データ構造とプログラミング技法（第８回）

ーデータの探索ー

データ探索

- レコードの集合(ファイル)から必要なレコードを探し出す処理.
  - フィールド数：単一、複数
  - 質問形式：一致型、区間型、最近接型、条件指定型
  - ファイル数：単一ファイル、複数ファイル

- 単純一致型：単一ファイルから、単一のフィールドの値を指定して、一致を条件としてレコードを探索する。

探索の方法

表探索: 順配置された線形リスト(表)からの探索

- 線形探索
- 二分探索
- ハッシュ法

木構造探索: ファイルが表ではなく木構造によって表現されている場合の探索

- 二分探索木
  - 平衡木／最適木
  - AVL木
- B木
- 桁探索木
  - トライ／パトリシア木

線形探索

1. データが極端に少ない場合
2. 格納の方が探索よりも圧倒的に多い場合
3. 新しく追加したキーへの探索要求のほうが頻度が高い場合

```
int linear_search(int K, int n)
{    int i;
     a[0] = K;
     for (i=n; ; i--) if (a[i] == K) break;
     return i;
}
```

図 8·1　線形探索

格納：*O(1)*，探索：*O(n)*

二分探索

整列化並びに対する探索法

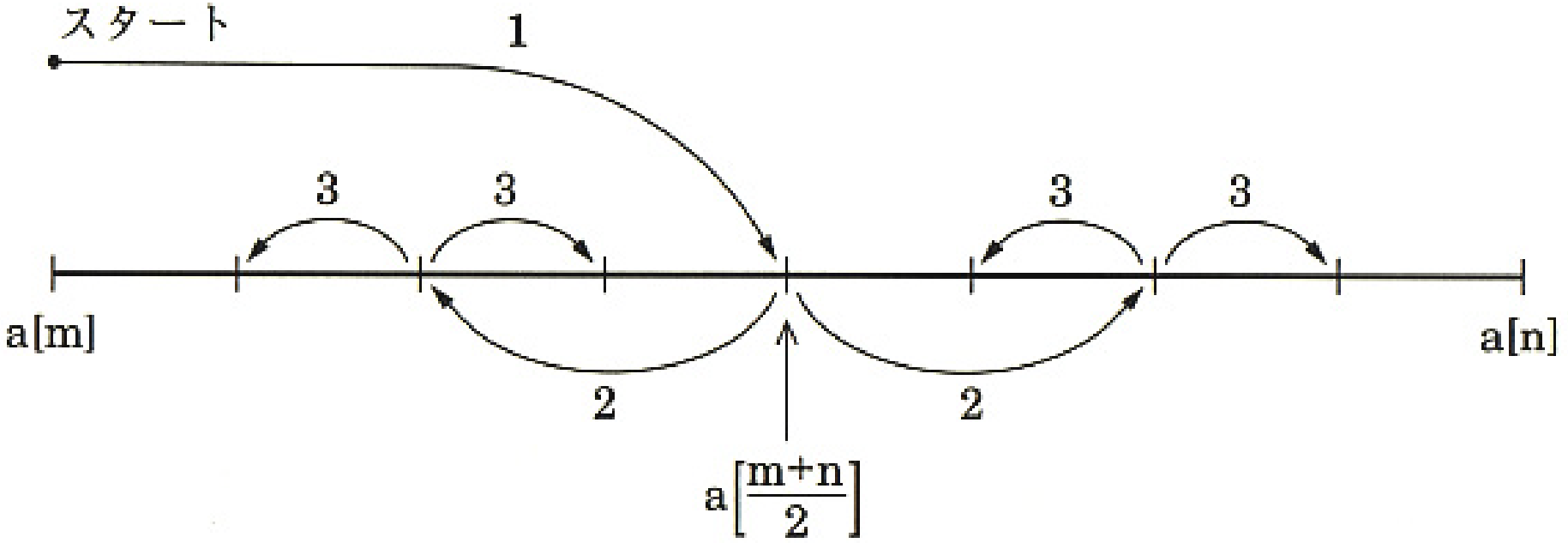


図 8·2　二分探索の探索過程

二分探索のアルゴリズム

```
int binary_search(int K, int m, int n)
{    int i,j,p;
     i = m; j = n;
     while (i <= j) {
          p = (i+j)/2;
          if (K < a[p]) j = p-1;
          else if (K > a[p]) i = p+1;
          else if (K == a[p]) return p;
     }
     return -1;
}
```

