テーマ4：幾何的データ構造

区分木と二重連結辺リスト

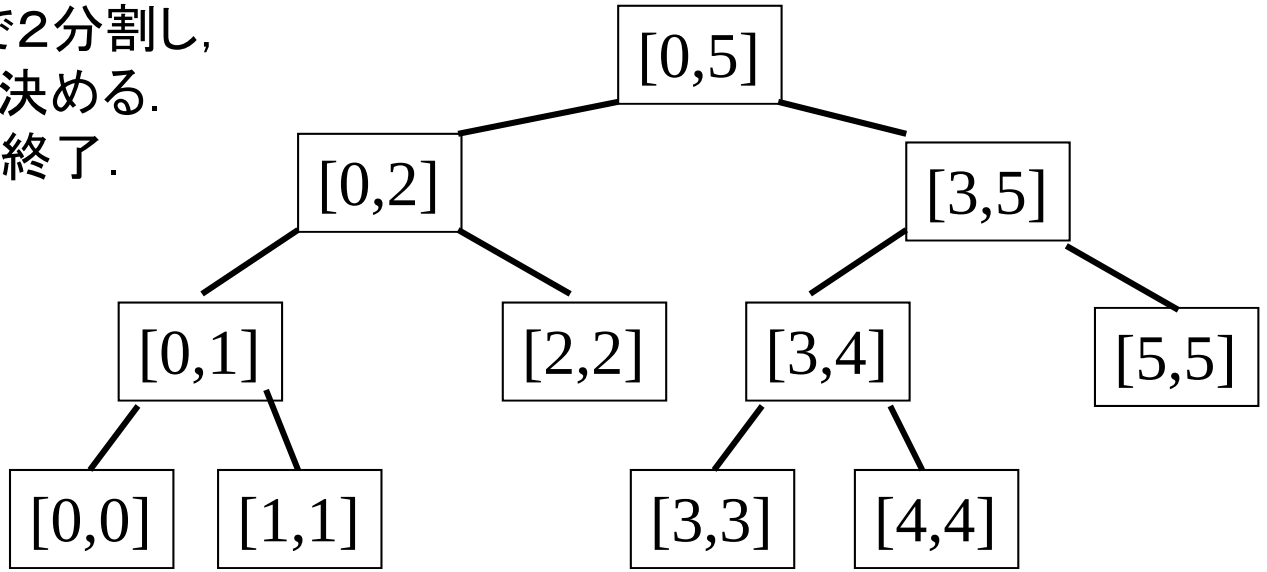
区間木の構造

区間木とは,

区間の集合を効率よく扱うためのデータ構造
全体の区間を[0, N]とするとき，次の操作を実行可能.
(1)新たな区間の登録（区間の挿入）,
(2)登録されている区間の削除,
(3)質問として与えられた値を含むすべての区間の列挙.

区間木の作り方

区間[a, b]を中央で2分割し,
左の子と右の子を決める.
[c,c]の形になれば終了.



演習：全体の区間が[0, 8]のときの区間木を構成せよ.

演習：全体の区間が[2, 10]のときの区間木を構成せよ.

区間木の構造(2)

区間木T(0, N)の定義

(1)区間木T(0, N)の根は，区間[0, N]に対応している.

(2)区間木T(a, b)は，b>aのとき，T(a, 【(a+b)/2】)を左部分木とし，
T(【(a+b)/2】+1, b)を右部分木とする2分木である.

(3)区間木T(a, a)は，区間[a, a]に対応する一つの節点からなる木である.

ただし，【x】はxの整数部分を表すものとする.

区間木T(a, b)の深さ(根から葉節点までの辺数＋1)をL(a, b)とすると，

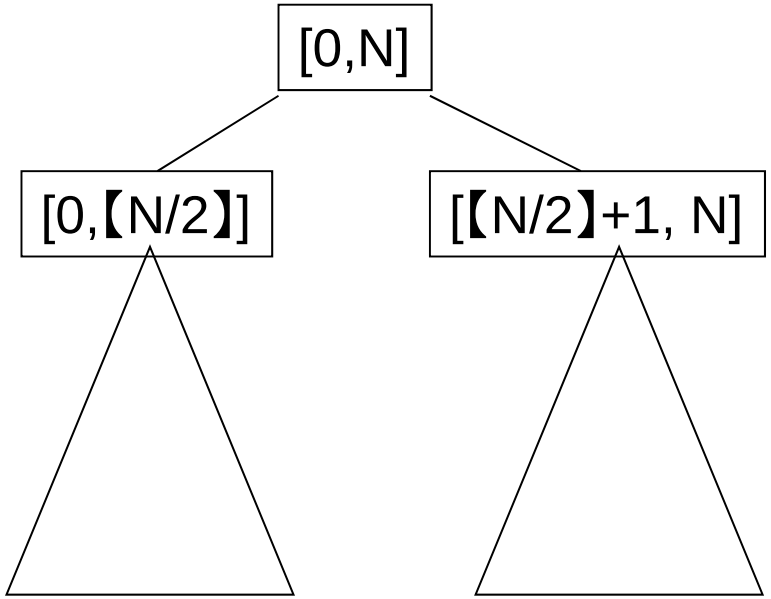
$L(0, N) \leqq L(0, [[(N+1)/2]]) + 1$

$L(0, 0) = 1$

であるから，

$L(0, N) = O(\log N)$

を得る.



演習 : 区間木 T(0, L) の深さがO(log N) であることを証明せよ.

区間木の構造(2)

性質： 区間木T(0, N)は高々2N+1 個の節点を持つ.

証明:

P(0, N)をT(0, N)の節点数とする.

定義より，次式を得る.

$$P(0,N) = P(0,\lfloor N/2 \rfloor) + P(\lfloor N/2 \rfloor + 1, N) + 1$$

区間$[0,\lfloor N/2 \rfloor]$と$[\lfloor N/2 \rfloor + 1, N]$は共通部分を持たないから,

$$P(0,N) \le P(0,0) + P(1,1) + \ldots + P(N,N) + N + 1$$

であることが分かる． よって， $P(0,N) \le 2N+1.$

性質: 区間木 T(0, N) はO(N)の時間で構成できる.

上の証明と全くと同様に証明できる.

区分木の定義とデータ構造

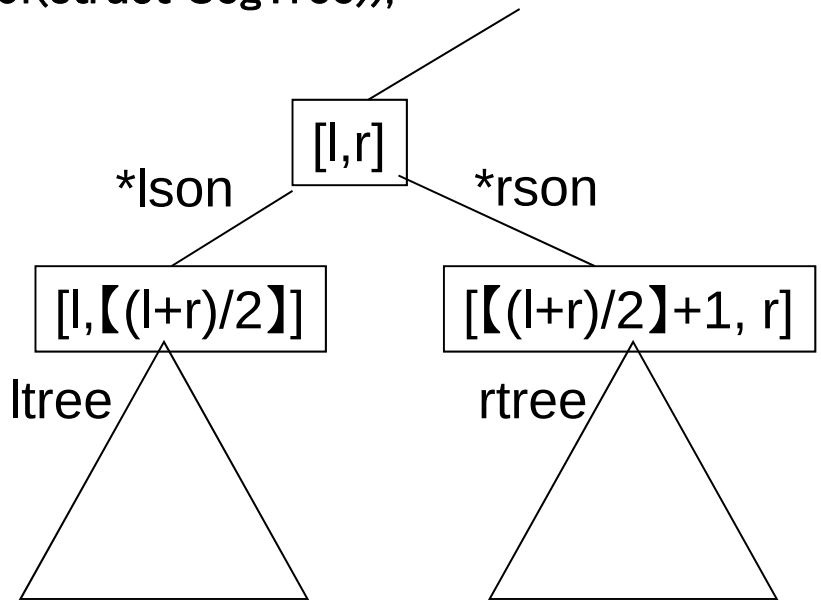
区分木(Segment tree)

区間木の各節点に，関連する区間のリストを蓄えるデータ構造

```
struct INTVAL{  //区間データ
    int left, right;
} interval[MAX];

struct IntNameList {  //節点に関する区間リスト
    int intname;
    struct IntNameList *next;
};

struct SegTree {  // 区分木の構造
    int left, right;
    struct IntNameList *first;
    struct SegTree *lson, *rson;
};

int n;    // 区間の個数
```

区分木の定義とデータ構造(2)

```
void ConsSegTree(int l, int r, struct SegTree *ptr)
{
    (ptr->left) = l;
    (ptr->right) = r;
    (ptr->first) = NULL;
    if(r - l > 0){
        ltree = (struct SegTree *) malloc(sizeof(struct SegTree));
        rtree = (struct SegTree *) malloc(sizeof(struct SegTree));
        ptr->lson = ltree;
        ptr->rson = rtree;
        ConsSegTree(l, (l+r)/2, ltree);
        ConsSegTree((l+r)/2+1, r, rtree);
    } else {
        (ptr->lson) = NULL;
        (ptr->rson) = NULL;
    }
}
```



