Introduction to BSCA package

서영오

서울대학교 통계학과

R 패키지 BSCA 개발 및 활용방안

R User Conference in Korea 2018

서영오

서울대학교 통계학과 다중척도연구실
collaboration with 박민정 (통계청 통계개발원)

목차

Introduction

Intro

- 본 발표에서 다루는 데이터는 지역의 행정계층에 따라 집계되는 여러 범주형 변수 조합에 대한 빈도 데이터이다.

시/도	시/군/구	동/읍/면	상세주소	연령	빈도수
서울특별시	관악구	봉천동	1597번지	20대	10
⋮	⋮	⋮	⋮	⋮	⋮

- 지역변수를 지징할 때 area 변수라고 하고 상위계층에서 하위계층 순으로 LA1, LA2, LA3, OA라 명명한다.
- 연령, 결혼상태, 교육수준 등 한 개인의 특성을 나타내는 범주형 변수를 key 변수라고 명명한다.

Intro

- 특정 area 수준에서 주어진 모든 key 변수의 조합에 따라 다음과 같은 빈도표를 만들 수 있는데, 이를 full table이라고 한다.

성별	결혼상태	Frequency
1	1	f_1
1	2	f_2
1	3	f_3
2	1	f_4
2	2	f_5
2	3	f_6

Input 자료구조

	LA1	LA2	LA3	OA	Key1	Key2
1	서울특별시	관악구	봉천동	1597번지	1	A
⋮	⋮	⋮	⋮	⋮	⋮	⋮
N	경기도	남양주시	진접읍	1300번지	2	B

- area변수 4개와 key변수 2개에 대하여 관측치가 N개 있는 빈도표.
- key1 변수의 범주는 1, 2이고, key2변수의 범주는 A, B, C가 있다고 가정.

Output 자료구조

서울특별시

key1	key2	빈도 (BSCA)
1	A	
1	B	
1	C	
2	A	
2	B	
2	C	

$\cdots$

1300번지

key1	key2	빈도 (BSCA)
1	A	
1	B	
1	C	
2	A	
2	B	
2	C	

- 모든 지역 범주에 대하여 모든 key변수 조합에 대한 BSCA 알고리즘이 적용된 빈도표를 만드는게 목적

작은 셀 조정(Small Cell Adjustment, SCA)

- $B = 5$: 5미만의 작은 셀은 0 또는 5로 반올림 한다.
- Margin, 상위 수준 빈도값은 full table(지역수준 OA에서 모든 변수들에 대한 빈도표)을 집계하여 계산한다.
- Magin을 구하는 방법에 따라 정보손실이 크거나, 숨기려던 작은 셀의 값이 드러날 수 있다.

original table

	M	F	
A	1	0	1
B	3	3	6
C	12	20	32
	16	23	39

masked table

	M	F	
A	0	0	0
B	0	5	6
C	12	20	32
	16	23	39

SCA의 정보손실 ($B = 3$)

-의도

$T/\tilde{T}$	$\tilde{f} = 0$	$\tilde{f} = 3$
$f = 1$	-1	2
$f = 2$	-2	1

SCA의 정보손실 ($B = 3$)

-실제

	OA				LA3			
IL	0	1	2	3+	0	1	2	3+
$p = 4$	93.6%	4.3%	2.1%	0.0%	73.0%	16.7%	8.1%	2.2%
$p = 3$	81.8%	10.4%	5.9%	1.8%	54.4%	18.9%	12.8%	13.9%
$p = 2$	55.7%	19.1%	12.8%	12.4%	28.5%	17.5%	14.2%	39.7%
$p = 1$	19.0%	19.2%	15.9%	45.9%	5.7%	10.2%	9.0%	75.1%

	LA2				LA1			
IL	0	1	2	3+	0	1	2	3+
$p = 4$	50.1%	11.9%	10.3%	27.8%	41.4%	8.2%	7.7%	43.0%
$p = 3$	36.2%	9.4%	8.3%	46.2%	30.8%	5.0%	4.8%	59.4%
$p = 2$	17.4%	5.7%	6.1%	70.8%	15.08%	1.2%	2.1%	81.0%
$p = 1$	1.8%	1.2%	3.6%	93.3%	3.0%	0.0%	0.0%	97.0%

BSCA 알고리즘

1. 각 OA마다 P개의 변수로 구성된 기저 빈도표를 생성한다.
2. T_{OA}^{vP} 각 빈도표마다 작은 셀들을 조정해 얻은 빈도표를 $\tilde{T}_{OA}^{vP}$라 하자.
3. $\tilde{T}_{OA}^{vP}$를 집계하여 $\tilde{T}_{LA}^{vp}$를 얻는다. (즉, $\tilde{f}_i = \sum \tilde{f}_{i,j}$)

4. $|\tilde{f}_i - f_i| \geq B$인 모든 셀에 대해
   - 4.1 다음과 같이 fixed grid rule을 적용해 $\tilde{f}_i$값을 업데이트 한다.
     - A. $f_i - 1 = a_i B + b_i, \;\; a_i \in Z, \;\; b_i \in \{0, 1, \ldots, (B-1)\}$
     - B. $\tilde{f}_i - 1 = a_i B + [B/2]$ 즉, $\tilde{f}_i = a_i B + [B/2] + 1$로 업데이트 한다.
   - 4.2 노출을 피하기 위해 B를 더하거나 빼어 $\tilde{f}_i$값을 업데이트 한다.
     - A. $\tilde{f}_i - [B/2] < k_i$ 이면 $\tilde{f}_i = \tilde{f}_i + B$
     - B. $u_i < \tilde{f}_i + [B/2]$ 이면 $\tilde{f}_i = \tilde{f}_i - B$

BSCA 적용결과

	OA				LA3			
IL	0	1	2	3+	0	1	2	3+
$p = 4$	93.6%	4.3%	2.1%	0.0%	73.2%	18.1%	8.4%	0.4%
$p = 3$	82.0%	11.5%	6.3%	0.2%	57.5%	28.8%	13.3%	0.4%
$p = 2$	58.2%	27.3%	14.0%	0.5%	39.7%	45.4%	14.7%	0.3%
$p = 1$	31.5%	51.8%	16.5%	0.2%	30.0%	61.0%	9.0%	0.0%

	LA2				LA1			
IL	0	1	2	3+	0	1	2	3+
$p = 4$	57.6%	31.7%	10.3%	0.4%	55.4%	36.8%	7.6%	0.1%
$p = 3$	50.7%	40.7%	8.4%	0.2%	49.6%	44.4%	5.8%	0.2%
$p = 2$	39.4%	55.1%	5.4%	0.1%	44.4%	53.3%	2.4%	0.0%
$p = 1$	34.6%	63.6%	1.8%	0.0%	39.4%	60.6%	0.0%	0.0%

알고리즘 Process

Process 1

Input data

	LA1	LA2	LA3	OA	Key1	Key2
1	서울특별시	관악구	봉천동	1597번지	1	A
⋮	⋮	⋮	⋮	⋮	⋮	⋮
N	경기도	남양주시	진접읍	1300번지	2	B

