Algorithm of dynamic convex hull

by Overmars and van Leeuwen

Martin Vavrek

Outline

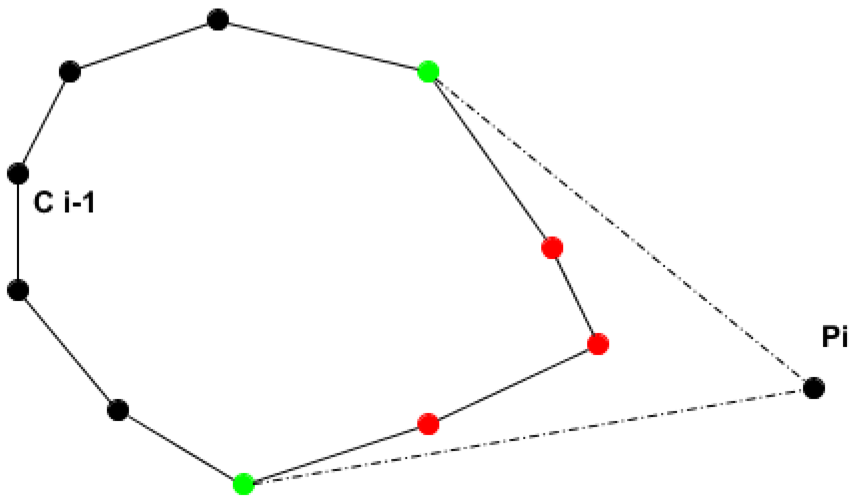
- Problem overview
- Basic approach
- Data Structures
- Procedures
  - Bridging Convex hulls
  - Inserting point
  - Deleting point
- Complexity

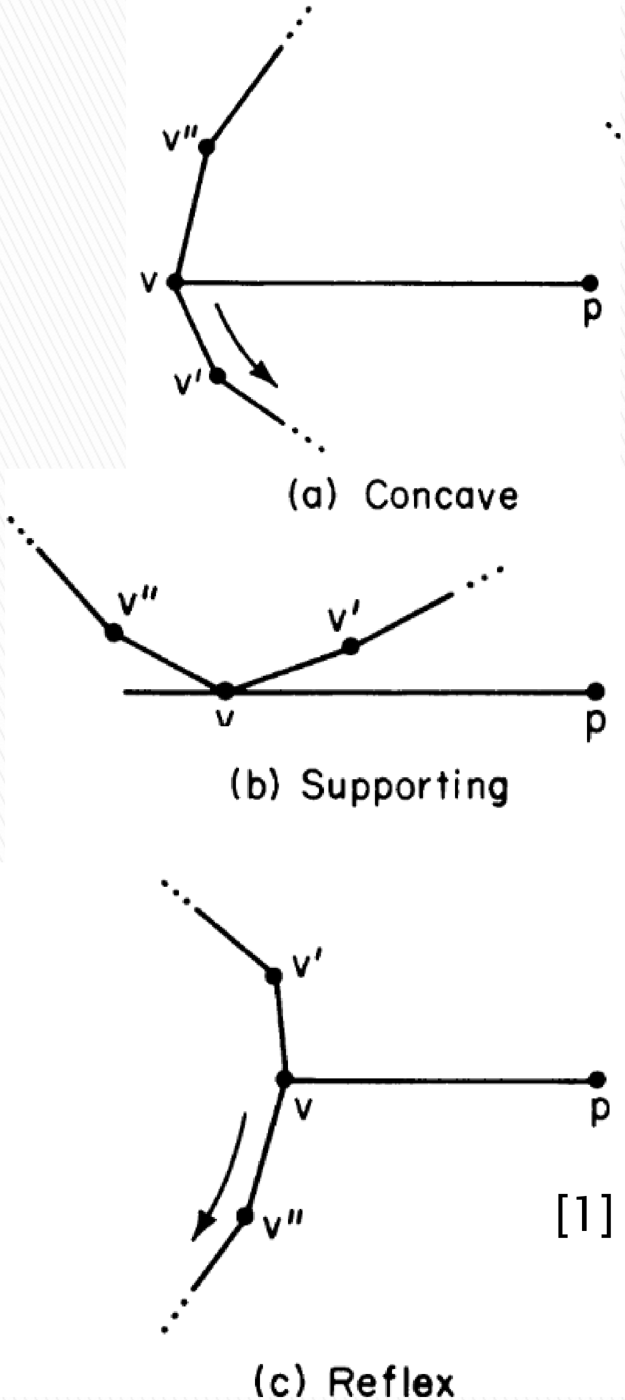
Problem overview

- Creating convex hulls online
  - Inserting points
  - Deleting points
  - Naive approach not satisfying
- Needed in many applications

Basic approach

- Looking for supporting points

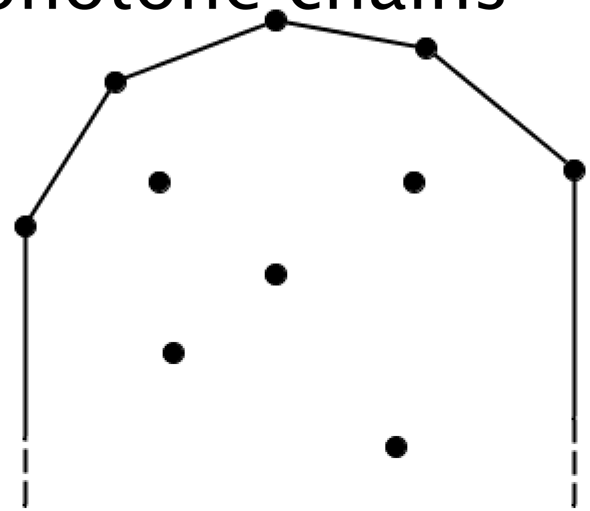




  - Looking for right data structure

Upper and Lower hull

- Convex hull is union of two monotone chains (lower and upper)
- $UH = CH(S \cup L_{-\infty})$
- $LH = CH(S \cup L_{+\infty})$

