



430.217

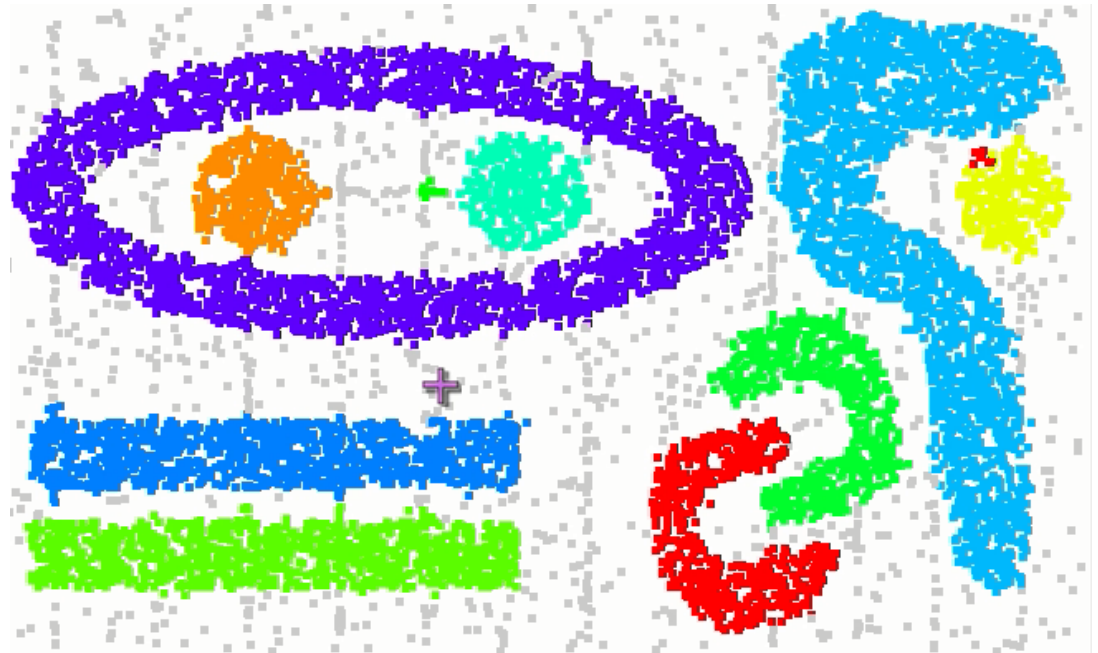
Introduction to Data Structures

Project

Seoul National University
Advanced Computing Laboratory

DBSCAN

- Density based spatial clustering of applications with noise
 - 밀도가 높으면 하나의 cluster로 간주



- 구현에 필요한 자료구조
 - kd-tree
 - Disjoint set
 - Container (array / linked list / stack / queue / heap 등)

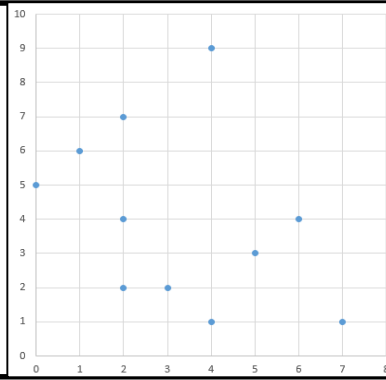
Project Details

kd-tree

DBSCAN (union-find)

DBSCAN Overview

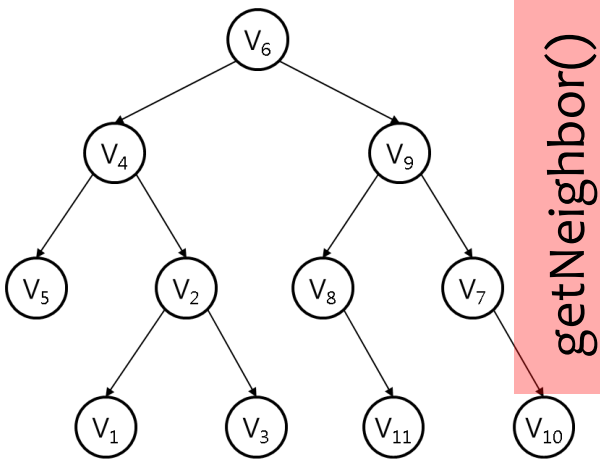
Coordinate data



Build a
kd-tree

float**

kd-tree



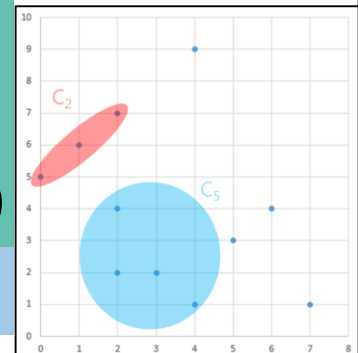
DBSCAN

Run()

Coordinate

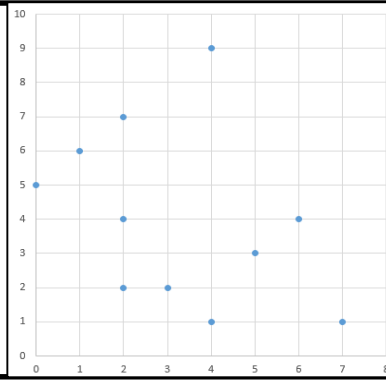
CoordinateSet

Check
conditions
&
Clustering
(Union & Find)



DBSCAN Overview: kd-tree

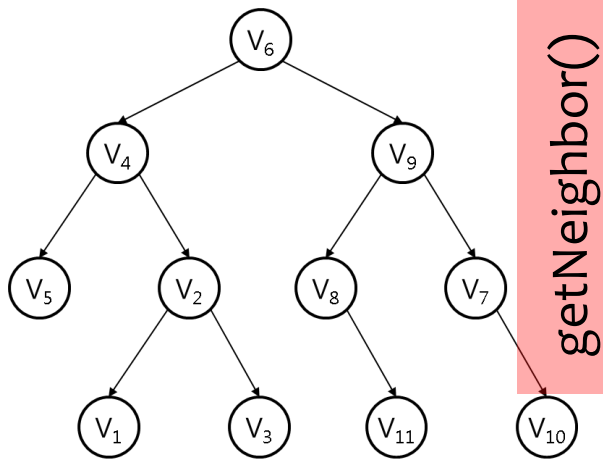
Coordinate data



Build a
kd-tree

float**

kd-tree



getNeighbor()

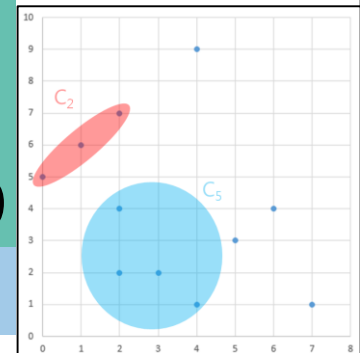
DBSCAN

Run()

Check
conditions
&
Clustering
(Union & Find)

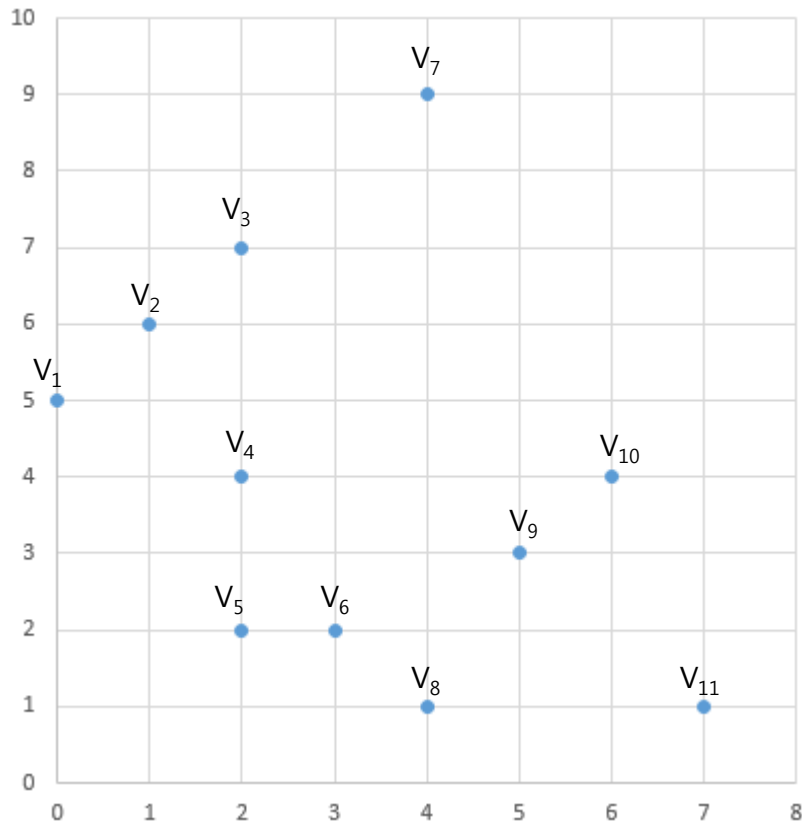
Coordinate

CoordinateSet



kd-tree

- k-dimensional tree
 - A general version of a BST (BST = 1D tree)
 - Stores k-dimension coordinates data (same dimension in a kd-tree)
 - Fast region search

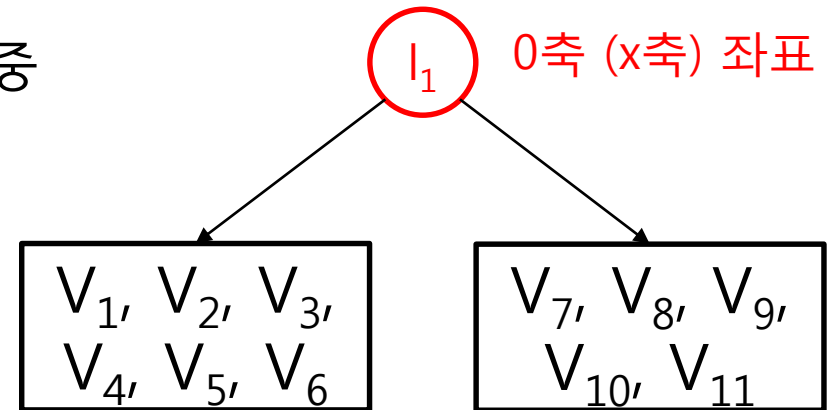
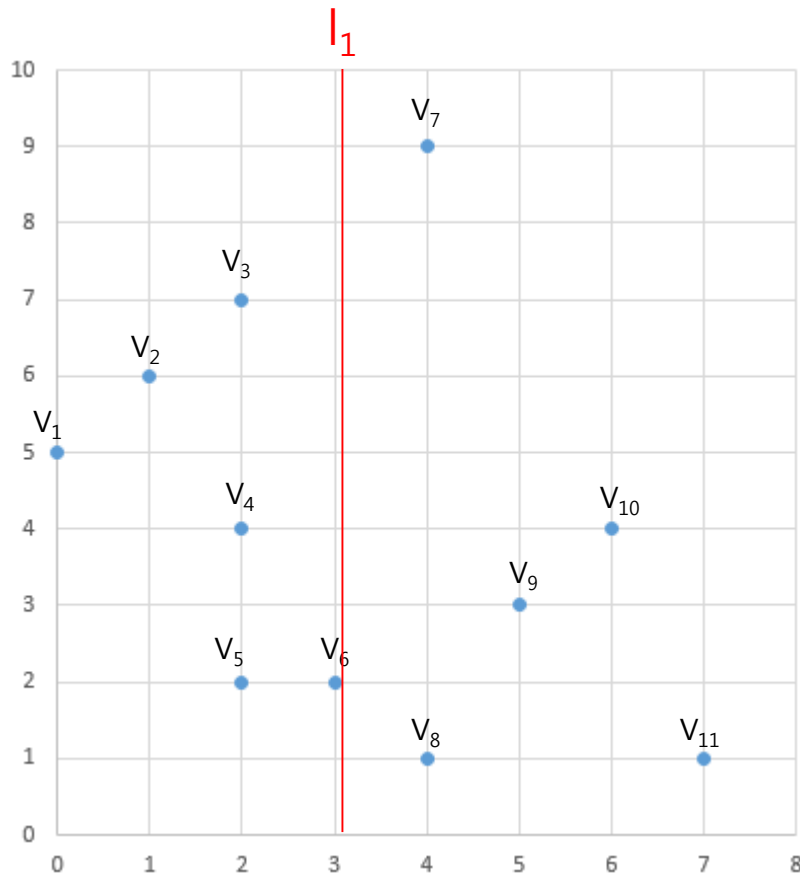


Build a kd-tree

■ Example: 2D

- Parent node는 hyperplane 정보 저장
(기준 point는 (depth % k) 축의 좌표 중
median 값을 가지는 point이다.)

