2018 年度 情報領域演習第三
— 第 12 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．今回は特にアルゴリズムを理解しないと解くことができないため，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．AVL 木の挿入や削除の動きについては，

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

  が参考になる (ただし，回転が 2 回起こるときの可視化が一部省略されているので注意)．
- 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/18jr3/12/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/18jr3/12/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/18jr3/12/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

前回，木の基本概念を復習し，構造体 `node` を用いて二分木を C 言語に実装する方法や応用について学習した．特に，二分探索木と呼ばれる二分木は探索を高速に行うことができるデータ構造であった．二分探索木は n 個の要素を二分木の節点に格納するため，平均的には高さ $\log_2 n$ 程度の二分木にすべての要素を格納することが可能であるため，探索に必要な平均時間計算量は $O(\log n)$ である．しかしながら，運悪く二分探索木が偏って作られてしまった場合には，高さ n の二分探索木が必要になり，線形探索と同じように探索に時間がかかってしまう．

今回実装するのは，**平衡探索木** (balanced search tree) とよばれる偏りのない木にデータを格納するアルゴリズムである．平衡探索木にはいくつか種類があるが，今回は AVL 木を実装してみよう．

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．なお，AVL 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．

[1] 子の数が 1 の場合も許す流儀もある．

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．

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，

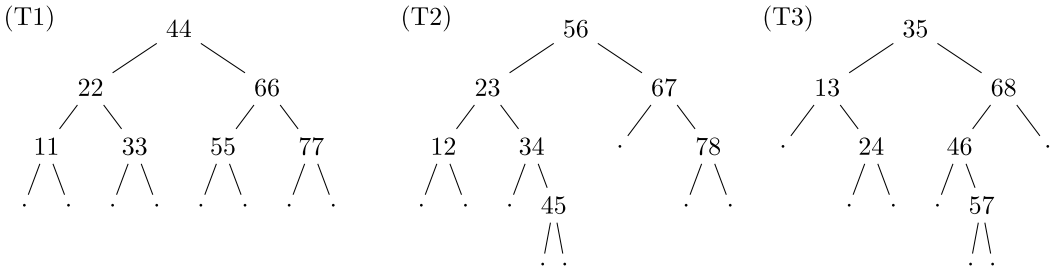
> どの節点についても，その節点のキーは左の部分木に含まれるキーよりも大きく，その節点のキーは右の部分木に含まれるキーより小さいか等しい

が言えるとき，この二分木を**二分探索木**という．

1.2　AVL 木

初めにも述べたように，偏りがある二分探索木の場合 (たとえば，すべての節点の左の子が葉である場合) は，線形探索と同じくらい探索時間が必要になってしまう．完全二分木のように偏りのない二分探索木が理想的ではあるが，要素の個数によっては完全二分木を作ることはできない [2]．

そこで，なるべく偏りのない二分探索木でありつづけるように挿入や削除ができるようにしたデータ構造として考案されたのが AVL 木である．**AVL 木** [3] とは，「どの節点も左右の子の木の高さの差が 1 以下」であるような二分探索木のことである (ただし，1 つの葉だけからなる木の場合も AVL 木とする)．たとえば，次の (T1) から (T3) はいずれも二分探索木であるが，(T1) と (T2) は AVL 木であり，(T3) は AVL 木ではない．



二分探索木 (T1) は完全二分木であり，どの節点についても左右の部分木の高さが等しい (高さの差が 0 である) ので AVL 木である．二分探索木 (T2) は一見偏りがあるように見えるが，根である 56 をキーとする節点は，左の部分木の高さが 3，右の部分木の高さが 2 であり，差は 1，23 をキーとする節点は，左の部分木の高さが 1，右の部分木の高さが 2 であり，差は 1，12 をキーとする節点は，左の部分木も右の部分木も葉で高さが 0 であり，差は 0，... のように見ていくとすべての節点の左右の部分木の高さの差が 1 以下なので，AVL 木である．二分探索木 (T3) は一見 (T2) と同程度に偏りがないように見えるが，68 をキーとする節点は，左の部分木の高さが 2，右の部分木の高さが 0 であり，差は 2 であるため，AVL 木ではない．AVL 木には「どの節点も左右の子の木の高さの差が 1 以下」という条件により大きな偏りがないため，探索に必要な時間計算量は常に $O(\log n)$ となることが理論的に証明できる．

前回，二分探索木に対する挿入や削除を実装したが，AVL 木に対して同じ方法で挿入や削除をしてしまうと，AVL 木の条件を満たさなくなってしまう．このため，データの挿入や削除は，AVL 木の条件を保つように実装する必要がある．そこで，木の高さがすぐにわかるように，AVL 木の節点を表す構造体 `avl_node` を次のように定義する：

