10.다차원 공간 화일

❖ 격자 화일(Grid file)

- 정적, 동적 상황에 모두 적합
  - 완전 일치 질의(exact match query)
  - 범위 질의(range query)

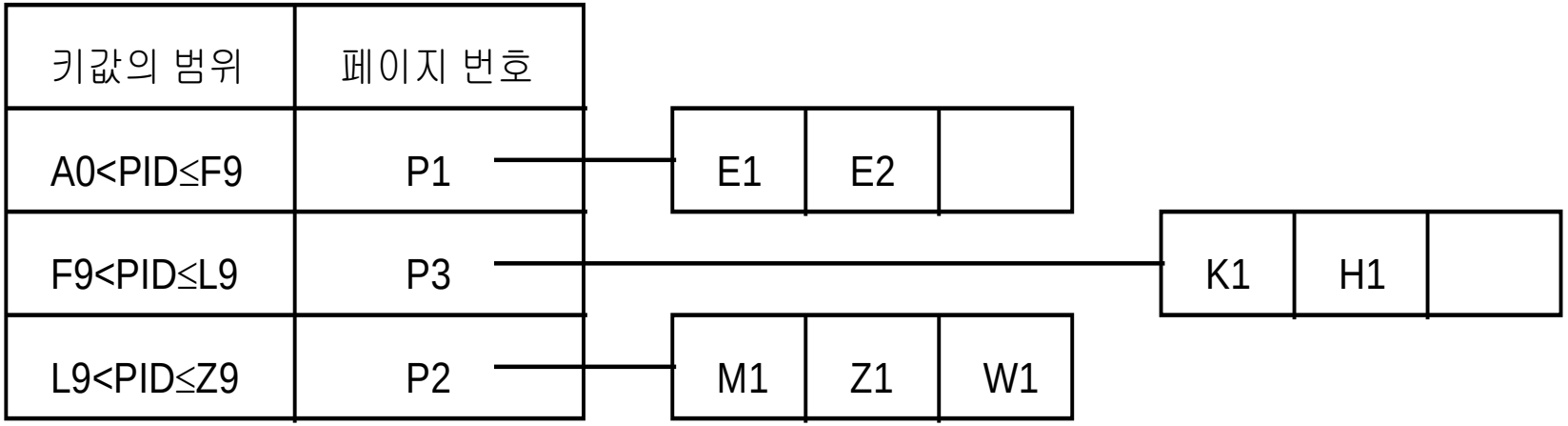
▶ 1 차원 인덱스 - X-Y 평면상의 점 화일

PID	X	Y
E1	2	4
M1	4	6
Z1	16	2
E2	7	2
W1	18	8
K1	9	4
H1	8	8
A1	13	9
Z2	6	4
L1	4	2
E2	15	8

- 인덱스 엔트리 = (키값의 범위, 페이지 번호)

i) 삽입

- 최초 범위는 PID의 최대 범위
- 오버플로시 분할(split)
- H1 삽입시 오버플로 발생, 분할



ii) 삭제

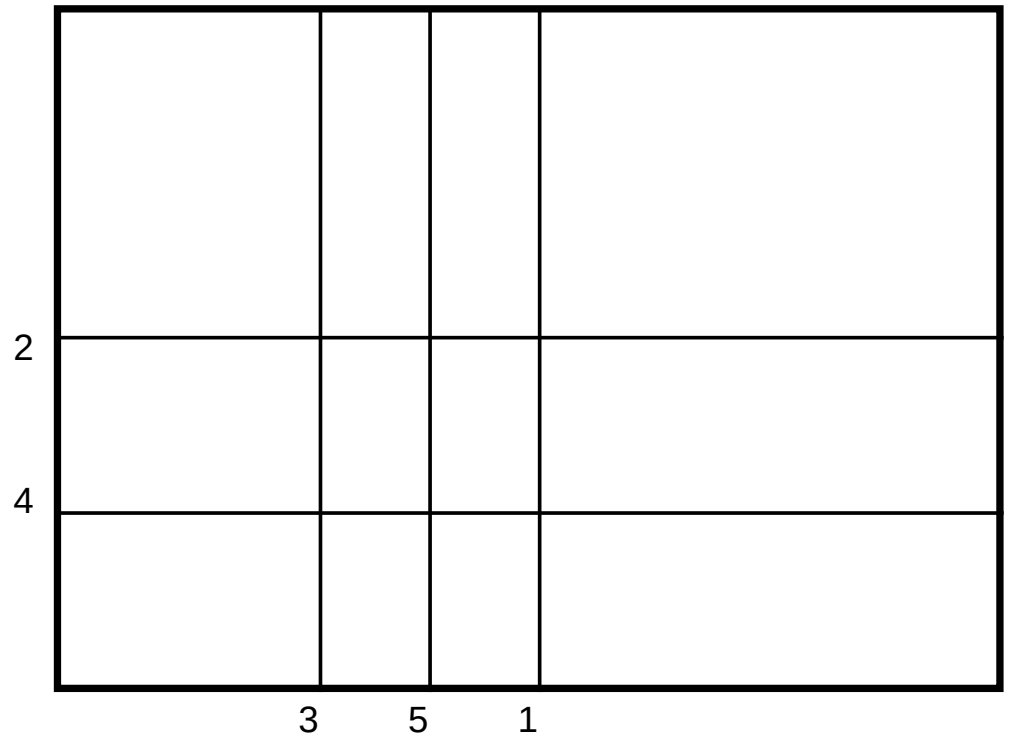
- 적재율이 낮으면 합병

▶ K 차원 인덱스

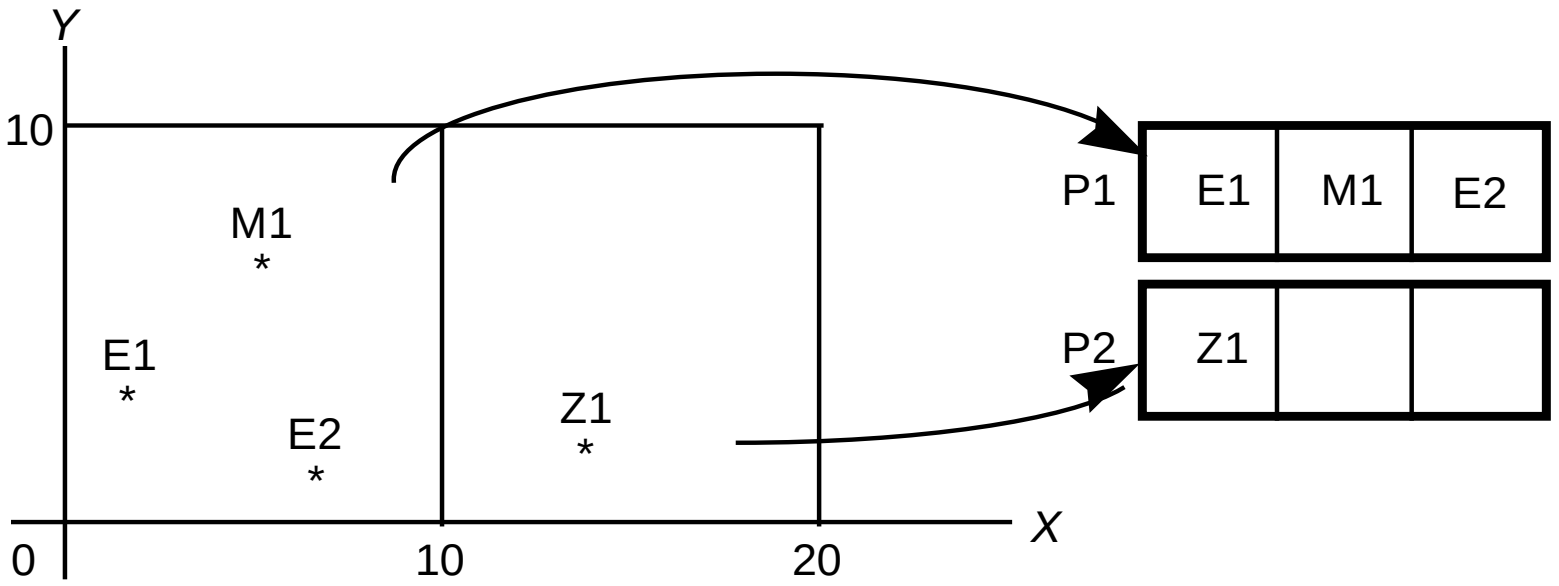
– 선형 눈금자, 격자 배열, 격자 블록, 격자 디렉토리

i) 삽입

– 격자 분할(각 축을 같은 빈도로 분할)