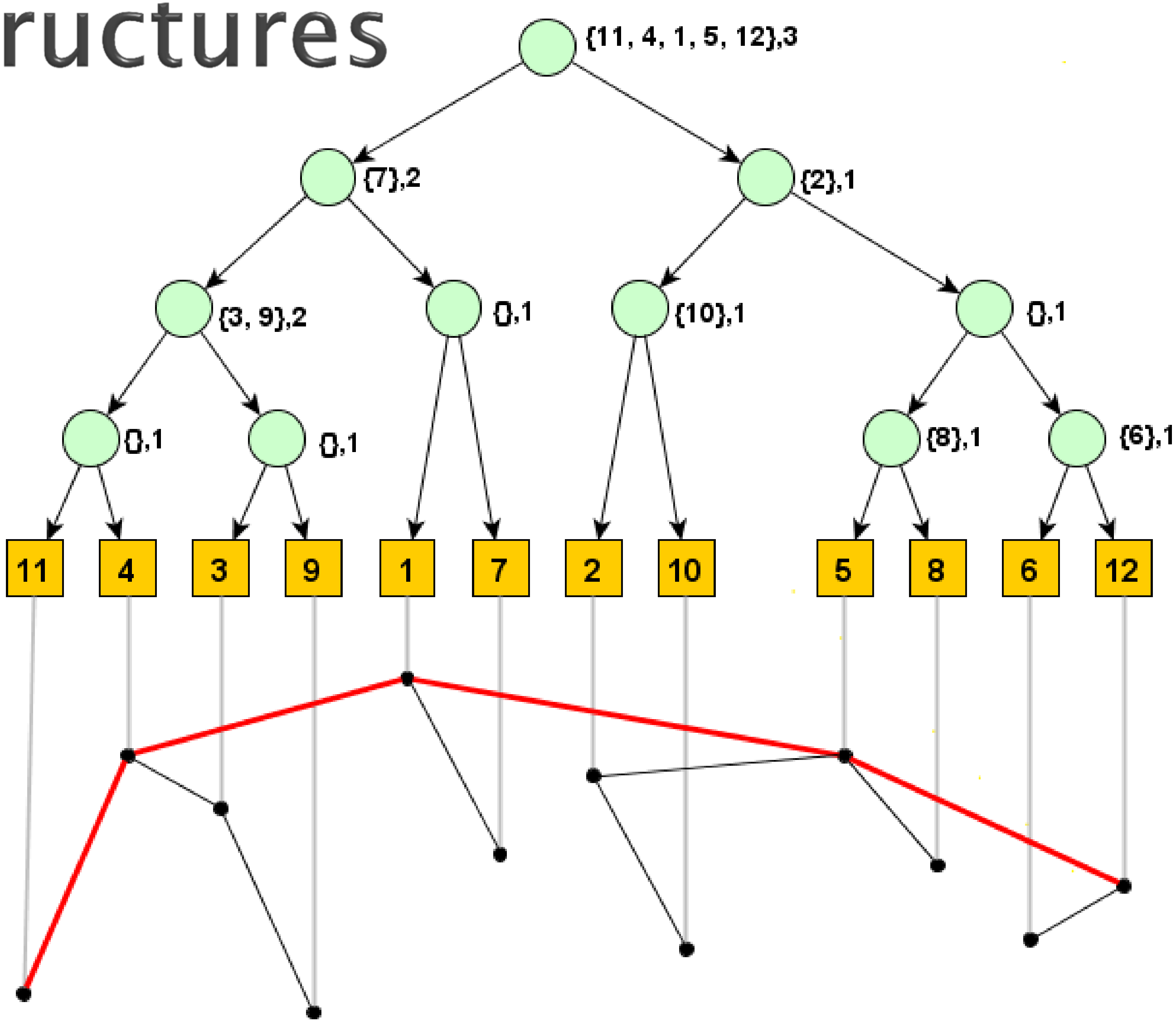


- We will focus on UH only, all operations are analogous for LH

Data Structures

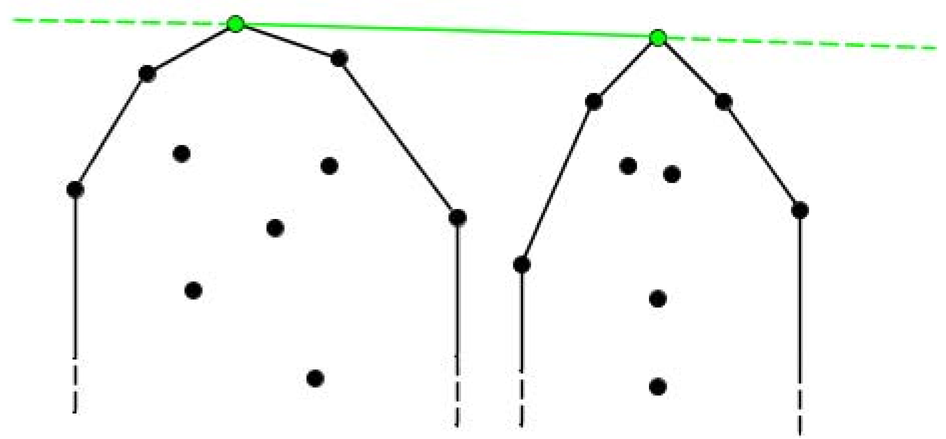
- Height balanced binary Search Tree
  - Representation of convex hulls in internal nodes
  - Points at leaves
- Concatenable queue
  - Represented by BST
  - Operations SPLIT and SPLICE in O(log i)
  - Represents Points