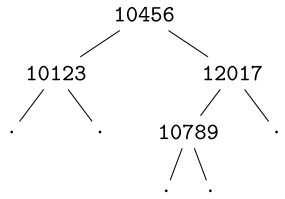
```
struct avl_node { datatype data; struct avl_node *left, *right; int height; };
```

[2] 葉を除く節点が 5 個や 6 個の完全二分木を作れるか考えてみよう．
[3] AVL は，このデータ構造を考案した 2 人の数学者 Adelson-Velskii と Landis の名前に由来する．

これは，前回二分探索木を実装するために用いた節点を表す構造体 `node` とほぼ同じ定義であるが，木の高さに関する情報がメンバ `height` として追加されている [4].

今回，AVL 木の各節点に格納するデータは構造体 `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高さとして適切な数値が与えられている (右は対応する AVL 木を図示したもの)：

```
[3]10456,David Beckham,77
[1]10123,Ichiro Suzuki,51
.
.
[2]12017,Osamu Dazai,50
[1]10789,Cristiano Ronaldo,7
.
.
.
```

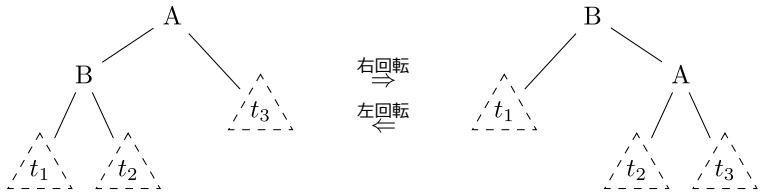


葉ではない節点の表す行の先頭の角括弧で囲まれた数値が高さを表しており，続く文字列は学籍番号 (5 桁の整数値) と名前と得点 (0 から 100 までの整数値) をカンマで区切ったものである．今回も，問題を簡単にするために標準入力から与えられた文字列を元に二分木を作る関数 `get_avl` を用意した．

```
struct avl_node* get_avl() {
  struct avl_node *t;
  /* 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 avl_node*)malloc(sizeof(struct avl_node));
    /* 高さを t->height に，学籍番号，名前，得点を t->data に格納 */
    sscanf(buf,"[%d]%d,%[^,],%d",&t->height,&t->data.id,t->data.name,&t->data.score);
    /* 左の子を t->left に，右の子を t->right に格納 */
    t->left = get_avl();
    t->right = get_avl();
    /* t を返す */
    return t;
  }
}
```

AVL 木の平衡条件を保ちながら挿入や削除を実現するための準備として，まず,「木の回転」を行う関数 `rotate_right` と `rotate_left` を実装しよう．

二分木の回転は二分探索木の平衡を保つために重要な操作である．具体的には部分木の位置を入れ替える操作であるが，次のように**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 [5]．回転操作を行っても，通りがけ順では節点の順序が変わっていないことに注意しよう．通りがけ順に節点を並べると，左右のどちらの二分木も，t_1 の節点，B，t_2 の節点，A，t_3

[4] `height` の代わりに左右どちらの高さが大きいか (または等しいか) を節点に記録する方式もあるが，ここでは簡単のため自身の高さを節点に記録する．どちらが大きいかは左右の子の節点のメンバ `height` (葉なら 0) を比較すればよい．

[5] これらの部分木は葉であることもありうる．

の節点の順になっていることがわかる．つまり，二分探索木における順序 (通りがけ順) が保たれているため，回転前に二分探索木の条件が満たされていれば，回転後も必ず二分探索木の条件が満たされていることになる．回転は少し複雑な操作に見えるが，足し算の結合則のような $x_1 + (x_2 + x_3)$ と $(x_1 + x_2) + x_3$ の間のやりとりだと思えばわかりやすいかもしれない．どんなに大きな二分探索木であっても定数回の操作で終了するということも，回転操作の重要な特徴の 1 つである．

右回転により左の部分木の高さは必ず小さくなり，左回転により右の部分木の高さは必ず小さくなるので，同じ情報を含む二分探索木でありながら部分木の高さを調節することが可能となる．たとえば，上の左の図において，t_1，t_2，t_3 の高さがいずれも 5 であるような AVL 木を考えよう．t_1 に値が挿入されることによって高さが 6 になったとしたら，B から下の左の部分木の高さが 7 になるため AVL 木の条件を満たさなくなるが，右回転の操作を行うことにより，AVL 木に戻すことができる．ただし，t_1 ではなく，t_2 に値が挿入されて高さが 6 になったとしたら，右回転しても AVL の条件が満たされることはないので，まず，B 以下の部分木で左回転してから A 以下の木に対して右回転を行えばよい．回転を利用した挿入アルゴリズムの詳細は問題 3 で紹介する．