図 8·3　二分探索

探索： *O(log n)*

ハッシュ法

キー-番地変換

Suzuki

Sは19番目のアルファベット

h_0(Suzuki)=19 mod 9 =1

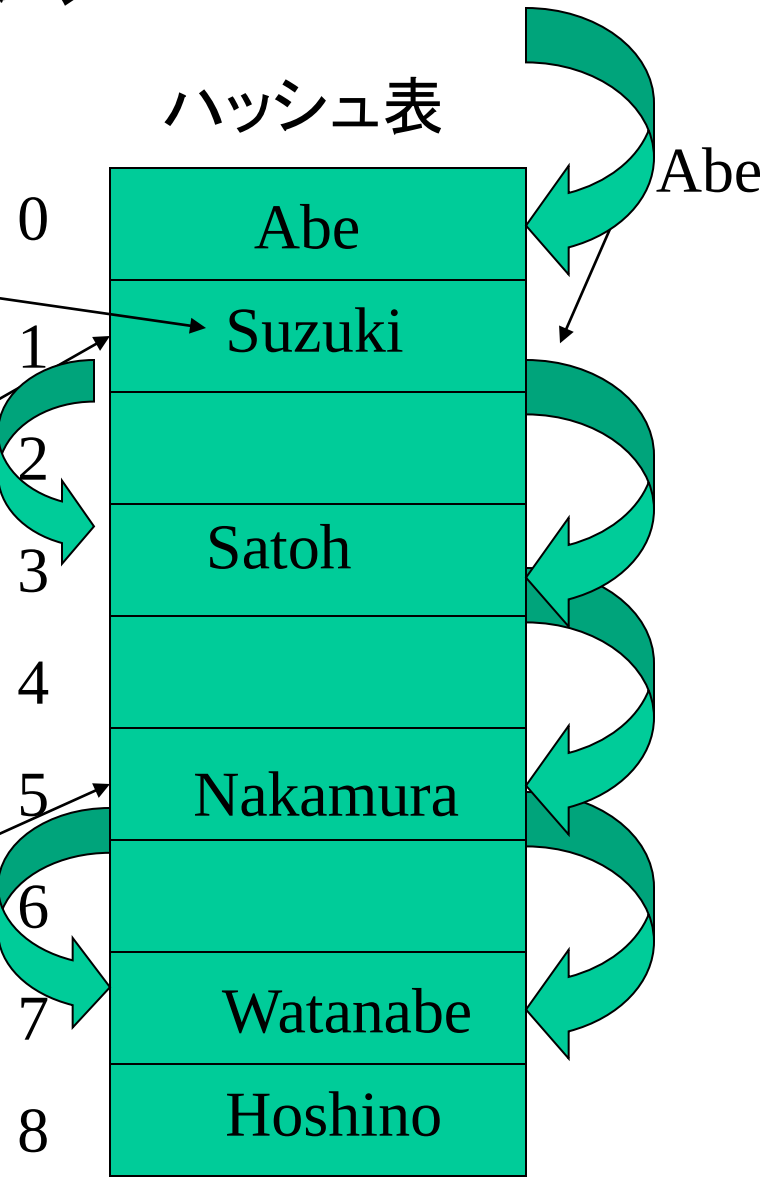
Satoh

衝突処理

$h_i(K) = [\, h_0(K) + d \times i] \bmod M$

（線形走査法:d=2,M=9）

Watanabe



クラスタ

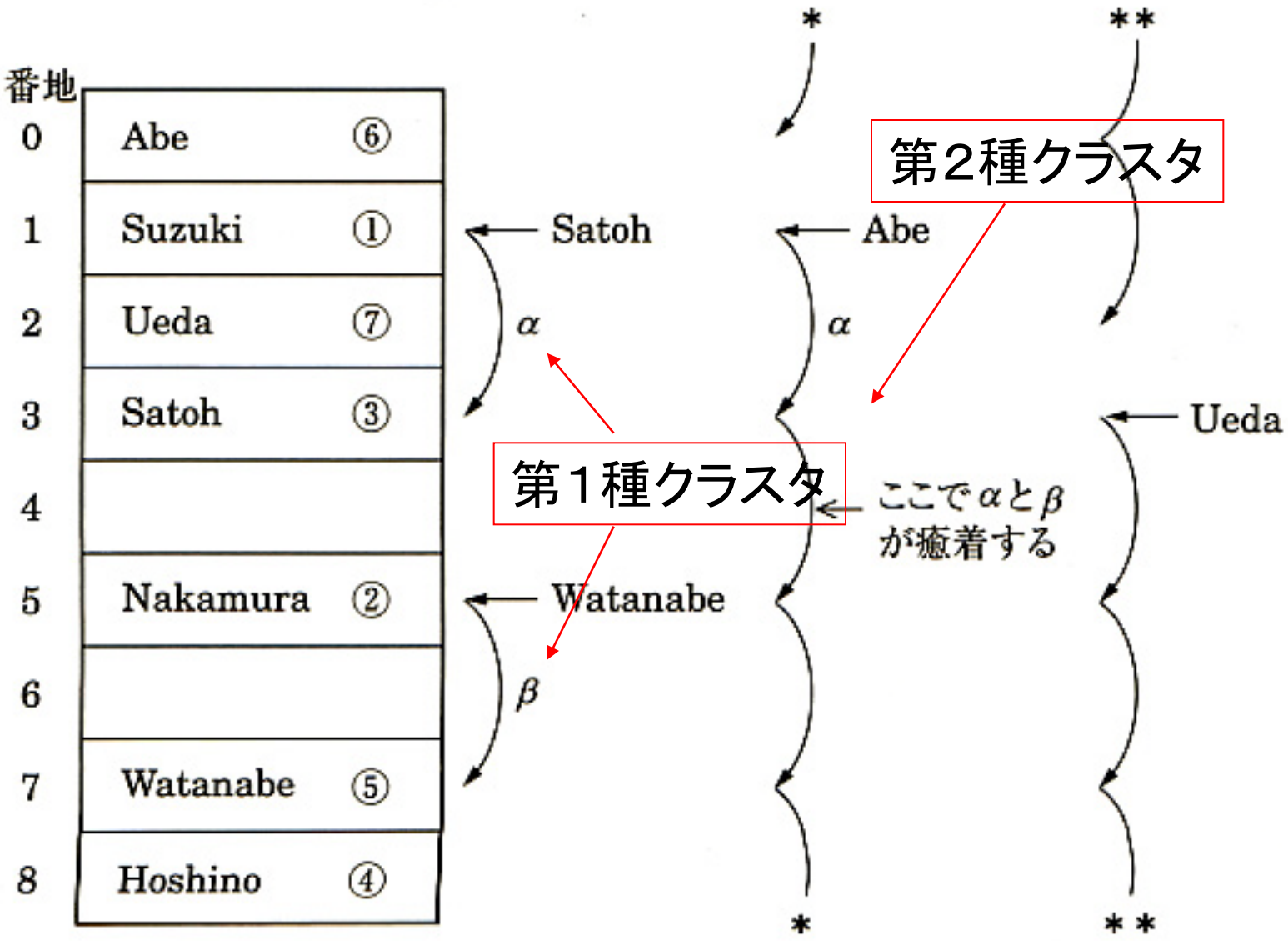


図 8·4　ハッシュ表

ハッシュ関数$h_0(K)$のいろいろ

除算法
平方採中法
折り返し法
桁解析法
etc

結局は問題依存なので、決め手になる方法はないと考えるのが無難。

衝突処理

- 開番地法
  - 線形走査法 : $h_i(K) = [h_0(K) + d \times i] \bmod M$
    (dとMが互いに素ならば、探索周期は全表的)
  - 2次走査法 : $h^i(K) = [h^0(K) + a \times i + b \times i^2] \bmod M$
    (第1種クラスタの発生を押さえることができる。一般に探索周期は全表的ではない。)
  - 2重分散法 : $h_i(K) = [h_0(K) + (2g(K)+1) \times i] \bmod M$
    (第1種2種クラスタの発生を押さえることができる。
    $M = 2^m$、$0 \leqq g(K) \leqq 2^{m-1}$ならば探索周期は全表的)
- 連鎖法
  - 連合連鎖法・分離連鎖法