– 오버플로 발생, 분할



ii) 검색

- 완전 일치 질의
  - 격자 배열 참조, 데이타 참조
- 범위 질의

iii) 삭제

- 트윈의 경우는 합병

- 인덱스의 키 갯수가 작고 균등 분포시 효과적

❖ K-D 트리

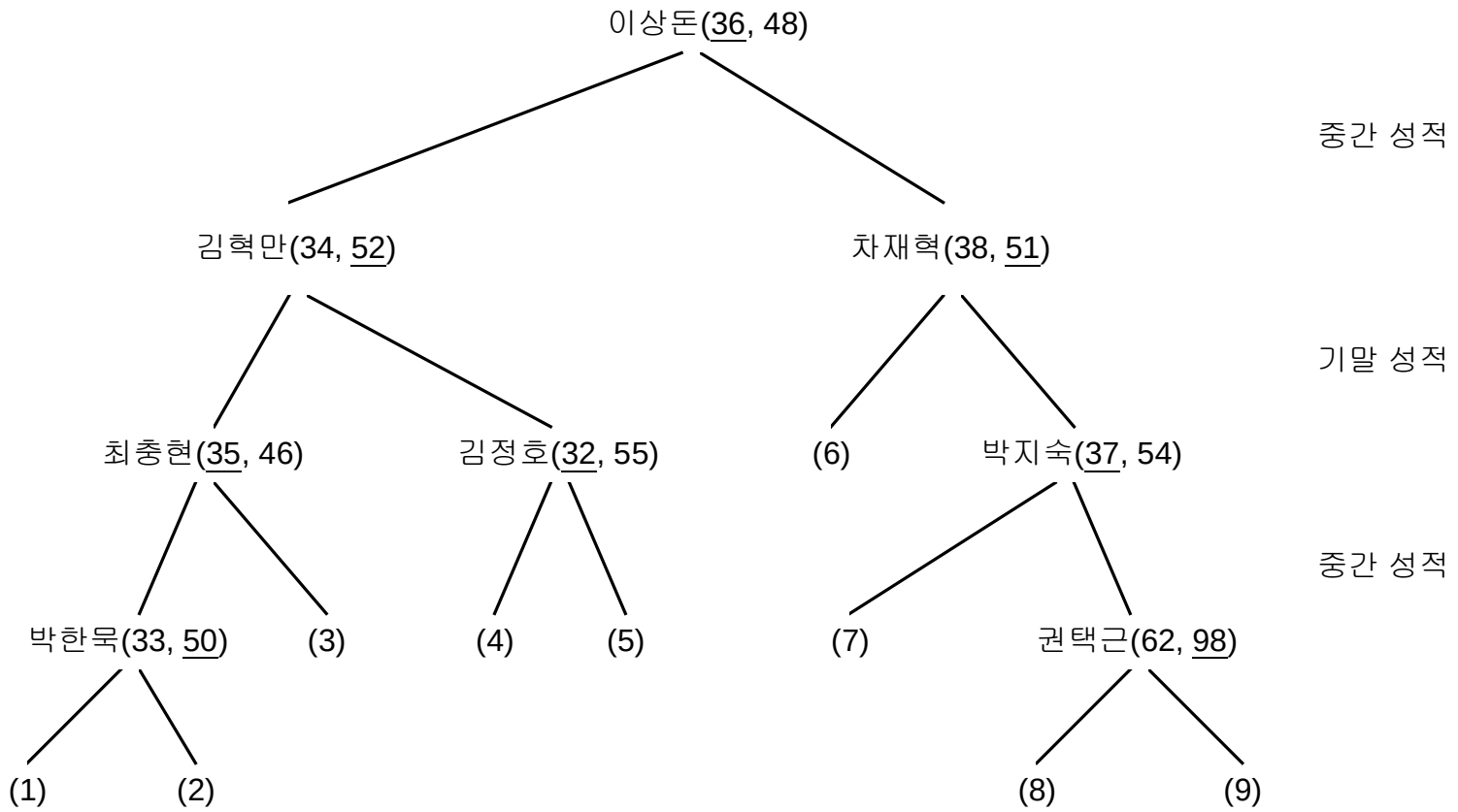
- 기본 키, 보조 키에 의한 검색
- K 개의 차원을 K 레벨마다 번갈아 비교.

▶ 2-D 트리 - 학생 성적 화일

이 름	중간성적	기말성정	…
이상돈	36	48	
김혁만	34	52	
차재혁	38	51	
박지숙	37	54	
김정호	32	55	
최충현	35	46	
박한묵	33	50	
권택근	62	98	

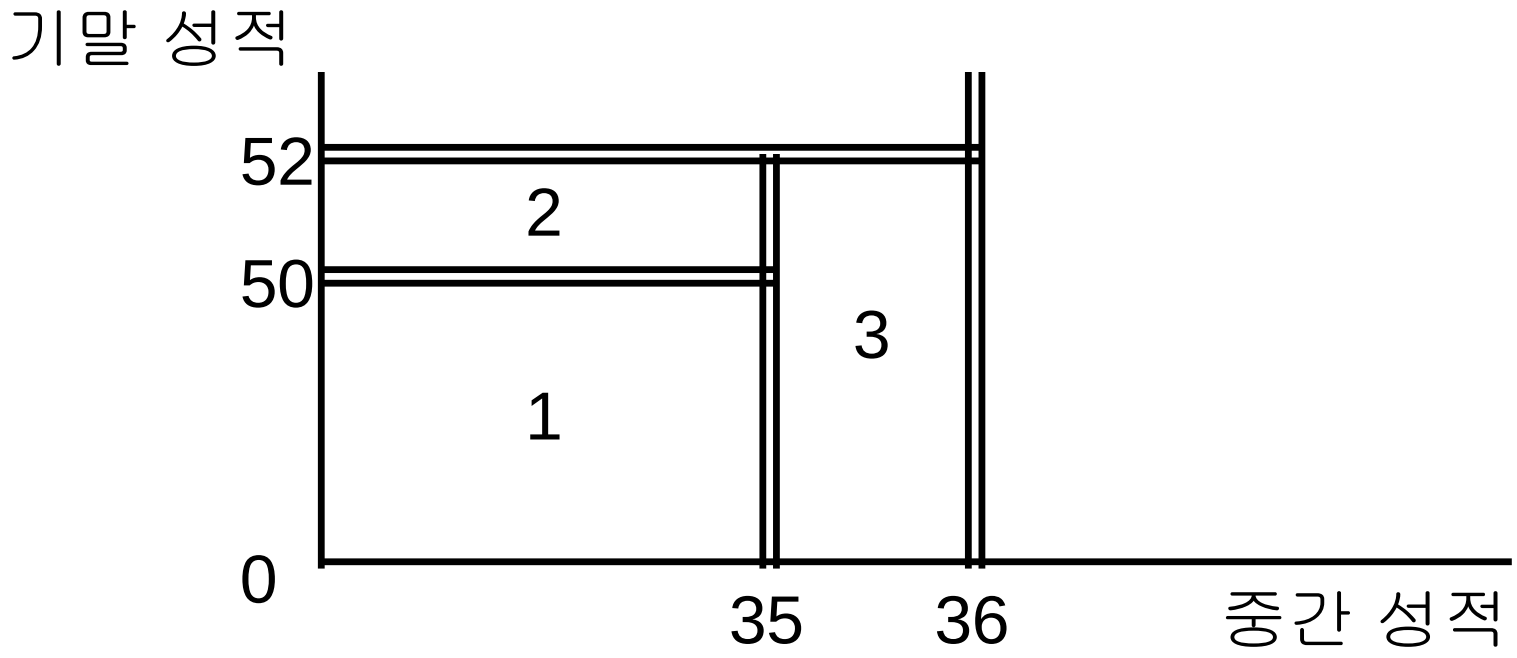
- 중간 성적과 기말 성적을 교대로 비교

i) 삽입



- 단말 노드는 인덱스 역할
- 탐색 공간을 키값에 의해 분할(≤과 >)한 소영역에 레코드 저장

– 탐색 공간의 분할



– 저장 버켓

이 름	버켓 번호
이상돈	3
김혁만	2
차재혁	6
박지숙	7
김정호	4
최충현	1
박한묵	1
권택근	8

ii) 검색

- **2-D** 트리의 노드를 따라가며 원하는 버켓을 탐색

❖ K-D-B 트리

- B 트리와 K-D 트리의 결합
- 완전 균형 트리
- 인덱스 (키0, 키1, …, 키K-1, 주소)
- 점 : 도메인0×도메인1×…도메인K-1의 한 원소
- 영역 : 같은 성질을 갖는 점들의 집합
- 노드는 루트 페이지와 페이지의 집합
  - 영역 페이지 : <영역, 페이지 ID>
  - 점 페이지 : <점, 주소>, 단말 노드

