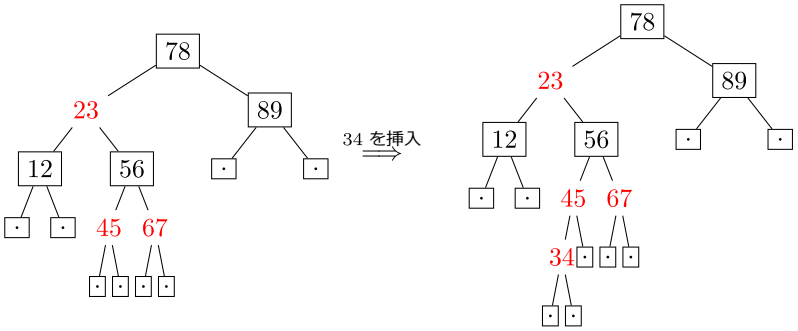
```
No.
条件 3 に違反しているため
```

作成すべきプログラムは以下の形式とする．※印を含むコメント部分を適切に書き換えればよいが，必要に応じて補助関数を定義してもよい．

```c
#include<stdio.h>
#include<stdlib.h>
char buf[128];

struct student { int id; char name[32]; int score; };
typedef struct student datatype; /* ← 格納するデータは構造体 student */
struct rb_node { datatype data; struct rb_node *left, *right; int black; };

struct rb_node* get_rbtree() {
  /* ※ここは問題 1 と同じ */
}

int is_rbtree(struct rb_node *t) {
  /* ※ここを適切なプログラムで埋める */
}

int main() {
  struct rb_node *t = get_rbtree();
  if(is_rbtree(t)) printf("Yes.\n");
  else             printf("No.\n");
  return 0;
}
```

1.5 赤黒木への挿入

赤黒木への挿入は通常の二分探索木と同様で,「根から開始して挿入すべき節点の学籍番号の方が小さければ左に，大きければ右に，といった具合に各節点の学生の学籍番号と比較していき，葉にたどり着いたらそこに挿入する」というものが基本である．ただし，赤黒木であり続けるためには条件が必要であるため，葉に挿入してから条件に合わせて調節を行う．**条件 1** は当然満たされているし，**条件 3** は赤い節点を挿入すれば崩れることはない．ただ，むやみに赤い節点を挿入すると，赤い節点の子の色が赤であることがありうるので，下のように**条件 2** が崩れてしまう (仮に 45 を黒い節点として挿入したら**条件 3** が崩れてしまう)．



そこで，次の使命は，赤い節点の挿入によって崩れた条件を回復するための関数を定義することである．以降，**条件 2** を崩すような赤い節点の親子の組を**赤親子**と呼ぶことにする．

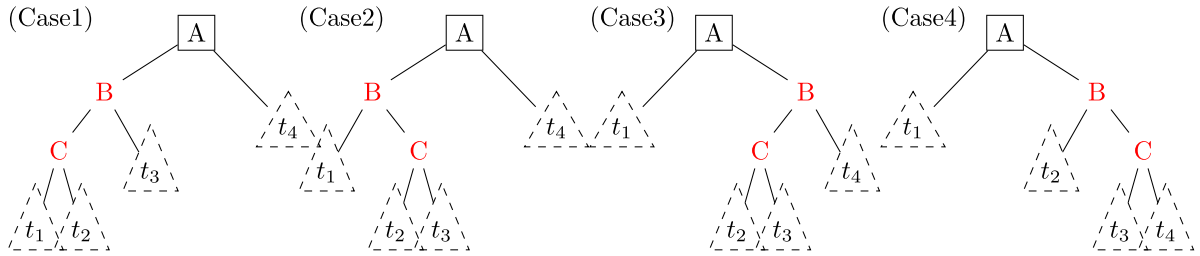
問題 3

この問題の目的は，赤親子を解消する関数 `resolve_red_pair` を定義することである．

```
struct rb_node* resolve_red_pair(struct rb_node *t) {
```

関数 `resolve_red_pair` は,「`t` の指す節点を根とする二分探索木において，根の子と孫に現れる赤親子を回転と色変えによって解消し，解消後の根の節点のアドレスを返す関数」である．子と孫の赤親子が存在しない場合はそのまま `t` を返す[4]．入力の二分探索木は赤親子は一箇所にしか現れないと仮定してよい．この関数は回転と色を変えるだけで実現されるため二分探索木としての性質は保たれたままであり，**条件 3** は崩さない．