2次走査法は思ったほど難しくはない

```
#include    <stdio.h>
#include    <string.h>
#define     M    300

char *tab[M];

int hash0(char *K) { 未定義 }

void quadratic_store(char *K, int a, int b)
{   int h,u,v;
    h = hash0(K);
    u = a+b; v = 2*b;
    while (tab[h] != NULL) {h = (h+u)%M; u = u+v;}
    tab[h] = K;
}
```

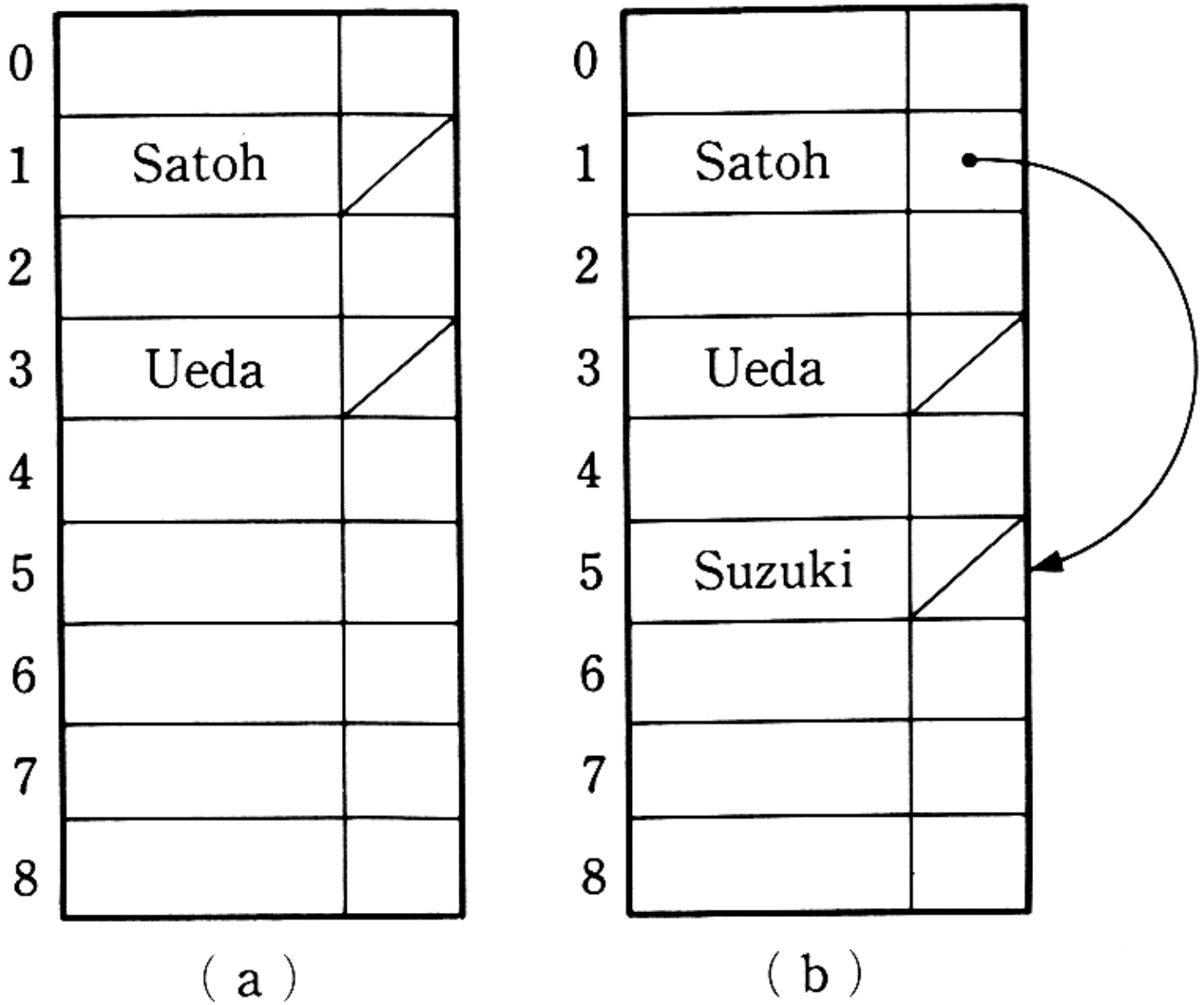
(a)

```
int quadratic_search(char *K, int a, int b)
{   int h,u,v;
    h = hash0(K);
    u = a+b; v = 2*b;
    while (tab[h] != NULL)
        if (strcmp(K, tab[h]) == 0) return h;
        else {h = (h+u)%M; u = u+v;}
    return -1;
}
```

(b)