問題 1

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

```
struct avl_node* rotate_right(struct avl_node *t);
struct avl_node* rotate_left(struct avl_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．`t` の指す節点を根とする二分木は二分探索木 (AVL 木とは限らない) であるが，各節点のメンバ `height` には部分木の高さの値が適切に格納されていると仮定してよい．`t` が `NULL` である場合や，根の左の部分木が葉であるのに右回転を行う場合，根の右の部分木が葉であるのに左回転を行う場合は，回転ができないので何も変更せずに `t` 自身を返す．これらの関数は，先ほどの図で表された節点 A や B のポインタの付け替えだけで実装することができるが，これらの節点のメンバ `height`が変わることに注意せよ．

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 AVL 木を表したものである．標準出力には，与えられた AVL 木を `R` なら右回転，`L` なら左回転を行なった後の AVL 木を入力と同じ形式 (最初の行を除く) で出力する．ただし，回転できない場合は元の AVL 木をそのまま出力するものとする．

入力例

```
R
[3]10456,David Beckham,77
[1]10123,Ichiro Suzuki,51
.
.
[2]12017,Osamu Dazai,50
[1]10789,Cristiano Ronaldo,7
.
.
.
```

出力例

```
[4]10123,Ichiro Suzuki,51
.
[3]10456,David Beckham,77
.
[2]12017,Osamu Dazai,50
[1]10789,Cristiano Ronaldo,7
.
.
.
```

入力例

```
L
[3]10456,David Beckham,77
[1]10123,Ichiro Suzuki,51
.
.
[2]12017,Osamu Dazai,50
[1]10789,Cristiano Ronaldo,7
.
.
.
```

出力例

```
[3]12017,Osamu Dazai,50
[2]10456,David Beckham,77
[1]10123,Ichiro Suzuki,51
.
.
[1]10789,Cristiano Ronaldo,7
.
.
.
```

入力例

```
R
[3]10456,David Beckham,77
.
[2]12017,Osamu Dazai,50
[1]10789,Cristiano Ronaldo,7
.
.
.
```

出力例

```
[3]10456,David Beckham,77
.
[2]12017,Osamu Dazai,50
[1]10789,Cristiano Ronaldo,7
.
.
.
↑ 左の部分木が葉だけなので右回転ができないため，
入力の木と同じ木が返される
```

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]; /* 関数 get_avl で用いるグローバル変数 */

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 avl_node { datatype data; struct avl_node *left, *right; int height; };

struct avl_node* get_avl() {
  /* ※ここは先述のとおり書けばよい */
}

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

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

void print_avl(struct avl_node *t) {
  if(t==NULL)
    printf(".\n");
  else {
    printf("[%d]%d,%s,%d\n",t->height,t->data.id,t->data.name,t->data.score);
    print_avl(t->left);
    print_avl(t->right);
  }
}

```

```
31 int main() {
32   struct avl_node *t;
33   char c;
34   scanf("%c␣",&c);
35   t = get_avl();
36   if(c=='R')
37     t = rotate_right(t);
38   else if(c=='L')
39     t = rotate_left(t);
40   print_avl(t);
41   return 0;
42 }
```

ヒント

- 回転操作は入力の大きさに関係なく定数時間で実現できるため，再帰や反復は必要がないことに注意せよ．
- 葉でも葉でなくても高さを知ることができる関数 `height` を用意すると便利かもしれない．`NULL` なら 0 を返し，`NULL` でなければメンバ `height` の値を返せばよい (この関数は後のすべての問題で活躍するだろう)．

  ```
  int height(struct avl_node *t);
  ```

- 左右の部分木の高さが正しく設定されているときにメンバ `height` を適切に設定し直す関数 `put_height` を定義するとスッキリ書ける．

  ```
  void put_height(struct avl_node *t);
  ```

左右の部分木がそれぞれ AVL 木の条件を満たしていて，その高さの差が 2 以下であるとき，2 回以下の回転によって全体として AVL 木の条件を満たすように変形することができる．この事実は挿入や削除を行う上で非常に重要である．これを実現する関数 `balance` を定義してみよう．

問題 2

二分探索木に対して AVL 木を満たすように変形する関数 `balance` を定義せよ．関数 `balance` の引数と返り値の型は以下の通りである．

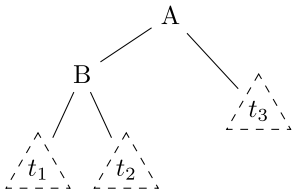
```
struct avl_node* balance(struct avl_node *t);
```