この操作は次のようなアルゴリズムで実現する．子と孫が赤親子であるのは以下の 4 通りしかない．

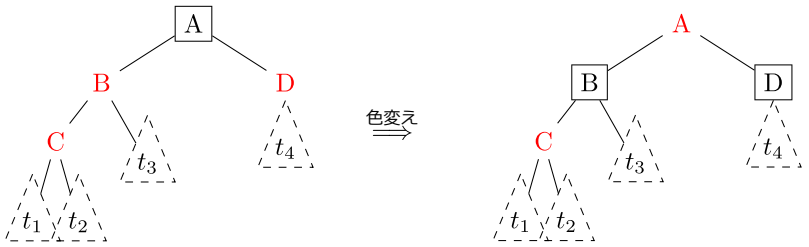


BC 以外に赤親子はないと仮定しているため，A が黒の場合しかあり得ない．それぞれのどのように赤親子を無くすかは以下の通り．

(Case1) t_4 の根の色によって対応が異なる．

- (Case1.1) t_4 の根が赤のとき，上の 3 つの色を変える．

[4] 子と孫からなる赤親子以外については考慮しない



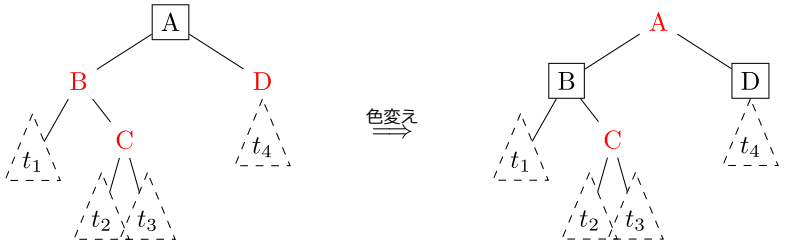
この色変えの前後で，根から葉にたどり着くまでに通る黒い節点の数は変わらないので，赤黒木の**条件 3** は崩れない．

- (Case1.2) t_4 の根が黒 (t_4 が葉の場合も含む) のとき，A で右回転して色変えを行う

この変形と色変えでも根から葉にたどり着くまでに通る黒い節点の数は変わらないので，赤黒木の**条件 3** は崩れない．また，元の木において赤親子は BC 以外に存在しないと仮定しているので，t_1，t_2，t_3 の根は全て黒であるため，変形と色変えの後で，新たに別の赤親子が現れることはない．

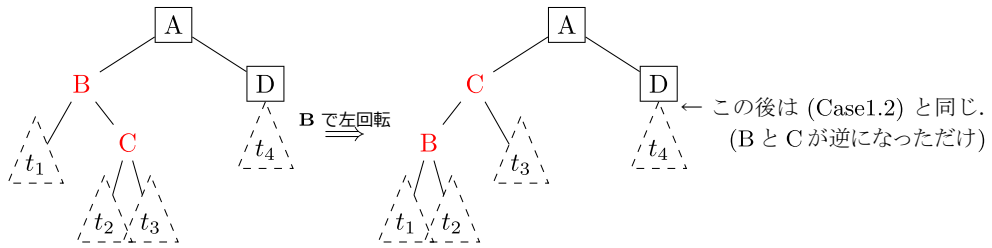
(Case2) t_4 の根の色によって対応が異なる．

- (Case2.1) t_4 の根が赤のとき，上の 3 つの色を変える．



この色変えを行っても根から葉にたどり着くまでに通る黒い節点の数は変わらないので，赤黒木の**条件 3** は崩れない．

- (Case2.2) t_4 の根が黒のとき，B で左回転させて (Case1.2) の場合に帰着する．



左の部分木の左回転を行っても根から葉にたどり着くまでに通る黒い節点の数は変わらないので，赤黒木の**条件 3** は崩れない．

(Case3) この場合は (Case2) を左右逆にして考えればよい．

(Case4) この場合は (Case1) を左右逆にして考えればよい.

作成すべきプログラムの形式は後に示す `main` 関数を含むものとし，関数 `resolve_red_pair` の定義を適切に埋めることによりプログラムを完成させよ．標準入力から与えられる入力は複数行に渡る文字列で，先述の形式で赤黒木を表したものである (入力において**条件 1** と**条件 3** は成り立っているが，**条件 2** は成り立っているとは限らない)．標準出力には，与えられた二分探索木の子と孫が赤親子である場合には，上の回転と色変えによって**条件 3** を保ったまま赤親子を解消し，解消後の赤黒木を出力する．入力される二分探索木に含まれる赤親子は 1 組以下であると仮定してよい.