l_i = hyperplane



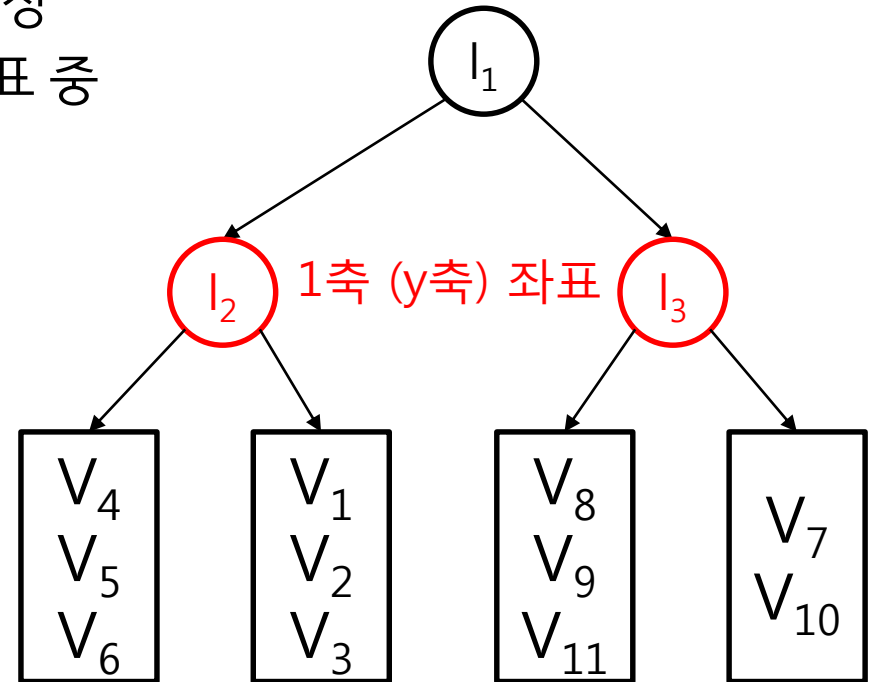
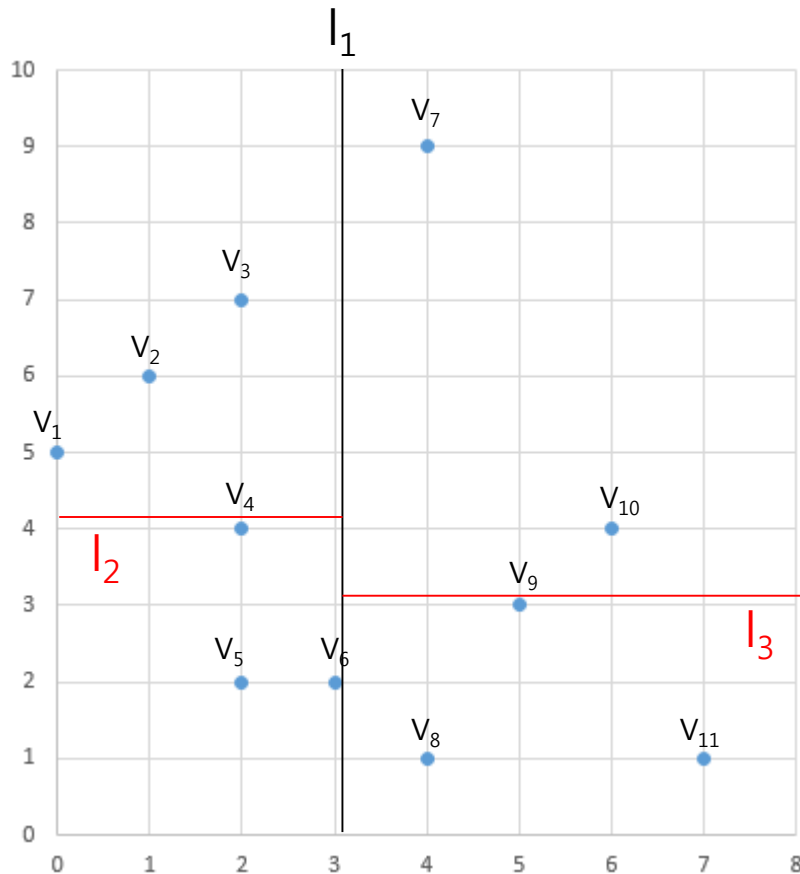
기준 point의 해당 축의 좌표보다
작거나 같으면 left subtree
크면 right subtree

Build a kd-tree

- Build a kd-tree (ex: 2D)

- Parent node는 hyperplane 정보 저장
(기준 point는 (depth % k) 축의 좌표 중 median 값을 가지는 point이다.)

l_i = hyperplane



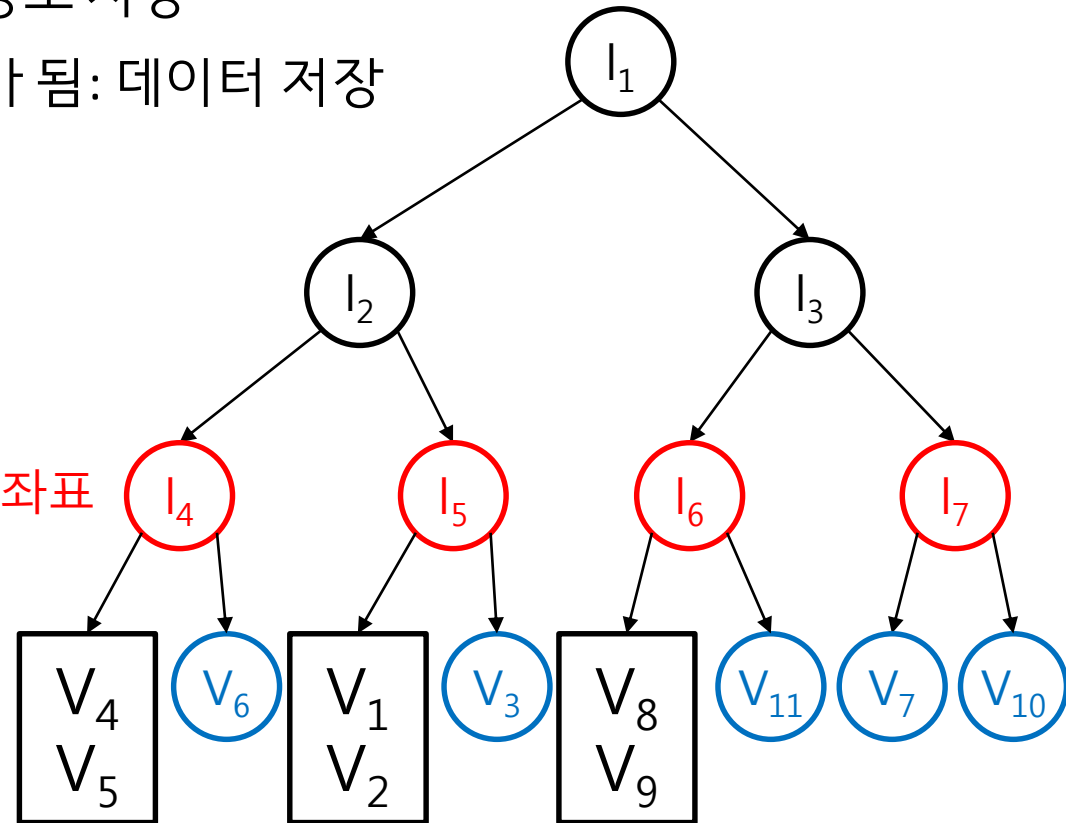
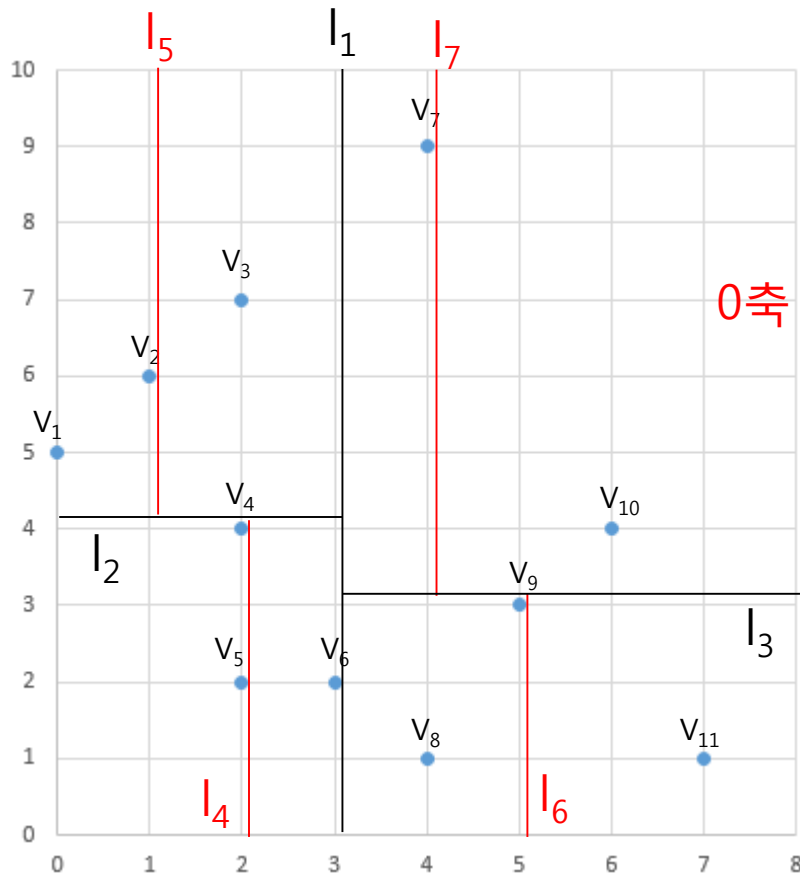
기준 point의 해당 축의 좌표보다
작거나 같으면 left subtree
크면 right subtree

Build a kd-tree

- Build a kd-tree (ex: 2D)

- Parent node는 hyperplane 정보 저장
- point가 1개 되면 leaf node가 됨: 데이터 저장

l_i = hyperplane

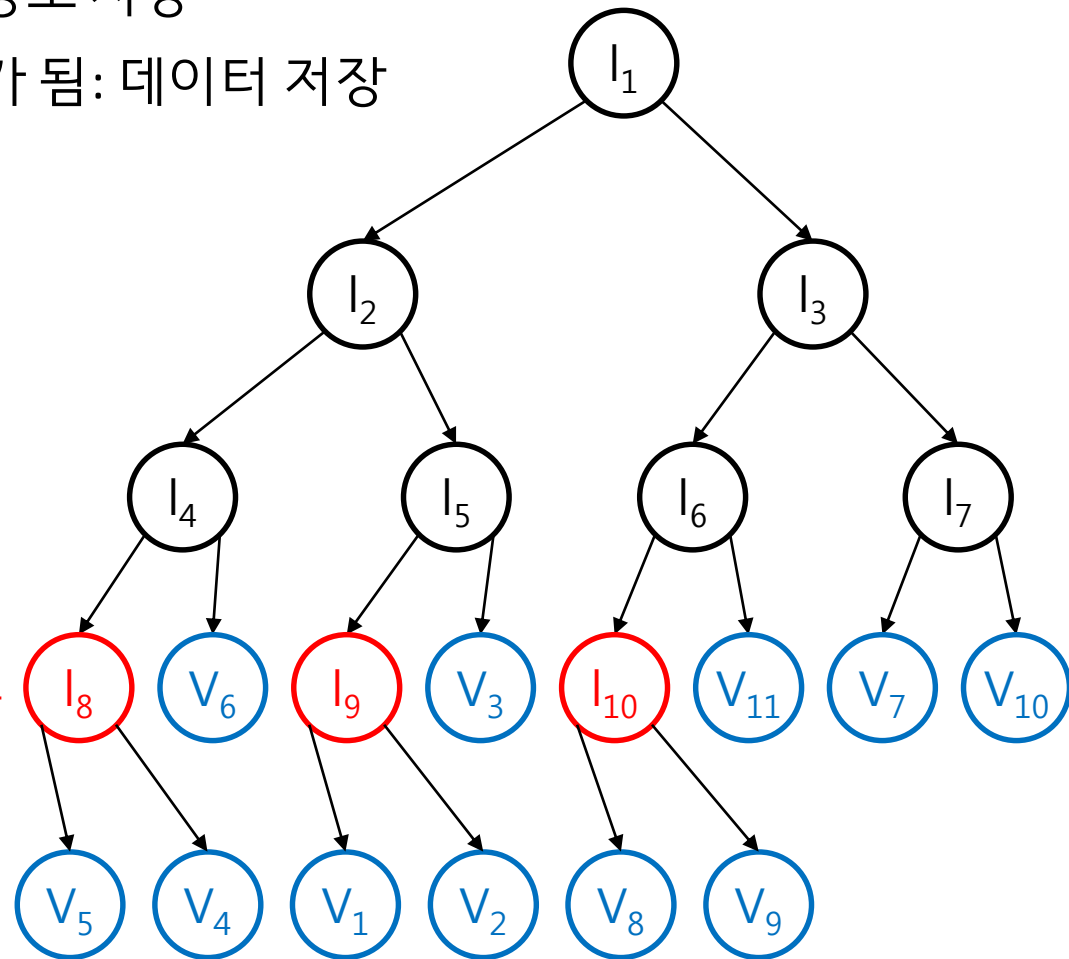
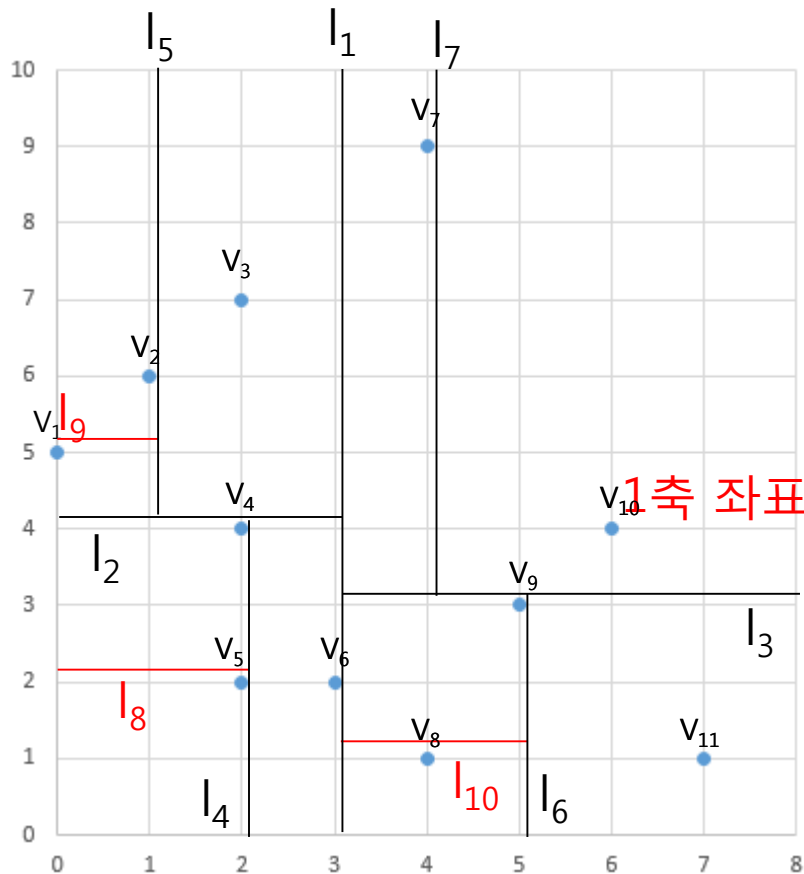


Build a kd-tree

■ Build a kd-tree (ex: 2D)