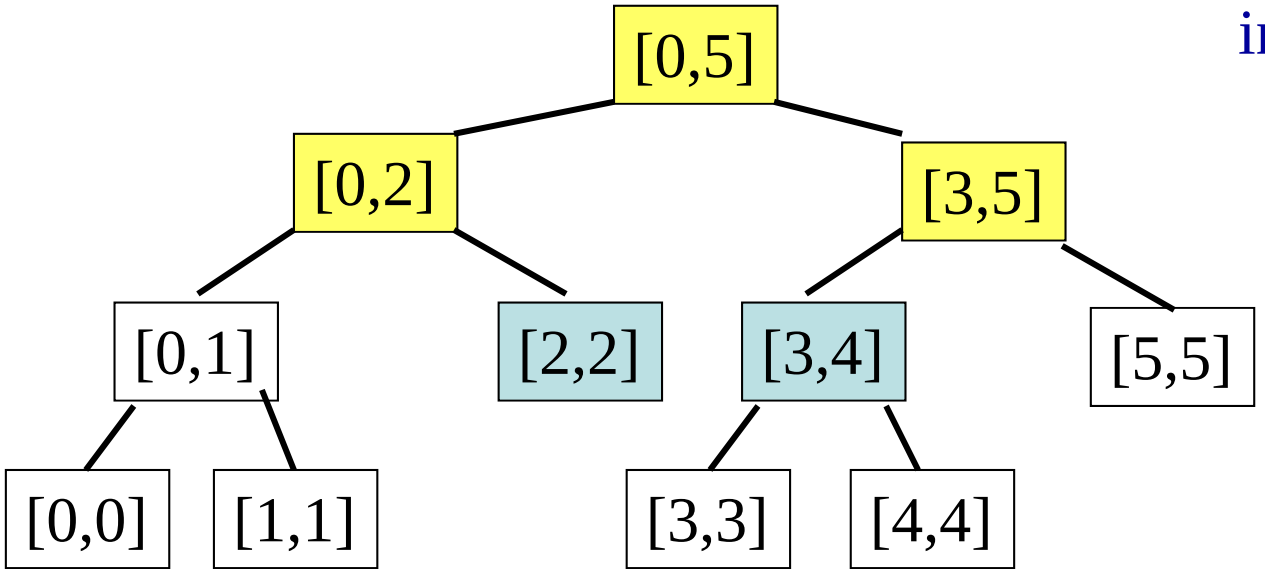
区間の登録

新たな区間の挿入　insert(l, r)

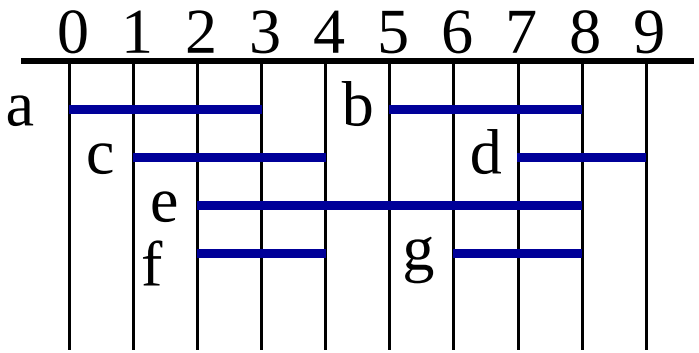
(1) 節点 u の区間が区間[l, r]に完全に含まれるなら，節点 u の区間リストにこの区間を加えて終り.

(2) 節点 u の左の子 v の区間と区間[l, r]が共通部分をもつなら，節点 v にも同じ操作 insert(l, r)を再帰的に適用する.

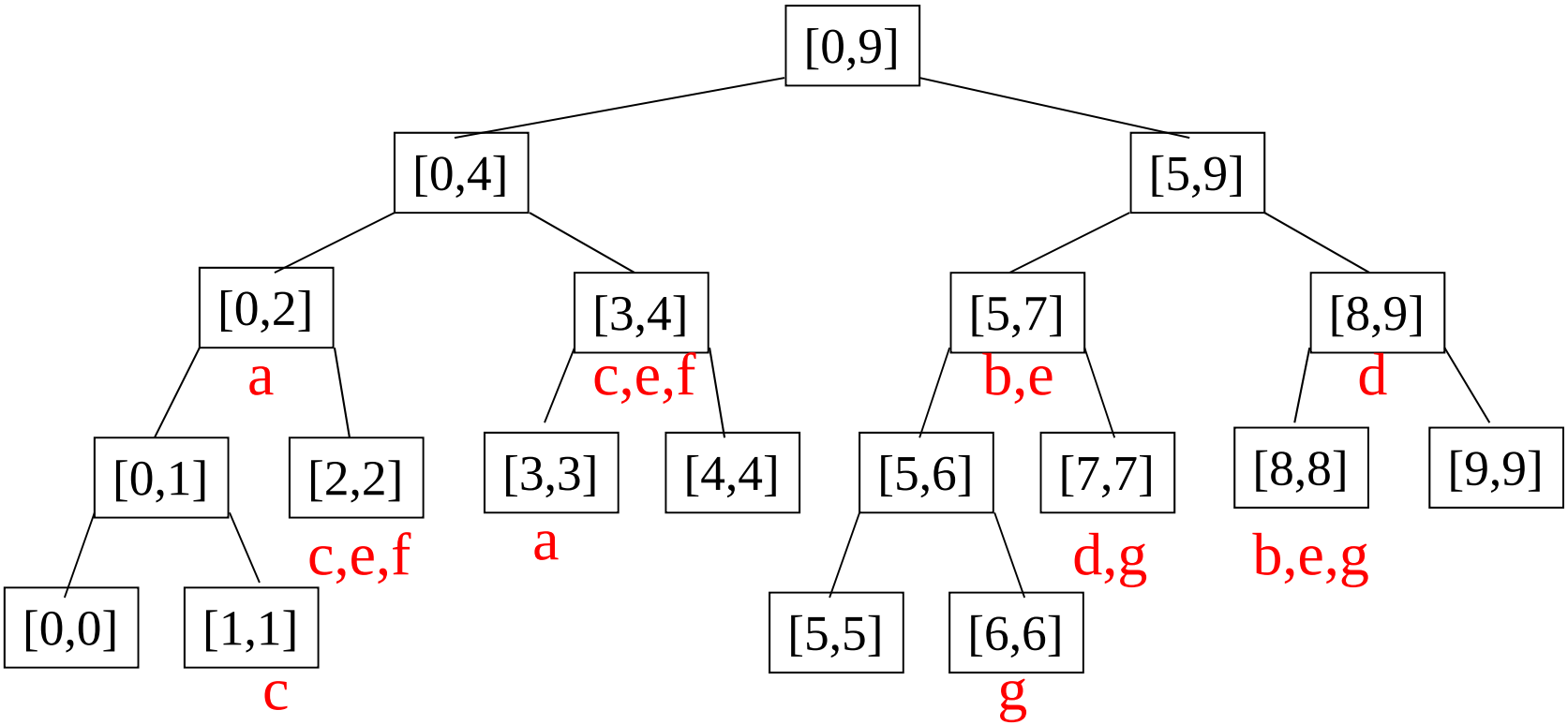
(3) 節点 u の右の子 w についても(2)と同様の操作を行う.



insert(2,4,root)


0 1 2 3 4 5 6 7 8 9
a
b
c
d
e
f
g

区分木


[0,9]
[0,4]
[5,9]
[0,2]
[3,4]
[5,7]
[8,9]
a
c,e,f
b,e
d
[0,1]
[2,2]
[3,3]
[4,4]
[5,6]
[7,7]
[8,8]
[9,9]
c,e,f
a
d,g
b,e,g
[0,0]
[1,1]
[5,5]
[6,6]
c
g

演習：前頁の区間木に区間[3, 9]を登録せよ.

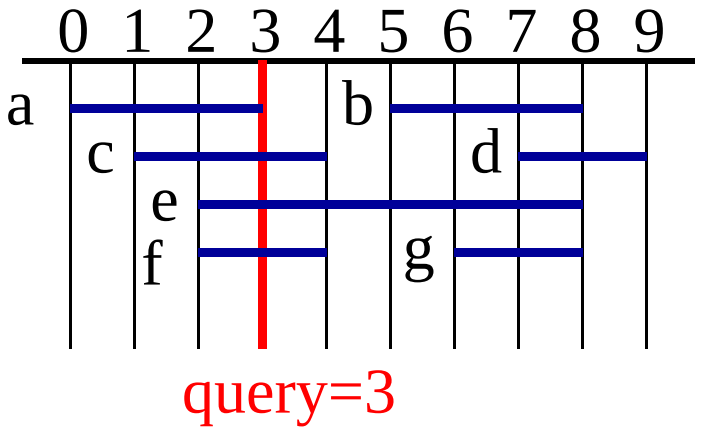
レポート課題1：前頁の区間木で，根を除くすべてのレベルの節点に
登録されるような区間はあるか．もしあれば，例をあげよ.
（ヒント：４つの節点に登録される区間を考えよ．）