서울특별시

key1	key2	빈도
1	A	
1	B	
1	C	
2	A	
2	B	
2	C	

⋯

1300번지

key1	key2	빈도
1	A	
1	B	
1	C	
2	A	
2	B	
2	C	

tb

1. Input data를 모든 지역변수와 key변수 조합에 대하여 자료구조 변환한다. 이 때, 각 지역별로 만들어진 table들을 tb라고 한다.

Process 2

OA_1

key1	key2	빈도
1	A	
1	B	
1	C	
2	A	
2	B	
2	C	

$\cdots$

OA_m

key1	key2	빈도
1	A	
1	B	
1	C	
2	A	
2	B	
2	C	

OA_1

key1	key2	빈도 (SCA 적용)
1	A	
1	B	
1	C	
2	A	
2	B	
2	C	

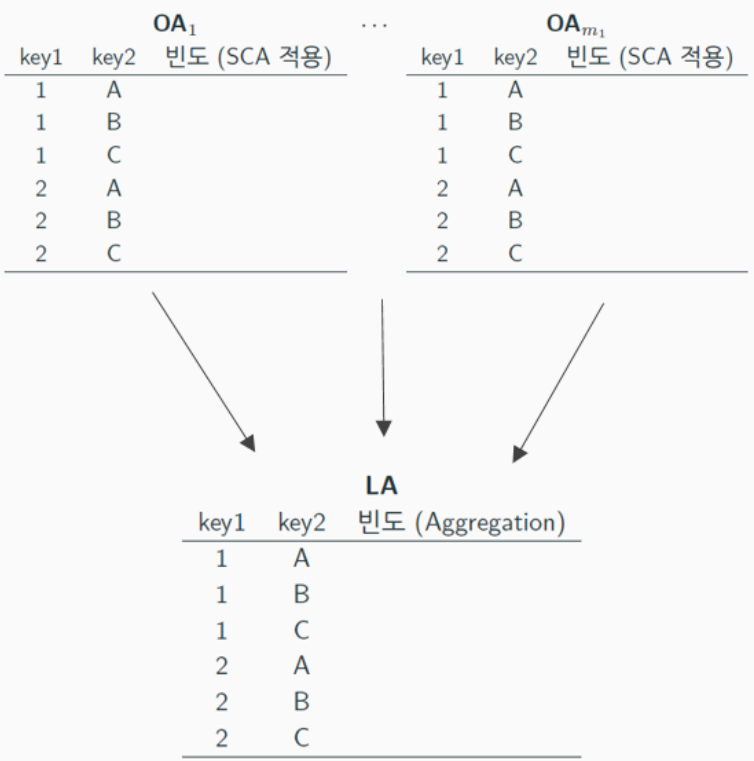
$\cdots$

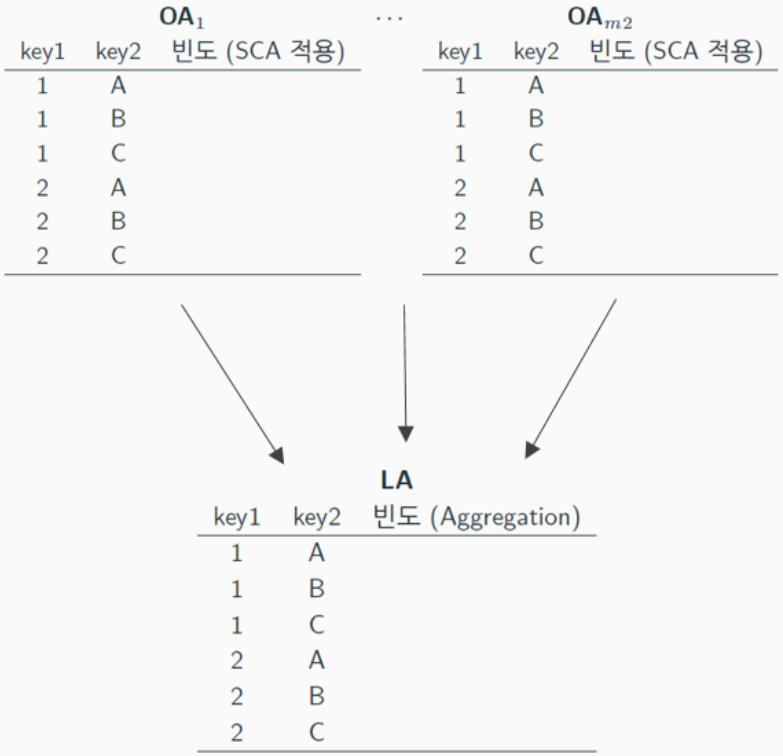
OA_m

key1	key2	빈도 (SCA 적용)
1	A	
1	B	
1	C	
2	A	
2	B	
2	C	

2. 1에서 만든 tb들 중에서 OA수준의 tb들에 대하여 SCA를 적용하여 빈도수를 변환한다.

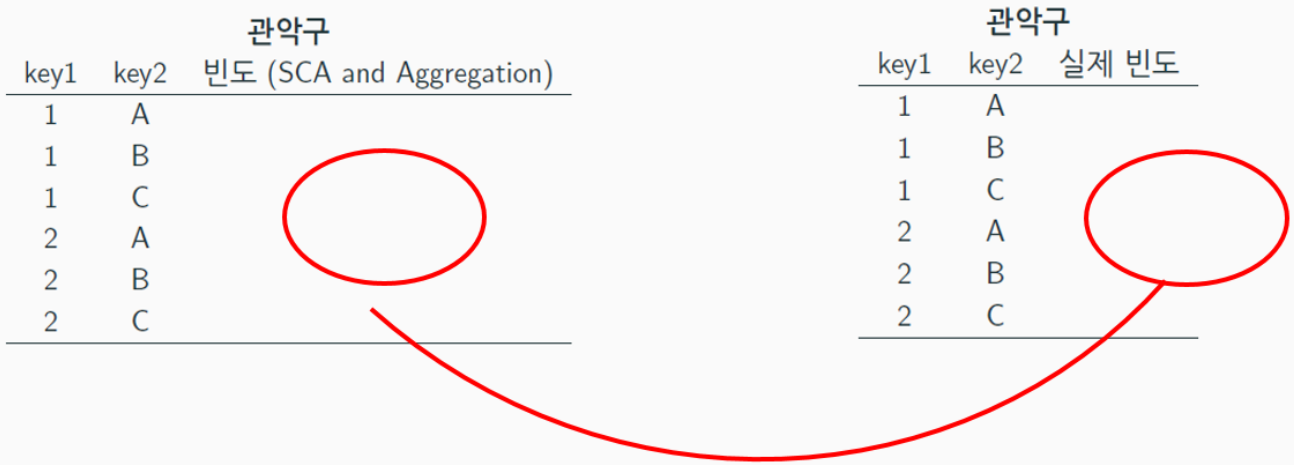
Process 3





3. SCA를 적용한 OA들을 aggregation하여 특정 LA수준의 빈도표를 만든다.

Process 4



관악구

key1	key2	빈도 (BSCA)
1	A	
1	B	
1	C	
2	A	
2	B	
2	C	

Output

4. 이제 각 각의 지역들에 대하여 실제 빈도수를 가지는 빈도표와 SCA 기법을 적용하여 aggregation한 새로운 빈도표가 만들어졌다. 두 빈도표의 빈도수를 비교하여 빈도차이가 3이상인 빈도수를 가지는 cell에 BSCA 알고리즘을 적용하여 새로운 빈도표를 만든다.