図 8·5　2 次走査法による開番地ハッシュ法
(a) 格納, (b) 探索

連鎖法の原理



連合／分離　連鎖

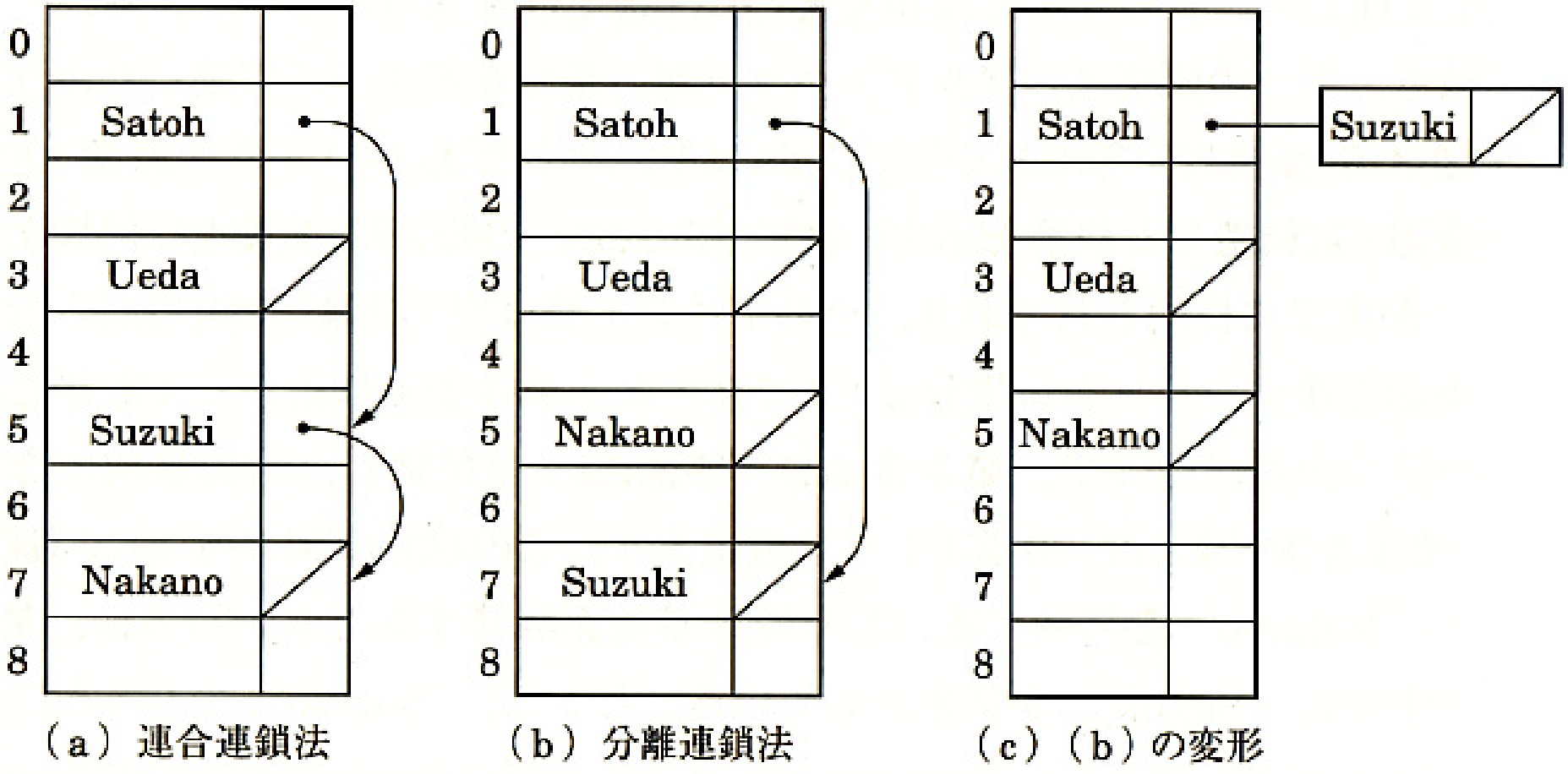


図 8·7　連鎖法の二つのタイプと同族リストを別途持つ方法

平均探査回数(α:表占有率)

表 8·1　成功/不成功探索の平均探査回数

	成功探索 E	不成功探索 E'
線形走査法 (8.1)	$\frac{1-\alpha/2}{1-\alpha}$	$\frac{1}{2}+\frac{1}{2(1-\alpha)^2}$
2 次走査法 (8.2)	$1-\ln(1-\alpha)-\frac{\alpha}{2}$	$\frac{1}{1-\alpha}-\alpha-\ln(1-\alpha)$
2 重分散法 (8.3)	$-\frac{1}{\alpha}\ln(1-\alpha)$	$\frac{1}{1-\alpha}$
連合連鎖法	$1+\frac{\alpha}{4}+\frac{1}{8\alpha}(e^{2\alpha}-1-2\alpha)$	$1+\frac{1}{4}(e^{2\alpha}-1-2\alpha)$
分離連鎖法	$1+\frac{\alpha}{2}$	$\alpha+e^{-\alpha}$