区間登録のプログラム

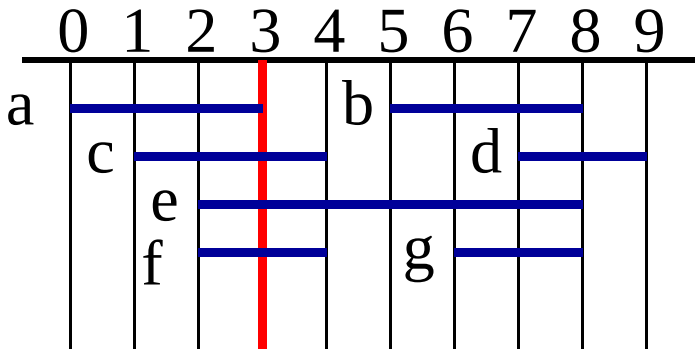
```
void insert(int n, int l, int r, SegTree *ptr)
{
    struct IntNameList *lp;

    if(l <= (ptr->left) && (ptr->right) <= r){
        lp = malloc(sizeof(struct IntNameList));
        lp->intname = n;
        lp->next = ptr->first;
        ptr->first = lp;
    } else {
        if( l <= (ptr->left + ptr->right)/2 ) insert(n, l, r, ptr->lson);
        if( (ptr->left + ptr->right)/2 < r )   insert(n, l, r, ptr->rson);
    }
}
```

質問点を含む区間の列挙

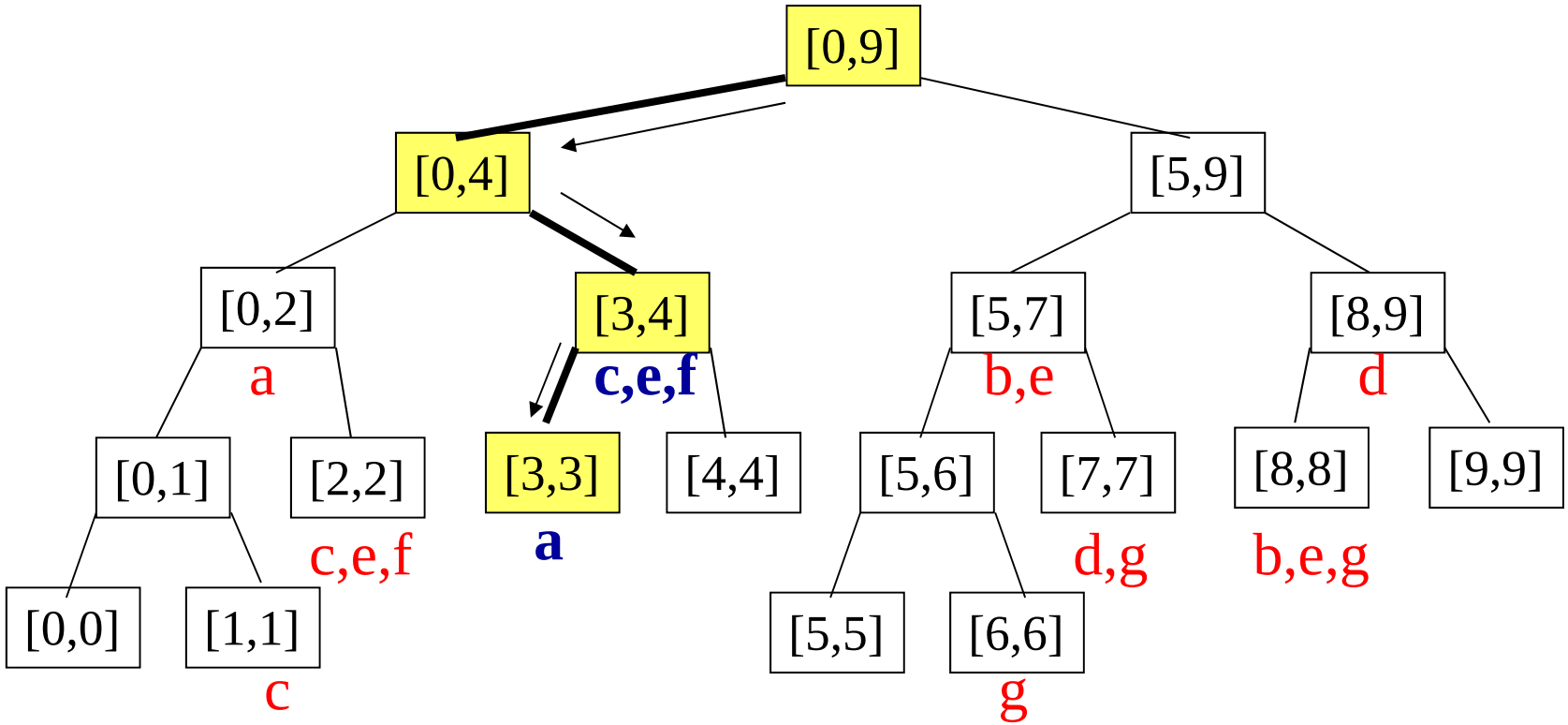
節点 v を根とする部分木に含まれる
区間の中で質問点を含む区間を
すべて列挙する.

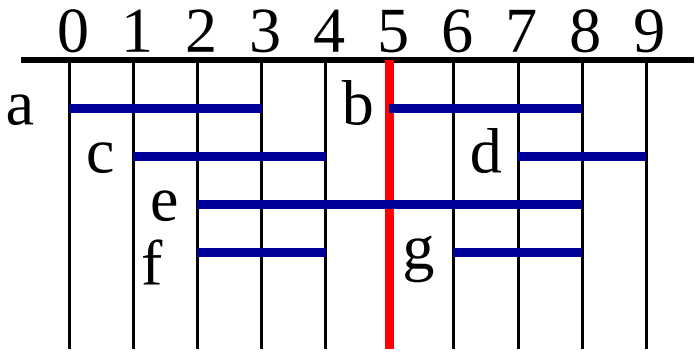



```
void report(SegTree *ptr, int x)
{
    struct IntNameList *p;

    if(ptr->left <= x && x <= ptr->right) {
        p = ptr->first;
        while(p != NULL){
            cout << " intreval " << p->intname;
            p = p->next;
        }
    }
    if(ptr->right - ptr->left > 0){
        if(x <= (ptr->left + ptr->right)/2 ) report(x, ptr->lson);
        else                                 report(x, ptr->rson);
    }
}
```


0 1 2 3 4 5 6 7 8 9
a
b
c
d
e
f
g
query=3

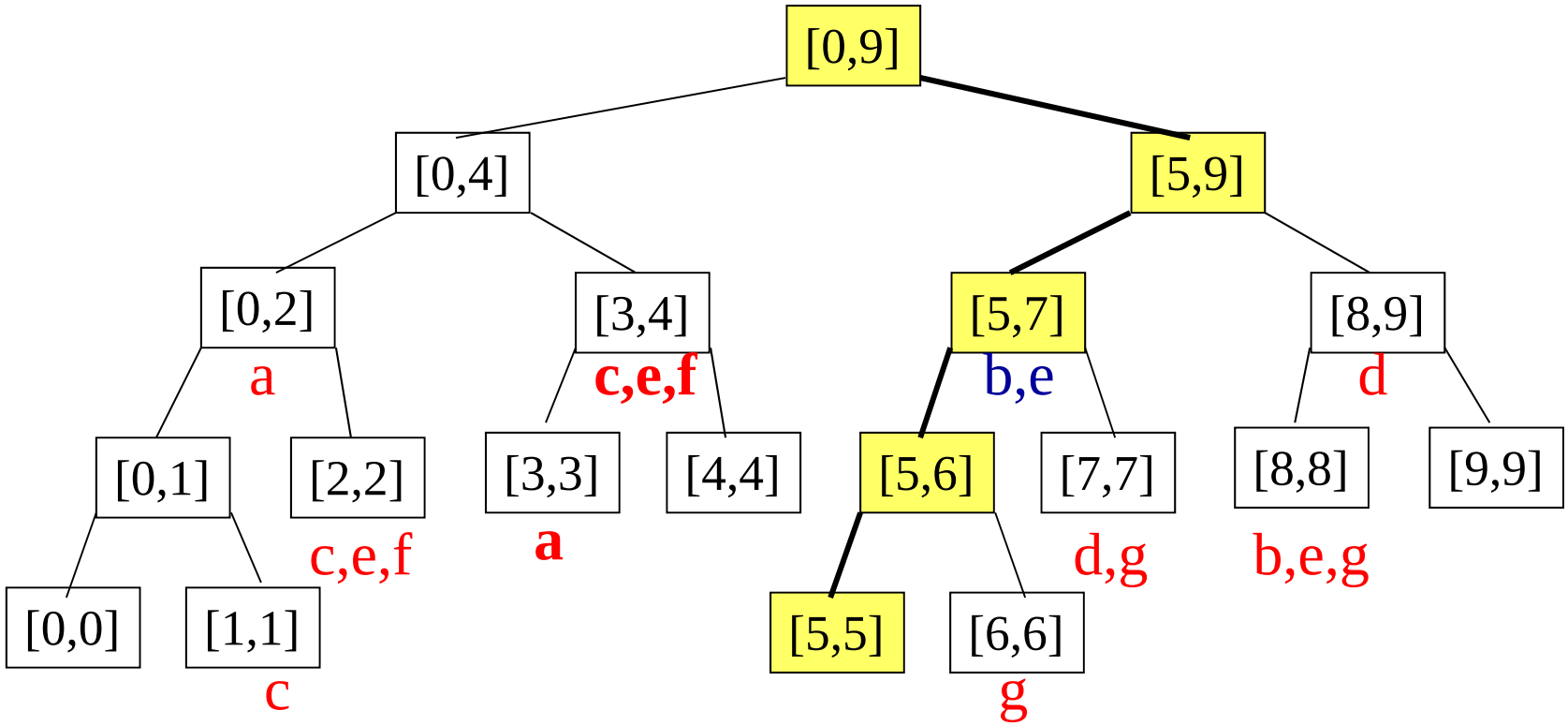
区分木


[0,9]
[0,4]
[5,9]
[0,2]
[3,4]
[5,7]
[8,9]
a
c,e,f
b,e
d
[0,1]
[2,2]
[3,3]
[4,4]
[5,6]
[7,7]
[8,8]
[9,9]
c,e,f
a
d,g
b,e,g
[0,0]
[1,1]
[5,5]
[6,6]
c
g