Data structures

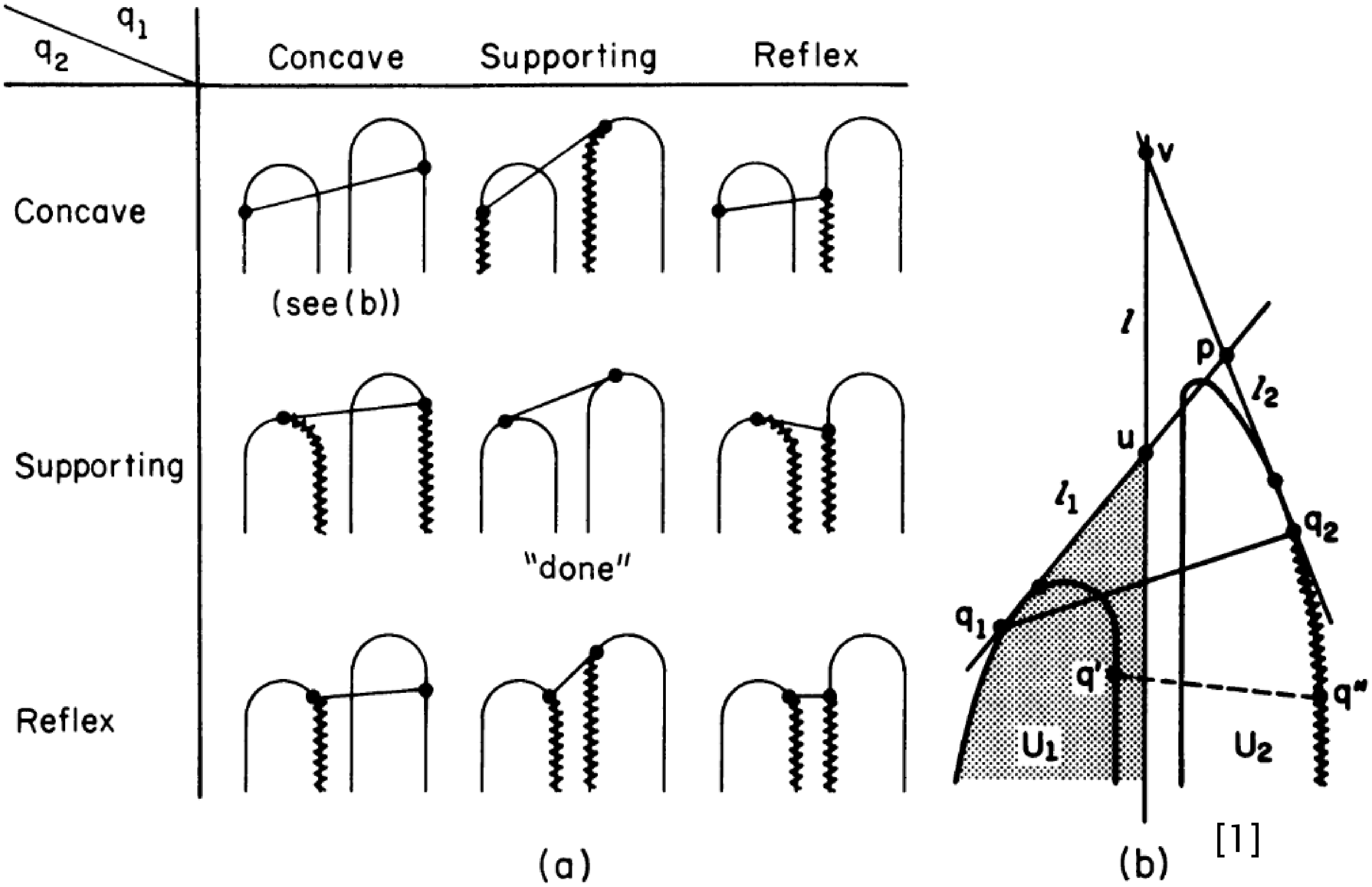


Bridging

- We need this operation when inserting/deleting point
- We have 2 UH, and want to their UH
- Looking for supporting points



Bridging



q_1
q_2
Concave
Supporting
Reflex
Concave
Supporting
Reflex
(see (b))
"done"
(a)
v
l
p
l_2
u
l_1
q_2
q_1
q'
q"
U_1
U_2
(b)

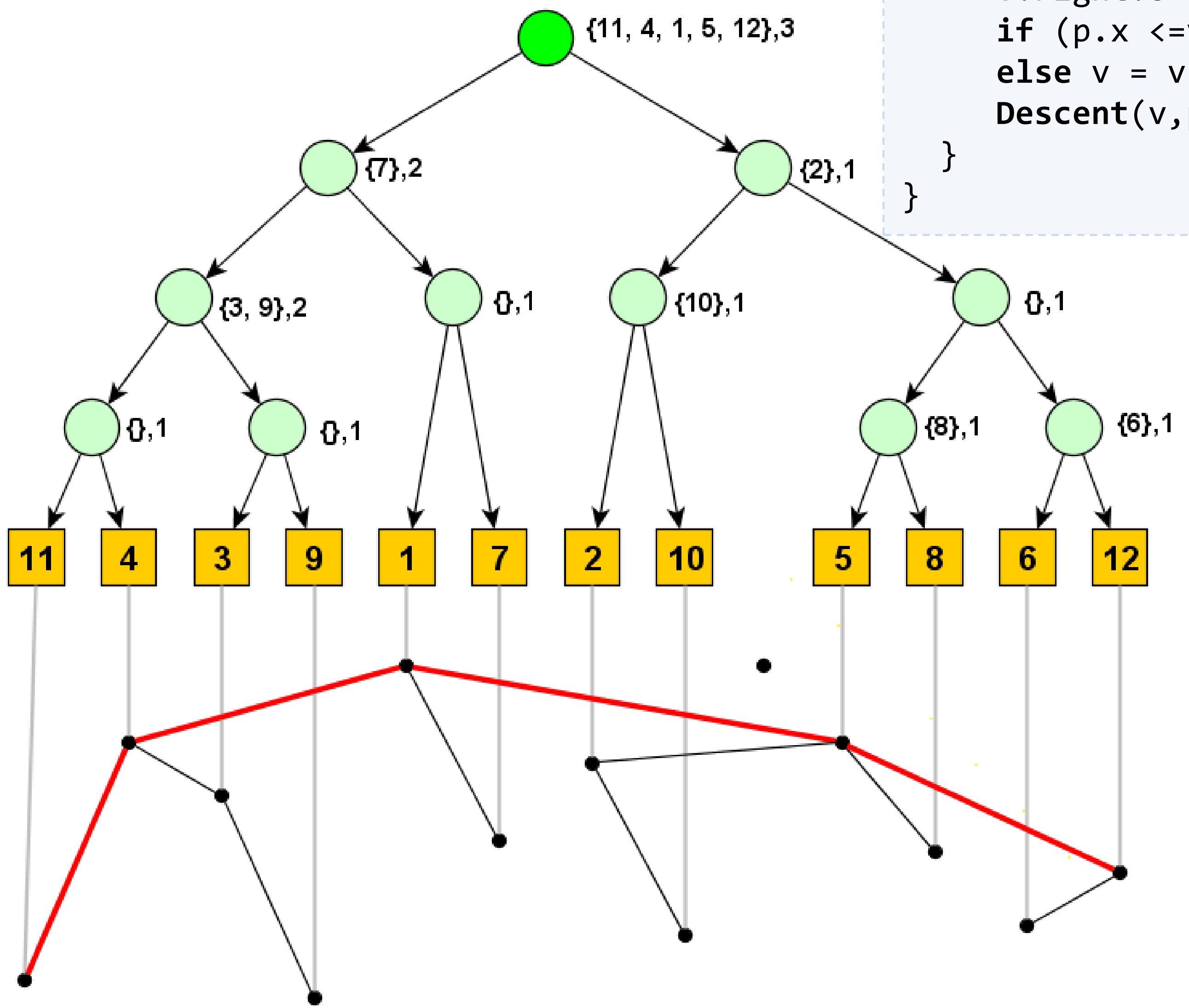
[1]

Adding a point

- First we need to find place for point in tree
  - Method Descent
- Then traverse back to root and reconstruct tree
  - Method Ascend
- Rebalancing Tree, if needed
  - Techniques described in Combinatorial algorithms by Edward M. Reingold, Jurg Nievergelt, Narsingh Deo ,1977