木探索
（データの動的な変更に適する）

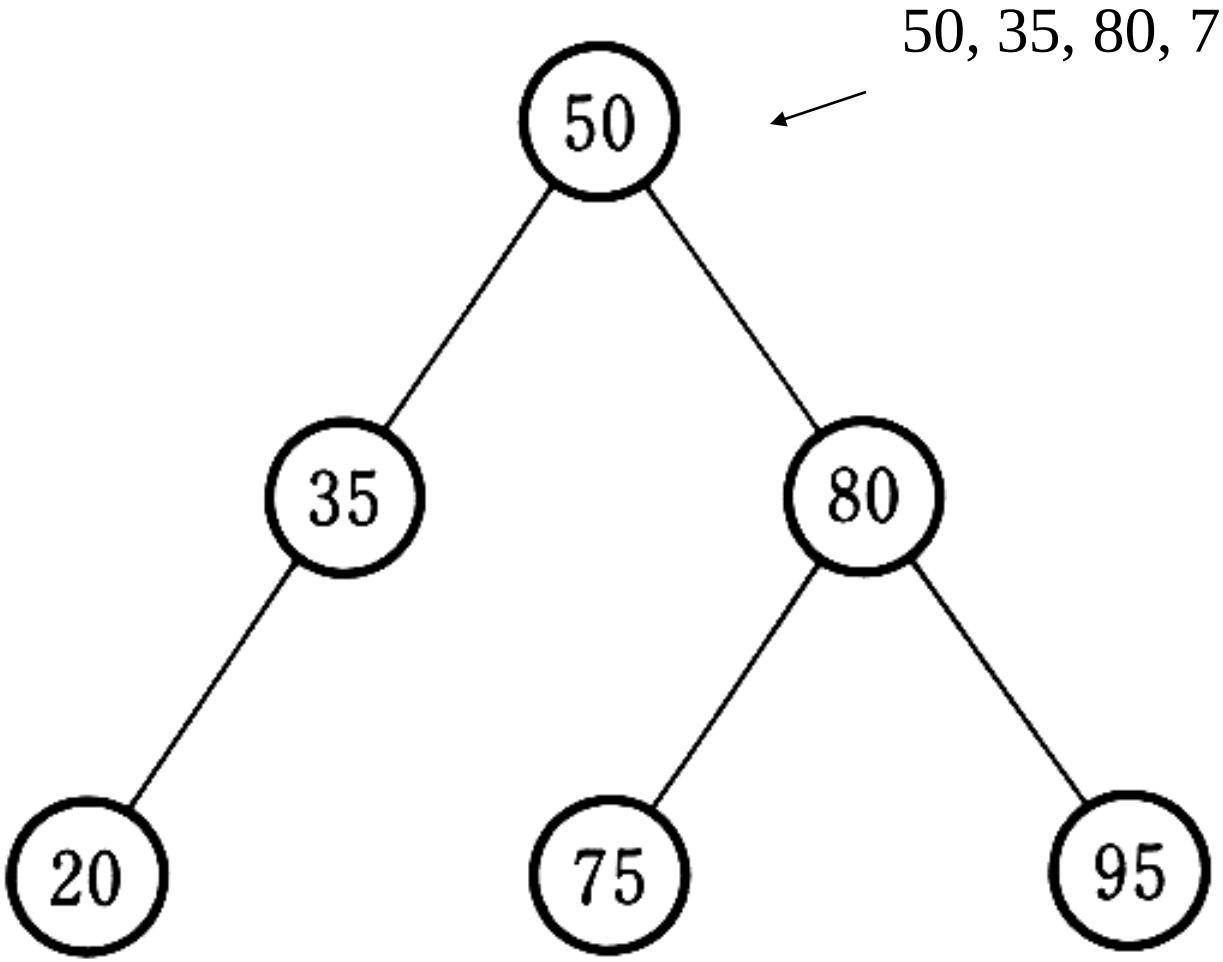


図 8.8　二分探索木

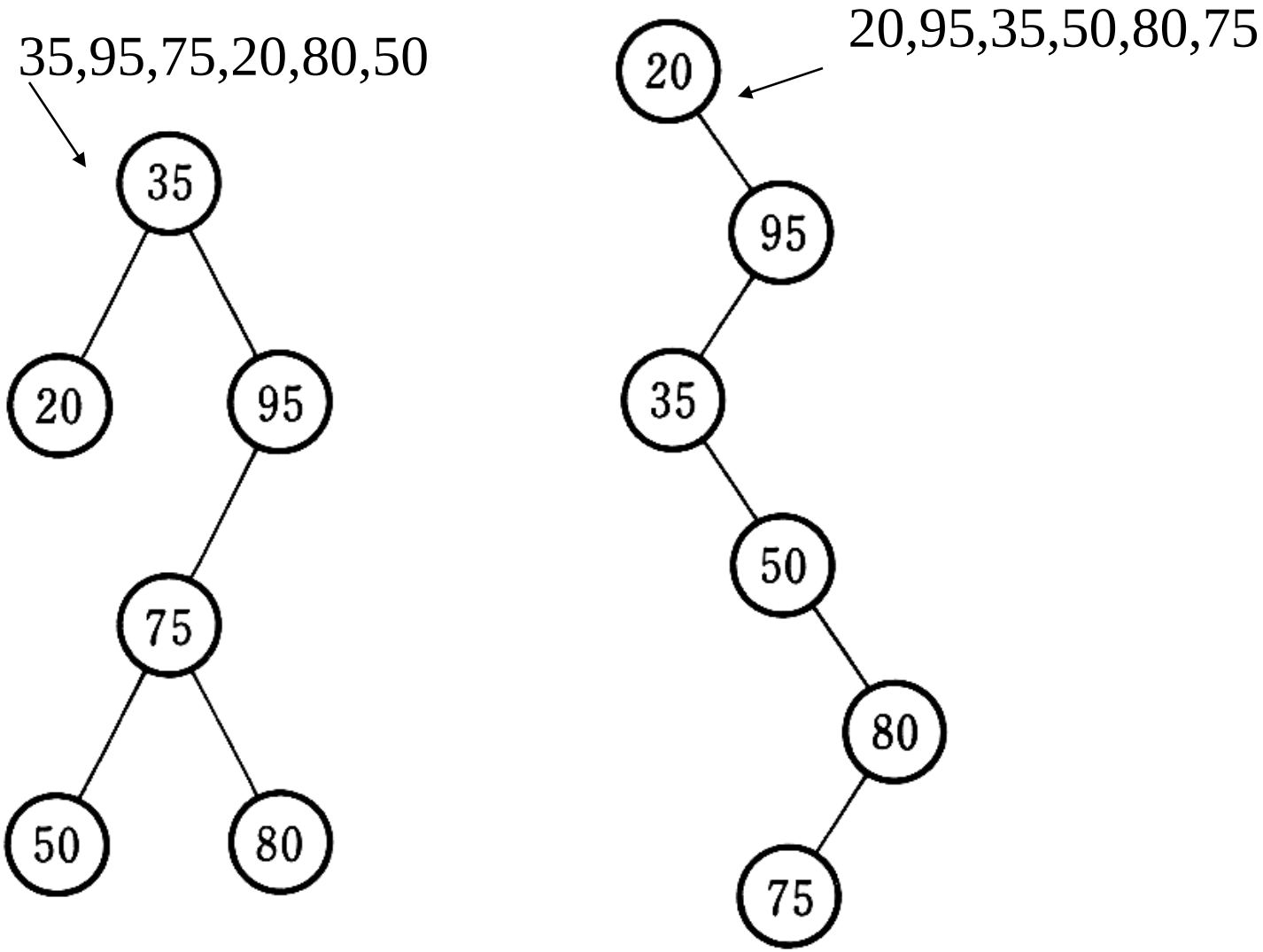
二分探索木の生成

```
procedure GenerateTree(n:integer);
begin
var Root, Candidate, NewNode: NodePointer;
      Root↑.key:=a[1]; Root↑.left:=nil; Root↑.right:=nil;
      for i := 2 to n do begin
        Candidate := SearchKey(Root, a[i]);
        if Candidate↑.key <> a[i] then begin
          NewNode := CreateNode();NewNode↑.key:=a[i];
          NewNode↑.left:=nil; NewNode↑.right:=nil;
          if Candidate↑.key >a[i] then begin
                Candidate↑.left:=NewNode;
          else Candidate↑.right:=NewNode
       end
     end
end;
```