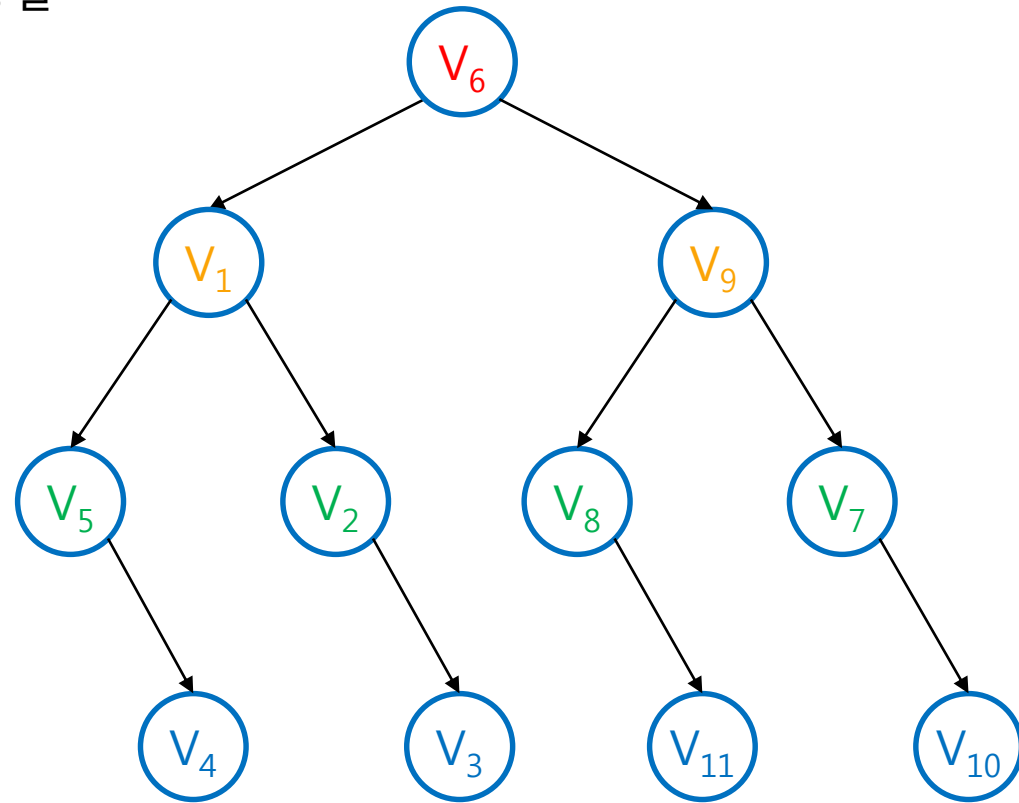
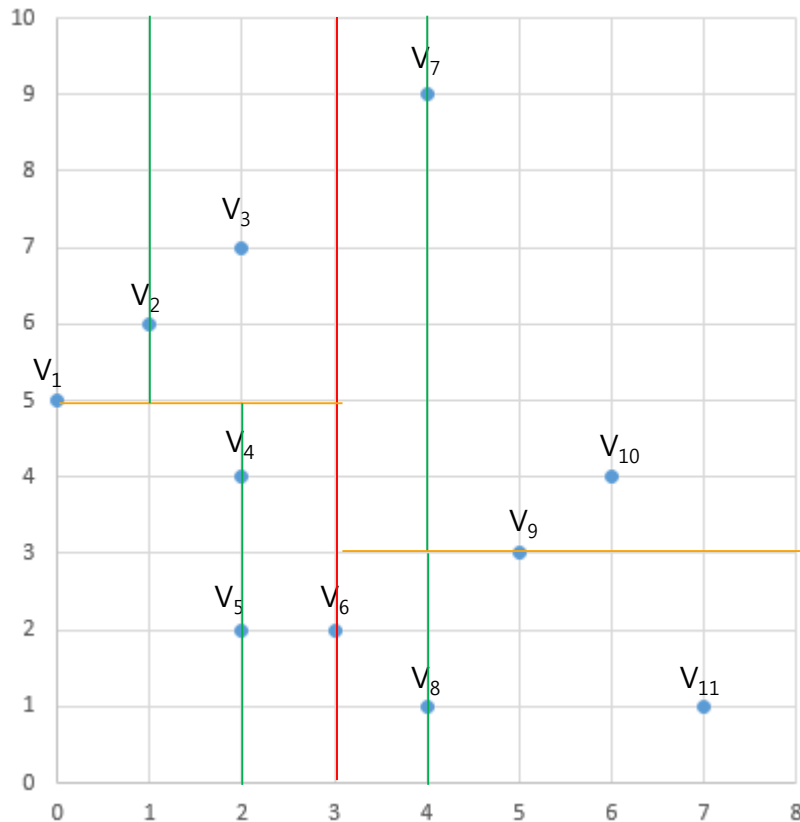
- Parent node는 hyperplane 정보 저장
- point가 1개 되면 leaf node가 됨: 데이터 저장

l_i = hyperplane



Another Way to Build a kd-tree

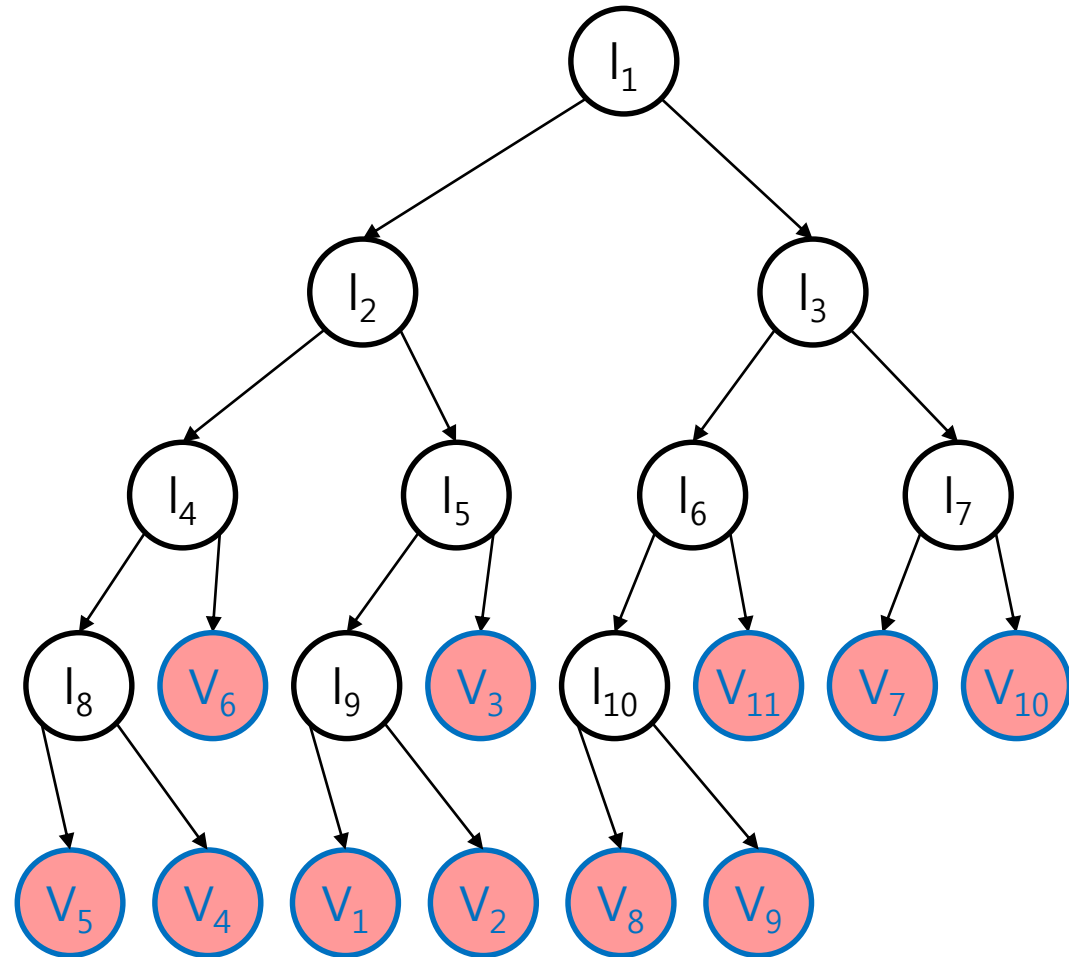
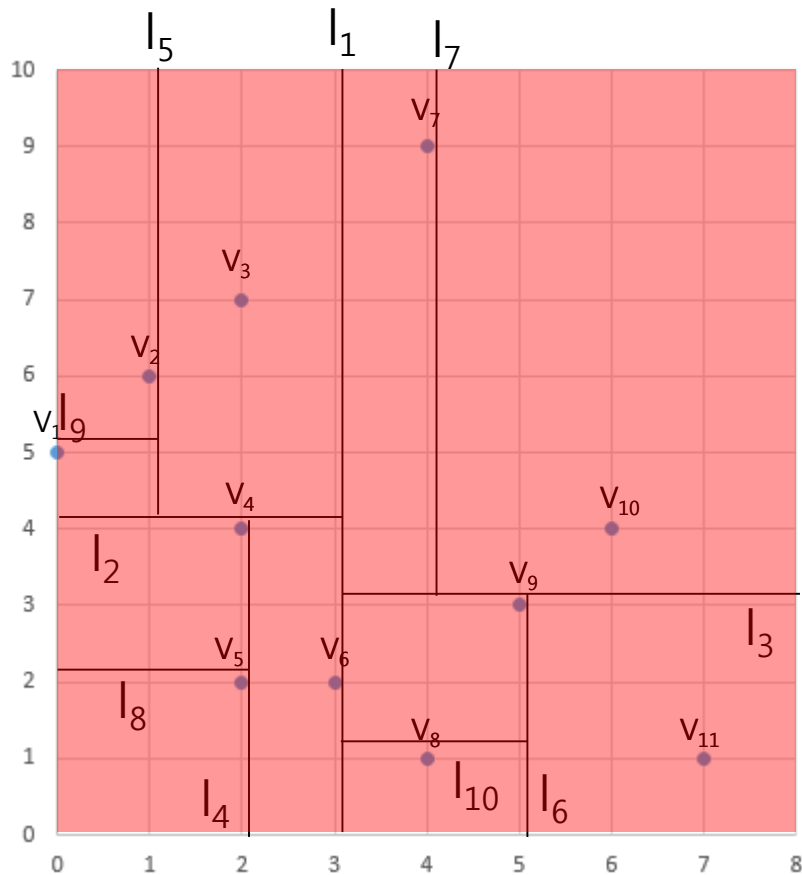
- Build a kd-tree 다른 방법 (기존의 BST에서 배운 방법대로)
 - 모든 노드가 좌표 데이터 저장
 - Subtree 설정은 앞 방법과 동일



※ 이 예시에서 V4와 V5 노드는 tree에서 자리가 바뀔 수도 있음

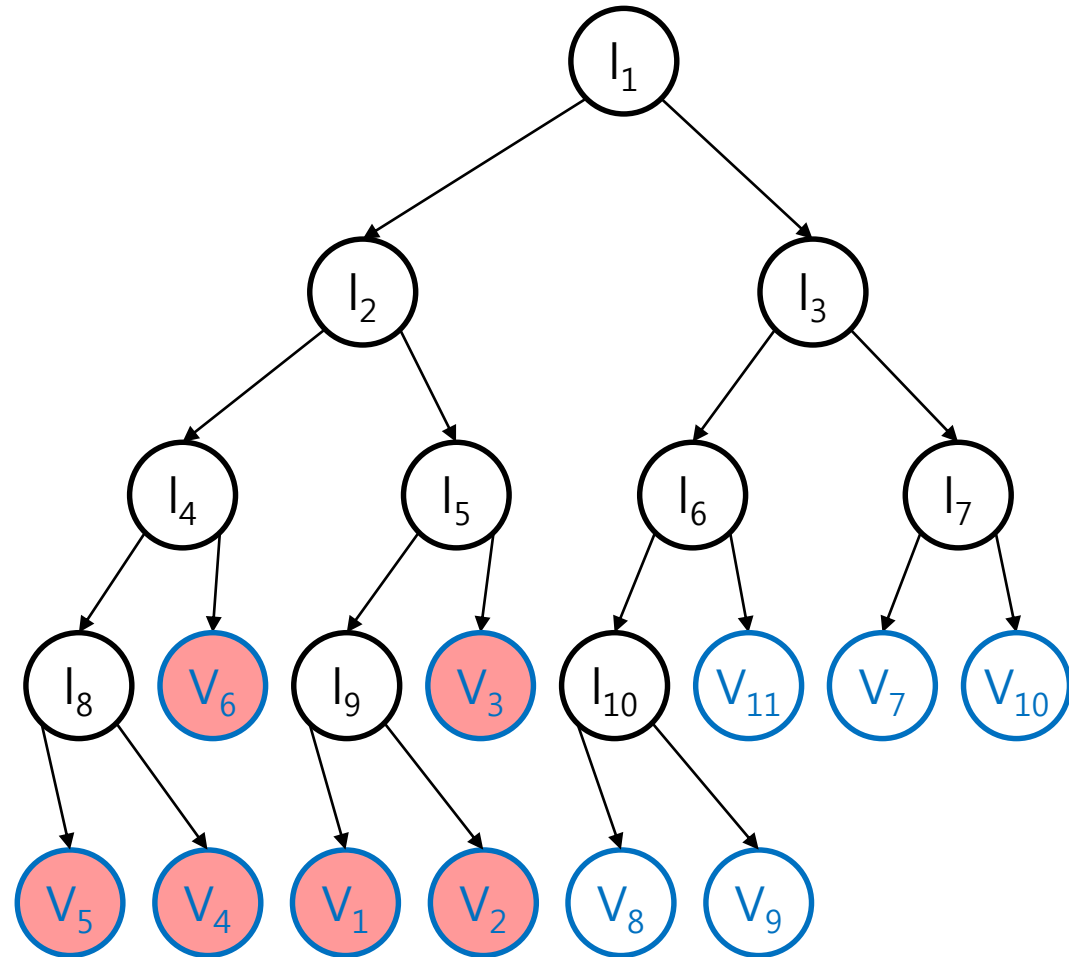
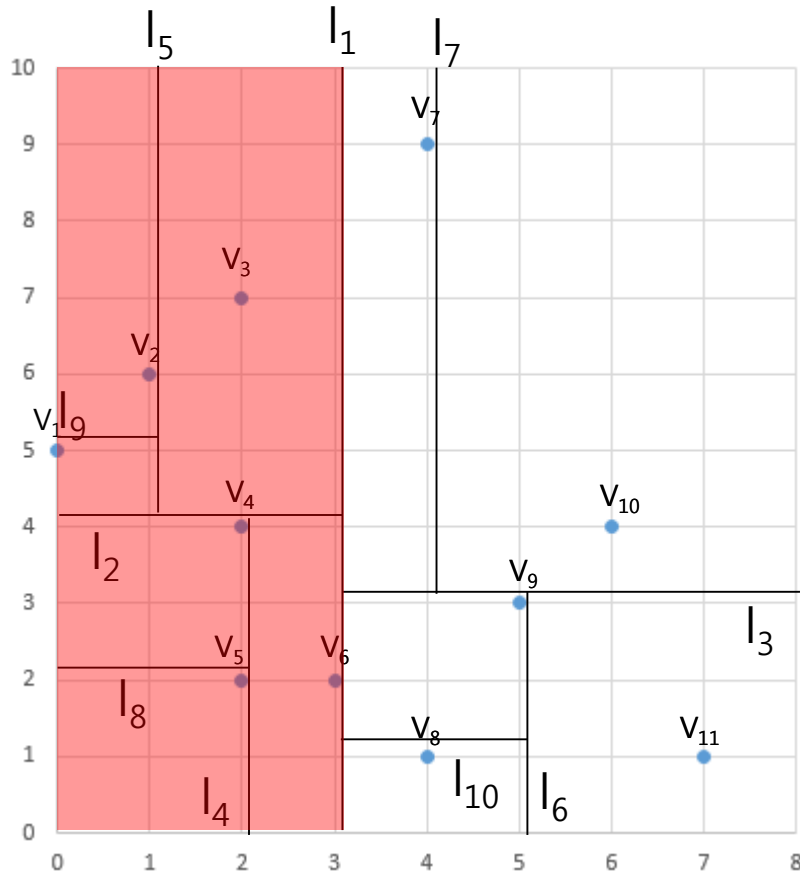
Geometric Meaning