Descend

```
Descent(node v, value p)
{
  if(v is not leaf)
  {
     (Q_L,Q_R) = SPLIT(v.U,v.J)
     v.left.U = SPLICE (Q_L, v.left.Q);
     v.right.U = SPLICE (v.right.Q, Q_R);
     if (p.x <=v.x) v=v.left;
     else v = v.right;
     Descent(v,p)
  }
}
```



v.U = {11,4,1,5,12}

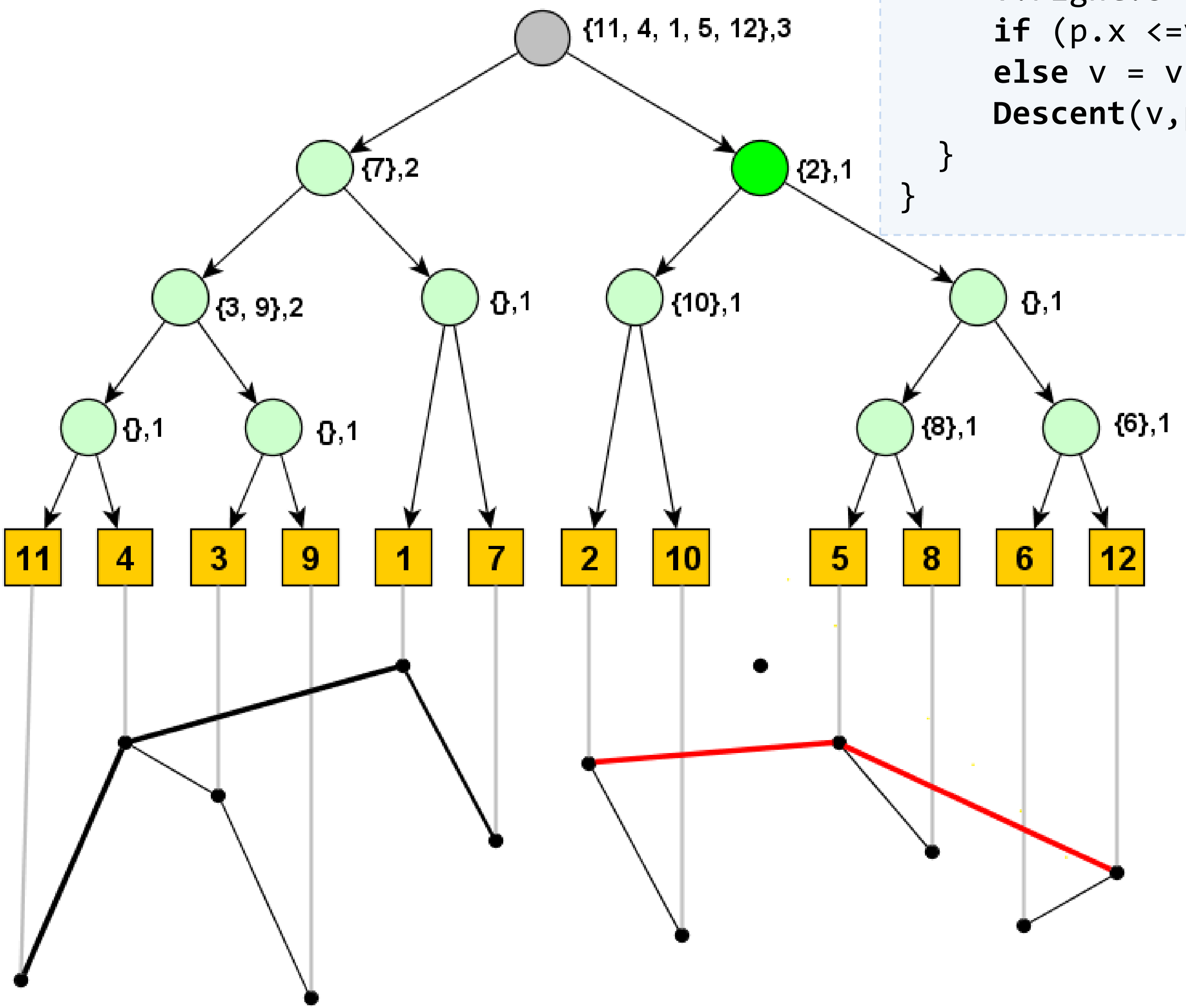
Q_L= {11,4,1}

Q_R= {5,12}

v.left.U={11,4,1,7}

v.right.U={2,5,12}

Descend



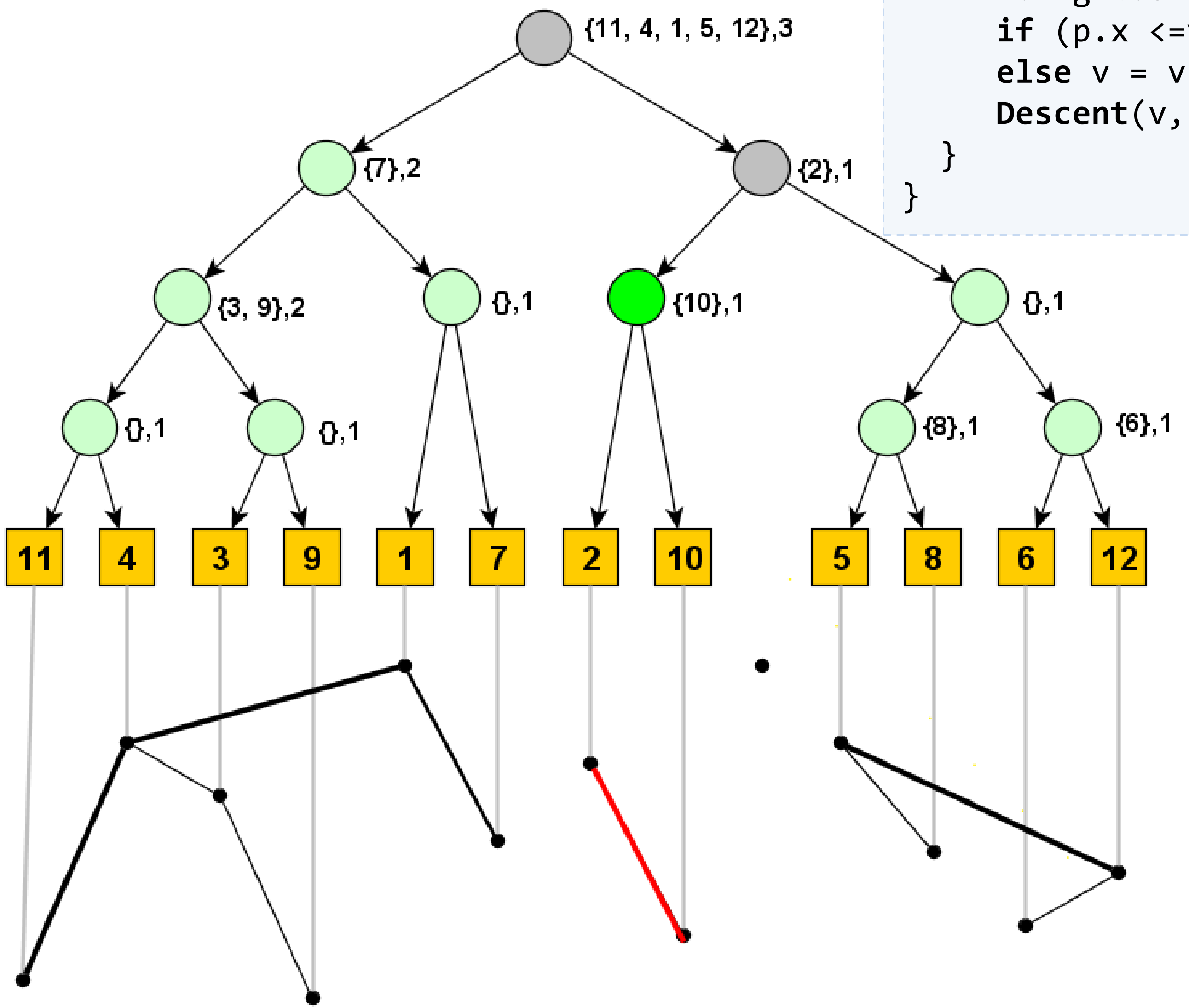

```
Descent(node v, value p)
{
  if(v is not leaf)
  {
     (Q_L,Q_R) = SPLIT(v.U,v.J)
     v.left.U = SPLICE (Q_L, v.left.Q);
     v.right.U = SPLICE (v.right.Q, Q_R);
     if (p.x <=v.x) v=v.left;
     else v = v.right;
     Descent(v,p)
  }
}
```