二分探索木の性質

- 親のキーよりも左の子に格納されたキーは小さく，右の子のキーは大きい．
- したがって，中順走査をすれば，ソート列が得られる．

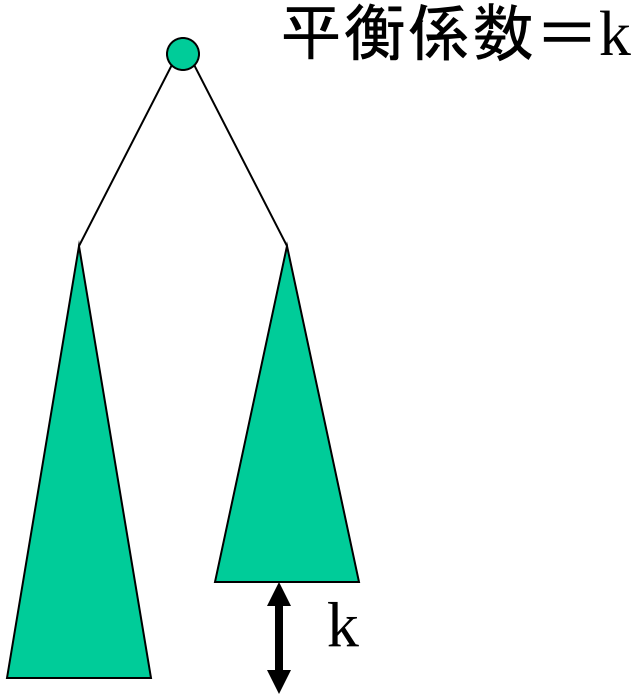
アンバランスな二分探索木



キー列と二分探索木の関係（演習問題：15分）

- 図８．８と同じ木構造を生成するようなキー列は何種類あるか答えなさい．
- レポート用紙に学籍番号と氏名，および上記の問題に対する解答を書き，提出しなさい．