▶ K-D-B 트리의 특성

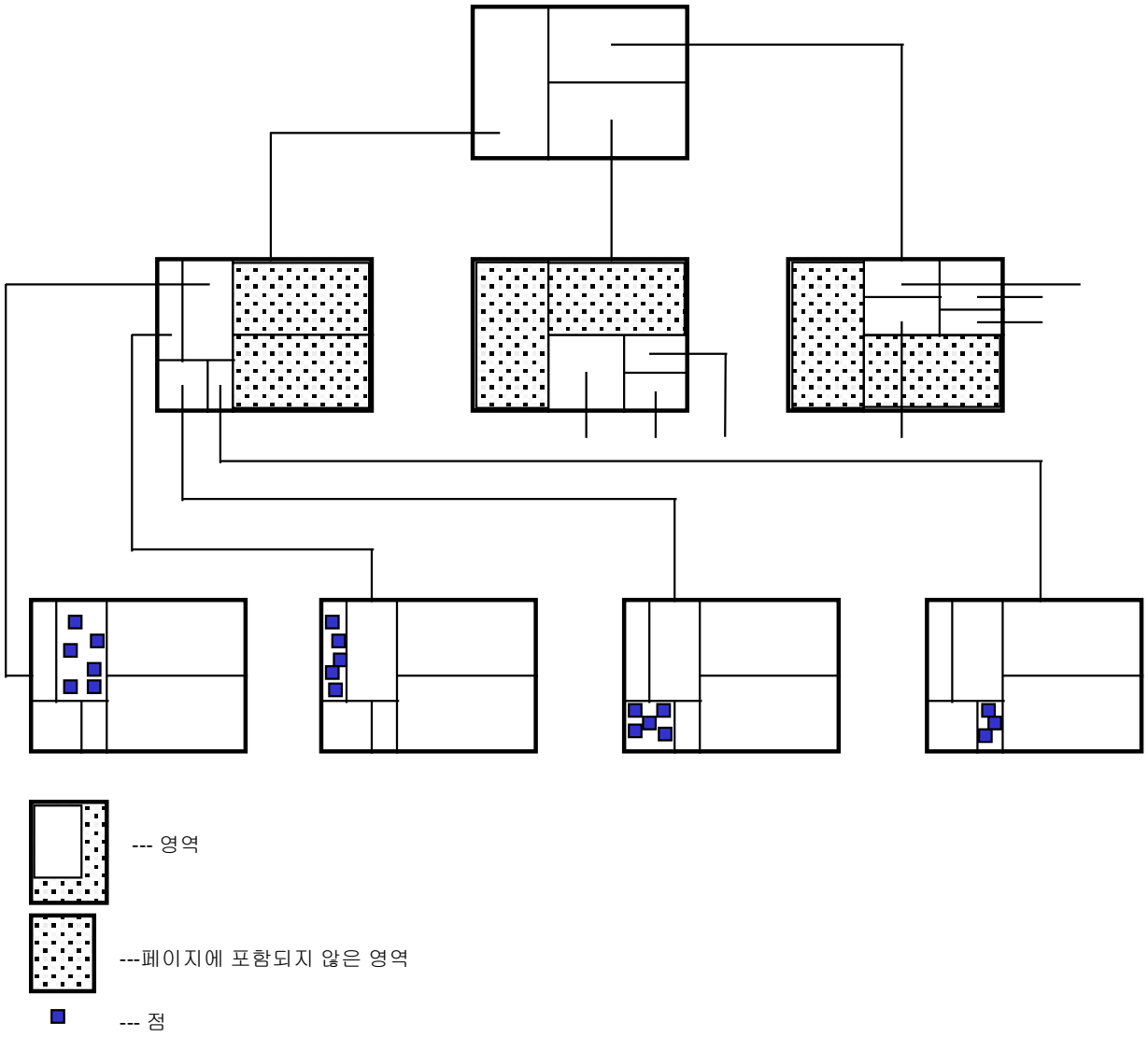
① 각 페이지를 노드, 페이지 **ID**를 노드 포인터로 갖는 다원 탐색 트리

② 모든 단말 페이지까지의 경로 길이는 동일

③ 모든 영역은 분리**(disjoint)**

④ 루트 페이지가 영역 페이지면, 영역들의 합은 영역 전체

▶ 2-D-B 트리



i) 질의

- **영역은 각 차원들의 간격의 교차곱(Ix×Iy)**
  - 부분 범위 질의 : 일부 간격들이 도메인 전체
  - 부분 일치 질의 : 일부 간격이 점이고 나머지가 전체 도메인
  - 완전 일치 질의 : 모든 간격들이 점

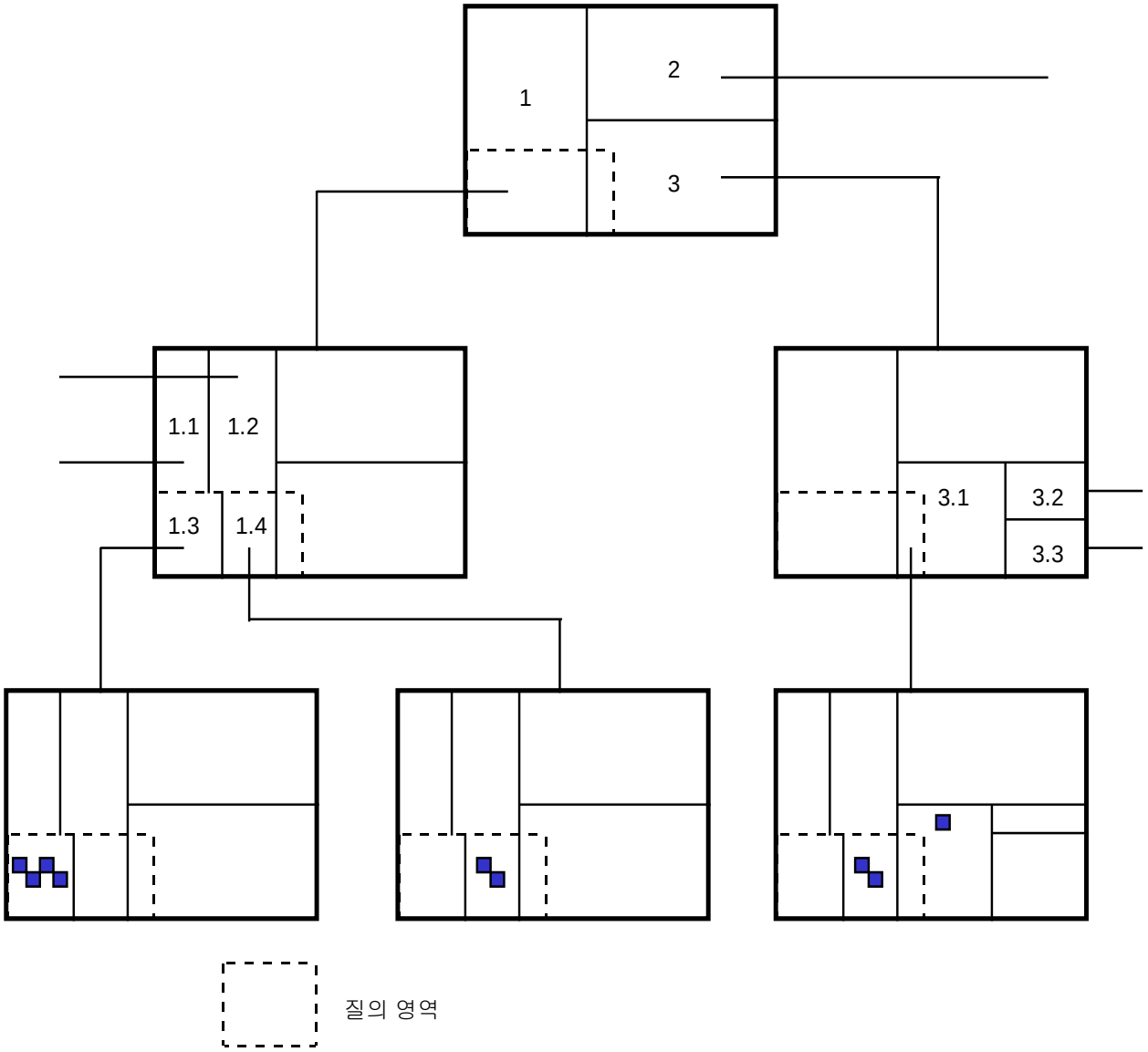
◆ 질의 알고리즘

① root-ID가 널이면 종료, 그렇지 않으면 변수 페이지는 루트 페이지를 가리키게 한다.