v.U = {2,5,12}
Q_L= {2}
Q_R= {5,12}

v.left.U = {2,10}
v.right.U = {5,12}

Descend

```
Descent(node v, value p)
{
  if(v is not leaf)
  {
     (Q_L,Q_R) = SPLIT(v.U,v.J)
     v.left.U = SPLICE (Q_L, v.left.Q);
     v.right.U = SPLICE (v.right.Q, Q_R);
     if (p.x <=v.x) v=v.left;
     else v = v.right;
     Descent(v,p)
  }
}
```

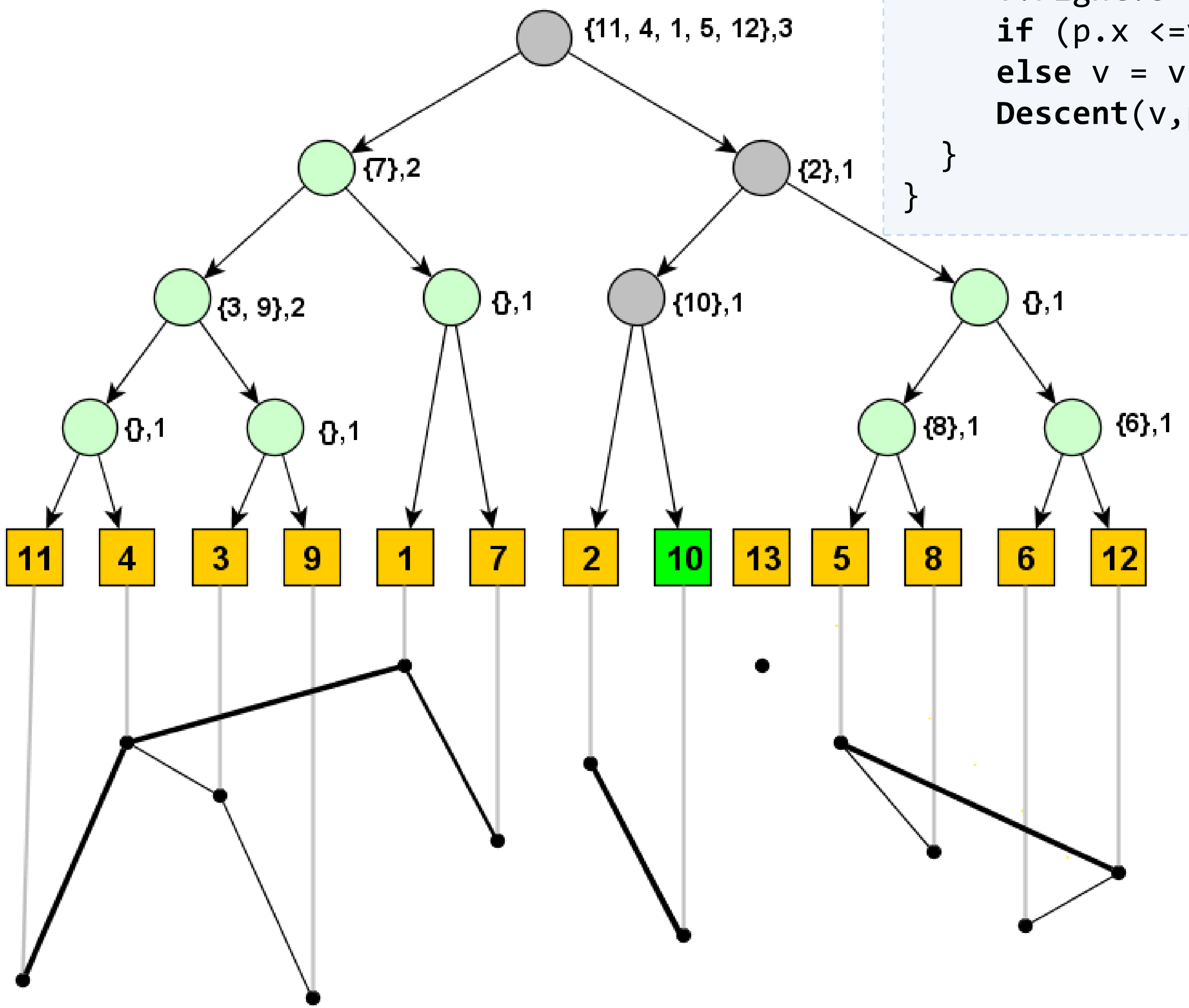


v.U = {2,10}
Q_L= {2}
Q_R= {10}

v.left.U = {2}
v.right.U = {10}

Descend

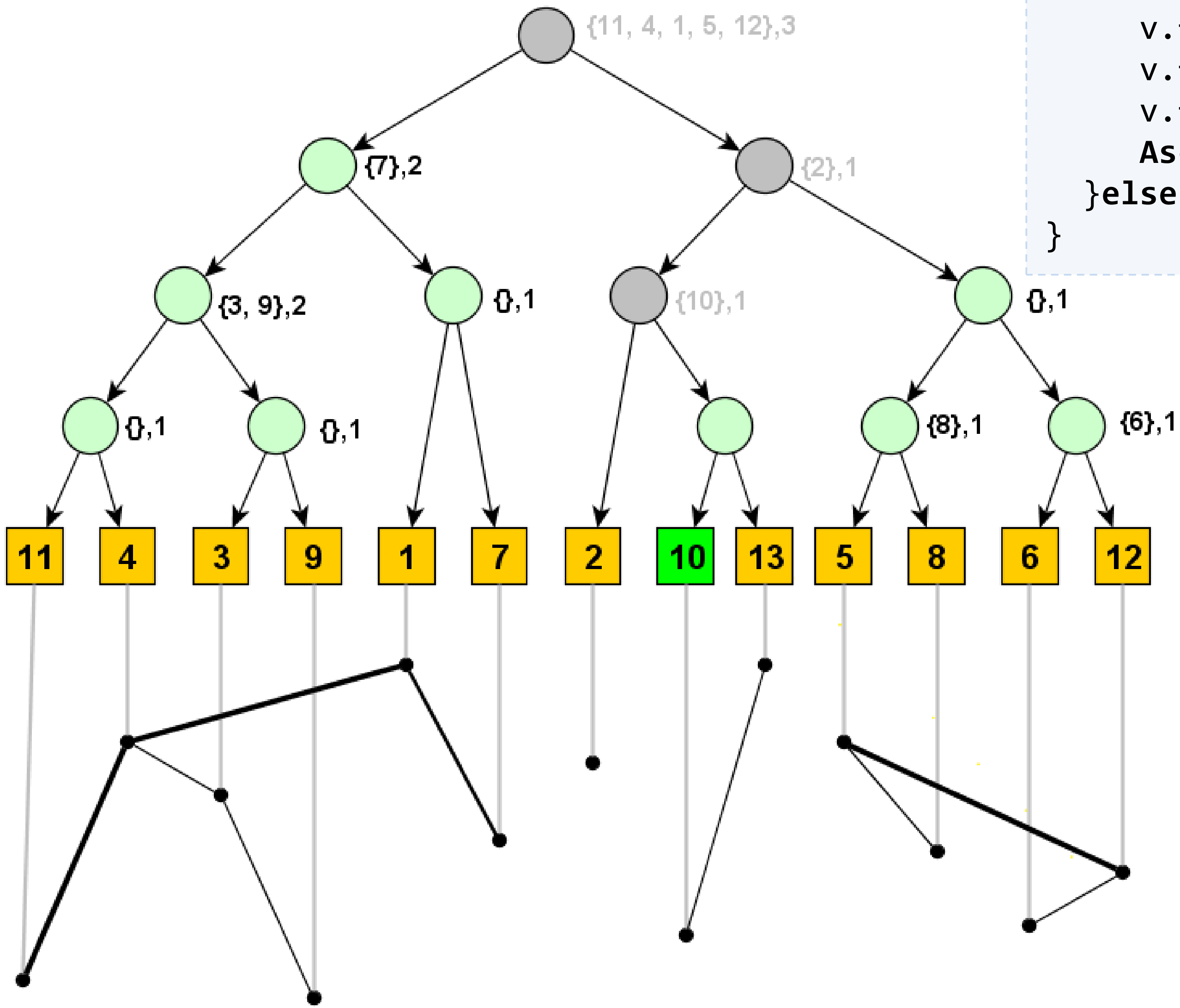
```
Descent(node v, value p)
{