この関数は，「構造体 `avl_node` のアドレス `t` の指す節点を根とする二分探索木について，左右の部分木が AVL 木であり，それらの高さの差が 2 以下であるとき，全体として AVL 木になるように変形し，その根の節点のアドレスを返す関数」である．引数で与えられるアドレスの指す節点を根とする木は二分探索木であり，左右の部分木 (葉の場合も含む) は AVL 木の条件を満たし，それらの高さの差は 2 以下であるものと仮定してよい．

この操作は以下のアルゴリズムによって実現できる．まず，左右の部分木の高さの差が 2 以下なので以下の 3 つの場合が考えられる．

- 左右の部分木の高さの差が 1 以下の場合
- 左の部分木の高さが右の部分木の高さよりちょうど 2 大きい場合
- 右の部分木の高さが左の部分木の高さよりちょうど 2 大きい場合

1 つめの場合は，すでに AVL なので引数の `t` をそのまま返せばよい．2 つめと 3 つめの場合の違いは左右が逆になっただけのことなので，ここでは 2 つめの場合だけを詳しく説明する．



左の部分木が右の部分木の高さよりちょうど 2 大きい場合，左の部分木には少なくとも 2 つの節点が含まれるので，元は右図のような形の二分探索木であり，B から下の部分木と t_3 がそれぞれ AVL 木の条件を満たしている．大雑把にいえば，「左の部分木の方が大きいので A から下の木 (つまり全体) に対して右回転すればよい」ということであるが，実際にはそれだけではうまくいかない場合がある．次の 2 つの場合に分けて考えてみよう．

- t_1 の高さが t_2 の高さより大きいか等しい場合
  このときは，A から下の木に対して右回転させれば，左の部分木の高さが 1 つ小さくなり，A から下の木において AVL 木の条件を満たす状態になる．

- t_2 の高さが t_1 の高さより大きい場合
  このまま，A から下の木に対して右回転してしまうと，t_2 が同じ深さのまま右の部分木に移動するだけで右の部分木が高くなってしまうので適切ではない．そこで，一旦 B から下の木に対して左回転させてから，A から下の木に対して右回転させると，A から下の木において AVL 木の条件を満たす状態になる．

逆に，右の部分木が左の部分木の高さよりちょうど 2 大きい場合には，上と左右逆の操作を行えばよい．

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 `balance` 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，問題 1 と同様の形式で二分探索木を表したものである．標準出力には，上のアルゴリズムを適用した後の AVL 木を同じ形式で出力する．入力は二分探索木であり，左右の部分木は AVL 木の条件を満たし，それらの高さの差は 2 以下であると仮定してよい．

入力例

```
[3]10789,Cristiano Ronaldo,7
[2]10456,David Beckham,77
[1]10123,Ichiro Suzuki,51
.
.
.
.
```

出力例

```
[2]10456,David Beckham,77
[1]10123,Ichiro Suzuki,51
.
.
[1]10789,Cristiano Ronaldo,7
.
.
```

入力例

```
[3]10567,Yuto Nagatomo,55
[2]10345,Shohei Ohtani,17
.
[1]10456,David Beckham,77
.
.
[1]10678,Shinji Kagawa,23
.
.
```

出力例

```
[3]10567,Yuto Nagatomo,55
[2]10345,Shohei Ohtani,17
.
[1]10456,David Beckham,77
.
.
[1]10678,Shinji Kagawa,23
.
.
↑もともと AVL 木なので変化なし
```

入力例

```
[4]10567,Yuto Nagatomo,55
[3]10345,Shohei Ohtani,17
[2]10123,Ichiro Suzuki,51
.
[1]10234,Makoto Hasebe,17
.
.
[1]10456,David Beckham,77
.
.
[1]10678,Shinji Kagawa,23
.
.
```

出力例

```
[3]10345,Shohei Ohtani,17
[2]10123,Ichiro Suzuki,51
.
[1]10234,Makoto Hasebe,17
.
.
[2]10567,Yuto Nagatomo,55
[1]10456,David Beckham,77
.
.
[1]10678,Shinji Kagawa,23
.
.
```

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]; /* 関数 get_avl で用いるグローバル変数 */

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 avl_node { datatype data; struct avl_node *left, *right; int height; };

struct avl_node* get_avl() {
  /* ※ここは問題 1 と同じ */
}

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

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

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

void print_avl(struct avl_node *t) {
  if(t==NULL)
    printf(".\n");
  else {
    printf("[%d]%d,%s,%d\n",t->height,t->data.id,t->data.name,t->data.score);
    print_avl(t->left);
    print_avl(t->right);
  }
}

int main() {
  /* 厳密にいえば入力は AVL 木とは限らないが,関数 get_avl で入力の木を作れる */
  struct avl_node *t = get_avl();
  t = balance(t);
  print_avl(t);
```

```
40    return 0;
41 }
```

問題 3

AVL 木に対してデータを挿入する関数 avl_insert を定義せよ．関数 avl_insert の引数と返り値の型は以下の通りである．