0 1 2 3 4 5 6 7 8 9
a
b
c
d
e
f
g

query=5

区分木


[0,9]
[0,4]
[5,9]
[0,2]
[3,4]
[5,7]
[8,9]
a
c,e,f
b,e
d
[0,1]
[2,2]
[3,3]
[4,4]
[5,6]
[7,7]
[8,8]
[9,9]
c,e,f
a
d,g
b,e,g
[0,0]
[1,1]
[5,5]
[6,6]
c
g

レポート2：
前頁では質問が一つの値で与えられたが，区間[p, q]で与えられて，
この質問区間と共通部分をもつ区間を列挙するように拡張することは可能か？

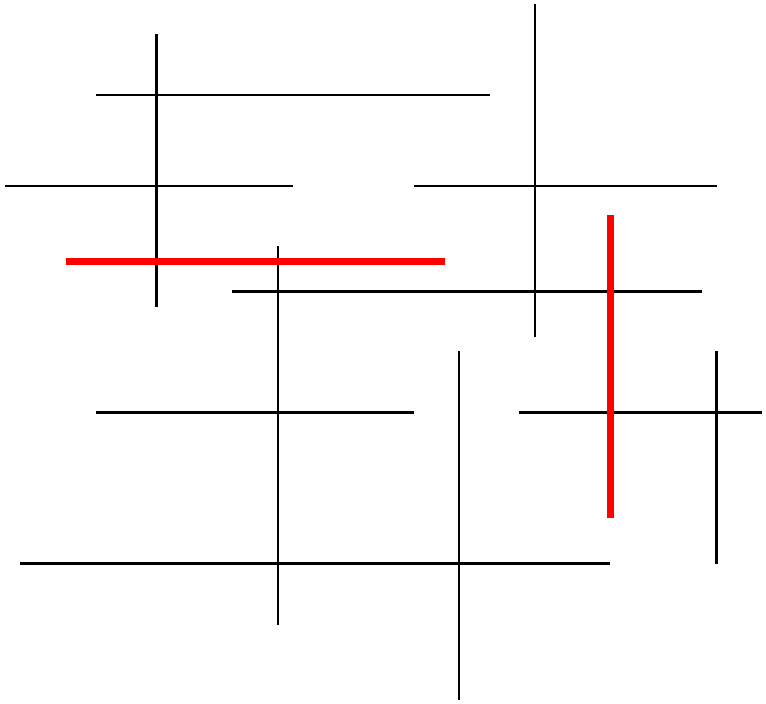
区分木の応用

水平線分と垂直線分の集合の管理

多数の水平線分と垂直線分からなる線分集合を蓄えておき，
質問として垂直または水平線分が与えられたとき，それと交差
する線分をすべて列挙する.

水平線分と垂直線分を別々に蓄える.

質問線分が水平線分なら，
垂直線分を蓄えた区分木を探索.
質問線分が垂直線分なら，
水平線分を蓄えた区分木を探索.



この方法では最悪の場合にO(n)の
時間がかかってしまう.

すべての線分が同じ左右の座標を
もつとき，O(n)の時間が必要.

区分木の応用(2)

各節点における区間集合の管理

単なる線形リストでは効率が悪い.

線形リストではなく，2分探索木に変更する.

水平線分を蓄える2分探索木ではy座標をキーとする.

探索の効率

質問線分が与えられたとき，O(log N)個の節点を調べる.

各節点では，O(log n)時間で交差線分を列挙できる.

ただし，Nは全体の区間の長さ，nは線分数.

よって，列挙交差のための時間は

$O(k + \log N \log n)$

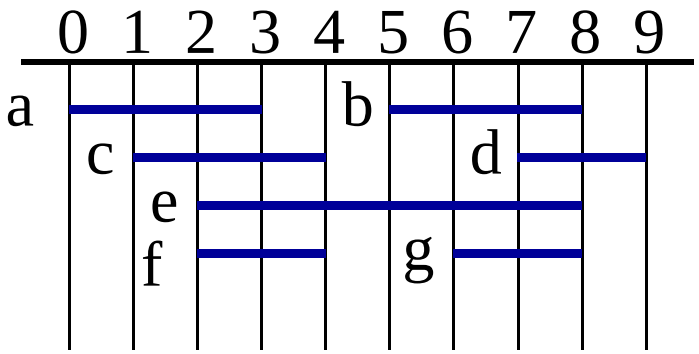
となる．ただし，kは列挙される線分の本数.

欠点:データ構造のメモリサイズは$O(N + n \log N)$.

もっと高速化が可能.

インタバル木

区間の集合 $I = \{[x_1, x_1'], [x_2, x_2'], \ldots, [x_n, x_n']\}$



I = {a[0,3], b[5,8], c[1,4], d[7,9],
e[2,8], f[2,4], g[6,8]}

$\{x_1, x_1', x_2, x_2', \ldots, x_n, x_n'\}$: 2n 個の端点

x_{mid} = 2n 個の端点の中央値

次の3つの集合を定義

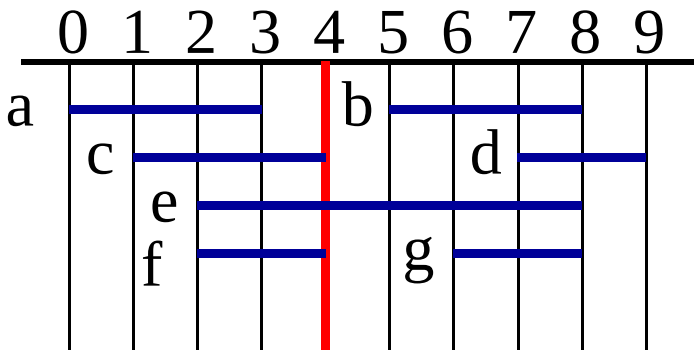
$I_{left} = \{[x_j, x_j'] \mid x_j' < x_{mid}\}$ x_{mid}よりにある左

$I_{right} = \{[x_j, x_j'] \mid x_j > x_{mid}\}$ x_{mid}よりにある右

$I_{mid} = \{[x_j, x_j'] \mid x_i \leqq x_{mid} \leqq x_j'\}$ x_{mid}を含む区間

インタバル木

区間の集合 $I = \{[x_1, x_1'], [x_2, x_2'], \ldots, [x_n, x_n']\}$



I = {a[0,3], b[5,8], c[1,4], d[7,9],
e[2,8], f[2,4], g[6,8]}

2n個の端点= {0,1,2,2,3,4,4,5,6,7,8,8,8,9}

中央値

$I_{left} = \{[x_j, x_j'] \mid x_j' < x_{mid}\}$ ={a[0,3]}

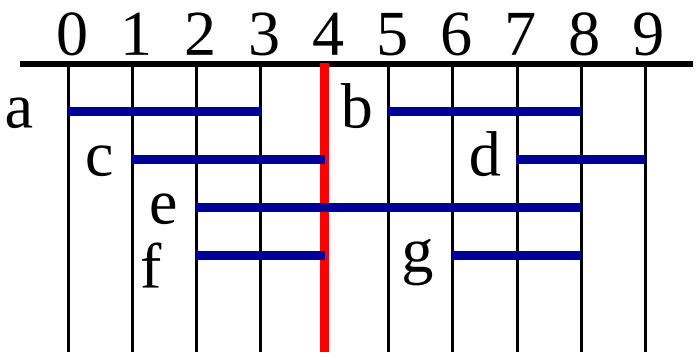
$I_{right} = \{[x_j, x_j'] \mid x_j > x_{mid}\}$ ={b[5,8],d[7,9], g[6,8]}

$I_{mid} = \{[x_j, x_j'] \mid x_j \leqq x_{mid} \leqq x_j'\}$={c[1,4], e[2,8], f[2,4]}

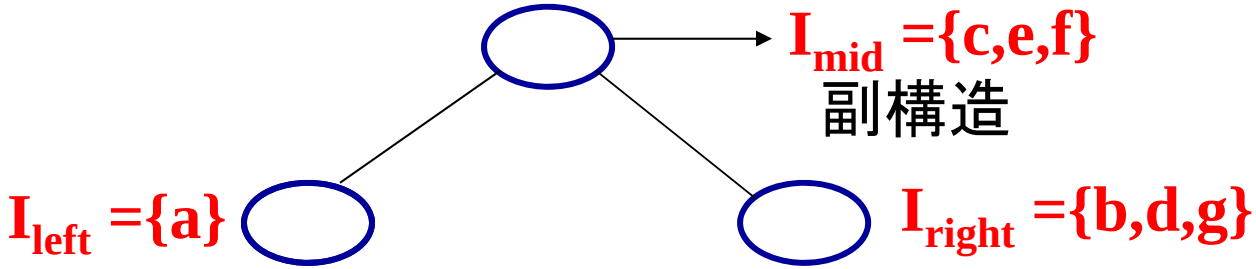
演習：中央値の定義を述べよ.