  if(v is not leaf)
  {
     (Q_L,Q_R) = SPLIT(v.U,v.J)
     v.left.U = SPLICE (Q_L, v.left.Q);
     v.right.U = SPLICE (v.right.Q, Q_R);
     if (p.x <=v.x) v=v.left;
     else v = v.right;
     Descent(v,p)
  }
}
```



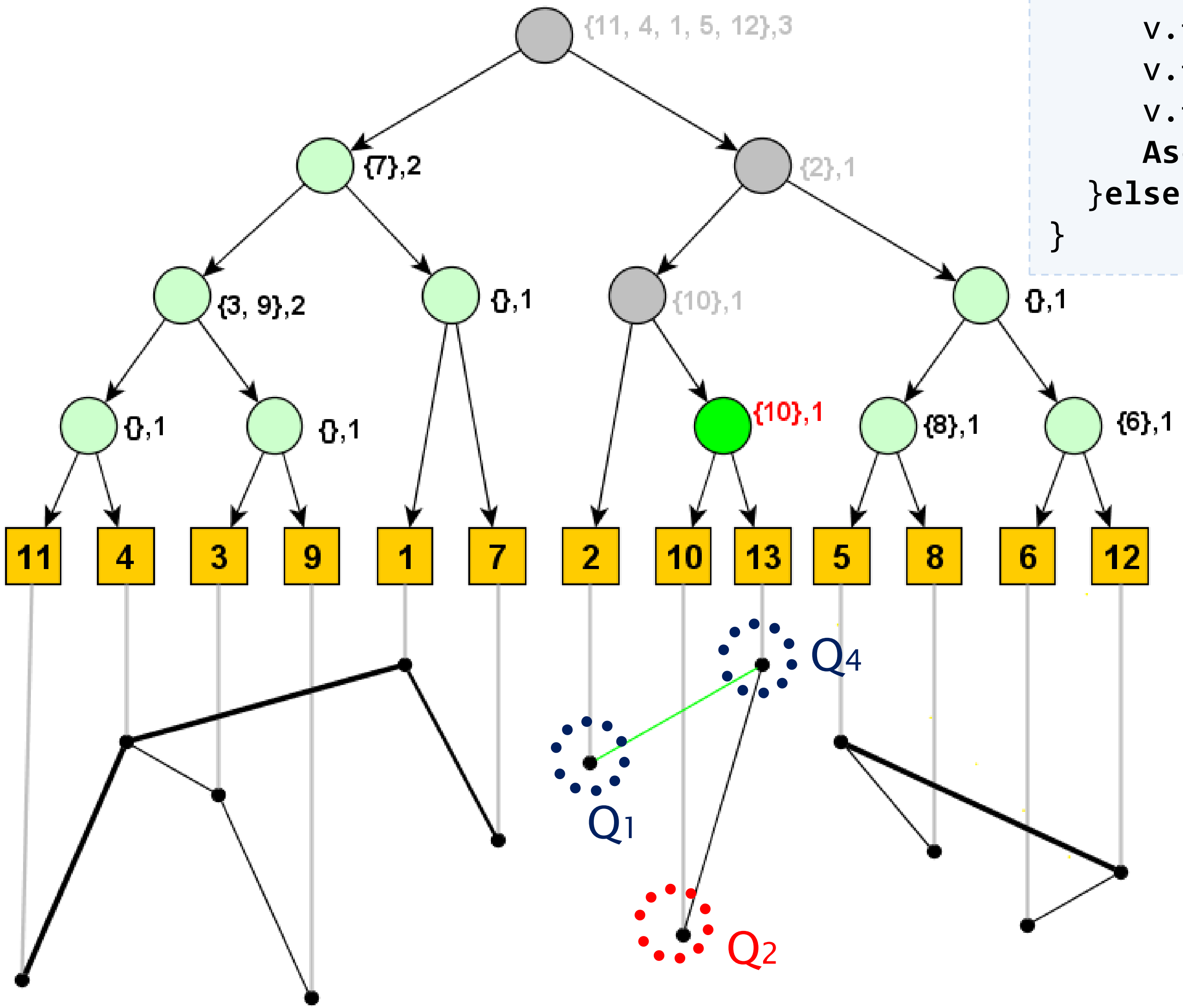
v is leaf

Ascend

```
Ascend(node v)
{
  if(v is not root)
  {
     (Q1,Q2,Q2,Q4,J) =
BRIDGE(v.U,v.sibling);
     v.father.left.Q = Q2;
     v.father.right.Q = Q3;
     v.father.U = SPLICE (Q1,Q4);
     v.father.J = J;
     Ascend(v.father)
  }else v.Q = v.U;
}
```