```
struct avl_node* avl_insert(struct avl_node *t, struct student d);
```

この関数は,「構造体 avl_node のアドレス t の指す節点を根とする AVL 木に，構造体 student の値 d を挿入して AVL 木の条件を満たすように調節し，挿入後の根の節点のアドレスを返す関数」である．引数 t の指す節点を根とする木は，AVL 木の条件を満たしており，各節点には適切な高さが格納されているものとし，挿入する d と同じ学籍番号の学生は AVL 木には存在しないものと仮定してよい．平衡を考えない二分探索木に対する挿入とは異なり，根の節点が変わることがあるため，返り値の型は void 型ではなく struct avl_node*型になっている．

平衡を保ちつつ AVL 木に挿入するアルゴリズムは再帰的に行うことができる．

1. t が葉 (NULL) かどうかによって場合分けを行う．
   - t が葉なら節点のメモリを確保し[6]，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，高さを 1 として，この節点のアドレスを返す．
   - t が葉ではない節点ならそのデータの学籍番号と d の学籍番号と比較する．
     - d の学籍番号の方が小さければ，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 の学籍番号の方が大きければ，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. t の指す節点のメンバ height を適切に更新する．
3. 挿入によって AVL 木の条件を満たさなくなっていたら,「回転によって調節」して AVL 木の条件を満たす形にしたのち，その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 avl_insert を使うため，この関数は再帰関数として定義される.「回転によって調節」は問題 2 で定義した関数 balance を用いればよい．

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 avl_insert 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 AVL 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

```
10789,Cristiano Ronaldo,7
10456,David Beckham,77
```

出力例

```
[2]10789,Cristiano Ronaldo,7
[1]10456,David Beckham,77
.
.
.
```

[6] (struct avl_node*)malloc(sizeof(struct avl_node)) で構造体 avl_node に必要なメモリを確保できる (get_avl() で既出であるが).

入力例

```
10789,Cristiano Ronaldo,7
10456,David Beckham,77
10123,Ichiro Suzuki,51
10345,Shohei Ohtani,17
10234,Makoto Hasebe,17
```

出力例

```
[3]10456,David Beckham,77
[2]10234,Makoto Hasebe,17
[1]10123,Ichiro Suzuki,51
.
.
[1]10345,Shohei Ohtani,17
.
.
[1]10789,Cristiano Ronaldo,7
.
.
```

入力例

```
10012,Hideki Matsui,55
10123,Ichiro Suzuki,51
10789,Cristiano Ronaldo,7
10234,Makoto Hasebe,17
10456,David Beckham,77
10345,Shohei Ohtani,17
```

出力例

```
[3]10234,Makoto Hasebe,17
[2]10123,Ichiro Suzuki,51
[1]10012,Hideki Matsui,55
.
.
.
[2]10456,David Beckham,77
[1]10345,Shohei Ohtani,17
.
.
[1]10789,Cristiano Ronaldo,7
.
.
```

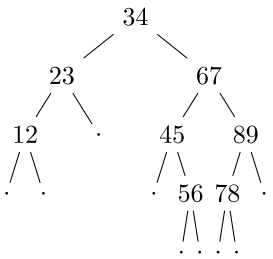
作成すべきプログラムは以下の形式とする．※印を含むコメント部分を適切に書き換えればよいが，必要に応じて補助関数を定義してもよい．

```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 */
struct avl_node { datatype data; struct avl_node *left, *right; int height; };

struct avl_node* rotate_right(struct avl_node *t) {
  /* ※ここは問題 1 と同じ */
}

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

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

void print_avl(struct avl_node *t) {
  if(t==NULL)
    printf(".\n");
  else {
    printf("[%d]%d,%s,%d\n",t->height,t->data.id,t->data.name,t->data.score);
```