- l_1 노드의 후손들은 모든 data



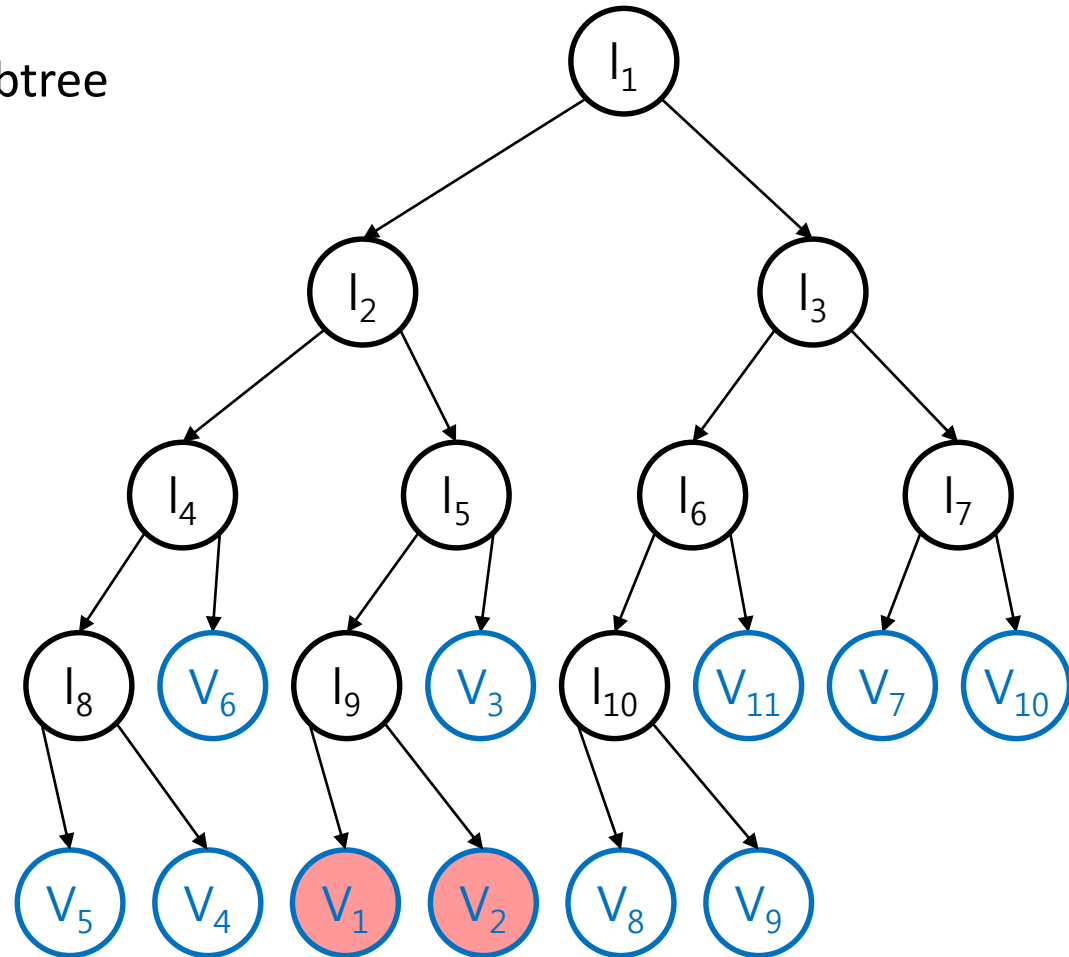
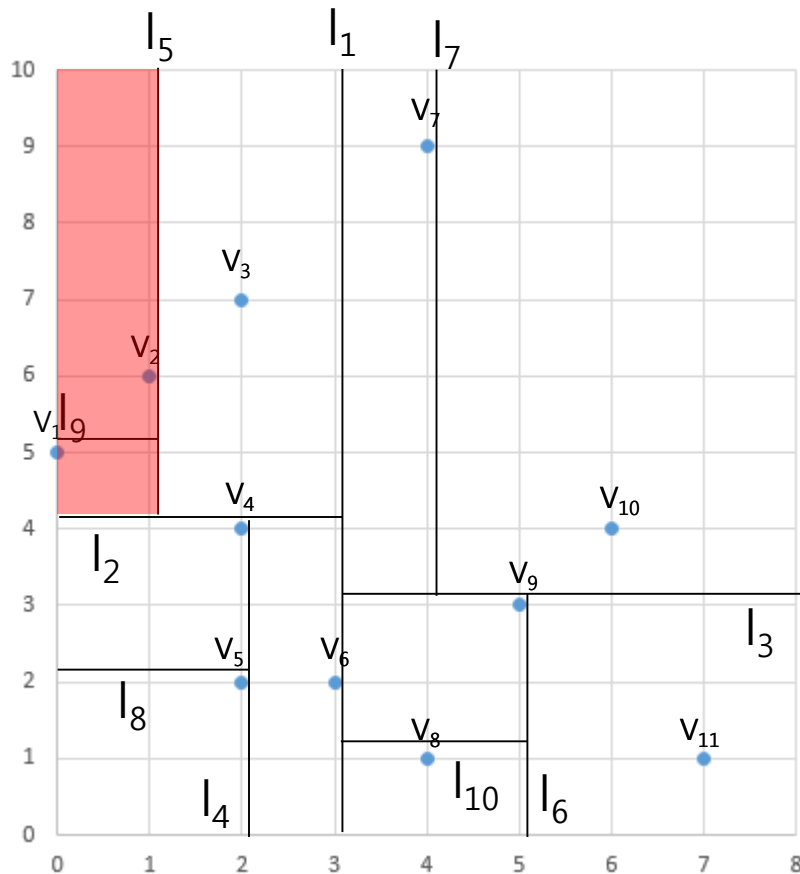
Geometric Meaning

- l_2 노드의 후손들은 0축의 값이 3보다 작거나 같은 data



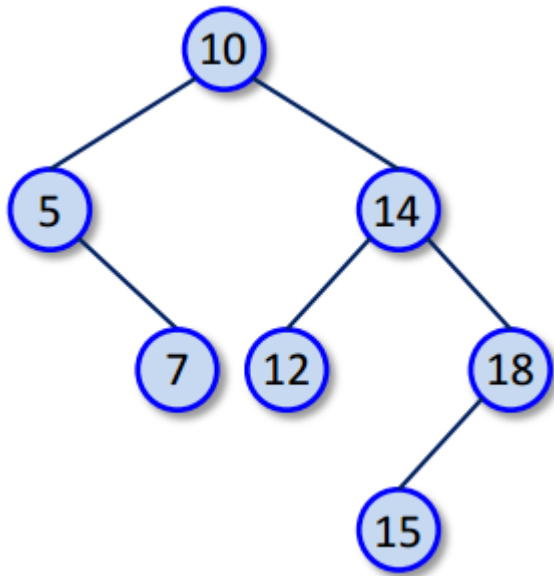
Geometric Meaning

- l_9 노드의 후손들은 ...
 - l_5 의 left subtree
 - (l_2 의 right subtree)의 left subtree



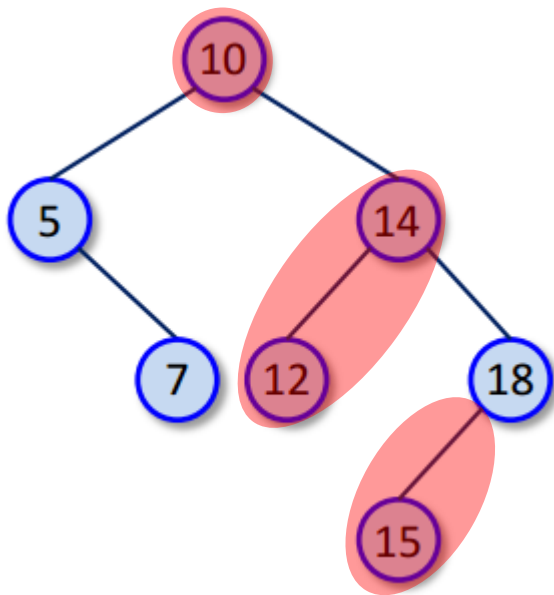
How to Search Region

- 수업시간에 BST는 find만 배웠음 → kd-tree의 find도 쉽고 빠르다!
- 구간 검색은 어떻게 할까.
 - Get data in [9, 15]



How to Search Region

- 수업시간에 BST는 find만 배웠음 → kd-tree의 find도 쉽고 빠르다!
- 구간 검색은 어떻게 할까.
 - Get data in [9, 15]

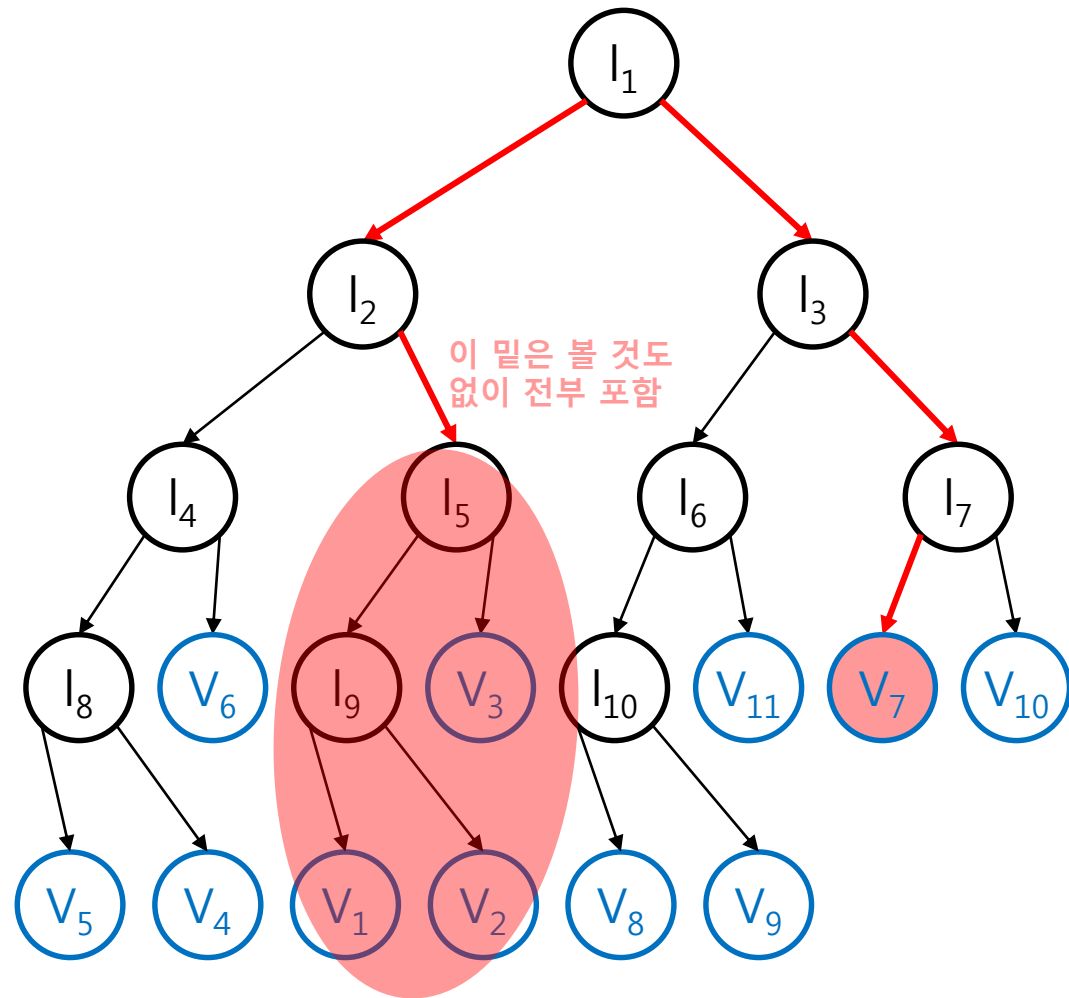
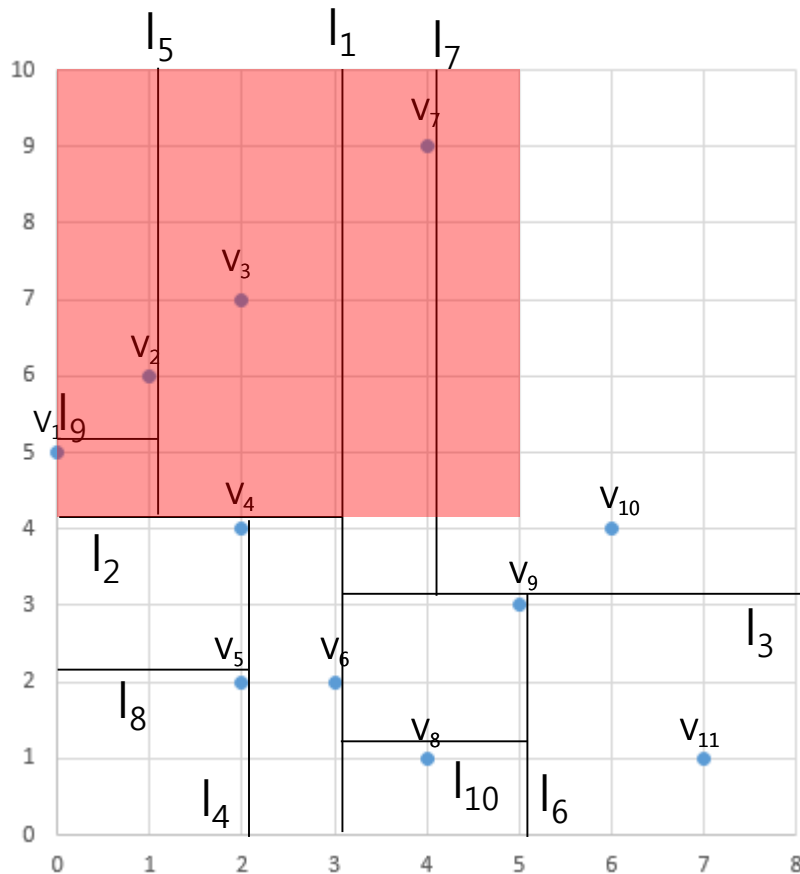


정렬할 필요 없음

10	12	14	15
----	----	----	----

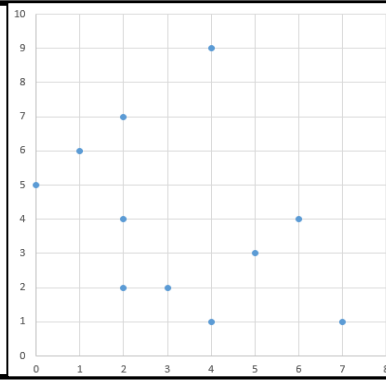
How to Search Region

- Get data in ($0\text{축} = [-\infty, 5]$, $1\text{축} = (4, \infty]$)