② 변수 페이지가 점 페이지를 가리키면 질의 영역에 속하는 <점, 주소>에 대해 주소에 있는 레코드를 검색, 출력

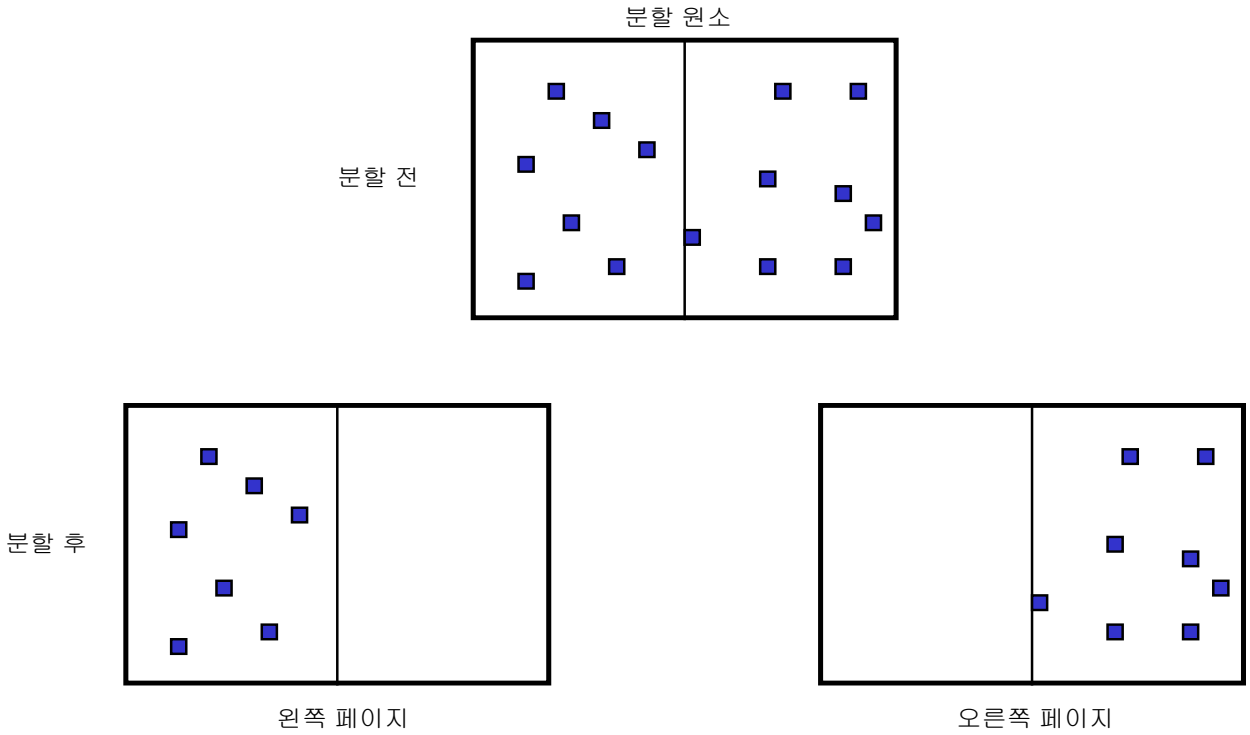
③ 영역 페이지인 경우는 <영역, 자식>에 대해 변수 페이지가 자식 ID에 의해 참조되는 페 이지를 가리키게 하고 ②에서 반복

질의 영역 검색 예



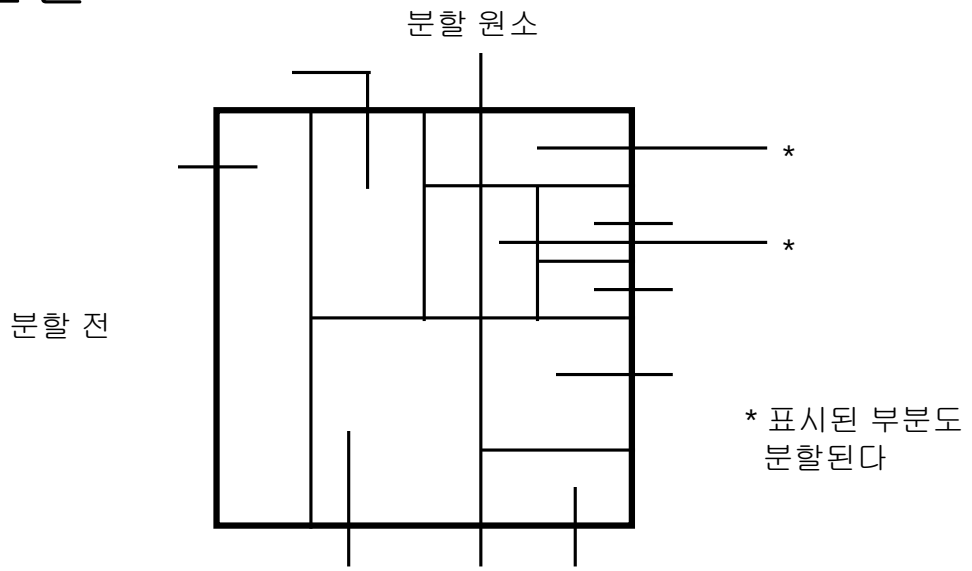
ii) 삽입

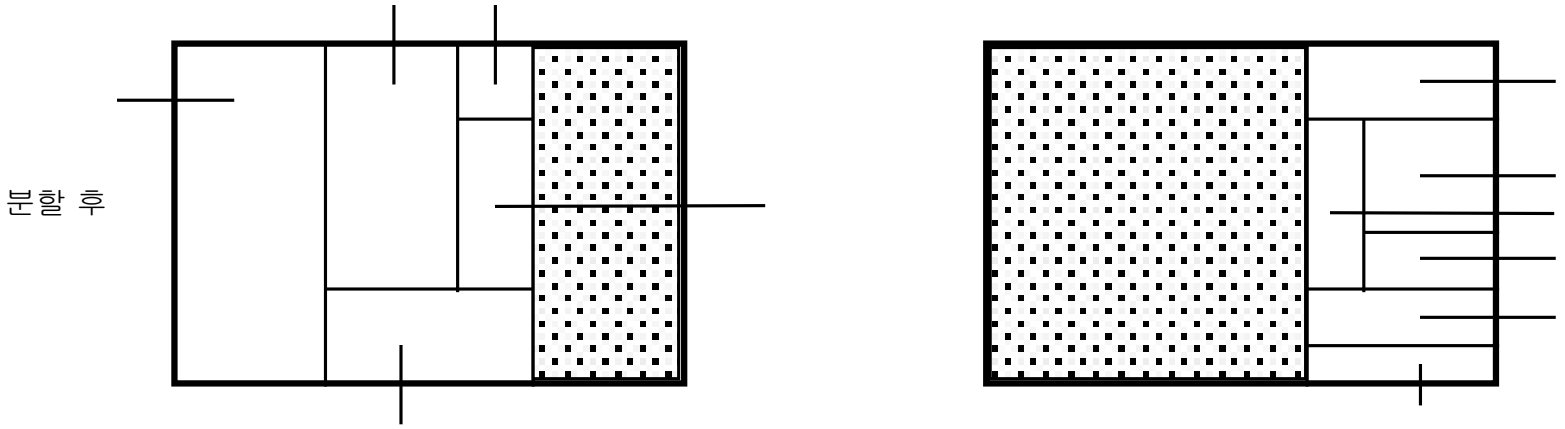
- 도메인의 값 X'이 영역에 포함되면 영역 분할
- 점 페이지 분할



- 원래 페이지 내의 모든 <점, 주소>쌍을 X'의 값에 따라 좌우 페이지로 이동한 후 원래 페이지 삭제

– 영역 페이지 분할





▶ 삽입 알고리즘

① **root-ID**가 널이면 <점, 위치>를 포함하는 점 페이지 생성

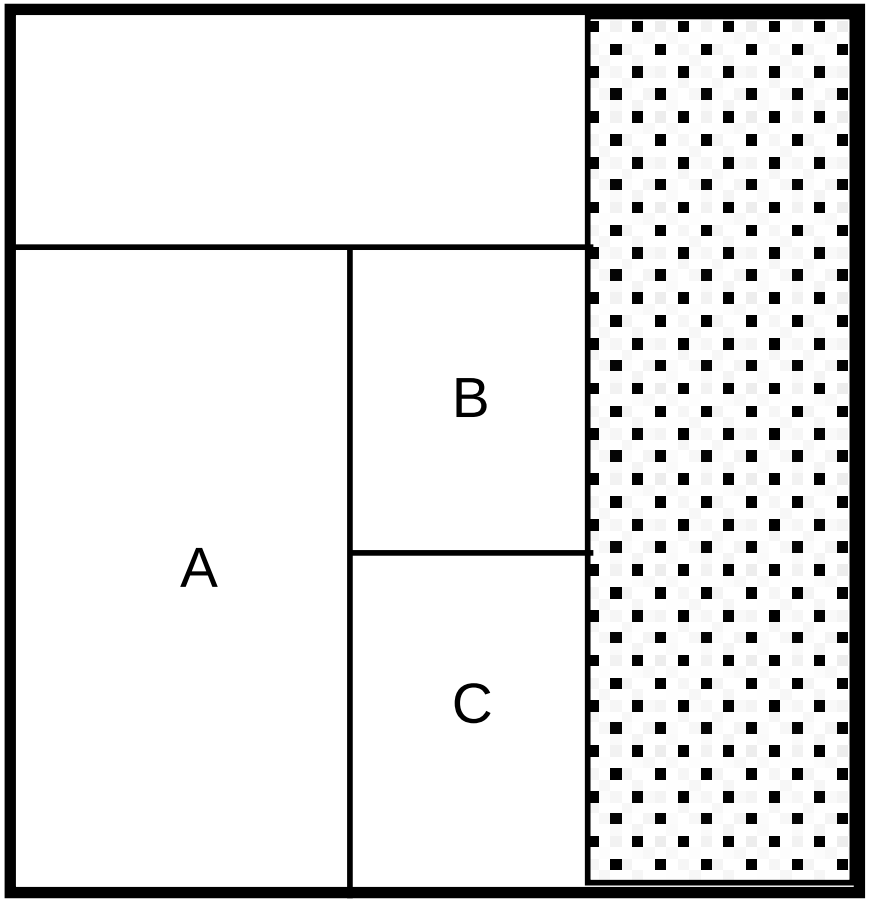
② 점이 첨가될 페이지 탐색(완전 일치 질의)