```
26      print_avl(t->left);
27      print_avl(t->right);
28    }
29  }
30
31  int main() {
32    struct student st;
33    struct avl_node *t = NULL;
34    while(fgets(buf,sizeof(buf),stdin)!=NULL) {
35      /* 学生の情報を読取り */
36      sscanf(buf,"%d,%[^,],%d",&st.id,st.name,&st.score);
37      /* AVL 木にそのデータを追加 */
38      t = avl_insert(t, st);
39    }
40    print_avl(t);
41    return 0;
42  }
```

ヒント　どの挿入で失敗しているかわからないときは，38 行めの直後にも `print_avl(t);` と書いて木を出力し動作確認してみるとよい (行きがけ順が見づらければ前回 (第 11 回) の問題 1 で作成したような木を表示する `print_tree` 関数を作ってもよい).

AVL 木におけるデータの削除は，通常の二分探索木に対するデータの削除と同様にやや煩雑である．二分探索木の中の節点を削除する際，その節点の左右の子のどちらかが葉であるなら，葉でない方の部分木で置き換えればよいが，左右の子ともに葉でない場合はどうすればいいだろうか．二分探索木においてそのような節点が持つ値の前後の値をもつ節点はどこにあるか想像してみよう．二分探索木では通りがけ順に並べると昇順に並んでいるため，根の節点の直前の値を持つ節点は左の部分木の最も右の子孫 (最大値) であり，根の節点の直後の値を持つ節点は右の部分木の最も左の子孫 (最小値) である．つまり，直前または直後の接点を根に持ってくることができれば，根のみを削除した二分探索木を作ることができる．そこで，まず補助関数として，最大値を削除する関数を実装してみよう．



問題 4

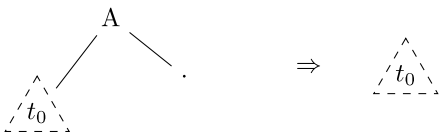
AVL 木に対して最大値を削除し，削除後の AVL 木の節点を返す関数 `delete_max` を定義せよ．関数 `delete_max` の引数と返り値の型は以下の通りである．

```
struct avl_node* delete_max(struct avl_node *t, struct avl_node **p);
```

この関数は,「構造体 `avl_node` のアドレス `t` の指す節点を根とする二分探索木から，最大値をもつ節点をその左の部分木で置き換え，削除した節点のアドレスを `p` の指すアドレスに格納し，削除後の AVL 木の根の節点のアドレスを返す関数」である．`t` 自身は葉 (`NULL`) でないと仮定してよい．二分探索木において最大値をもつ節点とは，葉でない最も右の節点，すなわち，右の子の右の子の，...，と繰り返して葉にたどり着く手前の節点のことである．

この操作も次のように再帰的なアルゴリズムによって実現できる．引数 `t` の表す木は葉ではないと仮定している (つまり，右の部分木が存在する).

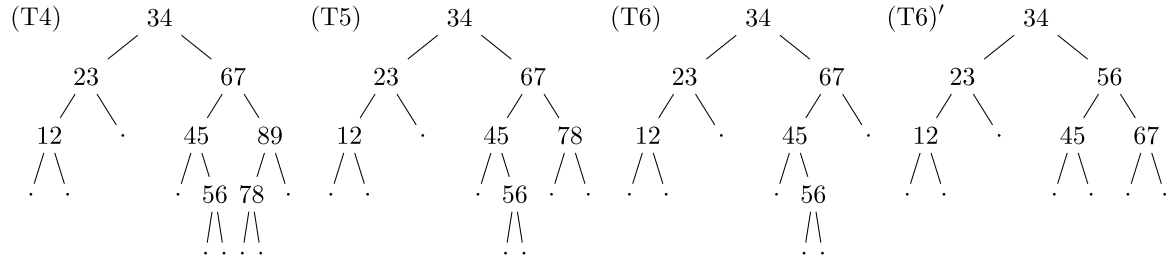
- 右の部分木が葉であるときは，その根の節点 (A) のアドレスを `*p` に入れ，左の部分木を返す.



- 右の部分木が葉でないときは,
  1. 右の部分木を「右の部分木から最大値を削除した木」で置き換える.
  2. 削除によって `t` の高さが変わる可能性があるので，メンバ `height` を適切に更新する.
  3. 削除によって AVL 木の条件が崩れている可能性があるので,「回転して調節」を行って AVL 木を作成し，その根の節点のアドレスを返す.

下線の「右の部分木から最大値を削除した木」を計算するために関数 `delete_max` を使うため，この関数は再帰関数として定義される.「回転によって調節」は問題 2 で定義した関数 `balance` を用いればよい.

たとえば，次の AVL 木 (T4) を引数に与えた場合，最も右の節点 (最大値 89 をキーとする節点) をその左の部分木 (78 をキーとする節点を根とする木) で置き換えることで，(T5) のような AVL 木が得られ，木から削除した 89 をキーとする節点が `*p` に格納される．AVL 木 (T5) を引数に与えた場合，最も右の節点 (最大値 78 をキーとする節点) をその左の部分木 (葉) で置き換えて (T6) のような二分探索木が得られるが，削除によって AVL の条件を満たさなくなるため，(T6)′ のような AVL 木に変更される．このとき木から削除した 78 をキーとする節点が `*p` に格納される.



作成すべきプログラムの形式は後に示す `main` 関数を含むものとし，関数 `delete_max` 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，問題 1 と同様の形式で二分探索木を表したものである．標準出力には，上のアルゴリズムを適用して最大値を削除した後の AVL 木を同じ形式で出力する．入力は葉ではない AVL 木であり，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