演習：n個の数が配列に蓄えられているとき，それらの中央値はどれだけの計算時間（のオーダー）で求めることができるか？ソーティングを用いた場合と比較せよ.

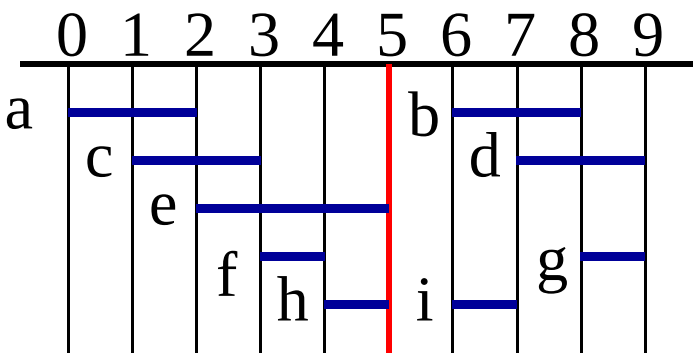
インタバル木の構成



I_{left} = {a[0,3]}
I_{right} = {b[5,8],d[7,9], g[6,8]}
I_{mid} = {c[1,4], e[2,8], f[2,4]}



再帰的に分割を続ける



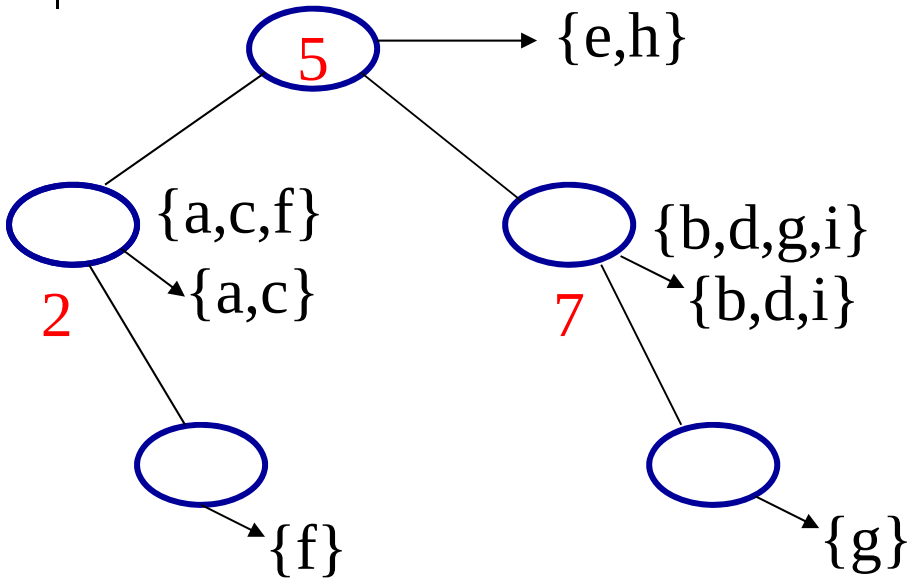
endpoints: 0,1,2,2,3,3,4,4,5,
5,6,6,7,7,8,8,9,9

中央値 = 5

I_{left} = {a,c,f}

I_{right} = {b,d,g,i}

I_{mid} = {e,h}



区間の集合Iに対するインタバル木を構成

I が空なら，対応するインタバル木は葉となる.

空でないなら，I_{left}, I_{right}, I_{mid};を求める

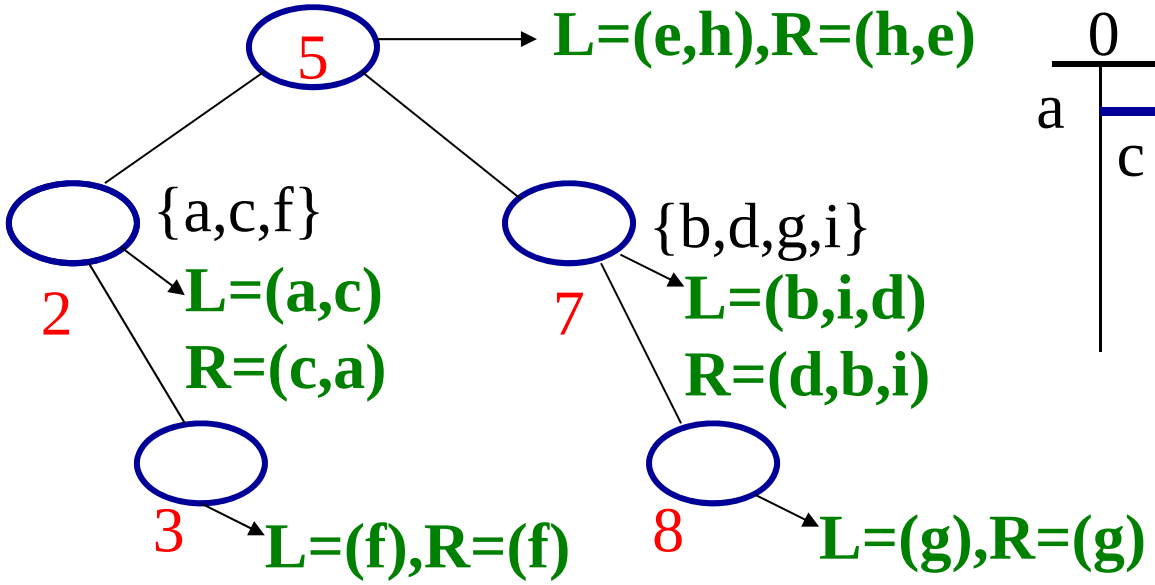
集合 I_{mid} は2回蓄える:

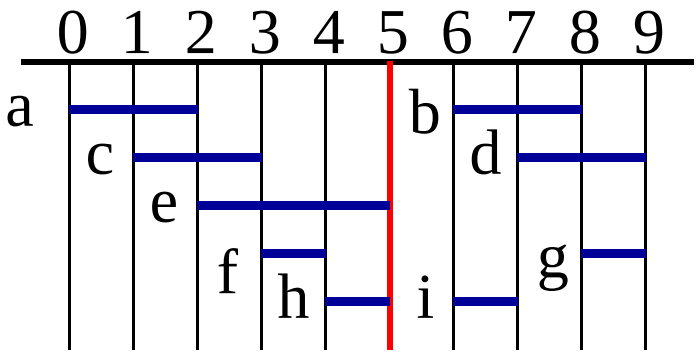
(1)左端点の昇順にソートした区間集合 L として,

(2) 右端点の降順にソートした区間集合 R として.

集合I_{left}に対するインタバル木として左部分木を構成.

集合I_{right}に対するインタバル木として右部分木を構成.





質問点qを含む区間の列挙

v: 節点

if(v is not a leaf)

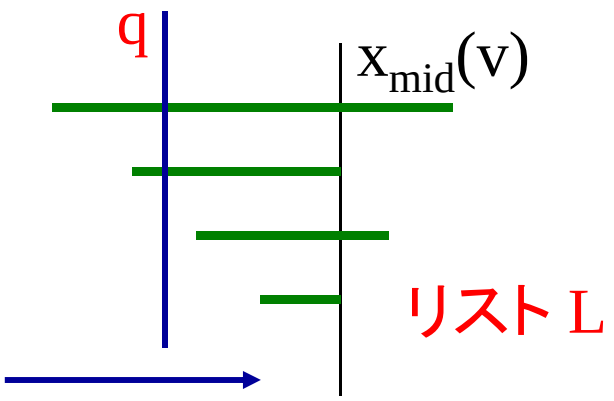
then if($q < x_{mid}(v)$)

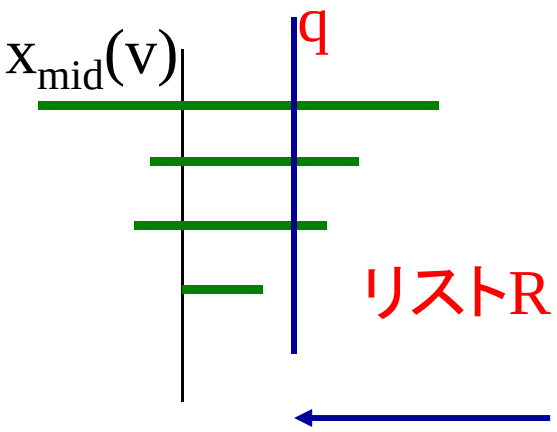
then 左のリスト **L** 上を左から右に調べ，qを含まなくなれば終了

左の子に対して質問を再帰的に適用;

else 右のリスト **R** 上を右から左に調べ，qを含まなくなれば終了

右の子に対して質問を再帰的に適用;