③ 점 페이지에 삽입하고 종료, 오버플로가 발생하면 분할

iii) 삭제와 재구성

- 완전 일치 질의로 탐색, 제거
- 공간 이용률을 높이기 위해 재구성
  - 접합 : 두 영역의 정보가 한 페이지로
  - 언더플로 : 두 영역간에 재분배
- 두 영역의 합이 영역이면 접합가능

▶ 접합이 불가능한 영역



❖ 사분트리

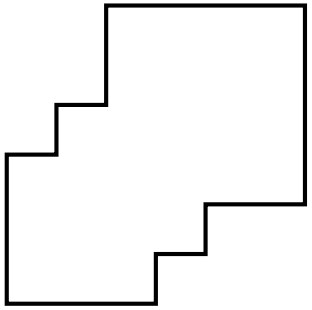
- 공간의 순환적 분해

◆ **사분트리의 분류 기준**
- 표현하고자 하는 자료의 유형
- 공간 분해 과정의 원칙
- 해상도(resolution) - 가변 또는 고정

▶ 영역 사분트리(region quadtree)

- 이차원의 영역 데이타 표현
- 이미지를 표현하는 이진수의 배열을 연속적으로 동일한 크기의 사분면들로 분할
- 루트 노드는 전체 배열, 자식 노드들은 각각 영역의 사분면 표현(**NW, NE, SW, SE**)
- 리프 노드는 영역의 내부 또는 외부 표현

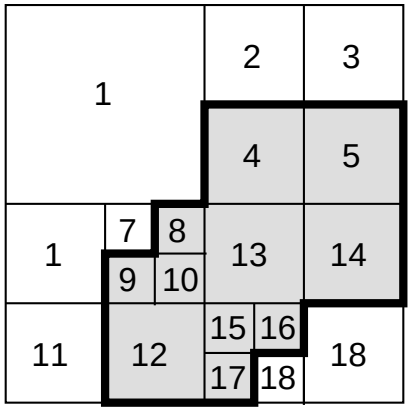
▶ 영역 사분트리를 이용한 이미지 표현



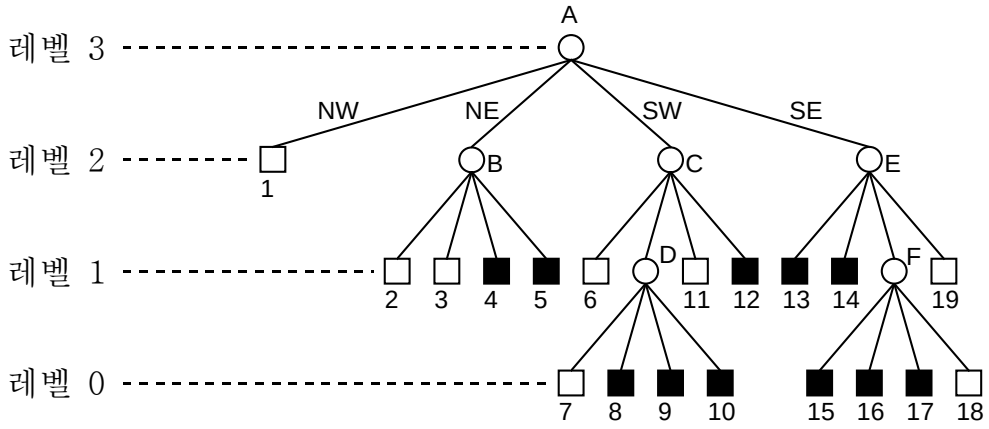
(a)

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	1	1	1	1
0	0	0	0	1	1	1	1
0	0	0	1	1	1	1	1
0	0	1	1	1	1	1	1
0	0	1	1	1	1	0	0
0	0	1	1	1	0	0	0

(b)



(c)



(d)

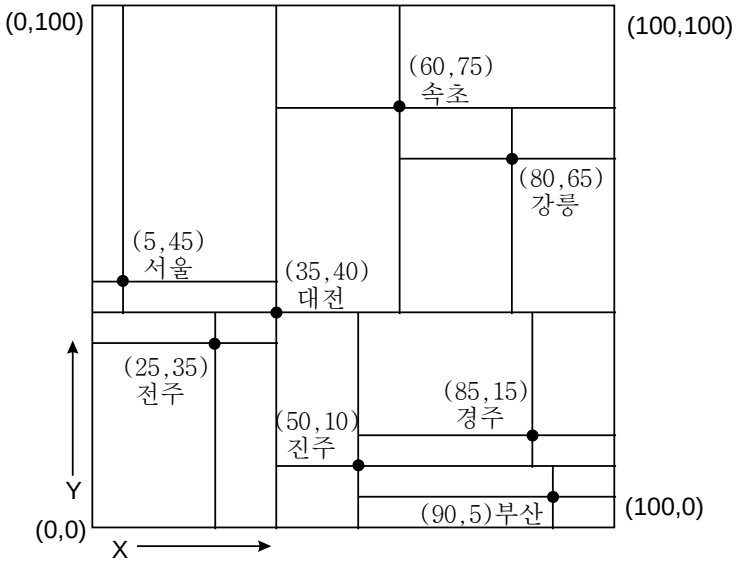
▶ 점 사분트리

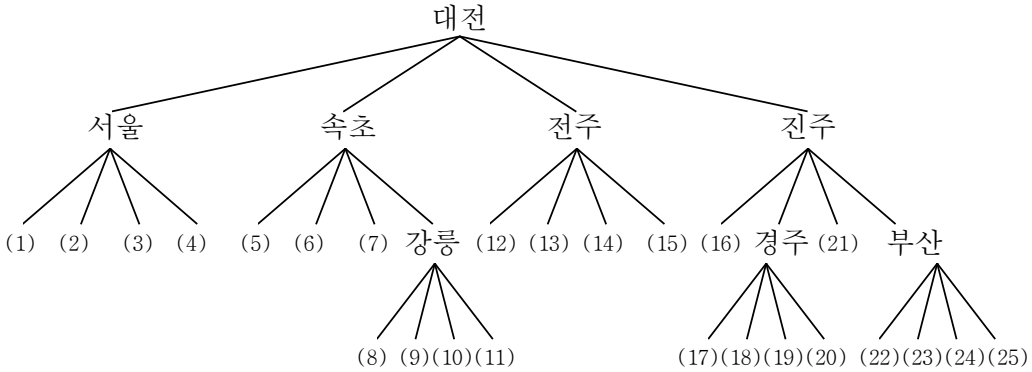
- 점 데이타의 표현
- 동일하지 않은 4개의 공간
- 이차원 점 데이타에 대한 인덱스로 활용
- 이진 탐색 트리의 일반화
- 이차원 점 데이타는 데이타 필드, x, y 좌표, 네 개의 포인터 필드를 가지는 노드로 표현

▶ 도시 데이타 레코드

도시명	X	Y	기타 정보
대전	35	40	…
진주	50	10	
속초	60	75	
강릉	80	65	
서울	5	45	
전주	25	35	
경주	85	15	
부산	90	5	