入力例

```
[b]10456,David Beckham,77
[r]10234,Makoto Hasebe,17
[r]10123,Ichiro Suzuki,51
.
.
.
[r]10678,Shinji Kagawa,23
.
.
```

出力例

```
[r]10456,David Beckham,77
[b]10234,Makoto Hasebe,17
[r]10123,Ichiro Suzuki,51
.
.
.
[b]10678,Shinji Kagawa,23
.
.
```

入力例

```
[b]10456,David Beckham,77
.
[r]10678,Shinji Kagawa,23
.
[r]10789,Cristiano Ronaldo,7
.
.
```

出力例

```
[b]10678,Shinji Kagawa,23
[r]10456,David Beckham,77
.
.
[r]10789,Cristiano Ronaldo,7
.
.
```

入力例

```
[b]10678,Shinji Kagawa,23
[r]10234,Makoto Hasebe,17
[b]10123,Ichiro Suzuki,51
.
.
[r]10456,David Beckham,77
[b]10345,Shohei Ohtani,17
.
.
[b]10567,Yuto Nagatomo,55
.
.
[b]10789,Cristiano Ronaldo,7
.
.
```

出力例

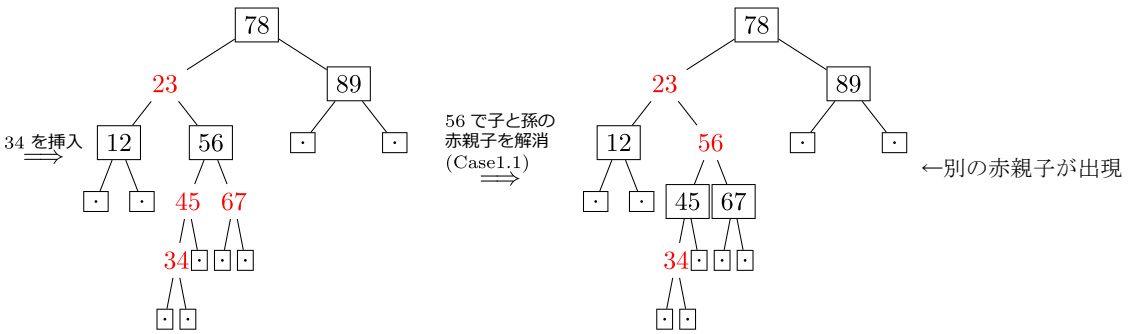
```
[b]10456,David Beckham,77
[r]10234,Makoto Hasebe,17
[b]10123,Ichiro Suzuki,51
.
.
[b]10345,Shohei Ohtani,17
.
.
[r]10678,Shinji Kagawa,23
[b]10567,Yuto Nagatomo,55
.
.
[b]10789,Cristiano Ronaldo,7
.
.
```

作成すべきプログラムは以下の形式とする．※印を含むコメント部分を適切に書き換えればよいが，必要に応じて補助関数を定義してもよい.

```c
#include<stdio.h>
#include<stdlib.h>
char buf[128];

struct student { int id; char name[32]; int score; };
typedef struct student datatype; /* ← 格納するデータは構造体 student */
```

```
7  struct rb_node { datatype data; struct rb_node *left, *right; int black; };
8
9  struct rb_node* get_rbtree() {
10   /* ※ここは問題 1 と同じ */
11 }
12
13 void print_rbtree(struct rb_node *t) {
14   /* ※ここも問題 1 と同じ */
15 }
16
17 struct rb_node* rotate_right(struct rb_node *t) {
18   /* ※ここも問題 1 と同じ */
19 }
20
21 struct rb_node* rotate_left(struct rb_node *t) {
22   /* ※ここも問題 1 と同じ */
23 }
24
25 struct rb_node* resolve_red_pair(struct rb_node *t) {
26   /* ※ここを適切なプログラムで埋める */
27 }
28
29 int main() {
30   struct rb_node *t = get_rbtree();
31   t = resolve_red_pair(t);
32   print_rbtree(t);
33   return 0;
34 }
```

赤黒木への挿入は，葉への挿入によって現れる赤親子を解消すれば実現できる．関数 `resolve_red_pair` によって赤親子は必ず解消されているように見えるかもしれないが，(Case1.1) や (Case2.1) の結果， A の節点が赤に変わったことで新たな赤親子が現れることがある．



上の例では，34 を挿入したことで赤親子が現れたため，その赤親子を子と孫にもつ 56 の節点で関数 `resolve_red_pair` を呼び出せばこの赤親子は解消できる．しかしながら，これによって新たな赤親子 (23 と 56) が現れるため，今度は 78 の節点で関数 `resolve_red_pair` を呼び出す必要がある．このように赤親子を解消しても別の赤親子が現れることがあるが，新たに現れる赤親子は多くても 1 つだけで，しかも根に近い方に現れるため，これを繰り返していけばいつかは根にたどり着くので，赤親子の解消処理は必ず終了する (ただし，根と子が赤親子にならないように根は必ず最後に黒にする)．