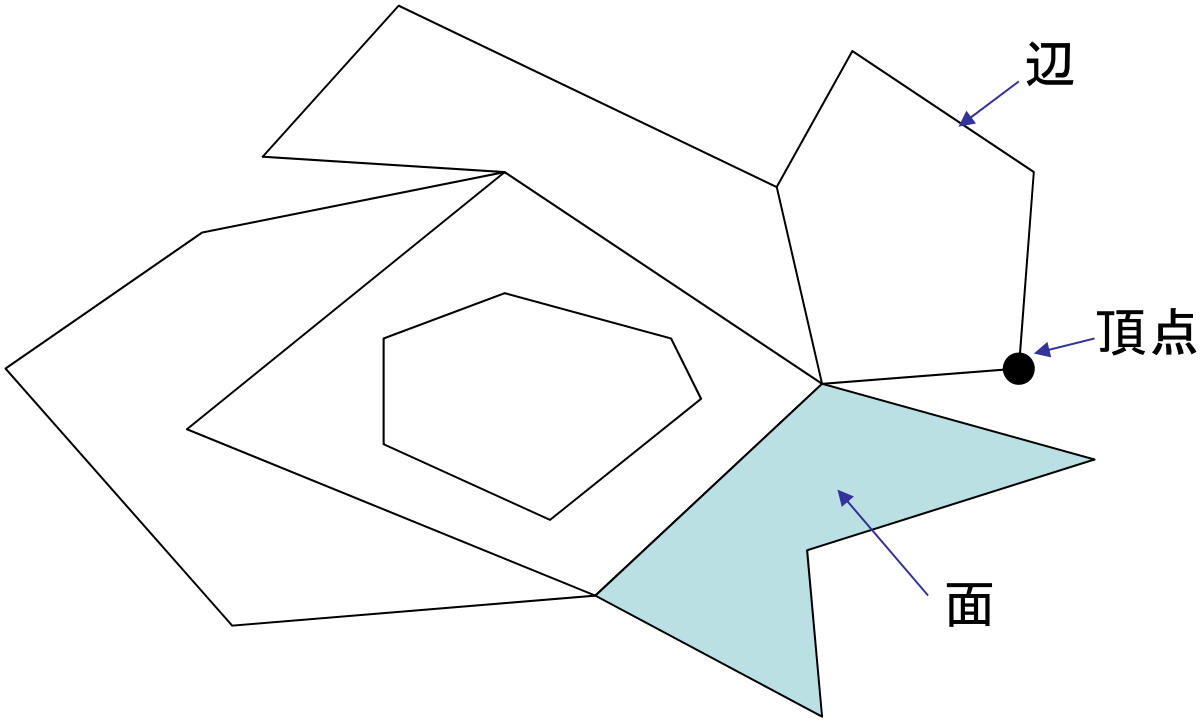




二重連結辺リスト

平面グラフ

グラフ:　頂点と頂点間を結ぶ辺によって表現される関係
平面グラフ:　辺が互いに交差しないように頂点の場所を決めて
　　描かれたグラフのこと，あるいは，そのように描けるグラフ．



二重連結辺リスト

二重連結辺リスト　(DCEL: Doubly-Connected Edge List)

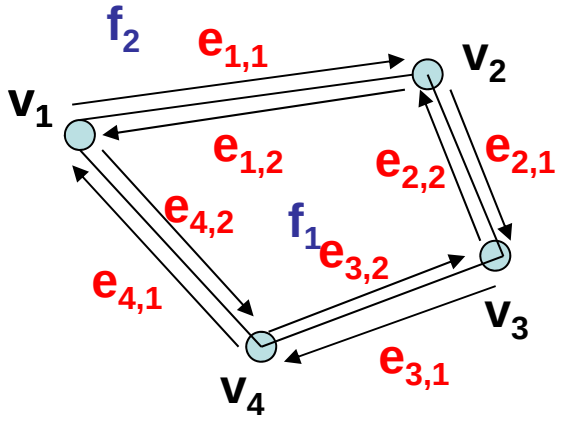
　平面に描かれた平面グラフを表現するのに適したデータ構造

平面グラフ G = (V, E),

　頂点集合　V = {v[1], v[2], ... , v[n]},

　辺集合　　E = {e[1], e[2], ... , e[m]}

　面集合　　F = {f[1], f[2], ... f[k]}



各頂点vに関する情報

　vの座標(x(v), y(v)),

　vから出る一つの辺 out_edge(v)（任意）

各面fに関する情報

　その外部境界上の一つの辺 outer_component(f)

　面の中のそれぞれの穴について，その境界上の一つの辺のリスト

　　inner_component(f)

	座標	接続辺
v1	(x(v1), y(v1))	$e_{1,1}$
v2	(x(v2), y(v2))	$e_{2,1}$
v3	(x(v3), y(v3))	$e_{3,1}$
v4	(x(v4), y(v4))	$e_{4,1}$

二重連結辺リスト

辺に関する情報

各辺について，

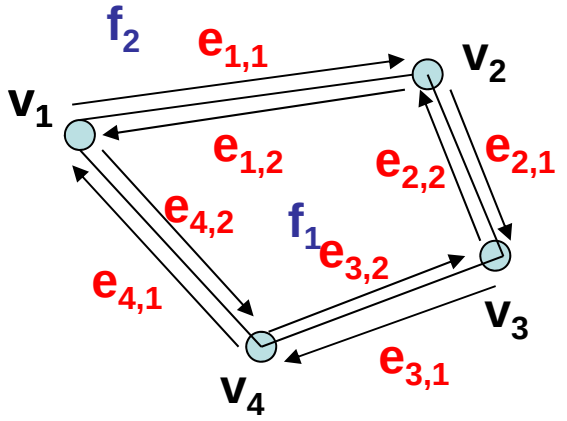
・異なる方向をもつ2つの有向辺を仮定．

・各辺の左にある領域の名前: face(e).

・各辺の始点と終点の頂点名．

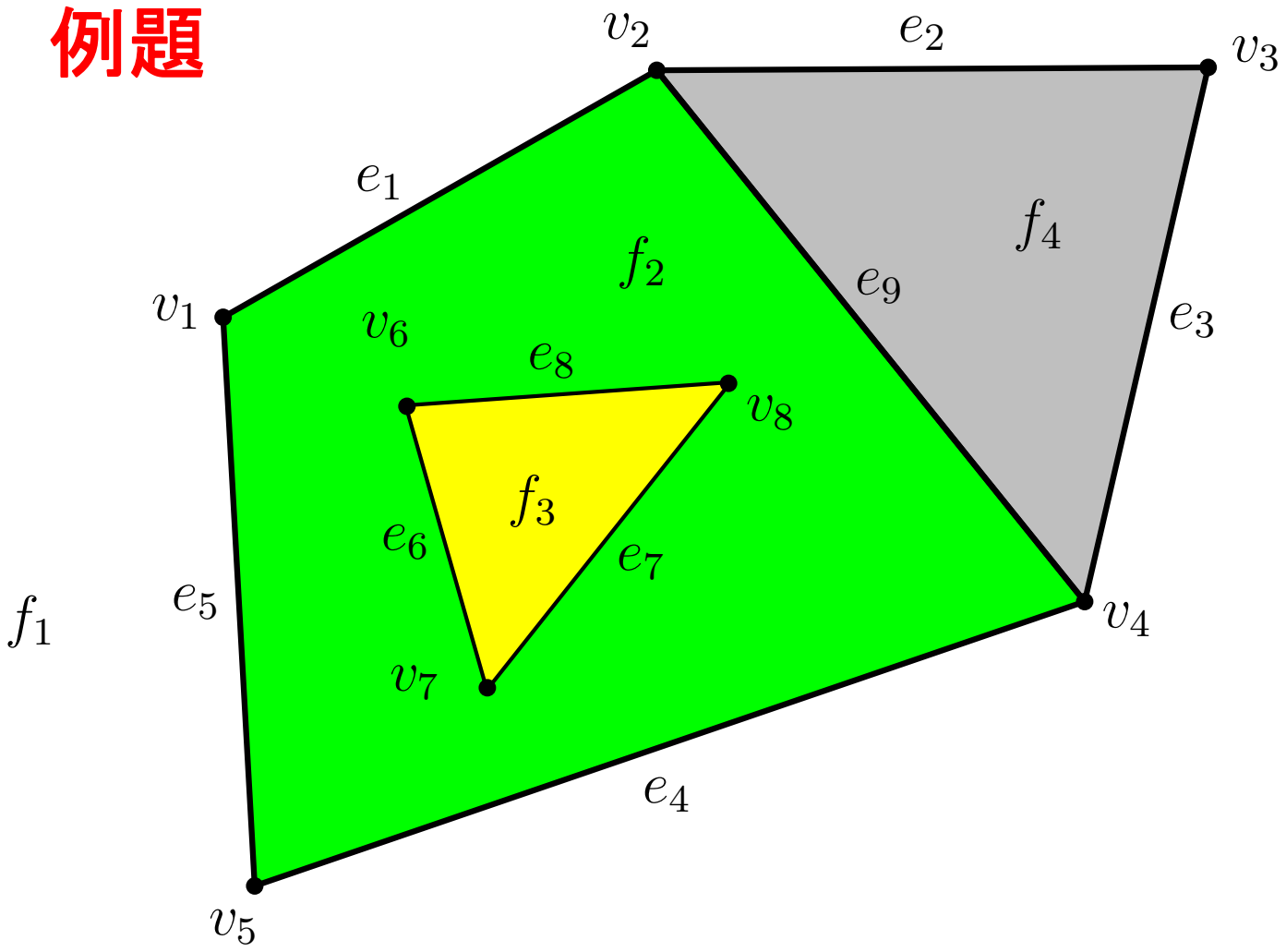
・各辺について，同じ面での次の辺へのポインタをもつ：NextEdge(e).