Key idea

1. 자료구조를 계층적으로 저장하여 탐색에 걸리는 시간을 단축한다.

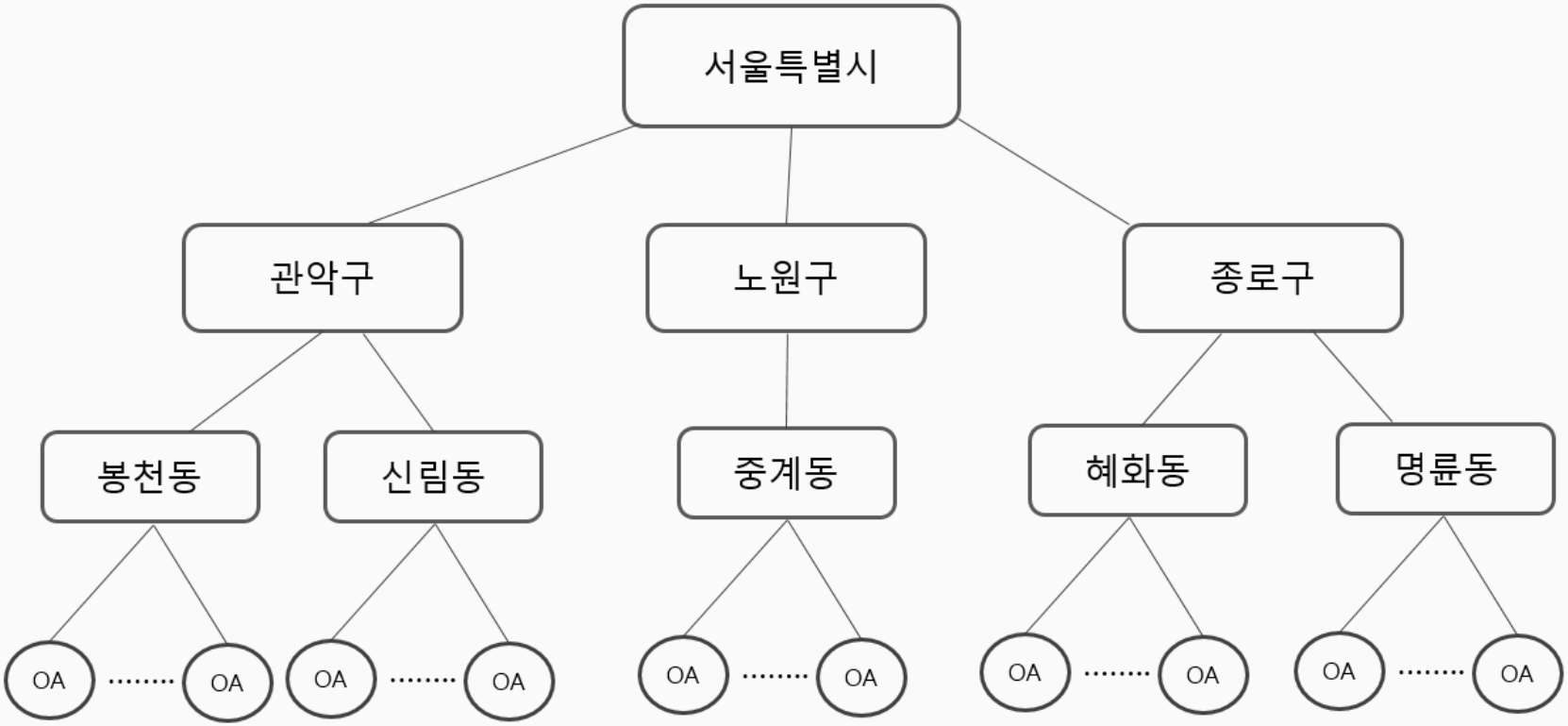
2. 재귀함수를 사용하여 지역의 개수를 4개 이상으로 좀 더 쉽게 일반화한다.

3. sapply와 lapply 함수를 적절하게 사용하여 for문으로 인한 시간을 좀 더 단축한다.

4. meta data를 이용하여, Process 3의 aggregation을 좀 더 효율적으로 한다.

Key idea 1 : 자료구조의 계층성 이용

자료구조의 계층성



- Input data의 자료구조가 위와 같다고 가정하자.

자료구조의 계층성

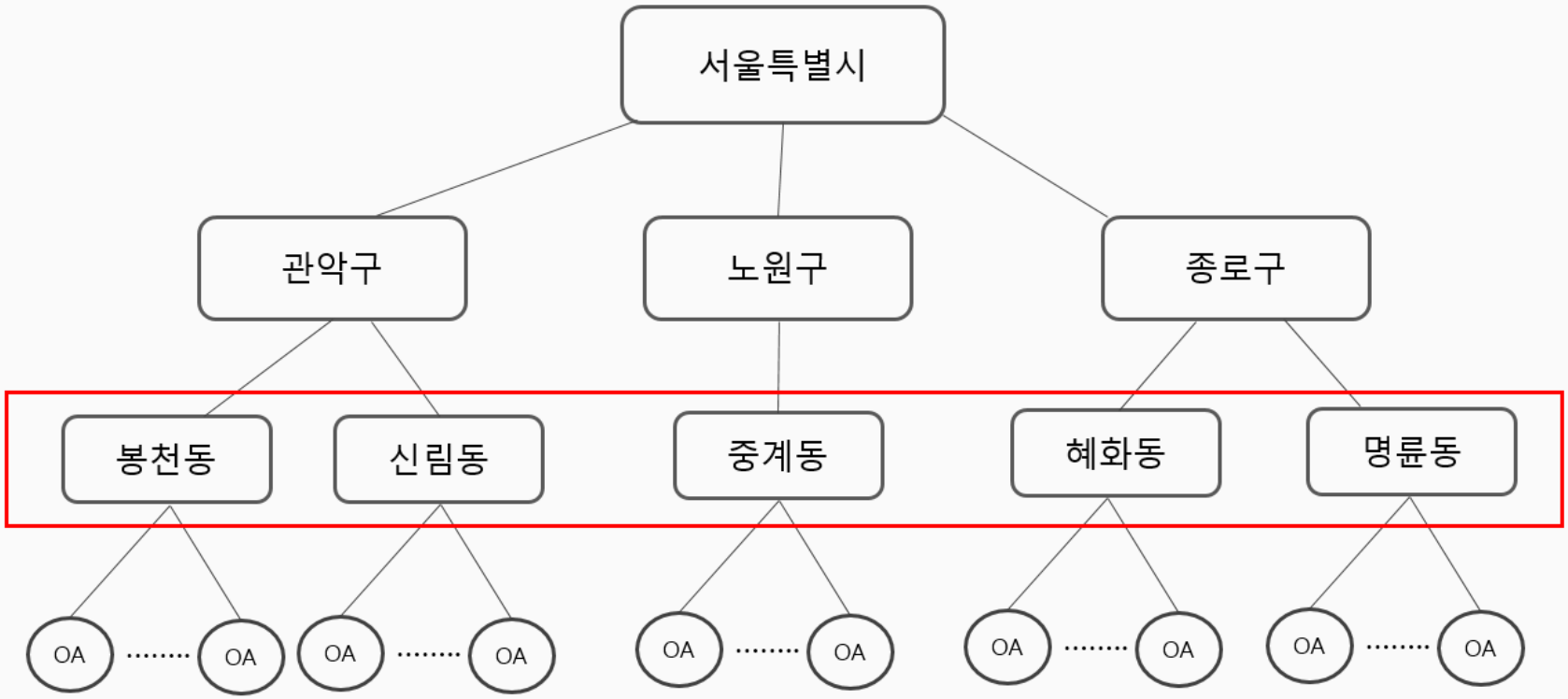
	LA1	LA2	LA3	OA	Key1	Key2
1	서울특별시	관악구	봉천동	1번지	1	A
2	서울특별시	종로구	혜화동	5번지	1	A
3	서울특별시	관악구	신림동	3번지	2	B
4	서울특별시	관악구	봉천동	10번지	1	C
⋮	⋮	⋮	⋮	⋮	⋮	⋮
N	서울특별시	노원구	중계동	32번지	2	A