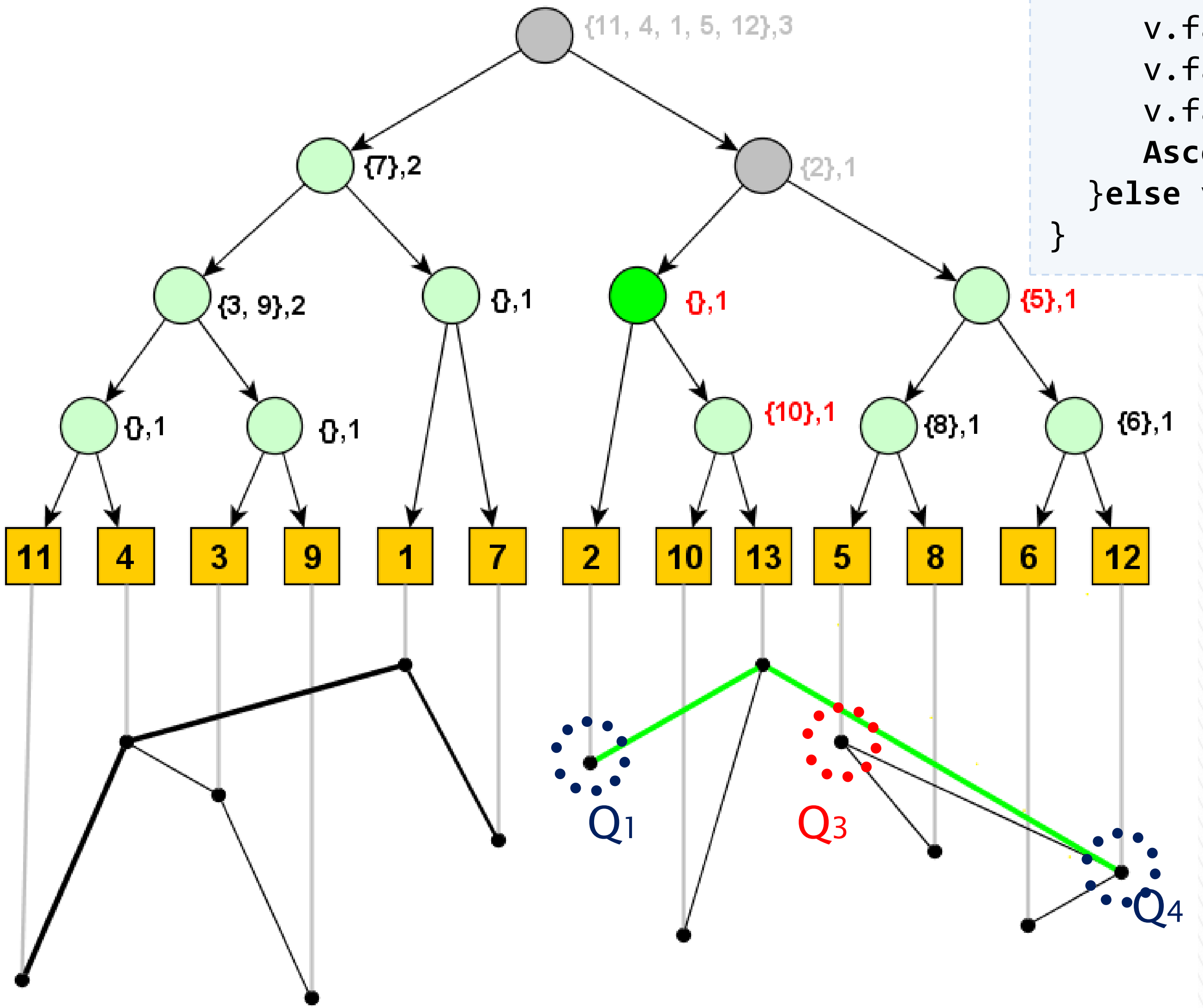
Ascend



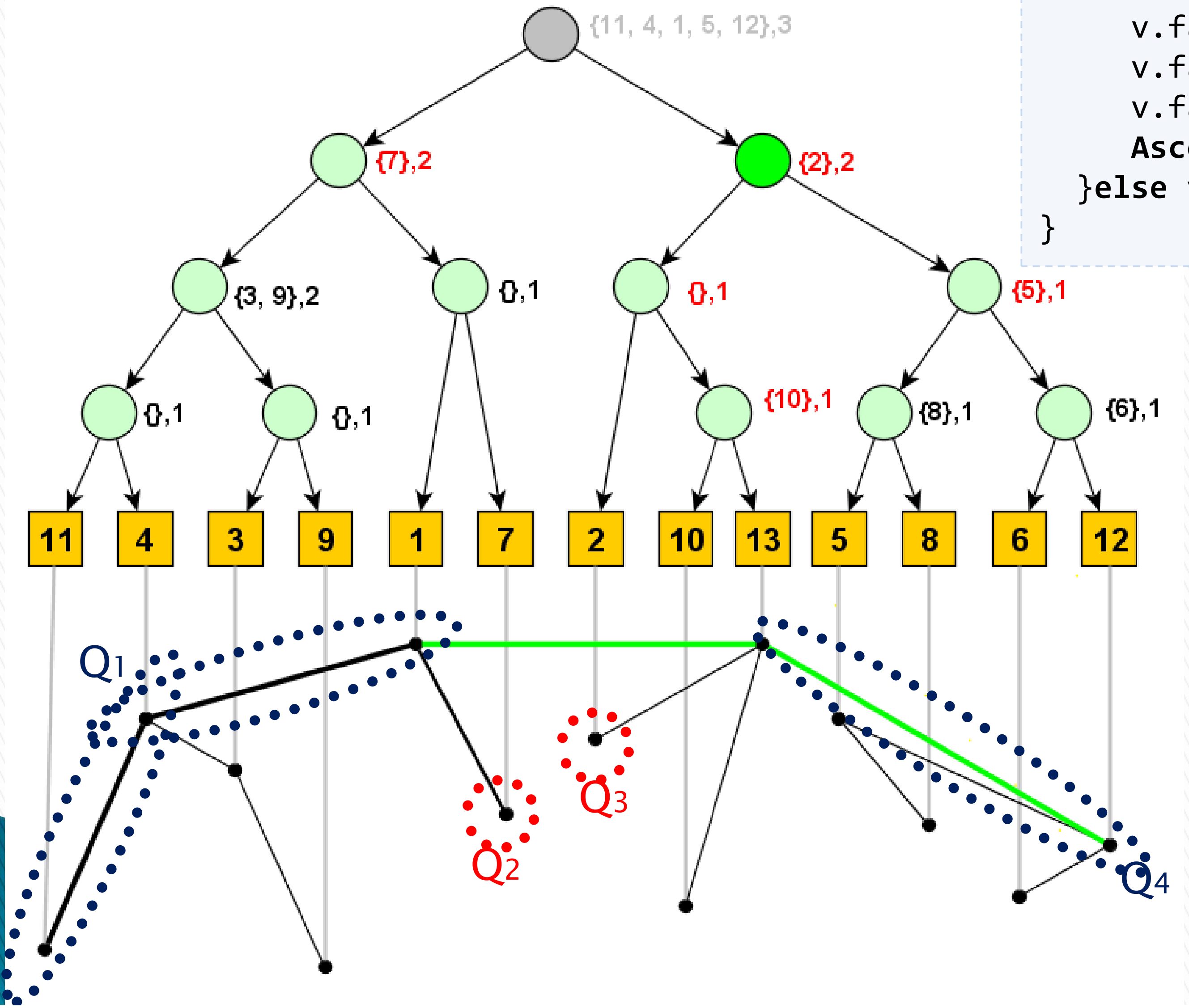

```
Ascend(node v)
{
  if(v is not root)
  {
     (Q1,Q2,Q2,Q4,J) =
BRIDGE(v.U,v.sibling);
     v.father.left.Q = Q2;
     v.father.right.Q = Q3;
     v.father.U = SPLICE (Q1,Q4);
     v.father.J = J;
     Ascend(v.father)
  }else v.Q = v.U;
}
```