DBSCAN Overview: DBSCAN

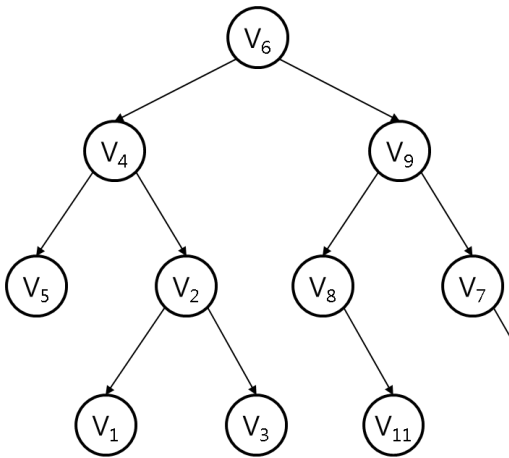
Coordinate data



Build a
kd-tree

float**

kd-tree



getNeighbor()

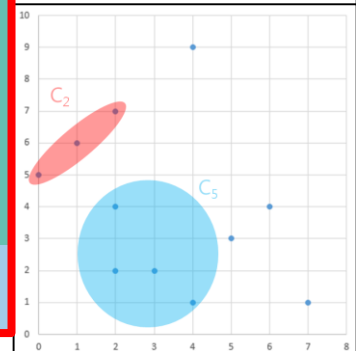
DBSCAN

Run()

Coordinate

CoordinateSet

Check
conditions
&
Clustering
(Union & Find)



DBSCAN: Project Algorithm

- Disjoint set 사용
 - Union & Find operation을 사용해 DBSCAN 알고리즘 진행

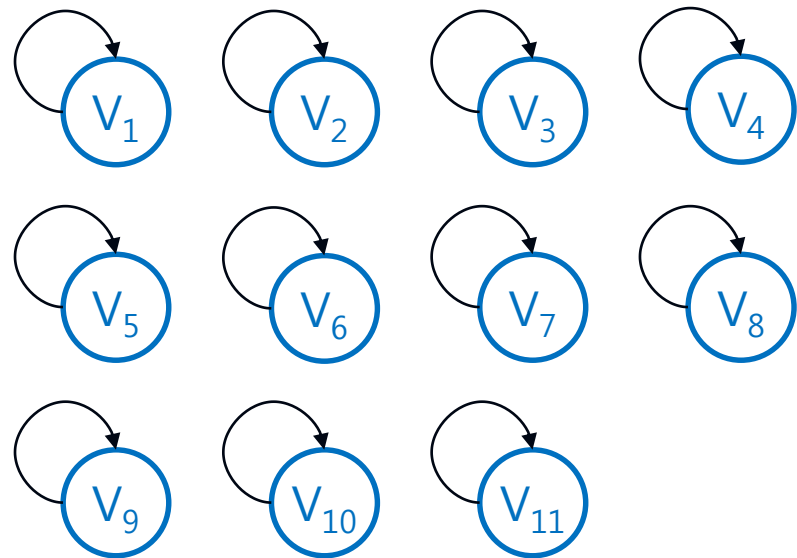
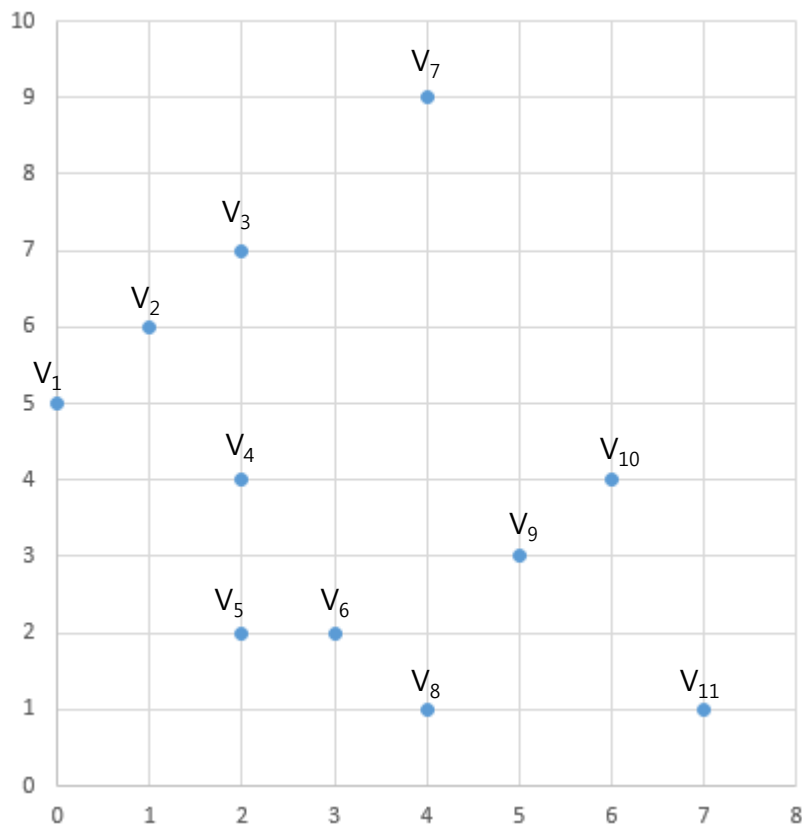
Algorithm 2 The disjoint-set data structure based DBSCAN Algorithm (DSDBSCAN). Input: A set of points X , distance eps , and the minimum number of points required to form a cluster, $minpts$. Output: A set of clusters.

```
1: procedure DSDBSCAN( $X, eps, minpts$ )
2:   for each point  $x \in X$  do
3:      $p(x) \leftarrow x$ 
4:     for each point  $x \in X$  do
5:        $N \leftarrow \text{GETNEIGHBORS}(x, eps)$ 
6:       if  $|N| \geq minpts$  then
7:         mark  $x$  as core point
8:         for each point  $x' \in N$  do
9:           if  $x'$  is a core point then
10:            UNION( $x, x'$ )
11:           else if  $x'$  is not yet member of any cluster then
12:            mark  $x'$  as member of a cluster
13:            UNION( $x, x'$ )
```

※ input point 순서대로
for문이 진행되도록 하면,
구현에 따라 결과 달라지지 않음.

DBSCAN: Project Algorithm

- 먼저 모든 point의 parent를 자기 자신으로
 - 수업시간의 Make-Set(x)와 동일한 역할

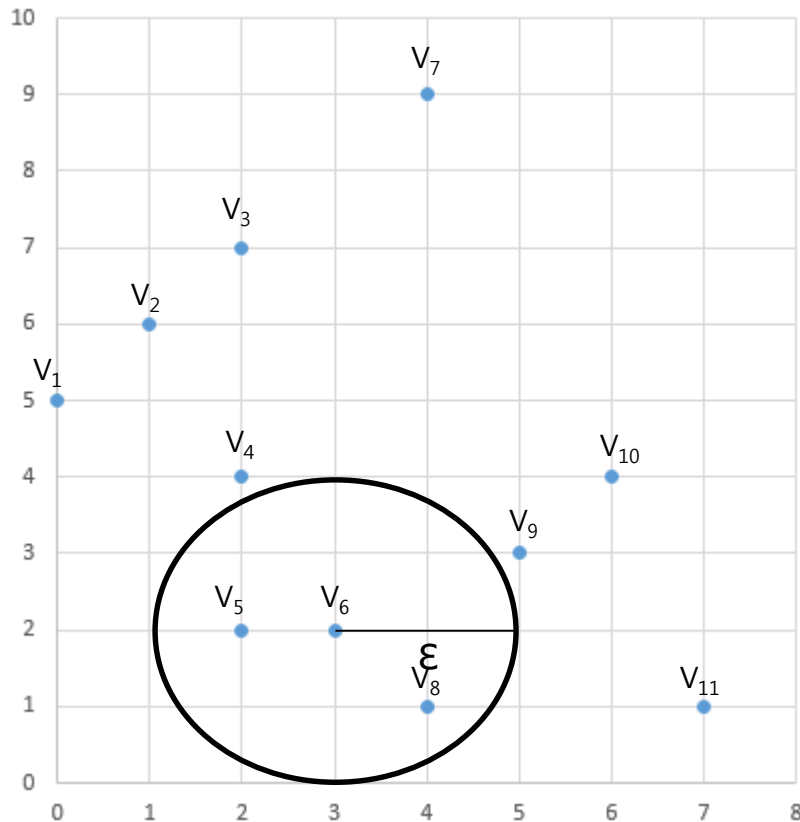


Algorithm 2 The disjoint-set data structure based DBSCAN Algorithm (DSDBSCAN). Input: A set of points X , distance eps , and the minimum number of points required to form a cluster, $minpts$. Output: A set of clusters.

```
1: procedure DSDBSCAN( $X, eps, minpts$ )
2:   for each point  $x \in X$  do
3:      $p(x) \leftarrow x$ 
4:   for each point  $x \in X$  do
5:      $N \leftarrow \text{GETNEIGHBORS}(x, eps)$ 
6:     if  $|N| \geq minpts$  then
7:       mark  $x$  as core point
8:       for each point  $x' \in N$  do
9:         if  $x'$  is a core point then
10:          UNION( $x, x'$ )
11:       else if  $x'$  is not yet member of any cluster then
12:         mark  $x'$  as member of a cluster
13:         UNION( $x, x'$ )
```

DBSCAN: Project Algorithm

- 각 point들의 ϵ -neighbor를 검색
 - kd-tree로부터 얻을 수 있음



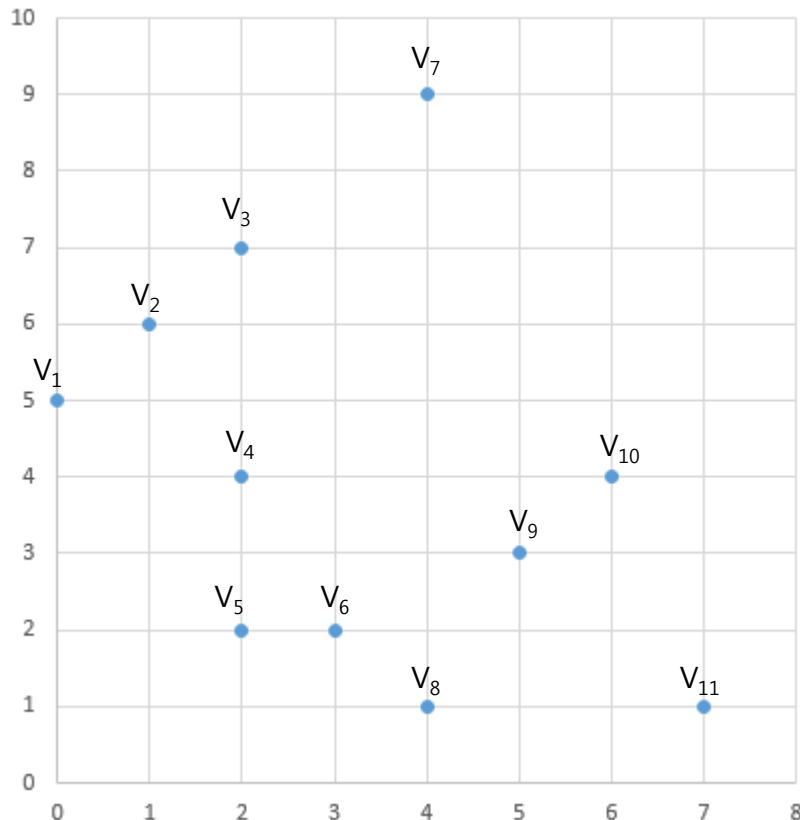
V_1 : V_2
 V_2 : V_1, V_3
 V_3 : V_2
 V_4 : V_5
 V_5 : V_4, V_6
 V_6 : V_5, V_8
 V_7 :
 V_8 : V_6
 V_9 : V_{10}
 V_{10} : V_9
 V_{11} :

Algorithm 2 The disjoint-set data structure based DBSCAN Algorithm (DSDBSCAN). Input: A set of points X , distance ϵ , and the minimum number of points required to form a cluster, $minpts$. Output: A set of clusters.

```
1: procedure DSDBSCAN( $X, \epsilon, minpts$ )
2:   for each point  $x \in X$  do
3:      $p(x) \leftarrow x$ 
4:   for each point  $x \in X$  do
5:      $N \leftarrow \text{GETNEIGHBORS}(x, \epsilon)$ 
6:     if  $|N| \geq minpts$  then
7:       mark  $x$  as core point
8:       for each point  $x' \in N$  do
9:         if  $x'$  is a core point then
10:          UNION( $x, x'$ )
11:       else if  $x'$  is not yet member of any cluster then
12:         mark  $x'$  as member of a cluster
13:         UNION( $x, x'$ )
```

DBSCAN: Project Algorithm

- Core point 조건 (if문) 을 만족하는 point들에 대해서 알고리즘 진행
 - Core point 만족조건 예시: neighbor가 2개 이상



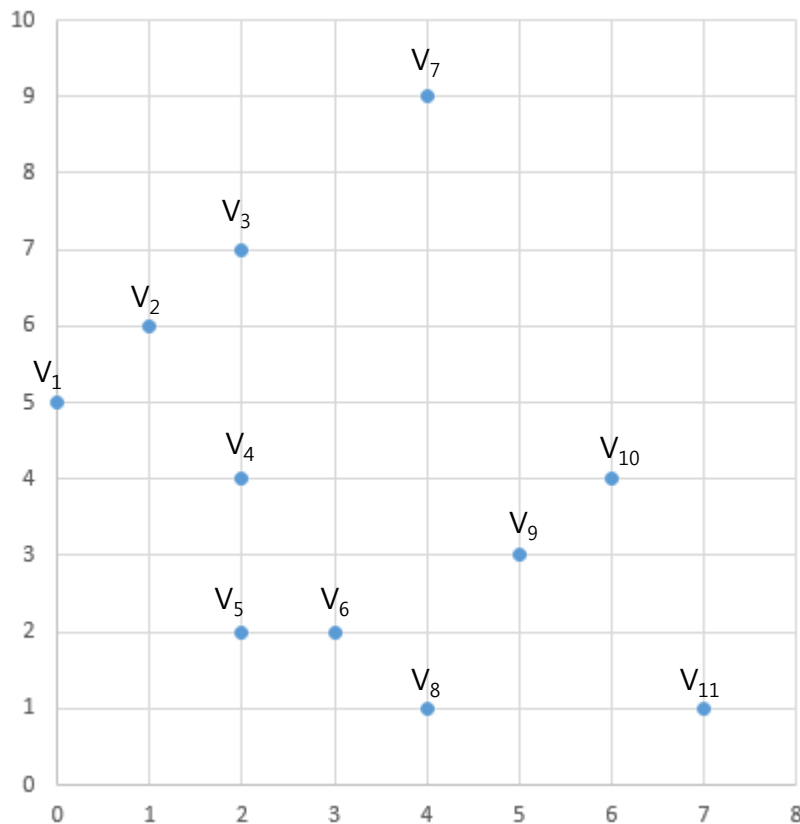
V₁: V₂
V₂: V₁, V₃
V₃: V₂
V₄: V₅
V₅: V₄, V₆
V₆: V₅, V₈
V₇:
V₈: V₆
V₉: V₁₀
V₁₀: V₉
V₁₁:

Algorithm 2 The disjoint-set data structure based DBSCAN Algorithm (DSDBSCAN). Input: A set of points X , distance eps , and the minimum number of points required to form a cluster, $minpts$. Output: A set of clusters.