問題 4

この問題の目的は，赤黒木に対して挿入を行う関数の補助関数 `rb_insert_rec` を定義することである．

```
struct rb_node* rb_insert_rec(struct rb_node *t, struct student d);
```

関数 rb_insert_rec は,「t の指す節点を根とする赤黒木に対して構造体 student の値 d を挿入し，根以外の赤親子を解消し，その根の節点のアドレスを返す関数」である．t の指す節点を根とする二分木は赤黒木であり，d と同じ学籍番号を持つ学生のデータは格納されていないと仮定してよい．

関数 rb_insert_rec による操作は，以下のアルゴリズムによって実現される．

- t が葉なら節点のメモリを確保し，この節点の左右の部分木を葉とし，データとして d を入れ，色を赤として，この節点のアドレスを返す．
- t が葉ではない節点の場合，
  1. t の指す節点にあるデータの学籍番号と d の学籍番号と比較する．
     - d の学籍番号が小さければ，左の部分木を「左の部分木に d を挿入した木」で置き換える．
     - d の学籍番号が大きければ，右の部分木を「右の部分木に d を挿入した木」で置き換える．
  2. 「子と孫に赤親子があれば解消」して，根の節点のアドレスを返す．

下線の「部分木に d を挿入した木」を計算するために関数 rb_insert_rec を使うため，この関数は再帰関数として定義される．また,「子と孫に赤親子があれば解消」は問題 3 で定義した関数 resolve_red_pair を用いればよい．ただし，この操作では根に赤親子が残ってしまう場合があるので，以下のように rb_insert_rec を呼んだ後で根の節点の色を黒にする関数 rb_insert を用意して，赤黒木全体に対して挿入するときには最初にこちらの関数を呼び出せばよい．

```
struct rb_node* rb_insert(struct rb_node *t, struct student d) {
  t = rb_insert_rec(t, d); /* rb_insert_rec で t に d を挿入 */
  t->black = 1;            /* 根の節点の色を黒にする */
  return t;
}
```

作成すべきプログラムの形式は後に示す main 関数を含むものとし，関数 rb_insert_rec の定義を適切に埋めることによりプログラムを完成させよ．標準入力から与えられる入力は複数行に渡る文字列で，各行は学籍番号と名前と得点をカンマで区切った文字列である．標準出力には，与えられた学生のデータを順に挿入して得られる赤黒木を問題 1 と同じ形式で出力する．入力されるデータにおいて同じ学籍番号をもつ学生は 1 度しか現れないと仮定してよい．

入力例

```
10123,Ichiro Suzuki,51
10234,Makoto Hasebe,17
10345,Shohei Ohtani,17
```

出力例

```
[b]10234,Makoto Hasebe,17
[r]10123,Ichiro Suzuki,51
.
.
[r]10345,Shohei Ohtani,17
.
.
```

入力例

```
10123,Ichiro Suzuki,51
10345,Shohei Ohtani,17
10567,Yuto Nagatomo,55
10789,Cristiano Ronaldo,7
10890,Yukio Mishima,100
10456,David Beckham,77
```

出力例

```
[b]10345,Shohei Ohtani,17
[b]10123,Ichiro Suzuki,51
.
.
[r]10789,Cristiano Ronaldo,7
[b]10567,Yuto Nagatomo,55
[r]10456,David Beckham,77
.
.
.
[b]10890,Yukio Mishima,100
.
.
```

入力例

```
10012,Hideki Matsui,55
10123,Ichiro Suzuki,51
10234,Makoto Hasebe,17
10345,Shohei Ohtani,17
10456,David Beckham,77
10567,Yuto Nagatomo,55
10678,Shinji Kagawa,23
10789,Cristiano Ronaldo,7
10890,Yukio Mishima,100
12017,Osamu Dazai,50
```

出力例

```
[b]10345,Shohei Ohtani,17
[b]10123,Ichiro Suzuki,51
[b]10012,Hideki Matsui,55
.
.
[b]10234,Makoto Hasebe,17
.
.
[b]10567,Yuto Nagatomo,55
[b]10456,David Beckham,77
.
.
[r]10789,Cristiano Ronaldo,7
[b]10678,Shinji Kagawa,23
.
.
[b]10890,Yukio Mishima,100
.
[r]12017,Osamu Dazai,50
.
.
```