```
[2]10345,Shohei Ohtani,17
[1]10123,Ichiro Suzuki,51
.
.
[1]10567,Yuto Nagatomo,55
.
.
```

出力例

```
10567,Yuto Nagatomo,55
[2]10345,Shohei Ohtani,17
[1]10123,Ichiro Suzuki,51
.
.
.
```

入力例

```
[4]10345,Shohei Ohtani,17
[2]10234,Makoto Hasebe,17
[1]10123,Ichiro Suzuki,51
.
.
.
[3]10678,Shinji Kagawa,23
[2]10456,David Beckham,77
.
[1]10567,Yuto Nagatomo,55
.
.
[2]10890,Yukio Mishima,100
[1]10789,Cristiano Ronaldo,7
.
.
.
```

出力例

```
10890,Yukio Mishima,100
[4]10345,Shohei Ohtani,17
[2]10234,Makoto Hasebe,17
[1]10123,Ichiro Suzuki,51
.
.
.
[3]10678,Shinji Kagawa,23
[2]10456,David Beckham,77
.
[1]10567,Yuto Nagatomo,55
.
.
[1]10789,Cristiano Ronaldo,7
.
.
```

入力例

```
[4]10345,Shohei Ohtani,17
[2]10234,Makoto Hasebe,17
[1]10123,Ichiro Suzuki,51
.
.
.
[3]10678,Shinji Kagawa,23
[2]10456,David Beckham,77
.
[1]10567,Yuto Nagatomo,55
.
.
[1]10789,Cristiano Ronaldo,7
.
.
```

出力例

```
10789,Cristiano Ronaldo,7
[3]10345,Shohei Ohtani,17
[2]10234,Makoto Hasebe,17
[1]10123,Ichiro Suzuki,51
.
.
.
[2]10567,Yuto Nagatomo,55
[1]10456,David Beckham,77
.
.
[1]10678,Shinji Kagawa,23
.
.
```

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]; /* 関数 get_avl で用いるグローバル変数 */

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 avl_node { datatype data; struct avl_node *left, *right; int height; };

struct avl_node* get_avl() {
  /* ※ここは問題 2 と同じ */
}

struct avl_node* rotate_right(struct avl_node *t) {
  /* ※ここも問題 2 と同じ */
}

struct avl_node* rotate_left(struct avl_node *t) {
```

```
18   /* ※ここも問題2と同じ */
19 }
20
21 struct avl_node* balance(struct avl_node *t) {
22   /* ※ここも問題2と同じ */
23 }
24
25 struct avl_node* delete_max(struct avl_node *t, struct avl_node **p) {
26   /* ※ここを適切なプログラムで埋める */
27 }
28
29 void print_avl(struct avl_node *t) {
30   if(t==NULL)
31     printf(".\n");
32   else {
33     printf("[%d]%d,%s,%d\n",t->height,t->data.id,t->data.name,t->data.score);
34     print_avl(t->left);
35     print_avl(t->right);
36   }
37 }
38
39 int main() {
40   struct avl_node *t = get_avl(), *u;
41   t = delete_max(t, &u);
42   /* u に格納された節点 (最大の学籍番号を含む) の学生の情報を表示 */
43   printf("%d,%s,%d\n",u->data.id,u->data.name,u->data.score);
44   print_avl(t);
45   return 0;
46 }
```

問題 5

AVL 木に対してデータを削除する関数 `avl_delete` を定義せよ．関数 `avl_delete` の引数と返り値の型は以下の通りである．

```
struct avl_node* avl_delete(struct avl_node *t, int id);
```

この関数は,「構造体 `avl_node` のアドレス `t` の指す節点を根とする AVL 木から，学籍番号が `id` である節点を削除して AVL 木の条件を満たすように調節し，削除後の根の節点のアドレスを返す関数」である．学籍番号が `id` である節点がないときは `t` をそのまま返すものとし，元の AVL 木には同じ学籍番号をもつ節点は 1 つしかないと仮定してよい．

平衡を保ちつつ AVL 木から節点を削除する操作は，挿入と同様に再帰的なアルゴリズムで実現することができる．

- `t` が葉 (`NULL`) のときは削除すべき節点がなかったということなので，そのまま `NULL` を返す．
- `t` が葉でないときは，

1. – `id` が `t` の根の節点の学生の学籍番号より小さいとき，左の部分木を「左の部分木から `id` の学生を削除した木」で置き換える．
   – `id` が `t` の根の節点の学生の学籍番号より大きいとき，右の部分木を「右の部分木から `id` の学生を削除した木」で置き換える．
   – それ以外 (`id` が `t` の根の節点の学生の学籍番号と等しい) で左の部分木が葉のとき，右の部分木を返す．
   – それ以外で右の部分木が葉のとき，左の部分木を返す．

  - それ以外の (左右の部分木が葉ではない) とき,「左の部分木から最大値の節点を削除した木」を t_L とし，削除した節点を根とし，左の部分木を t_L，右の部分木を `t` の右の部分木とするような木を `t` とする.
2. 削除によって高さが変わっている可能性があるので，`t` のメンバ `height` を適切に更新する.
3. 削除によって AVL 木の条件が崩れている可能性があるので,「回転して調節」を行って AVL 木を作成し，その根の節点のアドレスを返す.