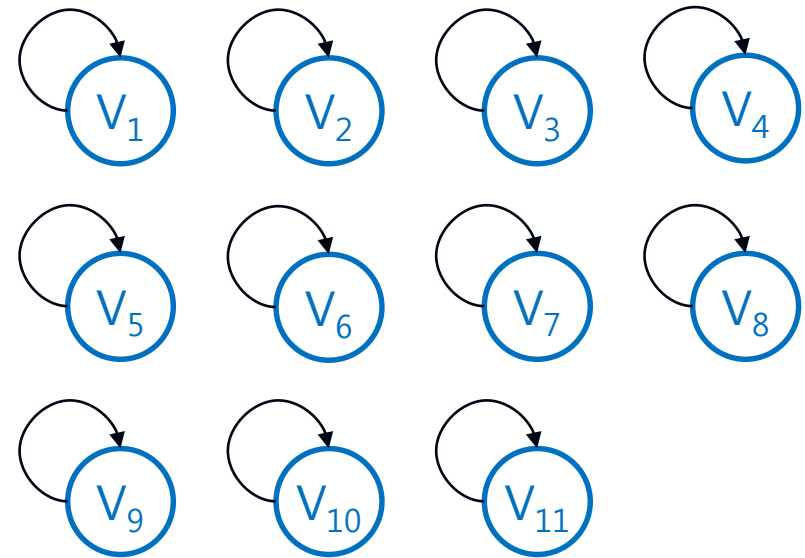
```
1: procedure DSDBSCAN( $X, eps, minpts$ )
2:   for each point  $x \in X$  do
3:      $p(x) \leftarrow x$ 
4:   for each point  $x \in X$  do
5:      $N \leftarrow \text{GETNEIGHBORS}(x, eps)$ 
6:     if  $|N| \geq minpts$  then
7:       mark  $x$  as core point
8:       for each point  $x' \in N$  do
9:         if  $x'$  is a core point then
10:          UNION( $x, x'$ )
11:       else if  $x'$  is not yet member of any cluster then
12:         mark  $x'$  as member of a cluster
13:         UNION( $x, x'$ )
```

DBSCAN: Project Algorithm

- Neighbor들을 union (pseudo code에 따라)
 - Union by rank
 - Path compression



$V_2: V_1, V_3$
 $V_5: V_4, V_6$
 $V_6: V_5, V_8$



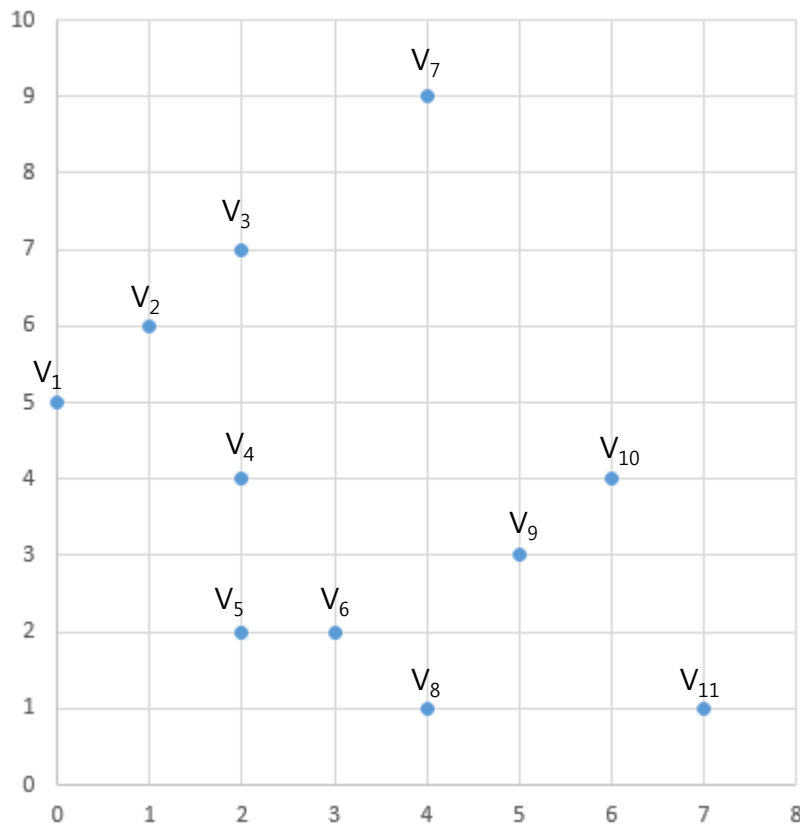
Algorithm 2 The disjoint-set data structure based DBSCAN Algorithm (DSDBSCAN). Input: A set of points X , distance eps , and the minimum number of points required to form a cluster, $minpts$. Output: A set of clusters.

```
1: procedure DSDBSCAN( $X, eps, minpts$ )
2:   for each point  $x \in X$  do
3:      $p(x) \leftarrow x$ 
4:   for each point  $x \in X$  do
5:      $N \leftarrow \text{GETNEIGHBORS}(x, eps)$ 
6:     if  $|N| \geq minpts$  then
7:       mark  $x$  as core point
8:       for each point  $x' \in N$  do
9:         if  $x'$  is a core point then
10:          UNION( $x, x'$ )
11:        else if  $x'$  is not yet member of any cluster then
12:          mark  $x'$  as member of a cluster
13:          UNION( $x, x'$ )
```

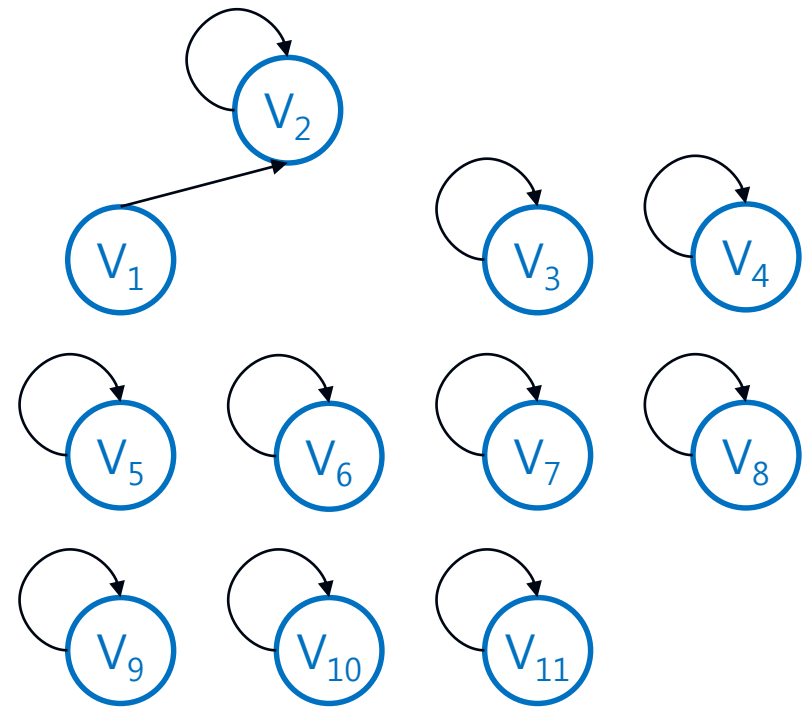
DBSCAN: Project Algorithm

■ Neighbor들을 union (pseudo code에 따라)

- Union by rank
- Path compression



$V_2: V_1, V_3$
 $V_5: V_4, V_6$
 $V_6: V_5, V_8$

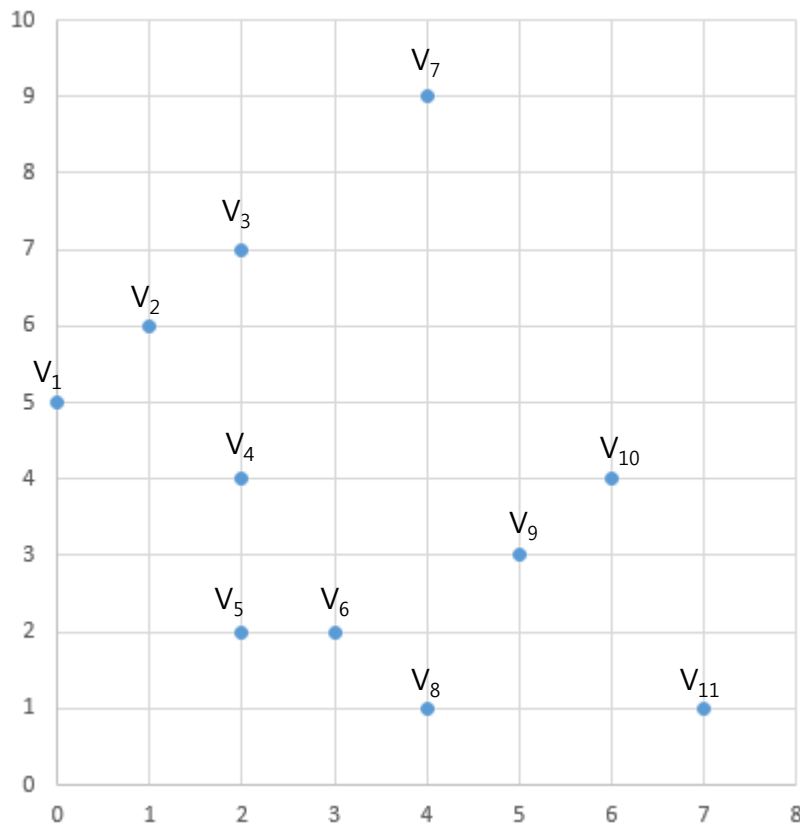


Algorithm 2 The disjoint-set data structure based DBSCAN Algorithm (DSDBSCAN). Input: A set of points X , distance eps , and the minimum number of points required to form a cluster, $minpts$. Output: A set of clusters.

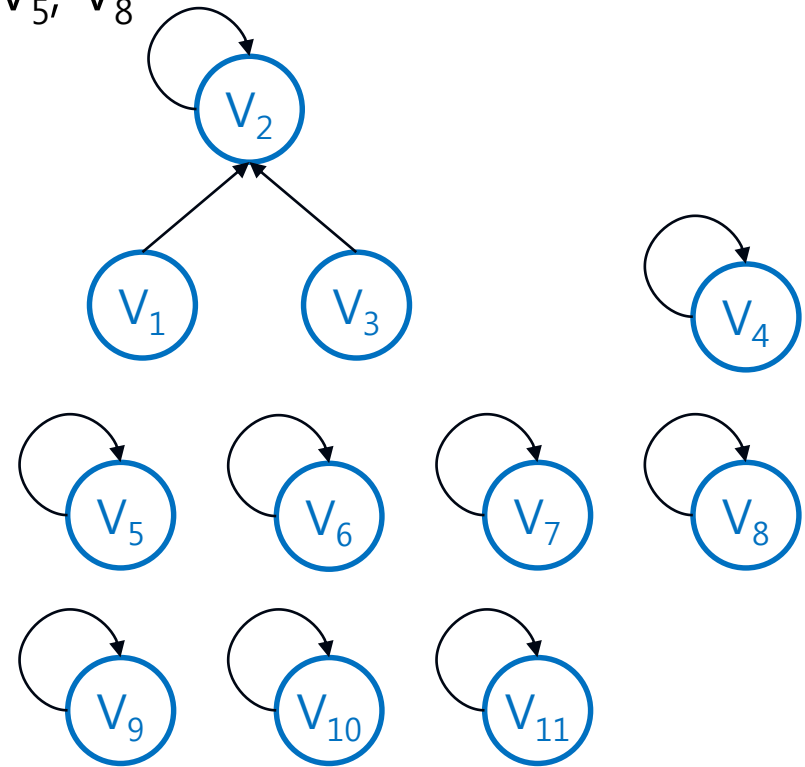
```
1: procedure DSDBSCAN( $X, eps, minpts$ )
2:   for each point  $x \in X$  do
3:      $p(x) \leftarrow x$ 
4:   for each point  $x \in X$  do
5:      $N \leftarrow \text{GETNEIGHBORS}(x, eps)$ 
6:     if  $|N| \geq minpts$  then
7:       mark  $x$  as core point
8:       for each point  $x' \in N$  do
9:         if  $x'$  is a core point then
10:          UNION( $x, x'$ )
11:        else if  $x'$  is not yet member of any cluster then
12:          mark  $x'$  as member of a cluster
13:          UNION( $x, x'$ )
```


DBSCAN: Project Algorithm

- Neighbor들을 union (pseudo code에 따라)
 - Union by rank
 - Path compression



$V_2: V_1, V_3$
 $V_5: V_4, V_6$
 $V_6: V_5, V_8$

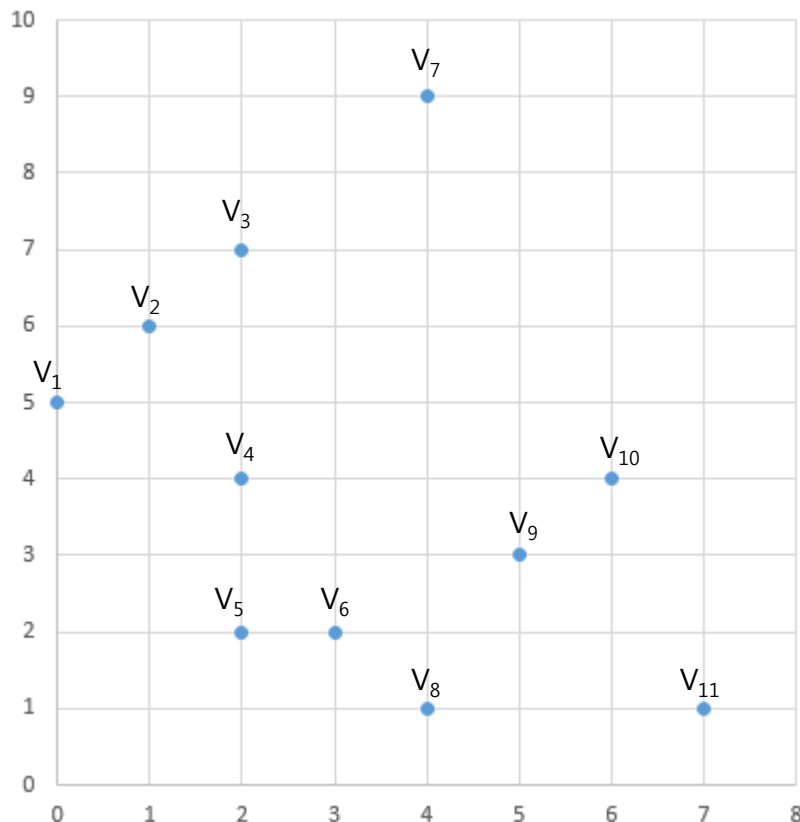


Algorithm 2 The disjoint-set data structure based DBSCAN Algorithm (DSDBSCAN). Input: A set of points X , distance eps , and the minimum number of points required to form a cluster, $minpts$. Output: A set of clusters.