作成すべきプログラムは以下の形式とする．※印を含むコメント部分を適切に書き換えればよいが，必要に応じて補助関数を定義してもよい．

```c
#include<stdio.h>
#include<stdlib.h>
char buf[128];

struct student { int id; char name[32]; int score; };
typedef struct student datatype; /* ← 格納するデータは構造体 student */
struct rb_node { datatype data; struct rb_node *left, *right; int black; };

void print_rbtree(struct rb_node *t) {
  /* ※ここは問題 1 と同じ */
}

struct rb_node* rotate_right(struct rb_node *t) {
  /* ※ここも問題 1 と同じ */
}

struct rb_node* rotate_left(struct rb_node *t) {
  /* ※ここも問題 1 と同じ */
}

struct rb_node* resolve_red_pair(struct rb_node *t) {
  /* ※ここは問題 3 と同じ */
}

struct rb_node* rb_insert_rec(struct rb_node *t, struct student d) {
  /* ※ここを適切なプログラムで埋める */
}

struct rb_node* rb_insert(struct rb_node *t, struct student d) {
  t = rb_insert_rec(t, d);
  t->black = 1;
```

```
32   return t;
33 }
34
35 int main() {
36   struct student d;
37   struct rb_node *t=NULL; /* 葉のみの赤黒木を用意 */
38   while(fgets(buf,sizeof(buf),stdin)!=NULL) {
39     sscanf(buf,"%d,%[^,],%d",&d.id,d.name,&d.score); /* 学生の番号を読取り */
40     t = rb_insert(t, d);                              /* そのデータを赤黒木に追加 */
41   }
42   print_rbtree(t);
43   return 0;
44 }
```

問題 5

この問題の目的は，各学生の総得点を計算するために赤黒木のデータを更新する関数 `rb_update` を定義することである．

```
struct rb_node* rb_update(struct rb_node *t, struct student d);
```

関数 `rb_update` は，「`t` の指す節点を根とする赤黒木に対し，構造体 `student` の `d` と同じ学籍番号がなければそれを挿入し，既に存在すればその節点のデータに `d` の得点を加え，得られた赤黒木の根の節点のアドレスを返す関数」である．挿入と同じ要領で探索し，同じ学籍番号の場合は単に得点を加えればよい．挿入のときと同様に更新後の赤黒木の根の節点の色は黒とする．

作成すべきプログラムの形式は後に示す `main` 関数を含むものとする．

入力例

```
10123,Ichiro Suzuki,51
10234,Makoto Hasebe,17
10345,Shohei Ohtani,17
10234,Makoto Hasebe,82
10123,Ichiro Suzuki,67
10234,Makoto Hasebe,24
```

出力例

```
[b]10234,Makoto Hasebe,123
[r]10123,Ichiro Suzuki,118
.
.
[r]10345,Shohei Ohtani,17
.
.
```

入力例

```
10345,Shohei Ohtani,17
10789,Cristiano Ronaldo,7
10567,Yuto Nagatomo,55
10890,Yukio Mishima,100
10123,Ichiro Suzuki,51
10890,Yukio Mishima,100
10456,David Beckham,77
10890,Yukio Mishima,100
10789,Cristiano Ronaldo,7
10567,Yuto Nagatomo,100
```

出力例

```
[b]10567,Yuto Nagatomo,155
[b]10345,Shohei Ohtani,17
[r]10123,Ichiro Suzuki,51
.
.
[r]10456,David Beckham,77
.
.
[b]10789,Cristiano Ronaldo,14
.
[r]10890,Yukio Mishima,300
.
.
```

入力例

```
10345,Shohei Ohtani,17
10345,Shohei Ohtani,25
10345,Shohei Ohtani,63
10789,Cristiano Ronaldo,7
10123,Ichiro Suzuki,51
10456,David Beckham,77
10345,Shohei Ohtani,1
```

出力例

```
[b]10345,Shohei Ohtani,106
[b]10123,Ichiro Suzuki,51
.
.
[b]10789,Cristiano Ronaldo,7
[r]10456,David Beckham,77
.
.
.
```

作成すべきプログラムの main 関数は以下の形式とする．プログラムの他の部分は問題 4 を参考にするとよい．

```
1  int main() {
2    struct student d;
3    struct rb_node *t=NULL; /* 葉のみの赤黒木を用意 */
4    while(fgets(buf,sizeof(buf),stdin)!=NULL) {
5      sscanf(buf,"%d,%[^,],%d",&d.id,d.name,&d.score); /* 学生の番号を読取り */
6      t = rb_update(t, d);                             /* そのデータで赤黒木を更新 */
7    }
8    print_rbtree(t);
9    return 0;
10 }
```