▶ 점 사분트리



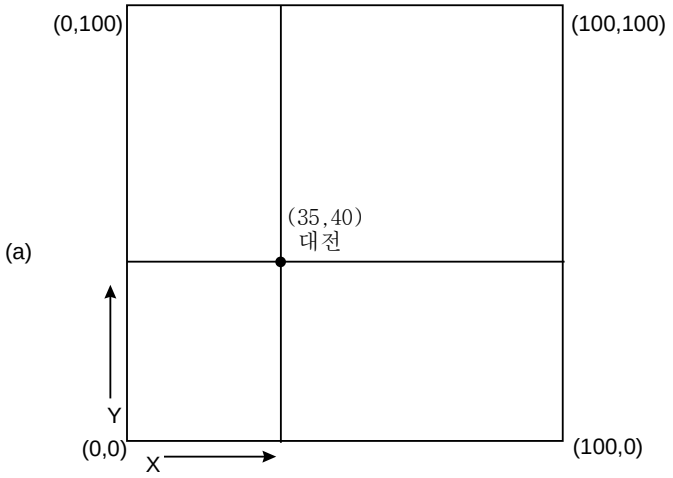


– 전체 레코드는 인덱스에 의해 참조되는 버켓에 저장

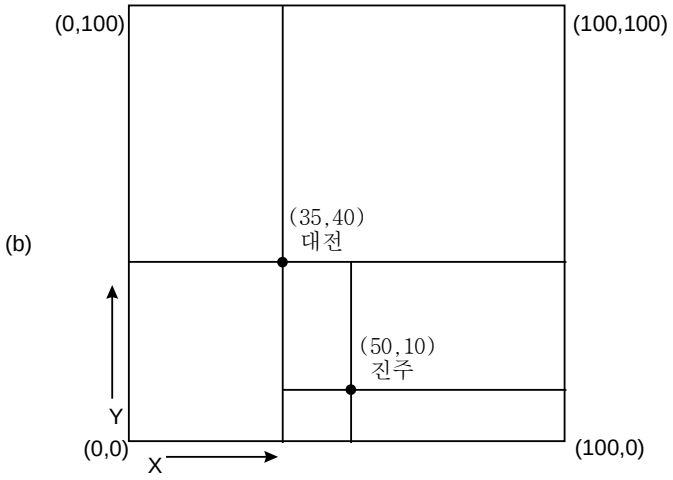
i) 삽입

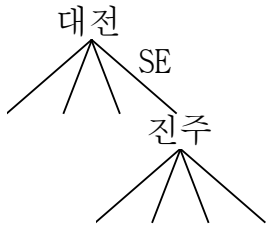
- **x, y** 좌표값에 의해 위치 탐색
- 리프 노드에 레코드 삽입
- 평균 삽입 비용 : $O(N\log_4 N)$
  탐색 비용 : $O(\log_4 N)$

▶ 점 사분트리의 삽입 예

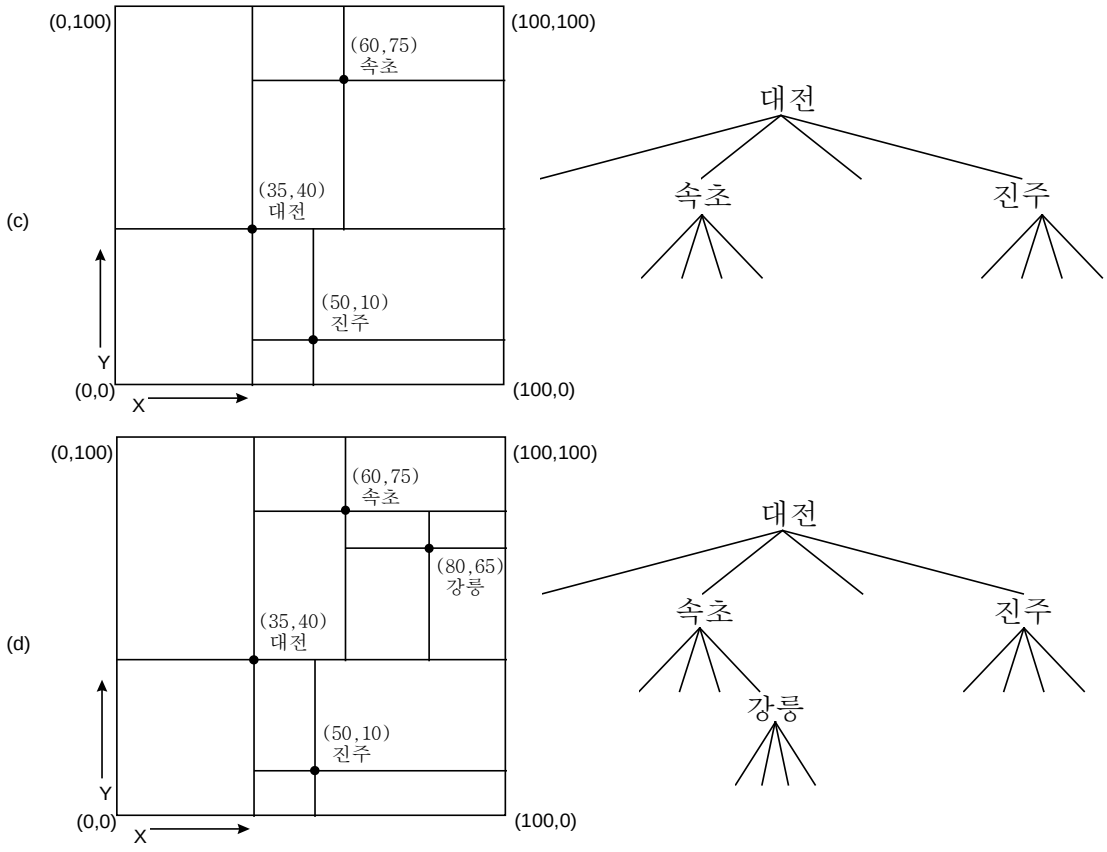

(a)
(0,100)
(100,100)
(35,40)
대전
Y
(0,0)
X
(100,0)


대전


(b)
(0,100)
(100,100)
(35,40)
대전
(50,10)
진주
Y
(0,0)
X
(100,0)


대전
SE
진주

▶ 점 사분트리의 삽입 예



- 최적 점 사분트리는 임의의 노드에 대해 어떤 서브트리도 전체 노드 수의 반 이상을 갖지 않는 트리

ii) 검색

- 탐색 공간은 사분의 일로 감소
- 범위 탐색, 근접 탐색에도 적합

iii) 삭제

- 삭제할 노드 **A(xA, yA)**를 노드 **B(xB, yB)**로 대치한 후 삭제
- 노드 **B**는 **x=xA**와 **x=xB**사이의 영역과 **y=yA**와 **y=yB**사이의 영역이 비어있는 노드
- 이런 **B**가 존재 않을 경우 영역에 존재하는 점 데이타들을 재삽입

❖ R-트리

– 데이타 객체를 여러 차원의 구간들에 의해 표현

◆ **R 트리 인덱스 구조**

– 인덱스 레코드로 구성되는 높이 균형 트리

– 리프 노드 : <I, 주소>

I = (I0, I1, …, IK-1)