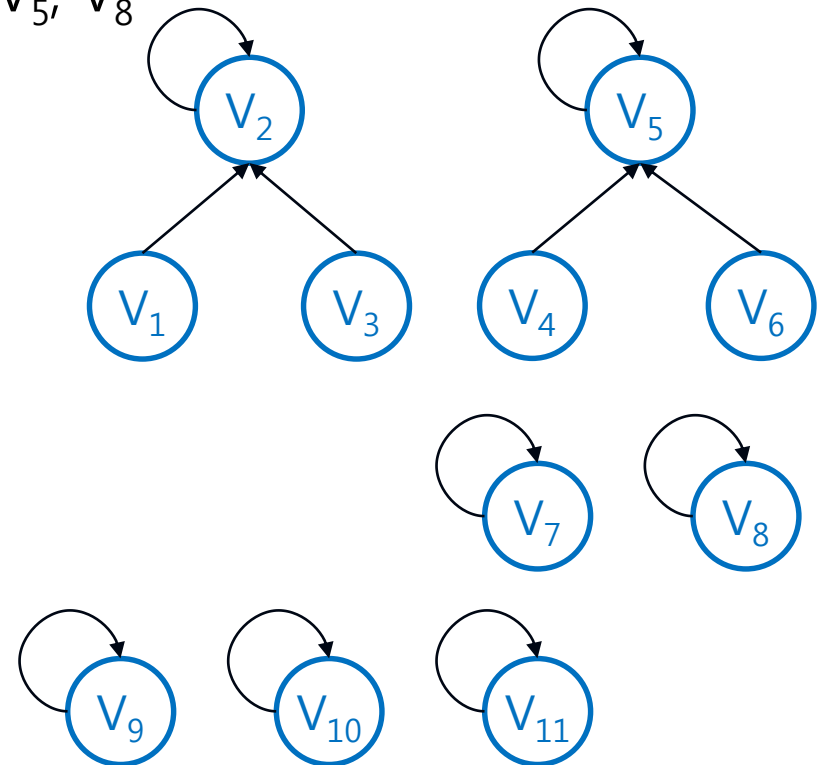
```
1: procedure DSDBSCAN( $X, eps, minpts$ )
2:   for each point  $x \in X$  do
3:      $p(x) \leftarrow x$ 
4:   for each point  $x \in X$  do
5:      $N \leftarrow \text{GETNEIGHBORS}(x, eps)$ 
6:     if  $|N| \geq minpts$  then
7:       mark  $x$  as core point
8:       for each point  $x' \in N$  do
9:         if  $x'$  is a core point then
10:          UNION( $x, x'$ )
11:        else if  $x'$  is not yet member of any cluster then
12:          mark  $x'$  as member of a cluster
13:          UNION( $x, x'$ )
```

DBSCAN: Project Algorithm

- Neighbor들을 union (pseudo code에 따라)
 - Union by rank
 - Path compression



$V_2: V_1, V_3$
 $V_5: V_4, V_6$
 $V_6: V_5, V_8$

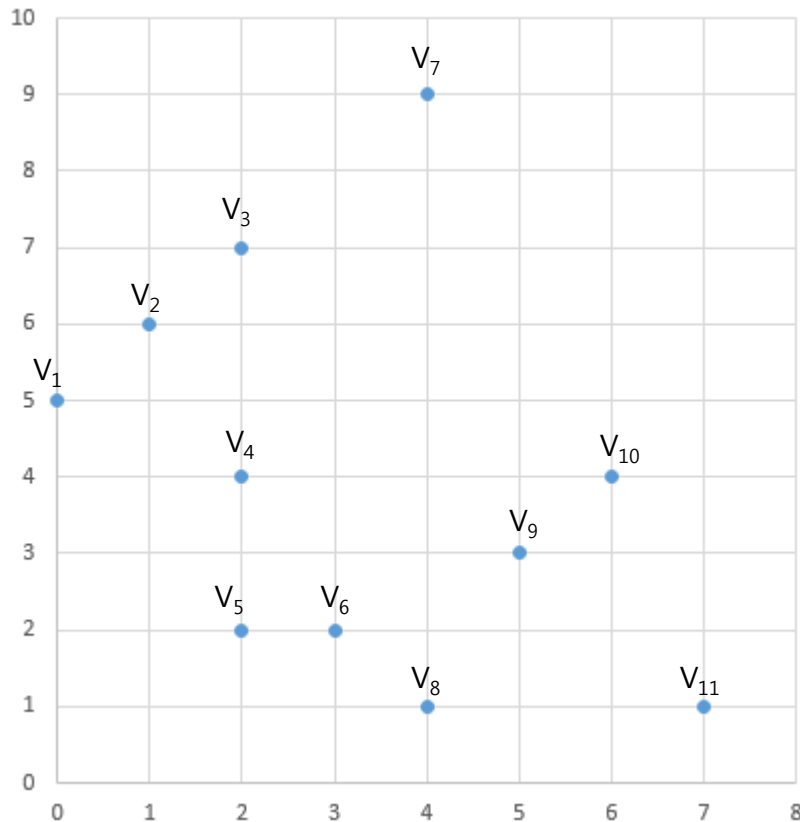


Algorithm 2 The disjoint-set data structure based DBSCAN Algorithm (DSDBSCAN). Input: A set of points X , distance eps , and the minimum number of points required to form a cluster, $minpts$. Output: A set of clusters.

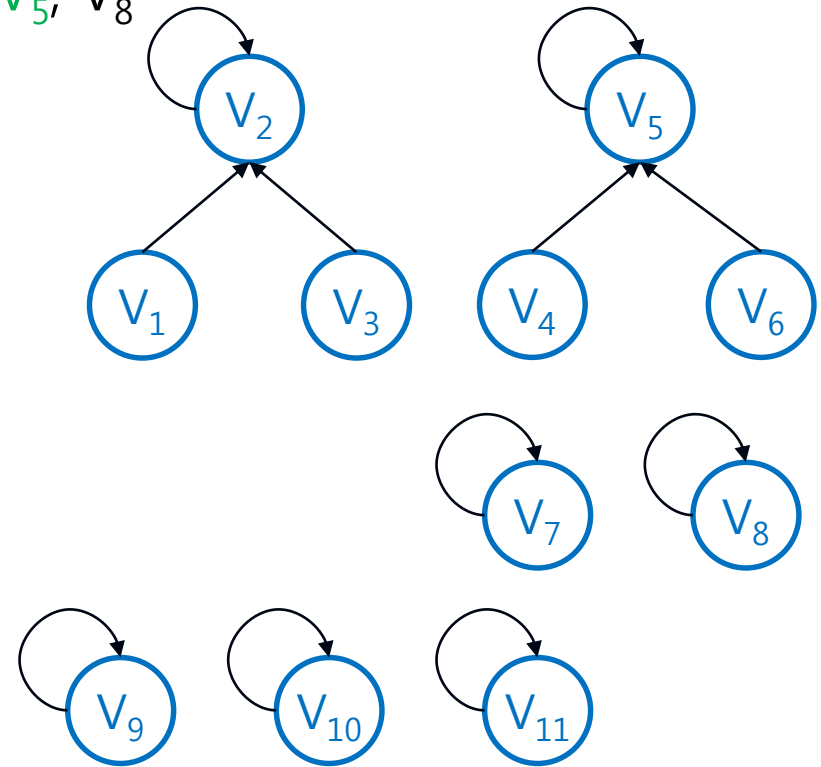
```
1: procedure DSDBSCAN( $X, eps, minpts$ )
2:   for each point  $x \in X$  do
3:      $p(x) \leftarrow x$ 
4:   for each point  $x \in X$  do
5:      $N \leftarrow \text{GETNEIGHBORS}(x, eps)$ 
6:     if  $|N| \geq minpts$  then
7:       mark  $x$  as core point
8:       for each point  $x' \in N$  do
9:         if  $x'$  is a core point then
10:          UNION( $x, x'$ )
11:        else if  $x'$  is not yet member of any cluster then
12:          mark  $x'$  as member of a cluster
13:          UNION( $x, x'$ )
```

DBSCAN: Project Algorithm

- Neighbor들을 union (pseudo code에 따라)
 - Union by rank
 - Path compression



$V_2: V_1, V_3$
 $V_5: V_4, V_6$
 $V_6: V_5, V_8$

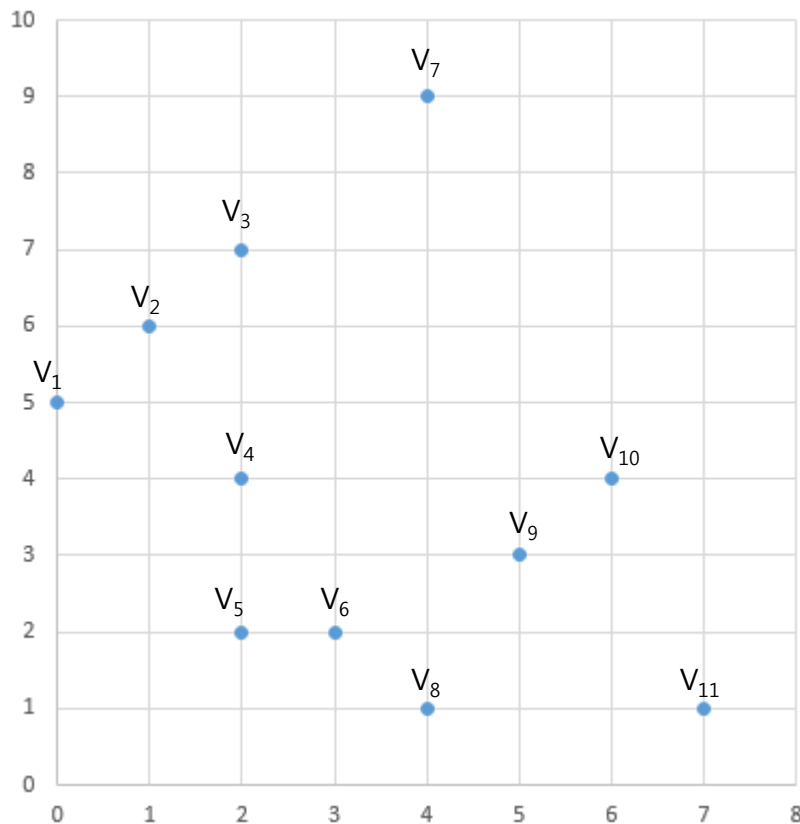


Algorithm 2 The disjoint-set data structure based DBSCAN Algorithm (DSDBSCAN). Input: A set of points X , distance eps , and the minimum number of points required to form a cluster, $minpts$. Output: A set of clusters.

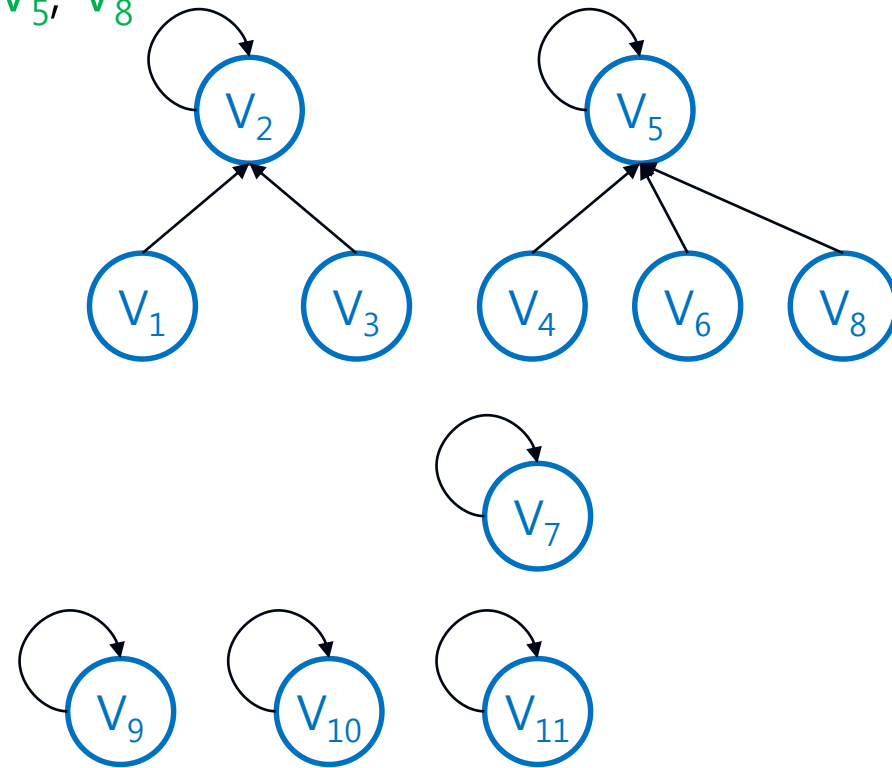
```
1: procedure DSDBSCAN( $X, eps, minpts$ )
2:   for each point  $x \in X$  do
3:      $p(x) \leftarrow x$ 
4:   for each point  $x \in X$  do
5:      $N \leftarrow \text{GETNEIGHBORS}(x, eps)$ 
6:     if  $|N| \geq minpts$  then
7:       mark  $x$  as core point
8:       for each point  $x' \in N$  do
9:         if  $x'$  is a core point then
10:          UNION( $x, x'$ )
11:        else if  $x'$  is not yet member of any cluster then
12:          mark  $x'$  as member of a cluster
13:          UNION( $x, x'$ )
```

DBSCAN: Project Algorithm

- Neighbor들을 union (pseudo code에 따라)
 - Union by rank
 - Path compression



$V_2: V_1, V_3$
 $V_5: V_4, V_6$
 $V_6: V_5, V_8$



Algorithm 2 The disjoint-set data structure based DBSCAN Algorithm (DSDBSCAN). Input: A set of points X , distance eps , and the minimum number of points required to form a cluster, $minpts$. Output: A set of clusters.