下線の「右の部分木から `id` の学生を削除した木」を計算するために関数 `avl_delete` を使うため，この関数は再帰関数として定義される.「左の部分木から最大値の節点を削除した木」は問題 4 で定義した関数 `delete_max` を用い,「回転によって調節」は問題 2 で定義した関数 `balance` を用いればよい.

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 `avl_delete` 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，最後の行以外は問題 1 と同様の形式で二分探索木を表したものであり，最後の行には削除すべき学籍番号 (5 桁) が整数値で入力される．標準出力には，上のアルゴリズムを適用して該当する学籍番号の節点を削除した後の AVL 木を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

```
[4]10567,Yuto Nagatomo,55
[3]10234,Makoto Hasebe,17
[1]10123,Ichiro Suzuki,51
.
.
[2]10345,Shohei Ohtani,17
.
[1]10456,David Beckham,77
.
.
[2]10678,Shinji Kagawa,23
.
[1]10789,Cristiano Ronaldo,7
.
.
10345
```

出力例

```
[3]10567,Yuto Nagatomo,55
[2]10234,Makoto Hasebe,17
[1]10123,Ichiro Suzuki,51
.
.
[1]10456,David Beckham,77
.
.
[2]10678,Shinji Kagawa,23
.
[1]10789,Cristiano Ronaldo,7
.
.
```

入力例

```
[4]10567,Yuto Nagatomo,55
[3]10234,Makoto Hasebe,17
[1]10123,Ichiro Suzuki,51
.
.
[2]10345,Shohei Ohtani,17
.
[1]10456,David Beckham,77
.
.
[2]10678,Shinji Kagawa,23
.
[1]10789,Cristiano Ronaldo,7
.
.
10123
```

出力例

```
[3]10567,Yuto Nagatomo,55
[2]10345,Shohei Ohtani,17
[1]10234,Makoto Hasebe,17
.
.
[1]10456,David Beckham,77
.
.
[2]10678,Shinji Kagawa,23
.
[1]10789,Cristiano Ronaldo,7
.
.
```

入力例

```
[4]10567,Yuto Nagatomo,55
[3]10234,Makoto Hasebe,17
[1]10123,Ichiro Suzuki,51
.
.
[2]10345,Shohei Ohtani,17
.
[1]10456,David Beckham,77
.
.
[2]10678,Shinji Kagawa,23
.
[1]10789,Cristiano Ronaldo,7
.
.
10567
```

出力例

```
[3]10456,David Beckham,77
[2]10234,Makoto Hasebe,17
[1]10123,Ichiro Suzuki,51
.
.
[1]10345,Shohei Ohtani,17
.
.
[2]10678,Shinji Kagawa,23
.
[1]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ること．

```c
#include<stdio.h>
#include<stdlib.h>
char buf[128]; /* 関数 get_avl で用いるグローバル変数 */

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 avl_node { datatype data; struct avl_node *left, *right; int height; };

struct avl_node* get_avl() {
  /* ※ここは問題４と同じ */
}

struct avl_node* rotate_right(struct avl_node *t) {
  /* ※ここも問題４と同じ */
}

struct avl_node* rotate_left(struct avl_node *t) {
  /* ※ここも問題４と同じ */
}

struct avl_node* balance(struct avl_node *t) {
  /* ※ここも問題４と同じ */
}

struct avl_node* delete_max(struct avl_node *t, struct avl_node **p) {
  /* ※ここも問題４と同じ */
}

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

void print_avl(struct avl_node *t) {
  if(t==NULL)
    printf(".\n");
  else {
    printf("[%d]%d,%s,%d\n",t->height,t->data.id,t->data.name,t->data.score);
```

```
38      print_avl(t->left);
39      print_avl(t->right);
40    }
41  }
42
43  int main() {
44    struct avl_node *t = get_avl();
45    int id;
46    scanf("%d␣", &id);
47    t = avl_delete(t, id);
48    print_avl(t);
49    return 0;
50  }
```

ヒント　入出例の AVL 木はこの課題の中で示した (T2) の AVL 木と同じ形をしているので，確認したいときは参考にするとよい (10234 なら 23，10567 なら 56 のように対応している).