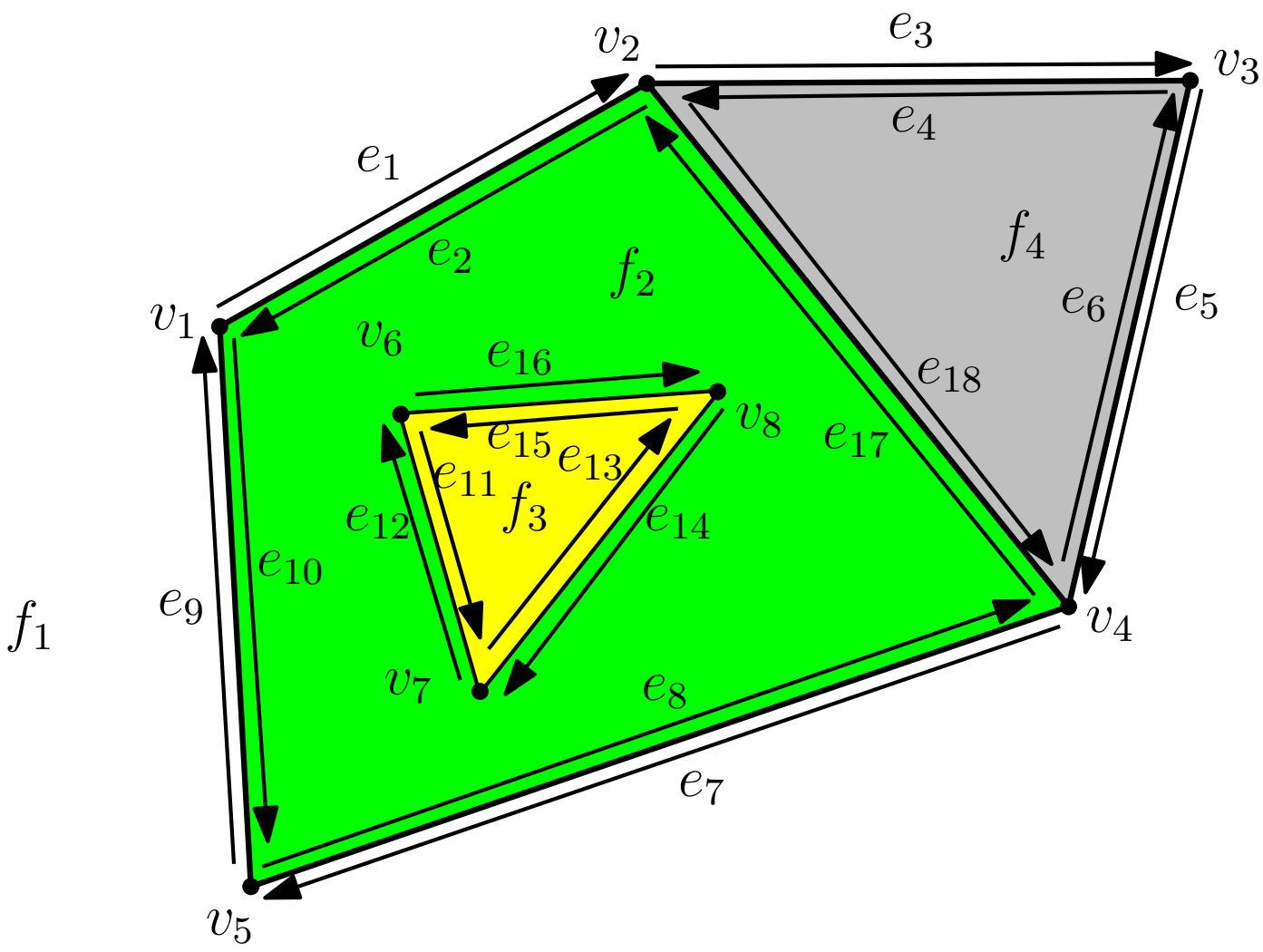
・各辺について，反対方向の辺へのポインタをもつ：TwinEdge(e).



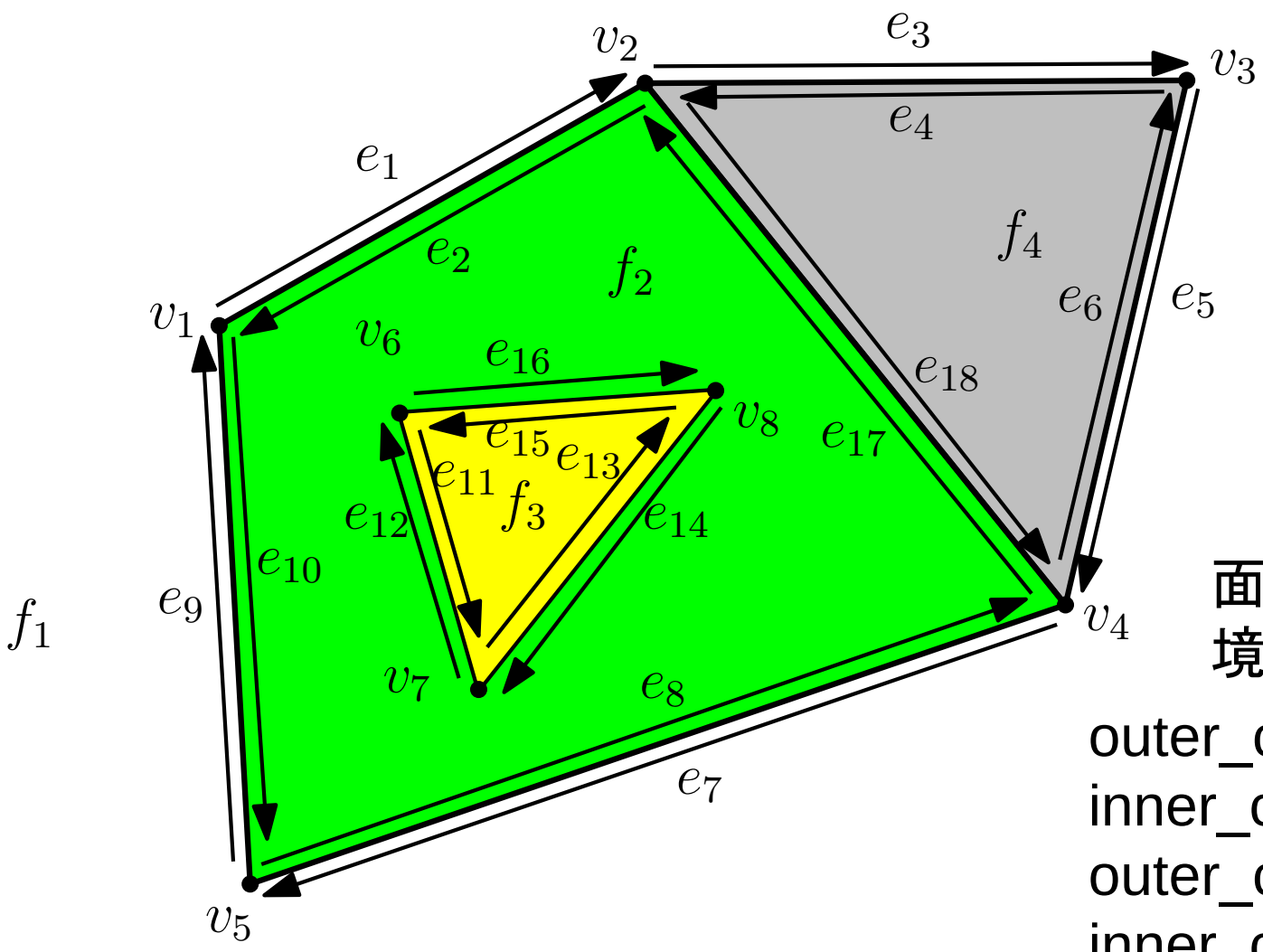
辺	左の領域名	始点	終点	次の辺	反対方向辺
$e_{1,1}$	f_2	v_1	v_2	$e_{2,1}$	$e_{1,2}$
$e_{1,2}$	f_1	v_2	v_1	$e_{4,2}$	$e_{1,1}$
$e_{2,1}$	f_2	v_2	v_3	$e_{3,1}$	$e_{2,2}$
$e_{2,2}$	f_1	v_3	v_2	$e_{1,2}$	$e_{1,2}$
$e_{3,1}$	f_2	v_3	v_4	$e_{4,1}$	$e_{3,2}$
……	……	……	…		

例題





各辺を異なる方向をもつ2辺で置き換える.



面f2のみ内部
境界をもつ.