K는 차원의 수, Ii는 폐쇄 경계 구간

– 중간 노드 : <I, 자식 포인터>

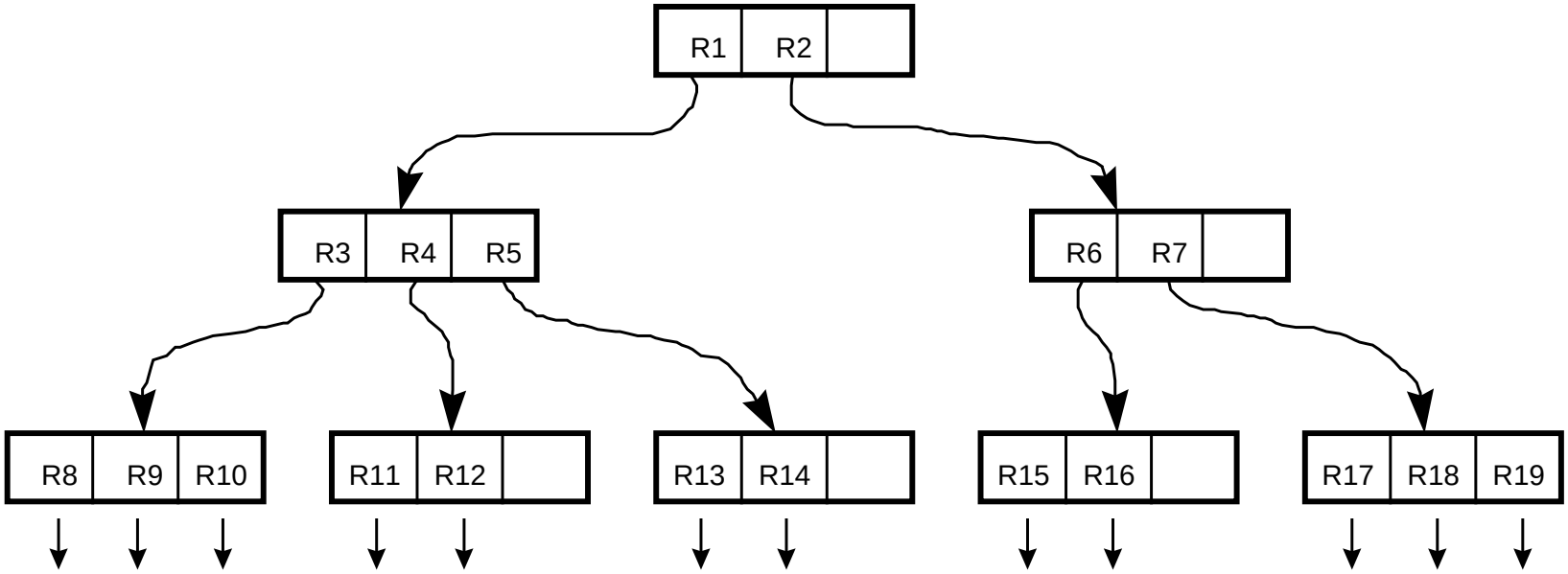
I는 하위 레벨 노드의 모든 사각형 포함

▶ R 트리의 특성

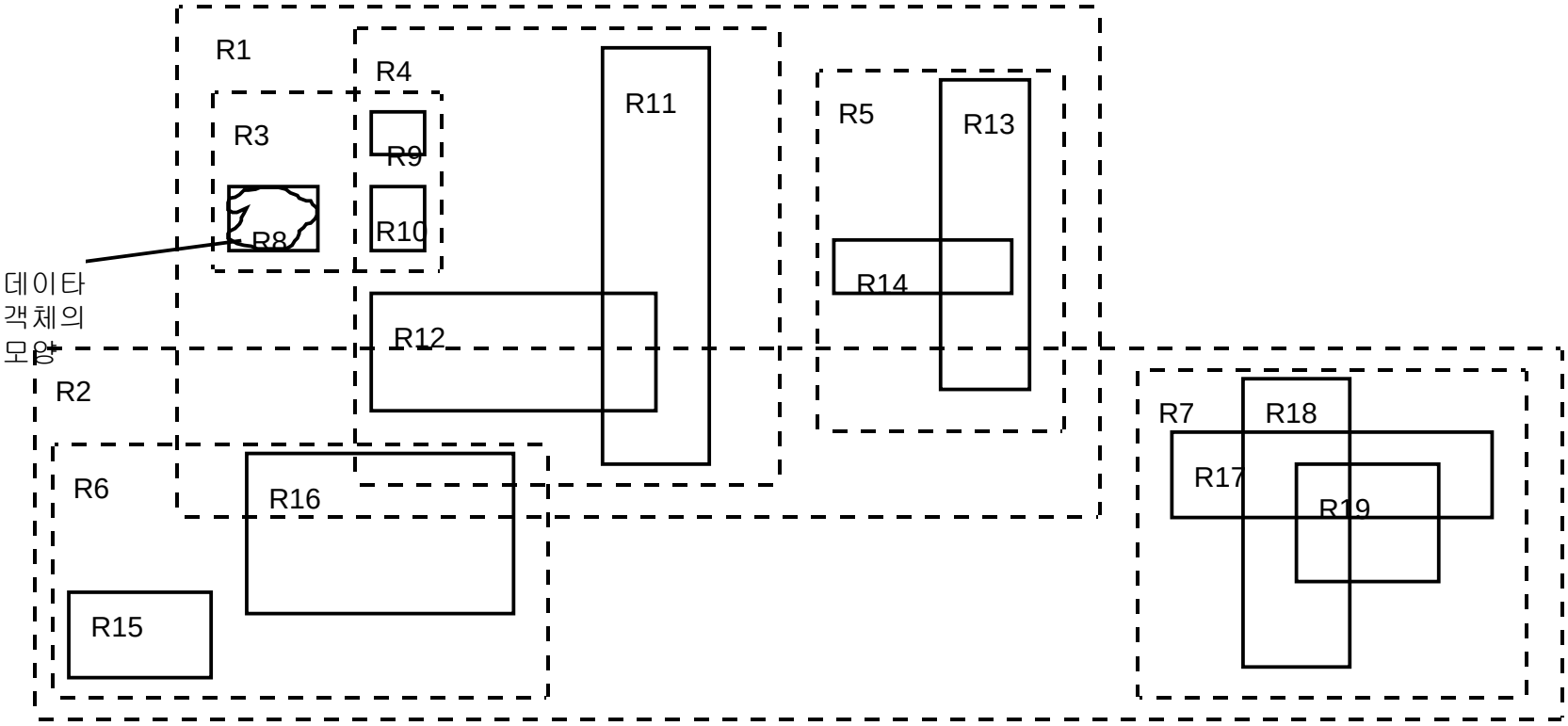
- 한 노드에 저장될 수 있는 엔트리의 최대수를 M이라 하고 m(≤M/2)이 엔트리의 최소수를 나타내는 매개변수일 때

  ① 모든 리프 노드는 루트가 아닌한 m과 M사이의 인덱스 포함
  ② 리프가 아닌 노드는 루트가 아닌한 m과 M사이의 자식 노드 포함
  ③ 루트 노드는 리프가 아니면 적어도 두개의 자식 노드
  ④ 모든 리프 노드는 동일 레벨에 위치

R- 트리의 예


R1
R2
R3
R4
R5
R6
R7
R8
R9
R10
R11
R12
R13
R14
R15
R16
R17
R18
R19
데이타 투플에 대한 포인터

(a)


R1
R4
R3
R9
R11
R5
R13
R8
R10
데이타
객체의
모임
R14
R12
R2
R7
R18
R6
R16
R17
R19
R15

(b)

i) 검색

- 인덱스 엔트리 E의 사각형 부분을 E.I로 하며 주소 또는 자식 포인터부를 E.p로 표시

- 루트가 T인 R 트리에서 탐색 사각형 S와 겹치는 공간 데이타 객체 탐색 과정

① 서브트리 탐색 : T가 리프가 아니면 E.I가 S와 겹치는 엔트리에 대해 E.p를 루트로 하는 서브트리에 대해 ①② 반복

② 리프 노드 탐색 : T가 리프이면 E.I가 S와 겹치면 E는 조건에 맞는 레코드를 가리키는 인덱스

ii) 삽입

- 새로운 인덱스 엔트리가 리프에 첨가, 오버플로시 분할
- R 트리에 인덱스 엔트리 E 삽입 과정

① 리프 노드 L 선택, L은 기존 경계 사각형을 최소 확장

② L에 공간이 있으면 E를 위치, 그렇지 않으면 분할

③ 분할시 트리 재구성, 부모 노드가 포함하는 사각형이 조정, 분할의 영향은 루트 방향으로 전달

iii) 삭제

- 삭제될 엔트리를 포함하는 노드를 찾아서 삭제
- 언더플로 발생시 노드를 제거하고 엔트리들을 트리에 재삽입

▶ R 트리의 분석

- **K** 차원의 데이타를 처리하기 위한 **B** 트리의 확장
- 중간 노드와 리프 노드로 구성된 높이 균형 트리
- 포함과 겹침 관계
- 최소 포함은 죽은 공간**(dead space)**의 양 감소
- **N**개의 인덱스를 갖는 **R** 트리의 높이는 최대 $\log_m N - 1$ **(**각 노드의 분기율이 적어도 **m)**
- 노드의 최대수 $N/m + N/m^2 + \ldots + 1$
- 루트를 제외한 모든 노드에서 최악의 공간 활용도 **(space utilization)**는 **m/N**