	LA1	LA2	LA3	OA	Key1	Key2
1	서울특별시	관악구	봉천동	1번지	1	A
2	서울특별시	관악구	봉천동	10번지	1	C
3	서울특별시	관악구	봉천동	12번지	2	B
4	서울특별시	관악구	봉천동	20번지	1	B
⋮	⋮	⋮	⋮	⋮	⋮	⋮
K	서울특별시	관악구	봉천동	31번지	2	A

- Process 1에서 각 지역별로 tb를 만들기 위해 위 그림과 같이 input data에서 각 지역을 찾아 정렬하는 경우 크게 두 가지 방법이 있다. 위의 예시는 봉천동의 tb를 만들기 위해 input data를 정렬하는 경우.

자료구조의 계층성

첫째, data[data[, 3] == '봉천동',] 식으로 input data에서 탐색을 실시하는 것이다. 이 때, 탐색은 다음의 범위에서 이루어진다.

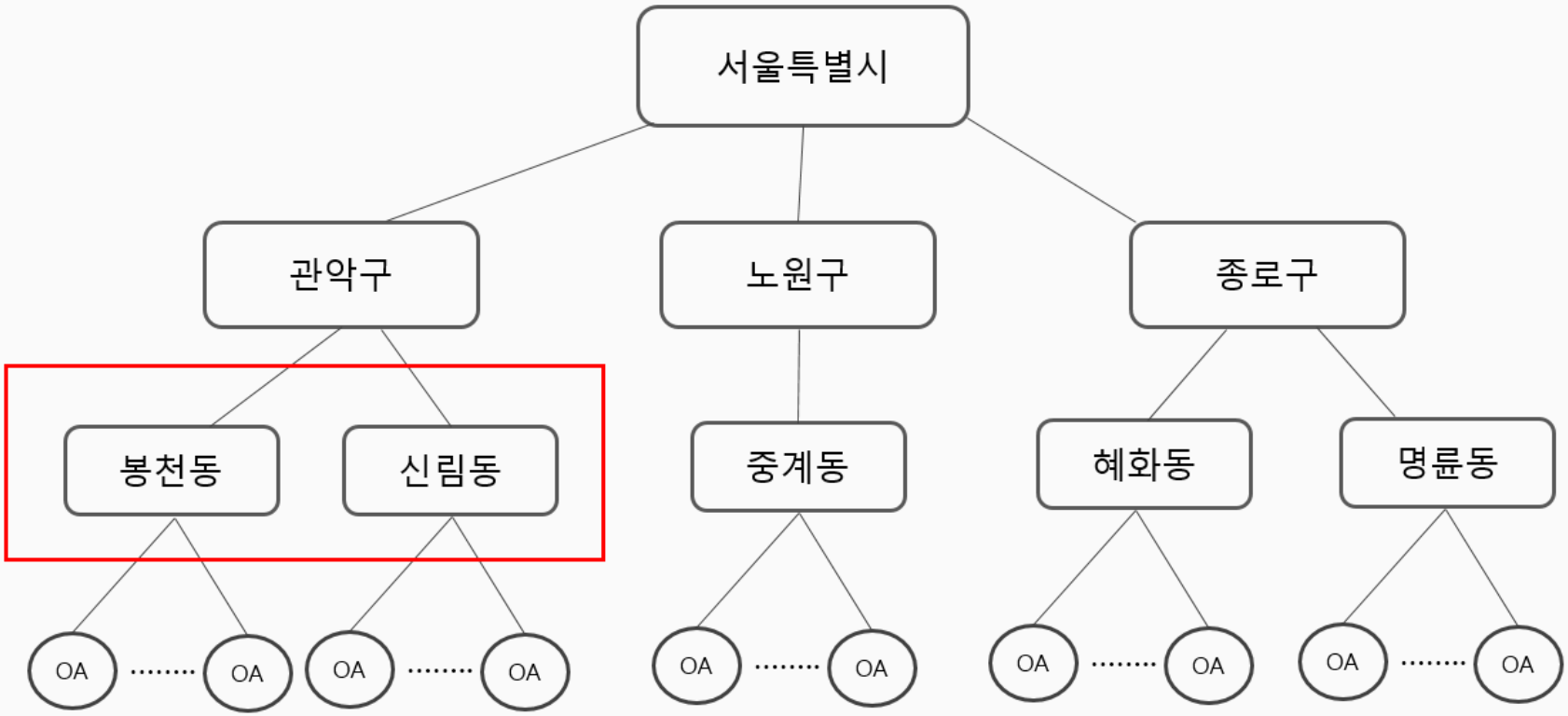


자료구조의 계층성

	LA1	LA2	LA3	OA	Key1	Key2
1	서울특별시	관악구	봉천동	1번지	1	A
2	서울특별시	종로구	혜화동	5번지	1	A
3	서울특별시	관악구	신림동	3번지	2	B
4	서울특별시	관악구	봉천동	10번지	1	C
⋮	⋮	⋮	⋮	⋮	⋮	⋮
N	서울특별시	노원구	중계동	32번지	2	A

자료구조의 계층성

둘째, 탐색의 범위를 미리 지정해주는 것이다. 탐색의 범위가 관악구로 지정이 되어있다면, 탐색은 다음과 같이 이루어 진다.