平衡木

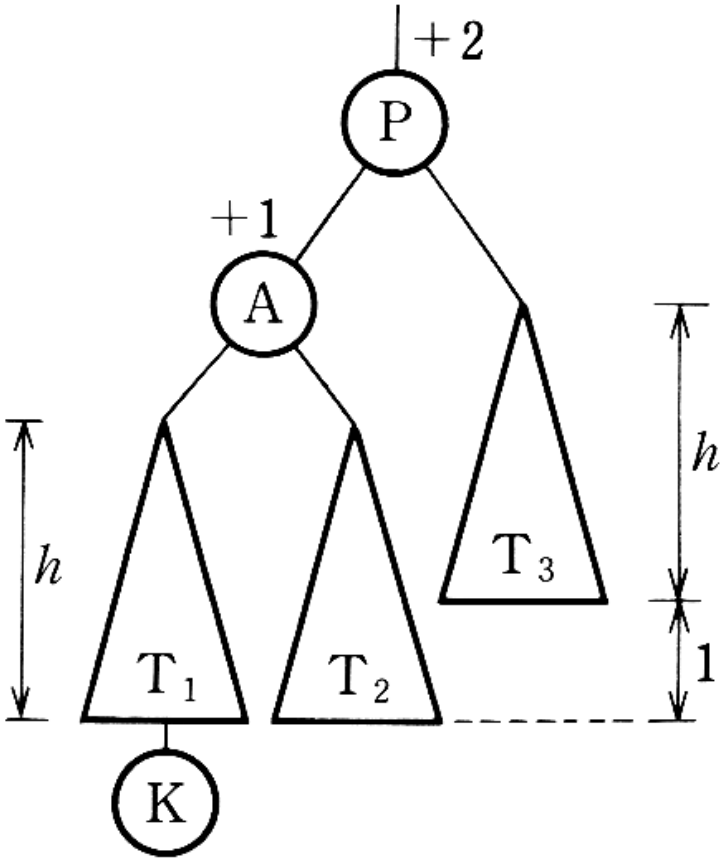


任意の節の平衡係数が
一定の範囲に入っている
二分探索木。

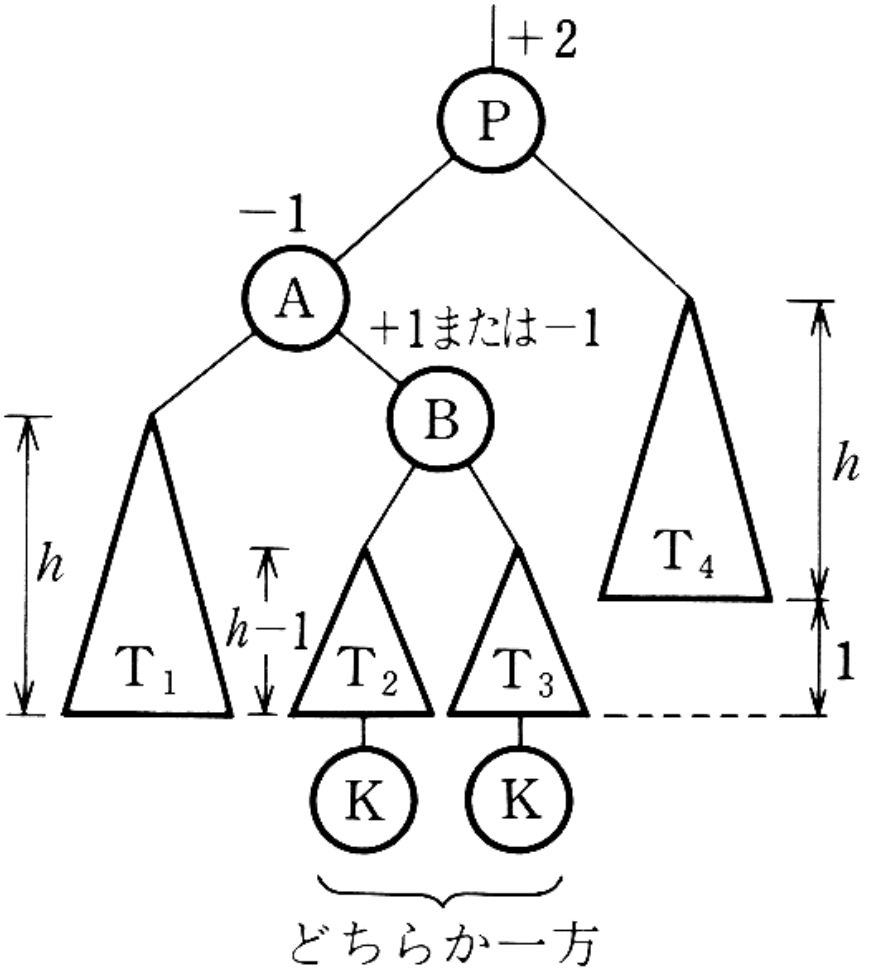
木の再平衡化：AVL木

二分探索木を作る段階で、平衡係数が[-1,1]の範囲を超えたときに、再平衡化を行い、平衡を保つようにして作られる。

平衡が崩れる場合

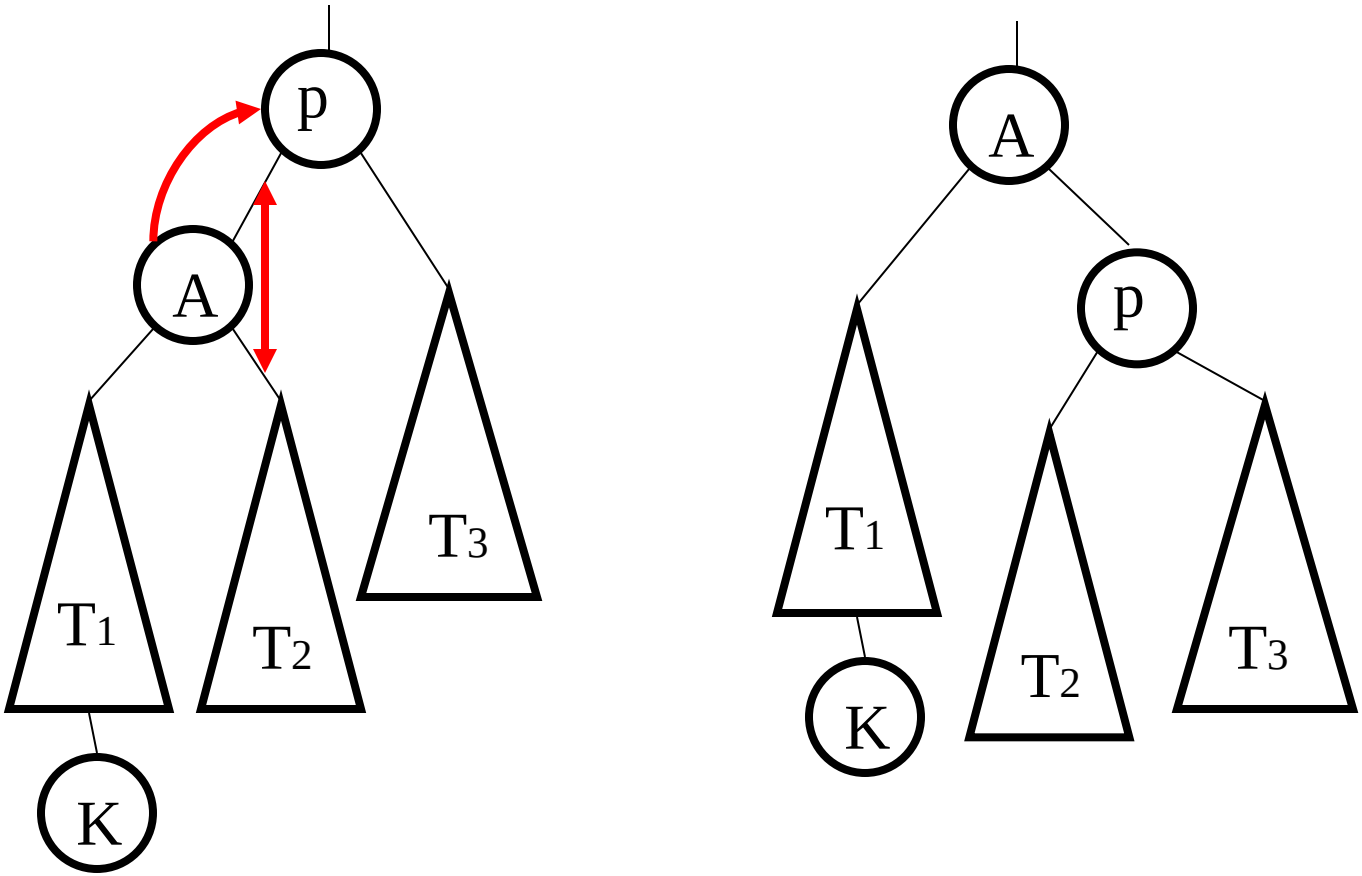