Ascend

```
Ascend(node v)
{
  if(v is not root)
  {
     (Q1,Q2,Q2,Q4,J) =
BRIDGE(v.U,v.sibling);
     v.father.left.Q = Q2;
     v.father.right.Q = Q3;
     v.father.U = SPLICE (Q1,Q4);
     v.father.J = J;
     Ascend(v.father)
  }else v.Q = v.U;
}
```



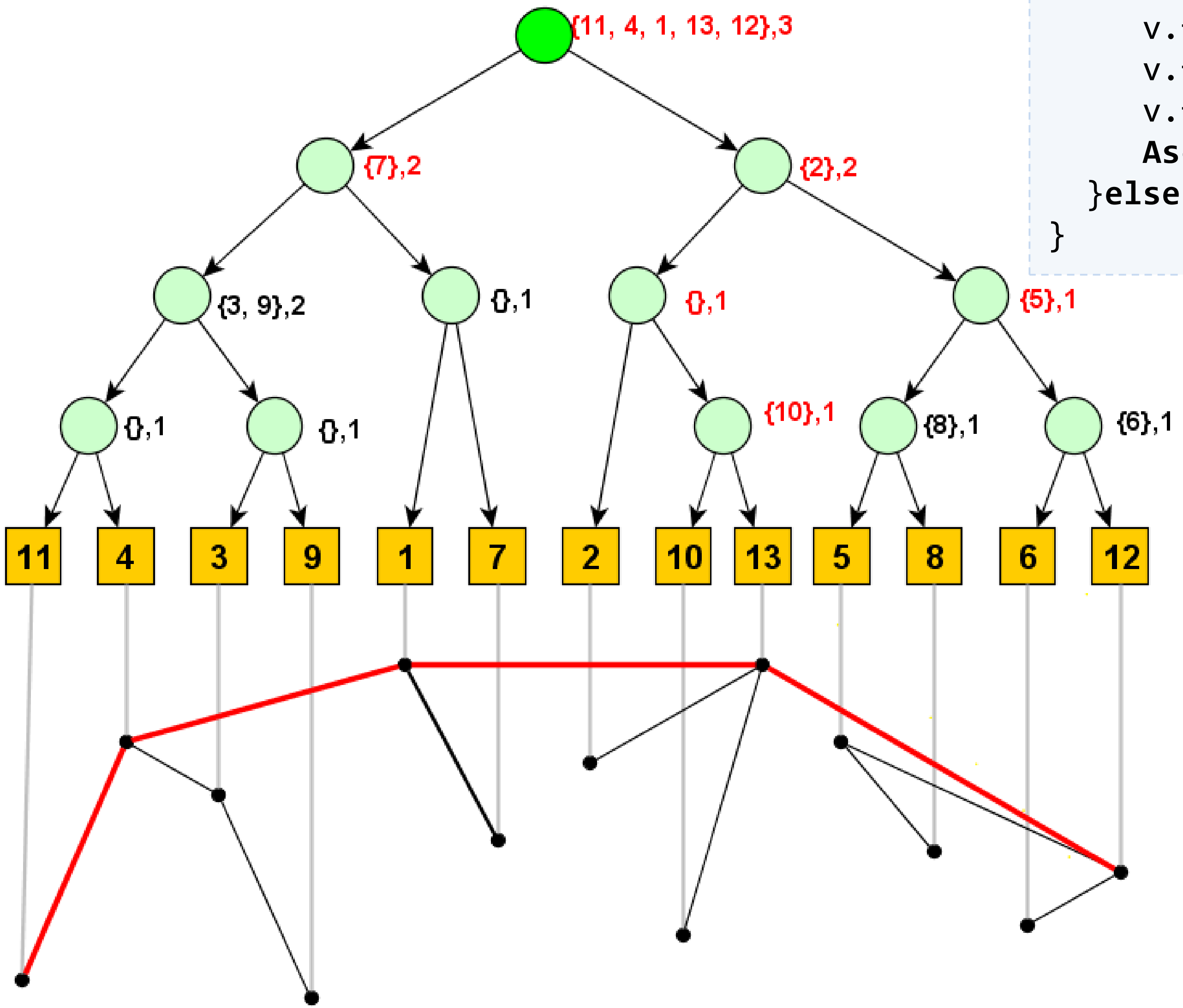
Ascend




```
Ascend(node v)
{
  if(v is not root)
  {
      (Q1,Q2,Q2,Q4,J) =
BRIDGE(v.U,v.sibling);
      v.father.left.Q = Q2;
      v.father.right.Q = Q3;
      v.father.U = SPLICE (Q1,Q4);
      v.father.J = J;
      Ascend(v.father)
  }else v.Q = v.U;
}
```

Ascend

```
Ascend(node v)
{
  if(v is not root)
  {
     (Q1,Q2,Q2,Q4,J) =
BRIDGE(v.U,v.sibling);
     v.father.left.Q = Q2;
     v.father.right.Q = Q3;
     v.father.U = SPLICE (Q1,Q4);
     v.father.J = J;
     Ascend(v.father)
  }else v.Q = v.U;
}
```



Deleting a point

- In same manner as Inserting
  - Traverse bottom and find point
  - Delete it from tree
  - Traverse up to fix tree

Complexity

- Memory O(N)
  - We have tree with N leaves and N-1 internal nodes
  - We have N-1 Concatenate queues (union of them is Convex hull)
- Time $O(N.\log^2(N))$
  - SPLIT and SPLICE in O(log k), where k<=N
  - Traversing tree in O(log(N))
  - Bridging in O(log(i)), where I <=N
    - =>Worst Case for Descend $O(\log^2(N))$
    - =>Worst Case for Ascend $O(\log^2(N))$

References

- [1] Franco P. Preparata, Michael Ian Shamos, *Computational Geometry, An Introduction; Springer; 1993*

Thank you for attention

- Any Questions ?