자료구조의 계층성

	LA1	LA2	LA3	OA	Key1	Key2
1	서울특별시	관악구	봉천동	1번지	1	A
2	서울특별시	관악구	신림동	3번지	2	B
3	서울특별시	관악구	봉천동	10번지	1	C
4	서울특별시	관악구	봉천동	23번지	2	A
⋮	⋮	⋮	⋮	⋮	⋮	⋮
n	서울특별시	관악구	신림동	1200번지	1	C

자료구조의 계층성

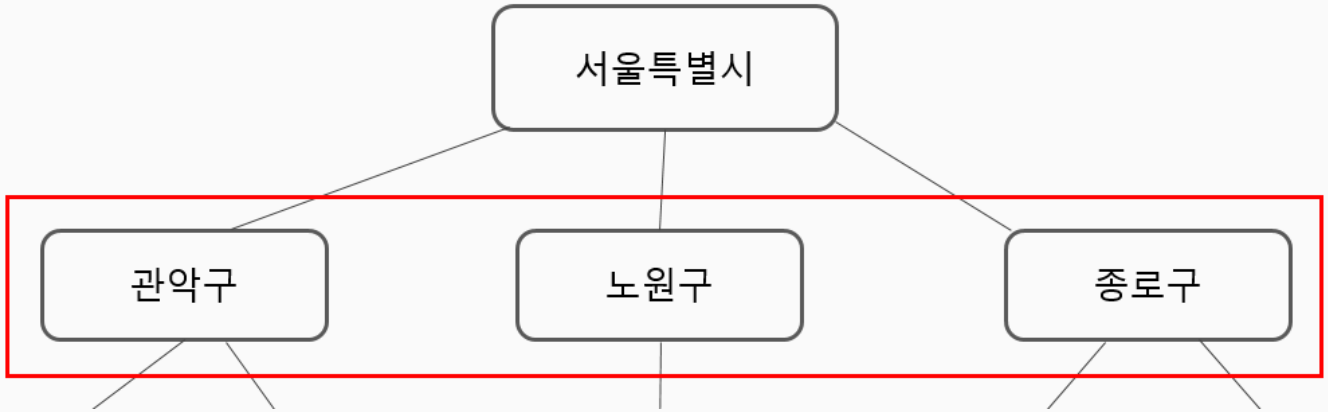
- 첫번째 방법으로 Process 1을 구현하였을 때, 처리시간이 480초가 걸린 반면, 두번째 방법으로 Process 1을 구현하였을 때, 처리시간이 186초 정도 걸리는 것을 확인 할 수 있었다.

- 두번째 방법으로 구현을 하는 것이 훨씬 효율적이라는 것을 알 수 있다.

자료구조의 계층성

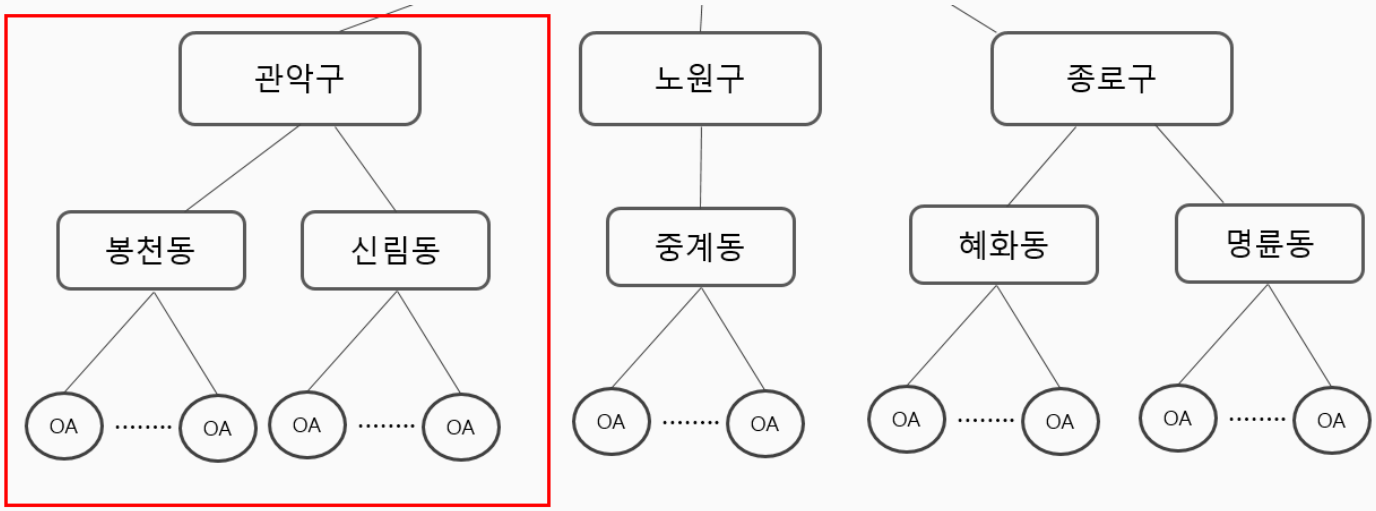
두번째 방법을 사용하기 위해서는 다음과 같이 탐색을 진행한다.

1. 우선 서울특별시의 tb를 만들면서 동시에 서울특별시 안의 3개의 구를 c(관악구, 노원구, 종로구) 형식으로 저장한다.



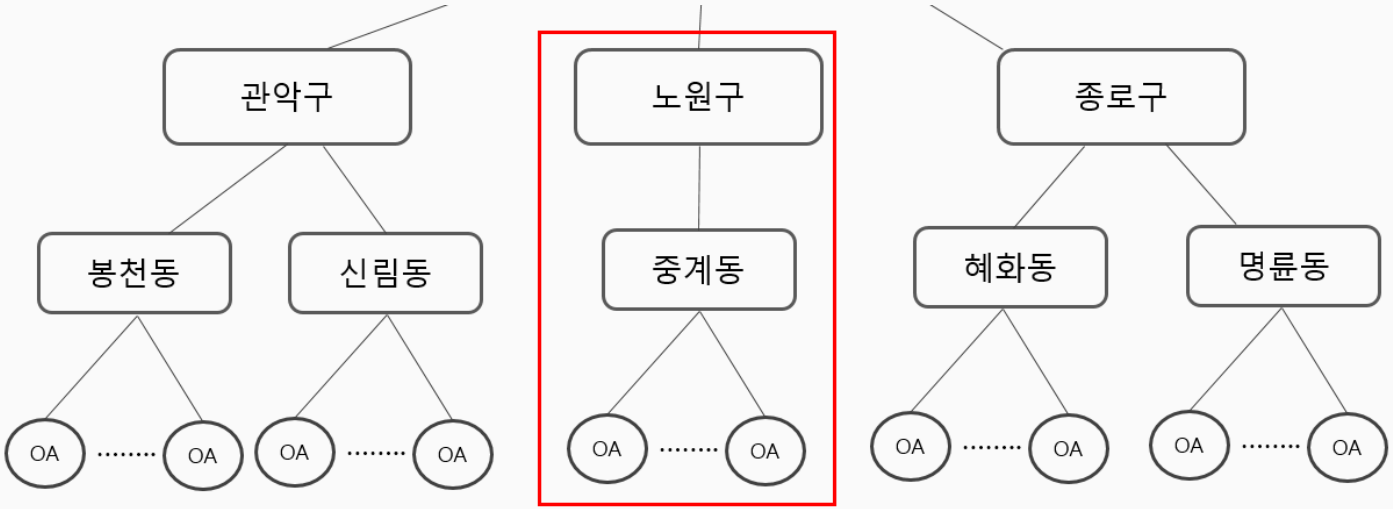
자료구조의 계층성

2-1. 관악구의 tb를 만들고 관악구 안의 2개의 동을 c(봉천동, 신림동) 형식으로 저장한다.
2-2. 봉천동의 tb를 만들고 봉천동 안의 OA들을 c(OA,. . . ,OA) 형식으로 저장한다.
2-3. 봉천동 안의 OA들의 tb를 순차적으로 만든다.
2-4. 신림동의 tb를 만들고 신림동 안의 OA들을 c(OA,. . . ,OA) 형식으로 저장한다.
2-5. 신림동 안의 OA들의 tb를 순차적으로 만든다.