（ａ） Ａの左部分木に追加された場合

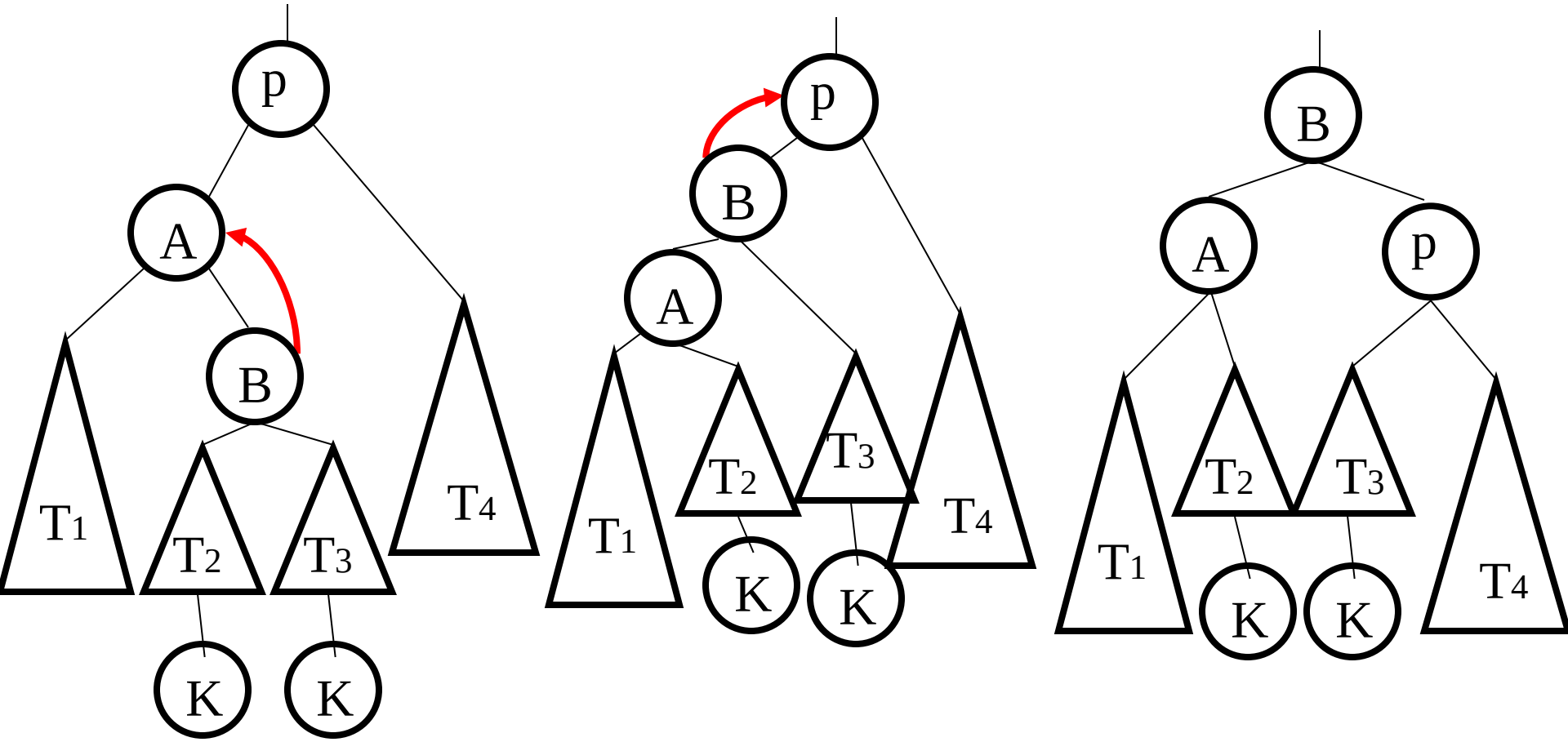


（ｂ） Ａの右部分木に追加された場合

単回転



複回転



得られるAVL木

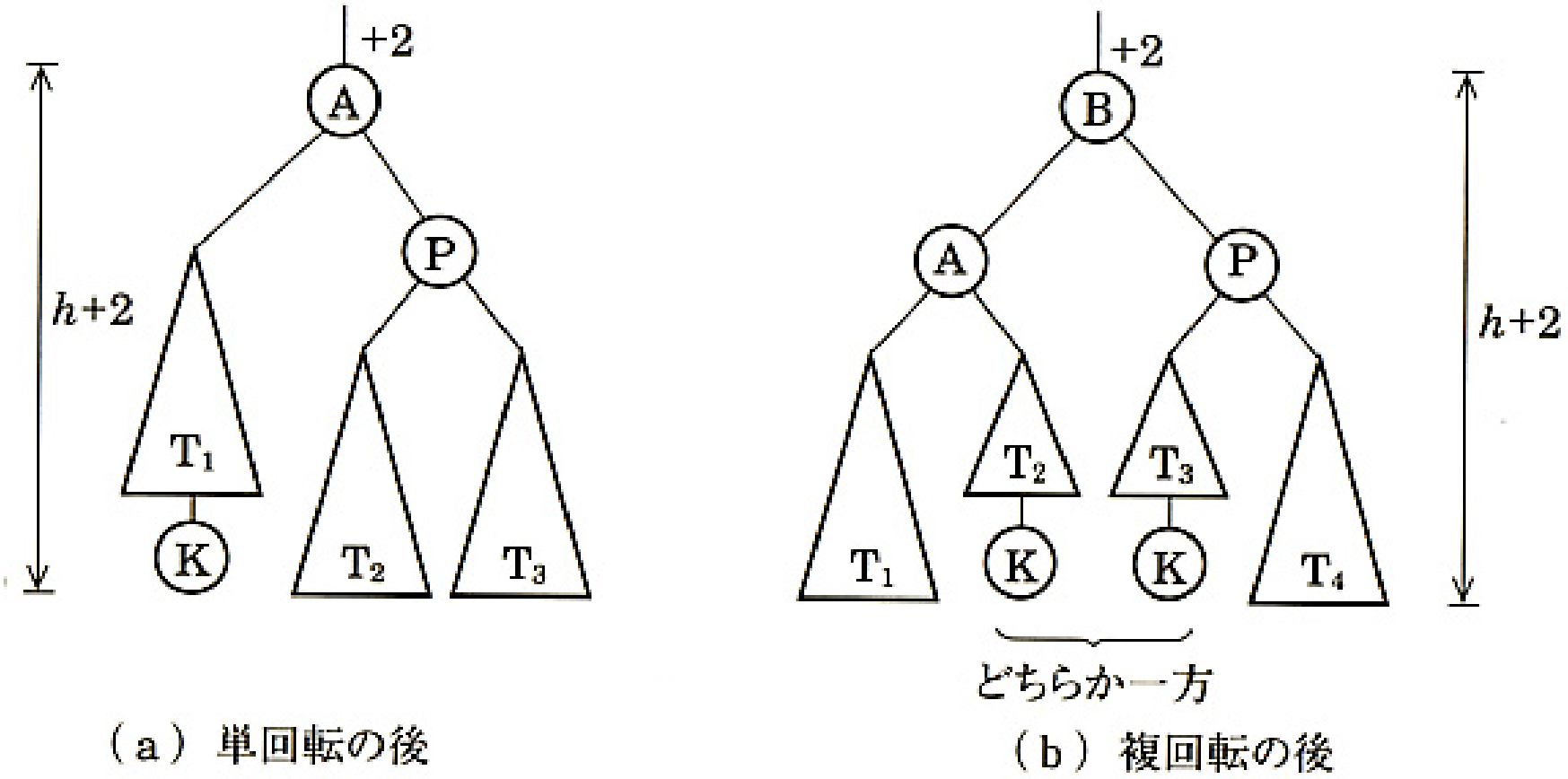


図 8·11　平衡回復した AVL 木

レポート課題

- 図８．９(a)および(b)のデータが与えられたときのＡＶＬ木の生成過程を図示しなさい．