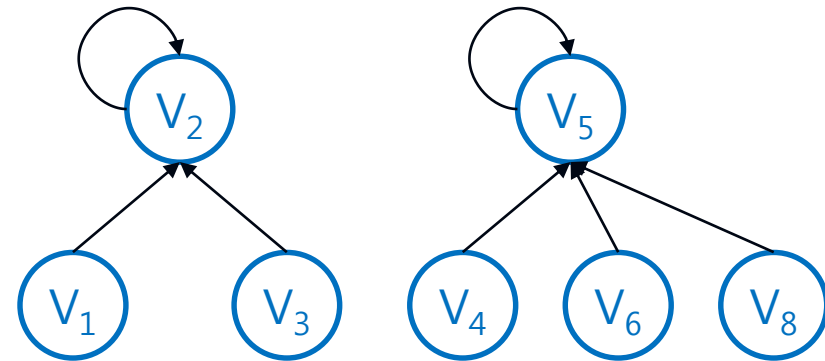
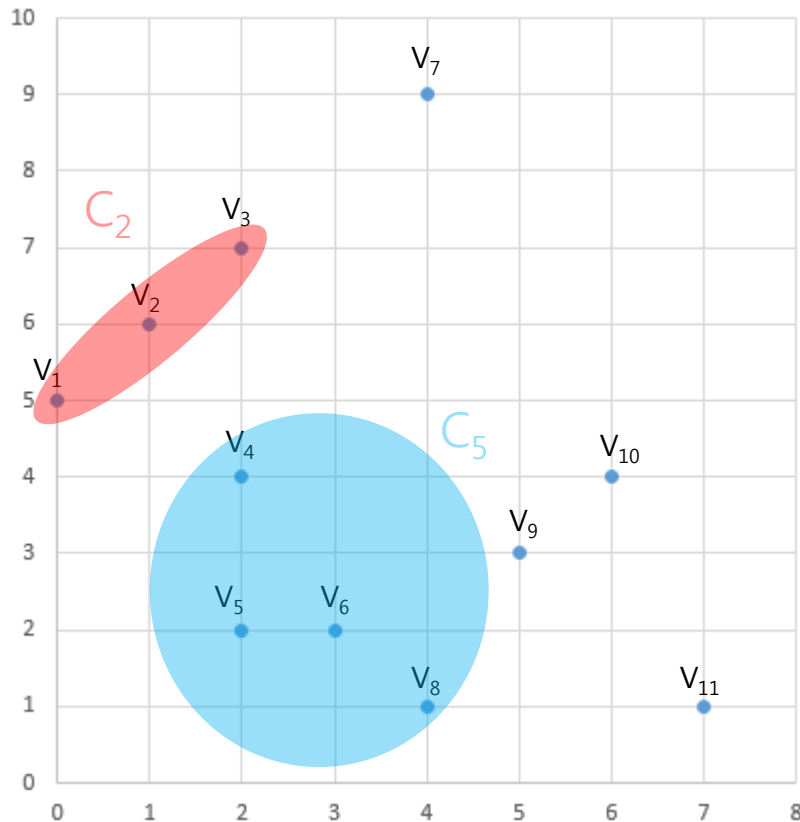
```
1: procedure DSDBSCAN( $X, eps, minpts$ )
2:   for each point  $x \in X$  do
3:      $p(x) \leftarrow x$ 
4:   for each point  $x \in X$  do
5:      $N \leftarrow \text{GETNEIGHBORS}(x, eps)$ 
6:     if  $|N| \geq minpts$  then
7:       mark  $x$  as core point
8:       for each point  $x' \in N$  do
9:         if  $x'$  is a core point then
10:          UNION( $x, x'$ )
11:        else if  $x'$  is not yet member of any cluster then
12:          mark  $x'$  as member of a cluster
13:          UNION( $x, x'$ )
```

DBSCAN: Project Algorithm

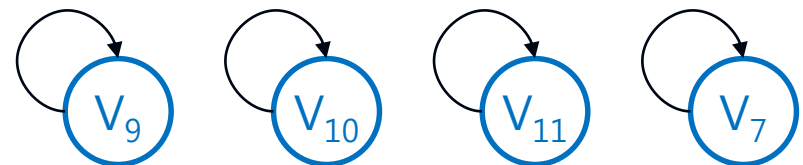
- Output은 각 point가 어떤 cluster에 속하는지 출력 (cout 이용)
 - Cluster id는 root point id로 출력
 - Noise는 -1로 출력

Algorithm 2 The disjoint-set data structure based DBSCAN Algorithm (DSDBSCAN). Input: A set of points X , distance eps , and the minimum number of points required to form a cluster, $minpts$.
Output: A set of clusters.

```
1: procedure DSDBSCAN( $X, eps, minpts$ )
2:   for each point  $x \in X$  do
3:      $p(x) \leftarrow x$ 
4:   for each point  $x \in X$  do
5:      $N \leftarrow \text{GETNEIGHBORS}(x, eps)$ 
6:     if  $|N| \geq minpts$  then
7:       mark  $x$  as core point
8:       for each point  $x' \in N$  do
9:         if  $x'$  is a core point then
10:          UNION( $x, x'$ )
11:       else if  $x'$  is not yet member of any cluster then
12:         mark  $x'$  as member of a cluster
13:         UNION( $x, x'$ )
```

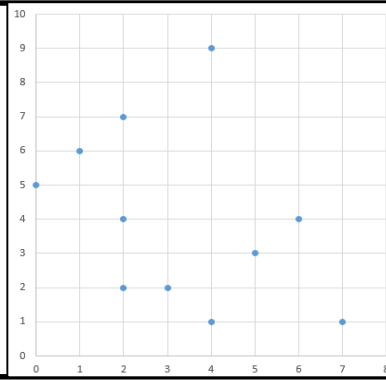


나머지는 noise



DBSCAN Overview

Coordinate data

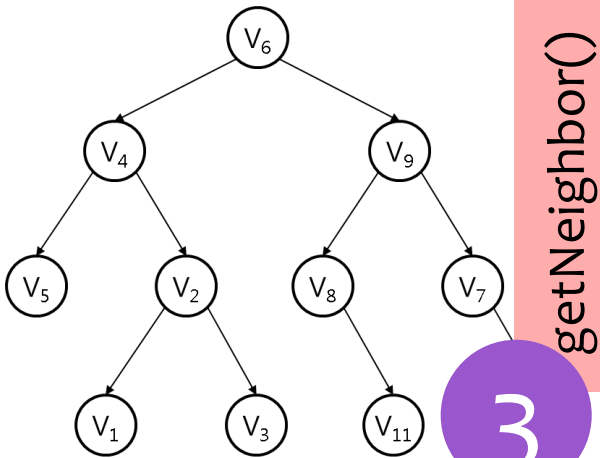


1

Build a
kd-tree

float**

kd-tree



getNeighbor()

3

2

DBSCAN

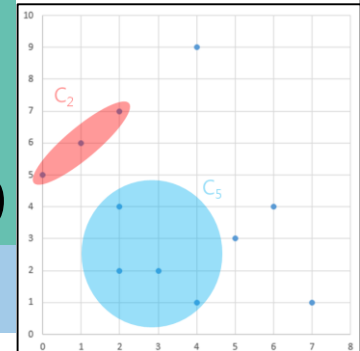
Run()

4

Check
conditions
&
Clustering
(Union & Find)

Coordinate

CoordinateSet



Project Information

Project

- 최소 필요 class
 - Coordinate
 - CoordinateSet 각 class들의 제시된 함수들 반드시 구현
 - KDtree
 - DBSCAN
 - Class들의 디자인 (member variables, functions) 은 직접 생각할 것.
 - 이 외 class 추가해도 된다. (disjoint set class, container class)

■ Member function들 input/output 예시

예: foo()

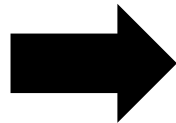
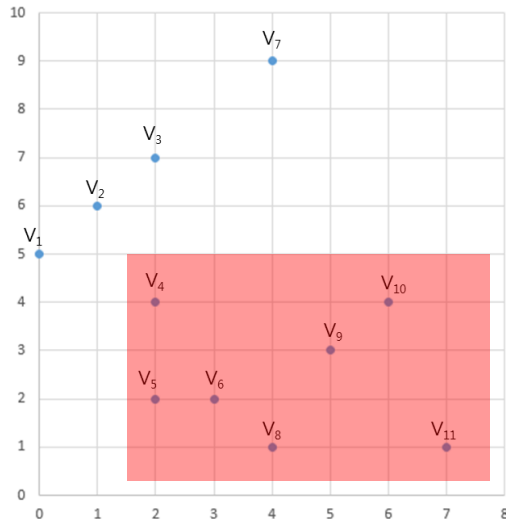
Input: int a, float b

Output: CoordinateSet cset

1. CoordinateSet foo(int a, float b)
2. CoordinateSet* foo(int a, float b)
3. CoordinateSet& foo(int& a, float* b)
4. void foo(int a, float b, CoordinateSet* cset)
5. void foo(int a, float b, CoordinateSet& cset)
6. ...

Project

- **Coordinate** class
 - 좌표 데이터를 저장하는 저장 class
- **CoordinateSet** class
 - 여러 Coordinate object (좌표데이터) 들을 저장하는 저장 class
 - print()
 - Input과 Output은 없으나, cout을 이용하여 다음과 같이 출력할 것



```
4 (2, 4)
5 (2, 2)
6 (3, 2)
8 (4, 1)
9 (5, 3)
10 (6, 4)
11 (7, 1)
```

Point_id (k dimesion 좌표들)

·
·
·

**CoordinateSet 안에 있는
point id 순서대로 출력**

예시에서는 2D이기 때문에 (x, y) 출력
만약 4D라면 (a, b, c, d) 로 출력

Project

- **KDtree** class
 - Constructor
 - Input: k-dimensional point들의 좌표 (**float**** points) → n x k 행렬
 - Input: dimension (int dimension)
 - Input: point 개수 (int numOfPoints)
 - getNeighbors()
 - Input: query point 좌표 (**Coordinate** queryPoint)
 - Input: query radius 길이 (float radius)
 - Output: query 결과 검색된 k-dim point들의 좌표 (**CoordinateSet** neighborPoints)

Project

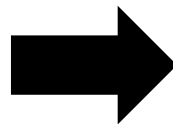
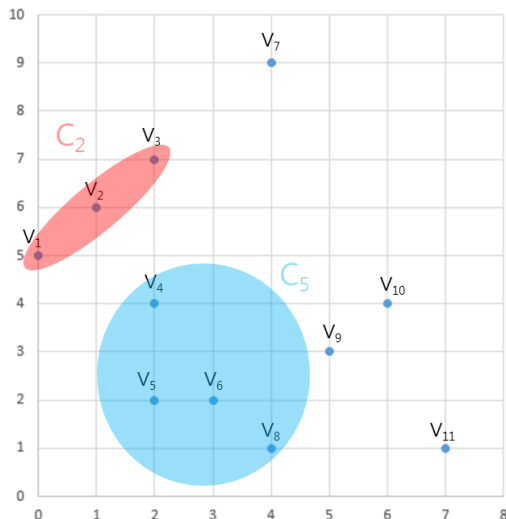
■ DBSCAN class (project algorithm 구현)

– Constructor

- Input: 만들어진 KDtree (**KDtree&** kdtree)
- Input: DBSCAN parameter ϵ (float epsilon)
- Input: core point가 되기 위한 neighbor의 최소 개수 (int minPoints)
(= cluster를 이루기 위한 최소 개수)

– run()

- Input과 Output은 없으나, cout을 이용하여 다음과 같이 출력할 것



```
1 2
2 2
3 2
4 5
5 5
6 5
7 -1
8 5
9 -1
10 -1
11 -1
```

Point_id cluster_id

·
·
·

Point id 순서대로 출력

Project 주의사항

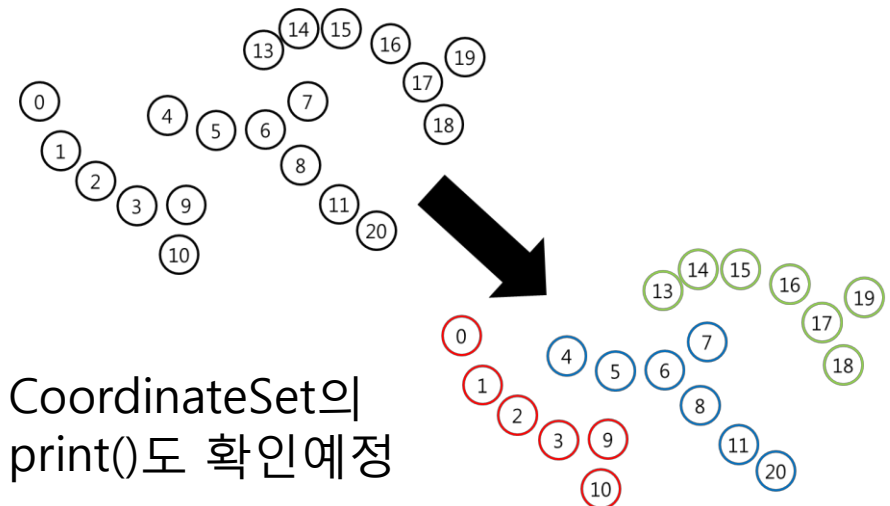
■ Input 관련

- Input은 k-dimensional point들의 좌표들이 float** 타입으로 main에서 주어질 것이고, 이는 KDtree를 생성할 때 parameter로 들어갈 것.
- Point의 index는 input 순서대로 할당하고 **0부터 시작한다.**
 - 예시에서는 이해를 돕기 위해 1부터 시작.

■ Test 관련

- 자신이 구현한 알고리즘을 확인하기 위해 여러 test를 해볼 것.
- Test 코드는 다음과 같다.

```
void test()
{
    float** data = ...;
    KDtree myTree(data, 2, 20);
    DBSCAN myDBSCAN(myTree, 0.5, 3);
    myDBSCAN.run();
}
```



Project 주의사항

- 제출 관련
 - 제출 압축파일명: 2015-12345.zip
 - visual studio project 폴더 (“빌드 → 솔루션 정리” 후)
 - Project 보고서
- Project 보고서
 - 개발자의 입장에서, 사용자들에게 배포할 문서라 생각
 - 간결하고 명확한 설명
 - Class들의 디자인 및 관계 (overview)
 - 함수들의 설명: 동작원리·역할 / 사용방법 / input과 output
 - 분량은 평가 요인 아님 (되도록 짧게)
 - 예시
 - <https://cran.r-project.org/web/packages/gputools/gputools.pdf>
 - <http://kr.mathworks.com/help/matlab/ref/plot.html>

Project 주의사항

■ 채점 관련

– 채점 방식: On/Off

- 코드 정확도

- 주어진 테스트셋에서 동작하면 On, 안 하면 Off

- 코드 효율성 (코드 동작 시 확인)

- 비 효율적 (Time, Memory) → Off

- 가이드 무시 → Off

- 코드 도용 및 공유

- 학칙에 의거 처리

■ 기한

- 제출: 12월 21일 (월) 23:59 까지

- 질문: 12월 21일 (월) 18:00 까지