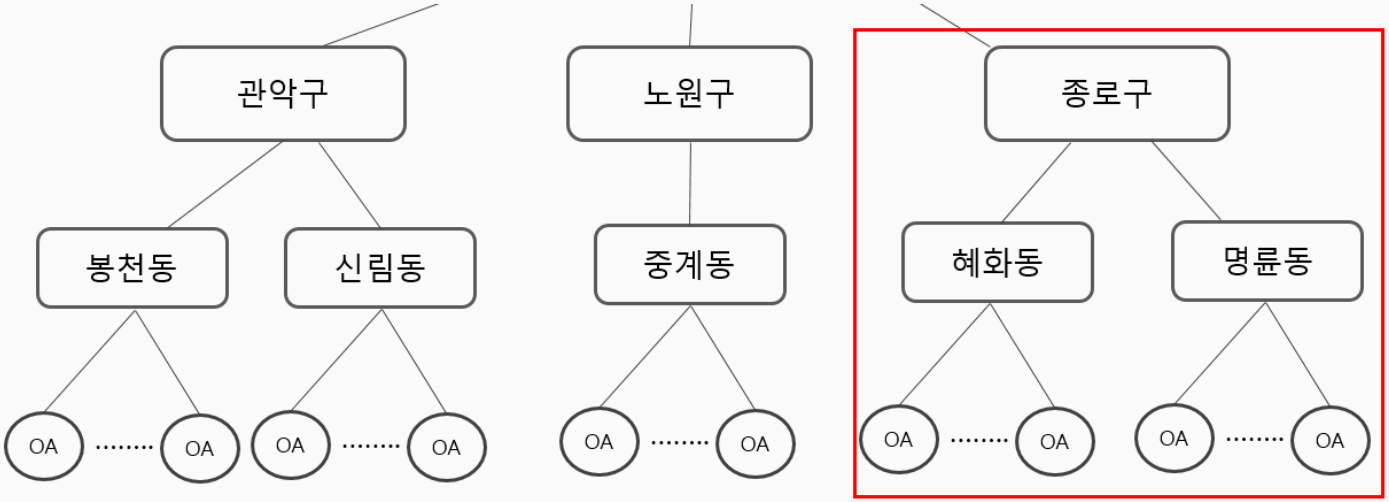
자료구조의 계층성

3. 2에서의 방식을 노원구에 대하여 똑같이 적용한다.



자료구조의 계층성

4. 2에서의 방식을 종로구에 대하여 똑같이 적용한다.



자료구조의 계층성

```
for(ii in 1:nn.LA2)
{
  # for the ii-th LA2 area unit
  area.LA2 = area[ area[, 'LA2'] == indx.LA2[ii], ]; area.LA2 = check.mtx(area.LA2, area)
  key.LA2 = key[ area[, 'LA2'] == indx.LA2[ii], ]; key.LA2 = check.mtx(key.LA2, key)
  tb.LA2 = cbind(tb.LA2, get.freq(key.LA2, ctgr0, weight))

  indx.LA3 = unique(area.LA2[, 'LA3'])
  nn.LA3 = length(indx.LA3)
  for(jj in 1:nn.LA3)
  {
    # for the jj-th LA3 area unit (in the ii-th LA2 area unit)
    area.LA3 = area.LA2[ area.LA2[, 'LA3'] == indx.LA3[jj], ]; area.LA3 = check.mtx(area.LA3, area)
    key.LA3 = key.LA2[ area.LA2[, 'LA3'] == indx.LA3[jj], ]; key.LA3 = check.mtx(key.LA3, key)
    tb.LA3 = cbind(tb.LA3, get.freq(key.LA3, ctgr0, weight))

    indx.OA = unique(area.LA3[, 'OA'])
    nn.OA = length(indx.OA)
    for(kk in 1:nn.OA)
    {
      # for the kk-th OA area unit (in the jj-th LA3 area unit in the ii-th LA2 area unit)
      area.OA = area.LA3[ area.LA3[, 'OA'] == indx.OA[kk], ]; area.OA = check.mtx(area.OA, area)
      key.OA = key.LA3[ area.LA3[, 'OA'] == indx.OA[kk], ]; key.OA = check.mtx(key.OA, key)
      tb.OA = cbind(tb.OA, get.freq(key.OA, ctgr0))
```

관악구, 노원구, 종로구

봉천동, … , 혜화동

OA

- 지금까지의 과정을 R코드로 구현하면 다음과 같이 구현할 수 있다.

- for문을 여러번 중첩적으로 사용하지만, 계층구조를 탐색하며 자료를 정렬하는 구조이기 때문에 지역을 전체 자료에서 찾아 정렬하는 방법보다 훨씬 시간이 단축된다.

자료구조의 계층성

- area변수의 갯수가 정해져 있을 경우 앞에서 처럼 for문을 중첩적으로 사용하여 구현하는 것이 가능하다.
- 하지만, area변수를 임의의 갯수로 일반화 할 경우, for문을 중첩적으로 사용하여 구현하는 것이 힘들다.
- 일반화를 위하여 재귀함수를 사용한다!

Key idea 2 : 재귀함수를 통한 일반화

재귀함수를 통한 일반화

- 재귀함수는 함수 내부에 같은 함수를 다시 사용하여 코드를 좀 더 간편하게 작성하는 방법이다.

```
facto = function(n)

{
  if (n == 0)

  {
    return(1)
  }

  else
  {
    result = n * facto(n-1)
    return(result)
  }
}
```

- 다음과 같이 $n!$을 계산하는 코드를 재귀함수를 이용하여 작성할 수 있다.

재귀함수를 통한 일반화

- Process 1에서 각 지역별로 tb를 만들어주는 tb.maker란 함수를 재귀함수를 이용하여 다음과 같이 만들었다.

```
tb.maker = function(area, key, level, flevel, ctgr0, weight)

{
  if (level == 1)

  {}

  if ((level > 1) & (level < flevel))

  {
    p.result = tb.maker(area, key, level-1, flevel, ctgr0, weight)
    p.result1 = p.result[[1]][[2]]
    result2 = p.result[[2]]
    result3 = p.result[[3]]
    freq = NULL
    meta = NULL
    tree = NULL
```

- 여기서 level은 area변수의 갯수를 의미한다.

재귀함수를 통한 일반화

- tb.maker는 모든 지역의 tb를 return하는 동시에 지역들의 계층구조를 return한다.

- 예를 들어 우리가 가정한 input data에서 level $= 2$인 경우, 다음과 같이 두 개의 list에 output을 저장한다.