❖ R* 트리

- R 트리의 변형으로 삽입이나 삭제시 부모 노드의 사각형이 효율적으로 확장

◆ **R* 트리의 고려 사항**

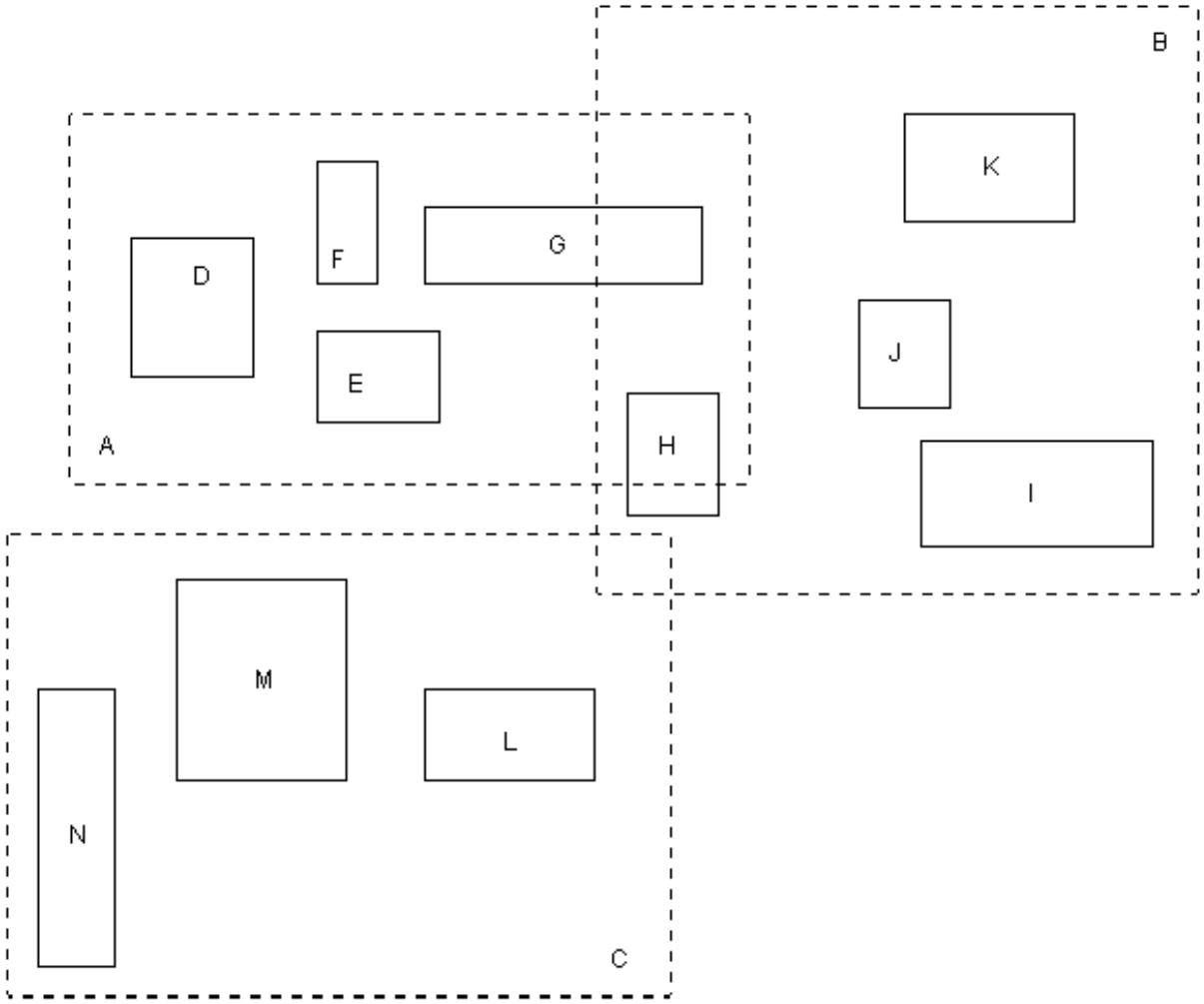
① 사각형의 면적 최소화
② 사각형들간의 겹침을 최소화
③ 사각형의 둘레 길이의 합을 최소화
④ 기억 장소 이용률을 최적화

- 오버플로 발생시 강제적 재삽입으로 사각형을 정사각형에 가깝게 조정
- 구현 비용에서는 R 트리보다 비효율적
- 점 데이타의 탐색, 탐색 윈도와 객체의 영역이 일치하는 객체의 탐색, 탐색 윈도의 크기가 작은 경우등에서 좋은 성능

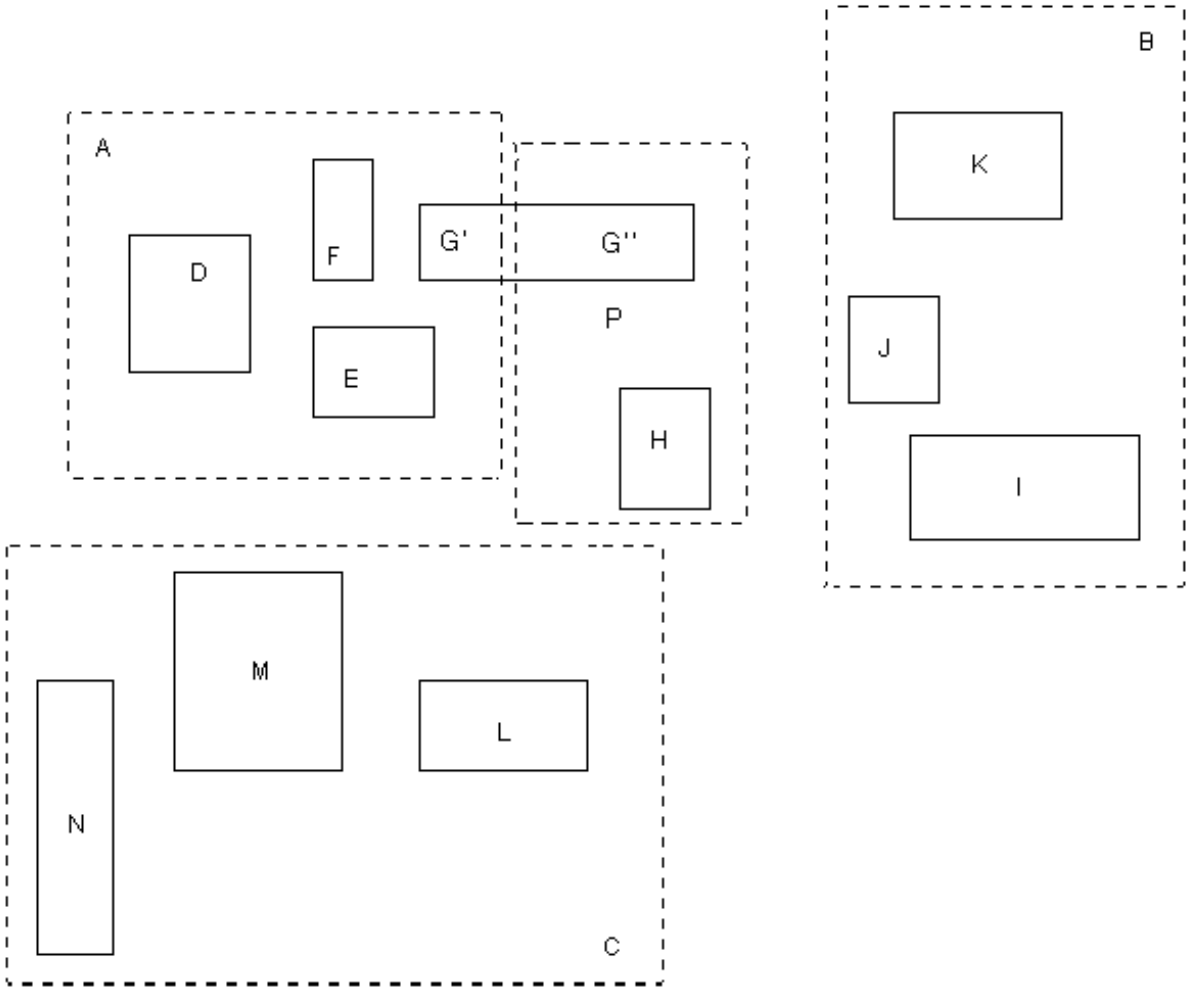
❖ R^+ 트리

- R^+ 트리는 겹침 관계를 제거한 R 트리
- 제로 겹침은 데이타가 점으로 표현
- 하나의 자료 사각형을 나누어 분할을 구성하면 중간 노드의 엔트리들에서 제로 겹침

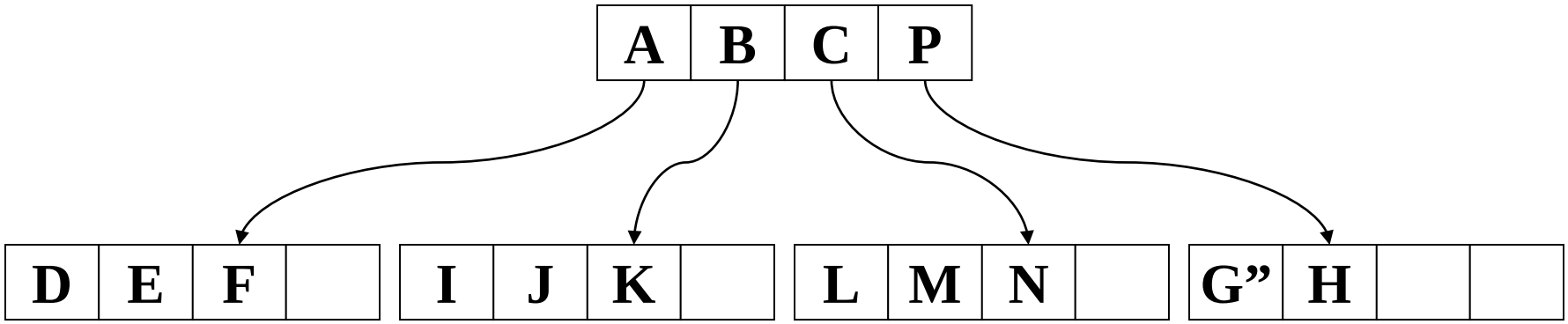
▶ R 트리를 위한 사각형들



▶ R^+ 트리를 위해 그룹핑한 예



▶ 구성된 R+ 트리



- 겹침을 없애는 대신 높이 증가
- **K-D-B** 트리에 비해 포함관계 감소
- **R** 트리에 비해 추가 공간이 필요한 대신 좋은 탐색 성능