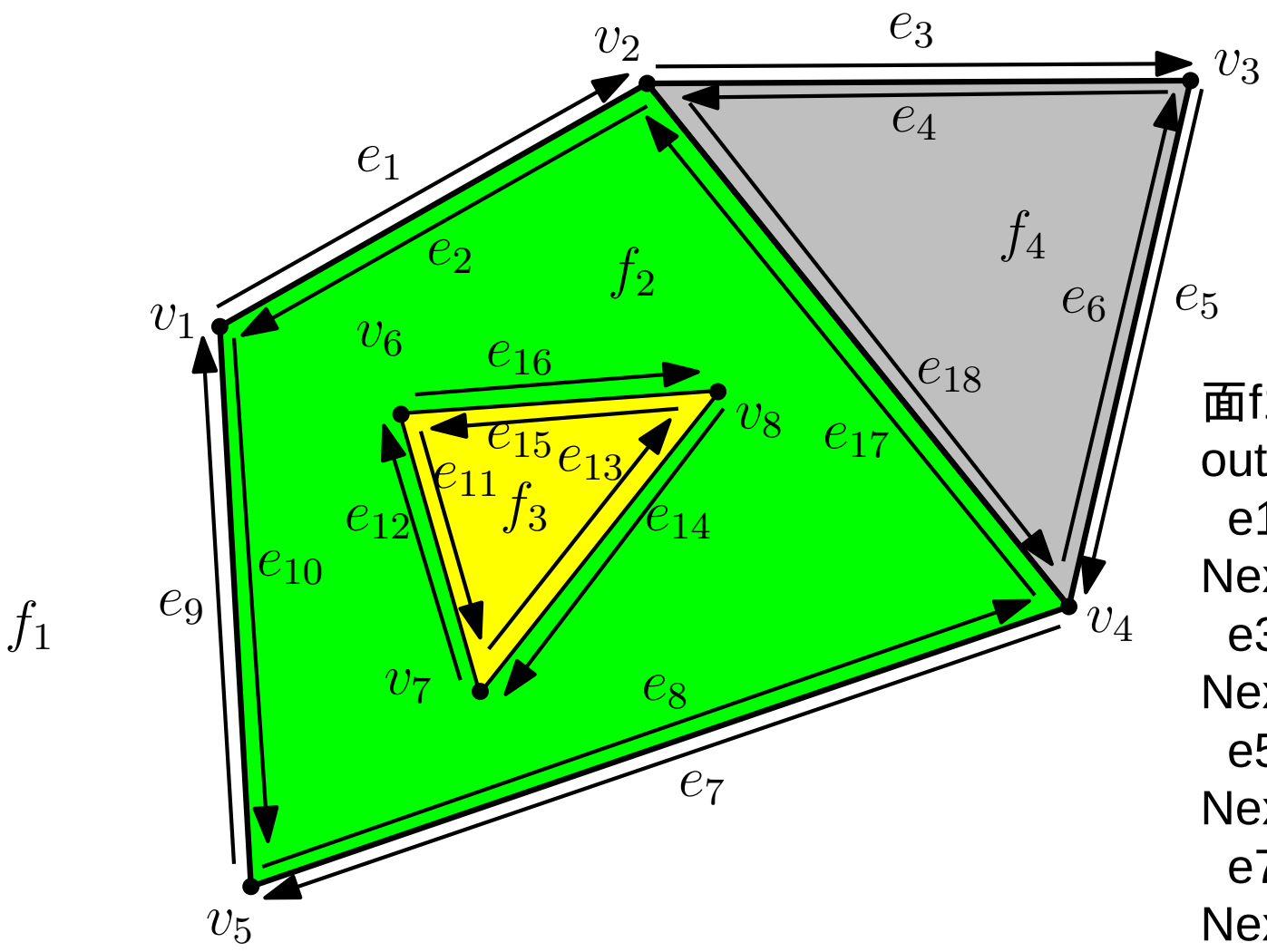
outer_comp(f1)=e1,
inner_comp(f1)=nil
outer_comp(f2)=e2
inner_comp(f2)=e12
outer_comp(f3)=e11
inner_comp(f3)=nil, ...

NextEdge(e1)=e3, NextEdge(e3)=e5, ...
TwinEdge(e1)=e2, TwinEdge(e2)=e1, ...

二重連結辺リスト

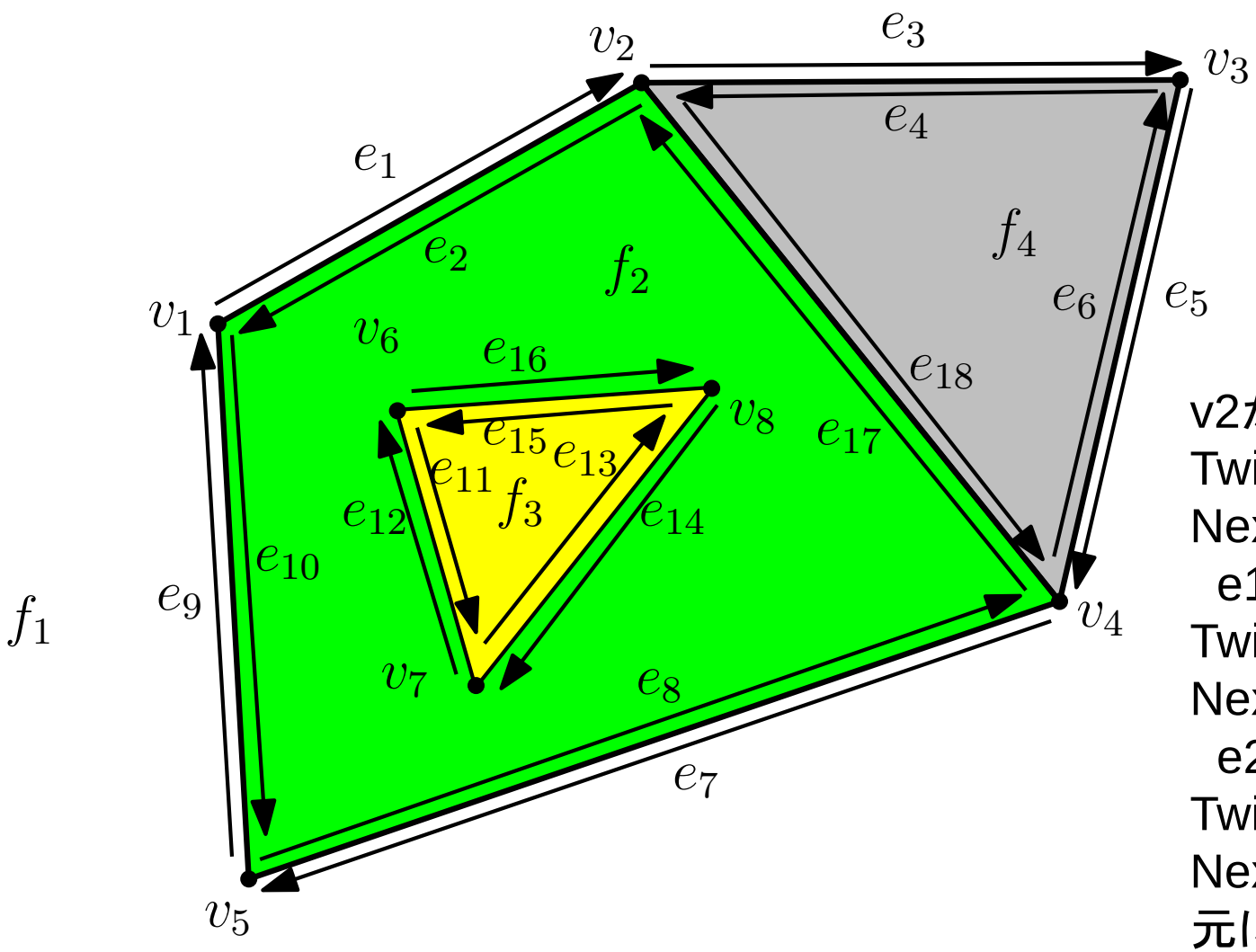
二重連結辺リストを用いてできること:

- 面fを与えると，一つの辺から始めて次の辺へのポインタをたどれば，面の境界上の辺を列挙できる.
- 頂点を与えると，その頂点に入るすべての辺を列挙できる.
- 頂点を与えると，その頂点から出るすべての辺を列挙できる.
- 辺を与えると，その左にある面の名前を出力できる.



一つの面の境界を辿る.

面f1を辿る:
outer_comp(f1)=e1
e1を出力
NextEdge(e1)=e3
e3を出力
NextEdge(e3)=e5
e5を出力
NextEdge(e5)=e7
e7を出力
NextEdge(e7)=e9
e9を出力
NextEdge(e9)=e1
元に戻ったので終了



v2から出る辺e3を出力
TwinEdge(e3)=e4
NextEdge(e4)=e18
　e18を出力
TwinEdge(e18)=e17
NextEdge(e17)=e2
　e2を出力
TwinEdge(e2)=e1
NextEdge(e1)=e3
元に戻ったので終了

たとえば，頂点v2から出る辺をすべて列挙するには？

演習：前ページでは二重連結辺リストを用いてできる操作を列挙したが，
単に頂点に接続する辺を番号順に蓄えるだけのデータ構造でそれらの
操作を実現しようとすると，どれだけの時間が必要か？