재귀함수를 통한 일반화

- List 1

[1]

서울특별시

key1	key2	빈도
1	A	
1	B	
1	C	
2	A	
2	B	
2	C	

[2]

관악구

key1	key2	빈도
1	A	
1	B	
1	C	
2	A	
2	B	
2	C	

노원구

key1	key2	빈도
1	A	
1	B	
1	C	
2	A	
2	B	
2	C	

종로구

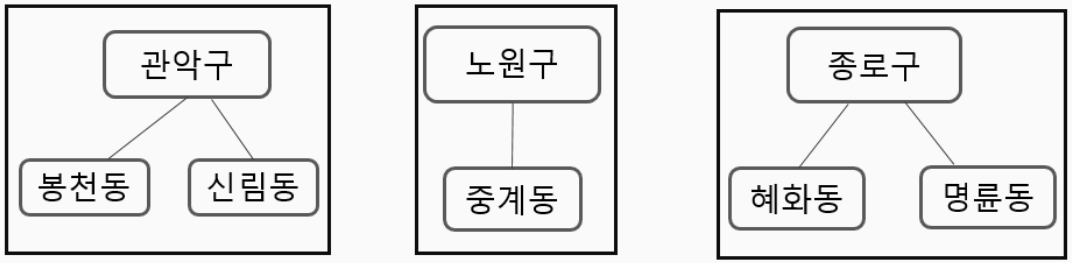
key1	key2	빈도
1	A	
1	B	
1	C	
2	A	
2	B	
2	C	

재귀함수를 통한 일반화

- List 2

[1]

[2]

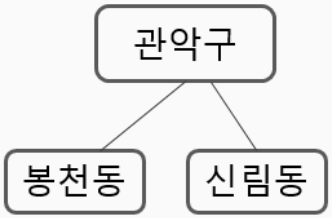


재귀함수를 통한 일반화

- 이제 level $= 3$인 경우 tb.maker를 실행시킨다고 가정하자.
- tb.maker는 재귀함수이기 때문에 level $= 2$인 경우의 tb.maker의 output을 참조한다.
- 이 때, List 2의 [2]에 저장되어 있는 계층구조를 참조하게 되고, 관악구에서 봉천동, 신림동을 탐색하여 각 각의 지역의 tb를 만들어 List 1의 [3]에 저장하고, 봉천동과 신림동의 계층구조를 List 2의 [3]에 다시 저장한다.
- 노원구와 종로구에 대해서도 차례로 같은 방식의 탐색을 통해 각 지역의 tb와 계층구조를 만들어 저장한다.

재귀함수를 통한 일반화

List 2 [2][1] 참조

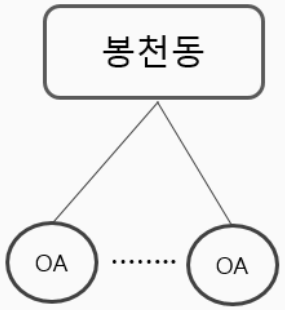


List 1 [3][1] 에 저장

봉천동

key1	key2	빈도
1	A	
1	B	
1	C	
2	A	
2	B	
2	C	

List 2 [3][1] 에 저장

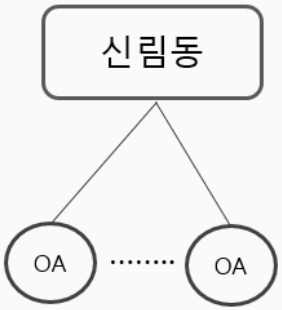


List 1 [3][2] 에 저장

신림동

key1	key2	빈도
1	A	
1	B	
1	C	
2	A	
2	B	
2	C	

List 2 [3][2] 에 저장



Key idea 3 : sapply와 lapply의 활용

sapply와 lapply의 활용

- sapply : 벡터, 리스트, 표현식, 데이터 프레임 등에 함수를 적용하고, 그 결과를 벡터 또는 행렬로 반환.

```
> sapply(c(1, 2, 3, 4), rep, times = 10)
      [,1] [,2] [,3] [,4]
 [1,]    1    2    3    4
 [2,]    1    2    3    4
 [3,]    1    2    3    4
 [4,]    1    2    3    4
 [5,]    1    2    3    4
 [6,]    1    2    3    4
 [7,]    1    2    3    4
 [8,]    1    2    3    4
 [9,]    1    2    3    4
[10,]    1    2    3    4
```

sapply와 lapply의 활용

- lapply : 벡터, 리스트 등에 함수를 적용하고 그 결과를 리스트로 반환.

```
> lapply(c(1, 2, 3, 4), rep, times = 10)
[[1]]
 [1] 1 1 1 1 1 1 1 1 1 1

[[2]]
 [1] 2 2 2 2 2 2 2 2 2 2

[[3]]
 [1] 3 3 3 3 3 3 3 3 3 3

[[4]]
 [1] 4 4 4 4 4 4 4 4 4 4
```

sapply와 lapply의 활용

tb.maker 함수 내부에 다음의 두 가지 함수를 사용.

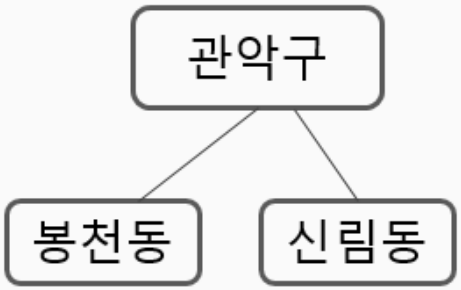
- sub.tb.maker.freq : 지역이 지정되면 그 지역의 tb를 만드는 함수
  예. sub.tb.maker.freq(관악구)

관악구

key1	key2	빈도
1	A	
1	B	
1	C	
2	A	
2	B	
2	C	

sapply와 lapply의 활용

- sub.tb.maker.tree : 지역이 지정되면 그 지역 안의 계층을 만드는 함수
  예. sub.tb.maker.tree(관악구)



sapply와 lapply의 활용

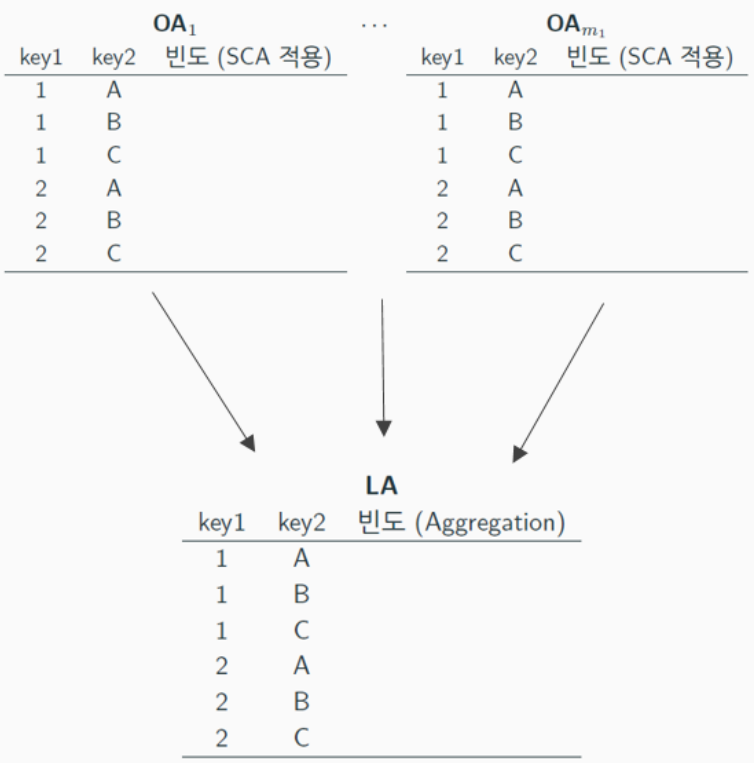
- 예를 들어, 관악구 안의 봉천동과 신림동의 tb를 만들고 계층구조를 저장한다고 가정하자.

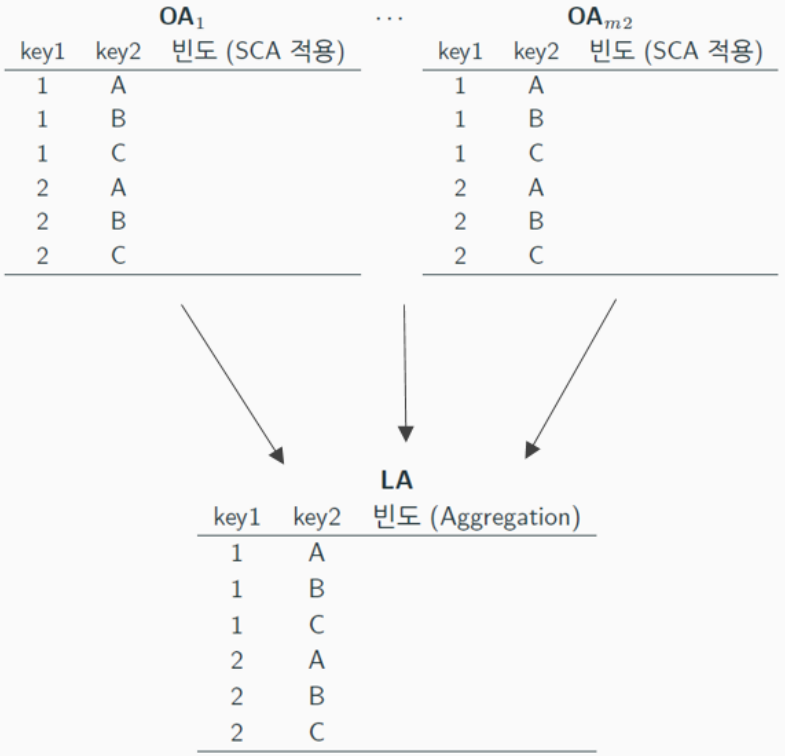
- indx라는 변수에 c(관악구, 봉천동)을 저장하면 for문을 사용하지 않고 sapply(indx, sub.tb.maker.freq)를 사용하여 좀 더 빠르게 관악구와 봉천동의 tb를 구할 수 있다.

- 또한, lapply(indx, sub.tb.maker.tree)를 사용하면 for문을 사용하지 않고 관악구와 봉천동의 계층구조를 만들어 list형태로 저장할 수 있다.(List 2)

Key idea 4 : Meta data의 활용

Meta data의 활용



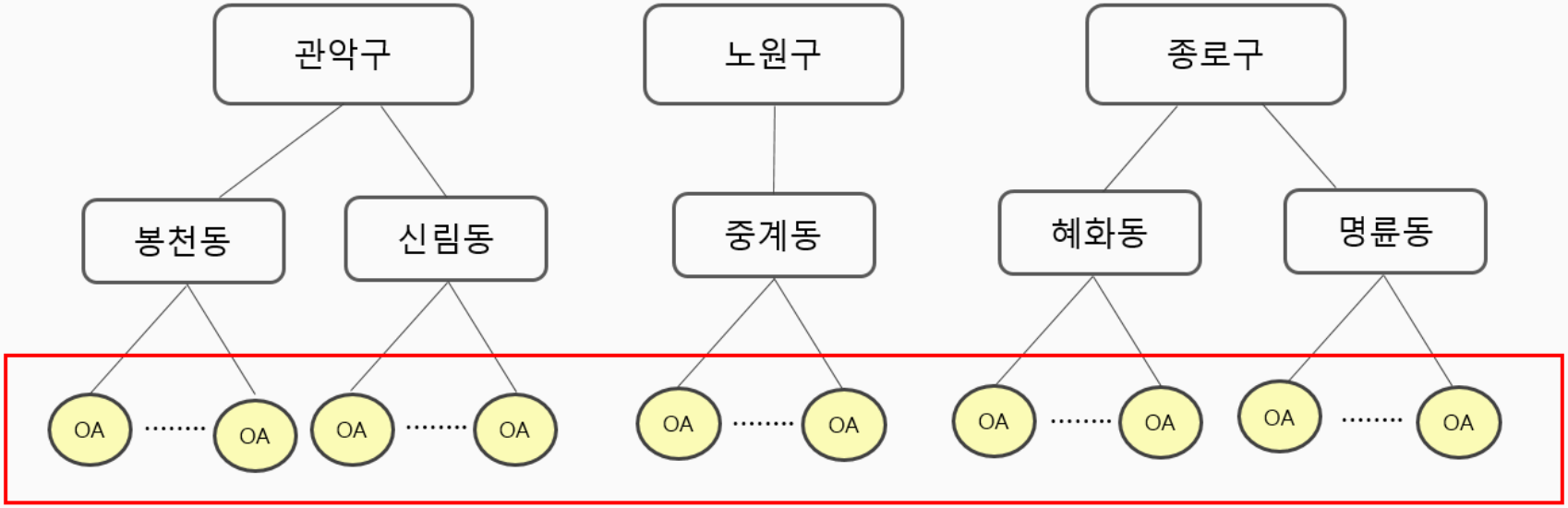


3. SCA를 적용한 OA들을 aggregation하여 특정 LA수준의 빈도표를 만든다.

Meta data의 활용

- SCA가 적용된 OA들을 aggregation해서 관악구, 노원구 , 종로구의 빈도표를 만드는 상황을 생각하자.

- Process 1에서 tb들을 계층적으로 접근하여 저장하였기 때문에, OA의 tb들은 관악구, 노원구, 종로구 순으로 저장이 되어있다.



Meta data의 활용

- Meta data 형식은 다음과 같다.

	관악구	노원구	종로구
OA 갯수	100	200	300

- OA의 tb들이 관악구, 노원구, 종로구 순으로 저장이 되어있기 때문에 관악구, 노원구, 종로구 순으로 OA를 aggreagtion하여 빈도표를 만들 때, 처음 100개의 OA tb를 aggregation하여 관악구의 빈도표를 만들고 그 다음 200개의 OA tb를 aggregation하여 노원구의 빈도표를 만들고, 마지막으로 300개의 OA tb를 aggregation하여 종로구의 빈도표를 만든다.

- 전체 data에서 OA를 직접 탐색하여 aggregation하는 것보다 훨씬 효율적이다.

마치며...

- 지금까지 R 패키지 개발과정에서 대용량 자료를 처리하는 부분에 대해서 중점적으로 다루었다.

- 대용량 자료의 구조를 파악하면 시간을 단축시킬 수 있는 idea를 손쉽게 얻을 수 있다.
  - 여기서는 계층구조의 특성을 이용하여 시간을 단축.

- for문을 여러번 사용하여 코드를 작성해야 하는 경우, 재귀함수를 사용하면 깔끔하게 작성을 할 수 있고 동시에 일반화도 용의하다.

- R에서의 for loop는 다른 언어에 비해 느리다. 따라서 apply계열의 함수를 사용하여 for문을 대체해주면 시간을 단축시킬 수 있다.

Thank You

Q&A