The bijection between forests and parking functions

Heesung Shin

Combinatorics Lab.
Department of Mathematics
Korea Advanced Institute of Science and Technology

August 10, 2006

Outline

KAIST

Outline

KAIST

History

The followings have same cardinality.

- T_{n+1} = trees with $n+1$ vertices
- F_n = forests with n vertices
- PF_n = parking functions with length n
- $\mathcal{S}_n$ = Shi arrangements of length n
- $\mathcal{F}_{(n+1)}$ = cycle factorizations of length $n+1$
- Q_n = allowable pairs of length n

We knew that the number of these are all $(n+1)^{n-1}$. Actually, we tried finding the bijection among them and, of course, there are many various bijections that we found.
But we need a new bijection for the next problem.

KAIST

History

The followings have same cardinality.

- T_{n+1} = trees with $n+1$ vertices
- F_n = forests with n vertices
- PF_n = parking functions with length n
- $\mathcal{S}_n$ = Shi arrangements of length n
- $\mathcal{F}_{(n+1)}$ = cycle factorizations of length $n+1$
- Q_n = allowable pairs of length n

We knew that the number of these are all $(n+1)^{n-1}$. Actually, we tried finding the bijection among them and, of course, there are many various bijections that we found.
But we need a new bijection for the next problem.

KAIST

Question

R. Stanley wrote the following open problem in his lecture note 'Hyperplane Arrangement'

Exercise

6.4 Find a bijection proof of Theorem 6.22, i.e., find a bijection φ between the set of all rooted forests on $[n]$ and the set PF_n of all parking functions of length n satisfying

$$\mathrm{inv}(F) = \binom{n+1}{2} - a_1 - \cdots - a_n$$

when $\varphi(F) = (a_1, \ldots, a_n)$.

NOTE. In principle a bijection φ can be obtained by carefully analyzing the proof of Theorem 6.22. However, this bijection will be of a messy recursive nature. *A "nonrecursive" bijection would be greatly preferred.*

Answer

- G.Kreweras connected recursively inversion enumerators for tree, $I_n(t)$, with parking functions (1980).

$$I_n(t) = \sum_{T \in T_{n+1}} t^{\mathrm{inv}(T)}$$

  where tree with $(n+1)$ vertices and root '0'.
- The recursive proof is also written in R.Stanley's lecture note (2004).

KAIST

Outline

KAIST

Objective

We'll find the answer of the previous question.

$$\begin{aligned} \varphi \quad : \quad F_n &\to PF_n \\ F &\hookrightarrow P = \varphi(F) \\ \mathrm{inv}(F) &= \mathrm{jump}(P) = \tbinom{n+1}{2} - |P| \\ \mathrm{lead}(F) &= \mathrm{lucky}(P) \end{aligned}$$

Moreover, we have

$$\sum_{F \in F_n} q^{\mathrm{inv}(F)} u^{\mathrm{lead}(F)} = \sum_{P \in PF_n} q^{\mathrm{jump}(P)} u^{\mathrm{lucky}(P)}.$$

KAIST

Objective

We'll find the answer of the previous question.

$$\begin{aligned} \varphi \quad : \quad F_n &\to PF_n \\ F &\hookrightarrow P = \varphi(F) \\ \mathrm{inv}(F) &= \mathrm{jump}(P) = \tbinom{n+1}{2} - |P| \\ \mathrm{lead}(F) &= \mathrm{lucky}(P) \end{aligned}$$

Moreover, we have

$$\sum_{F \in F_n} q^{\mathrm{inv}(F)} u^{\mathrm{lead}(F)} = \sum_{P \in PF_n} q^{\mathrm{jump}(P)} u^{\mathrm{lucky}(P)}.$$

KAIST

Outline

KAIST

Tree I

Definition (Tree)

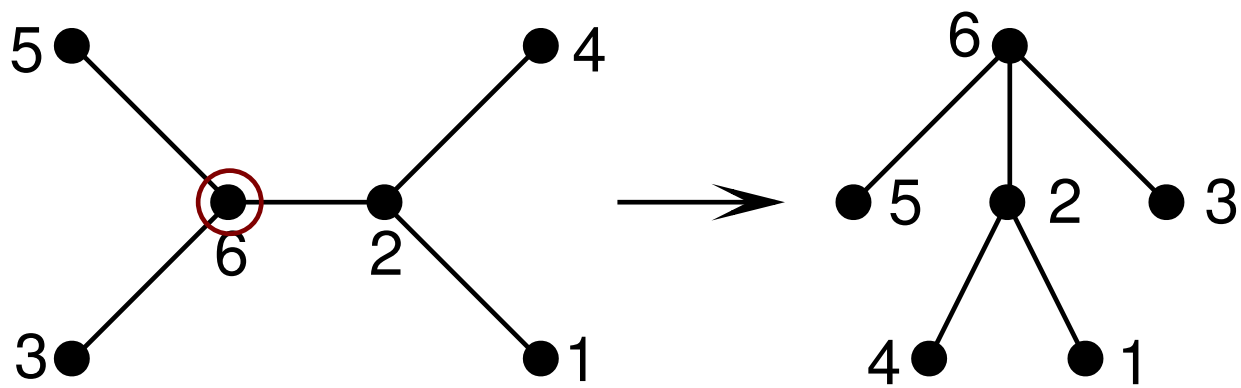
- Tree = simple graph, connected, no cycles
- Labeled Tree = tree, vertices are labeled
- Rooted Tree = tree, only one vertex is chosen as root.

NOTE. If we need to deal with a root in an unrooted labeled tree, then we consider the maximum vertex as a root, that is, an unrooted labeled tree is assumed as a rooted labeled tree with the root.

KAIST

Tree II

EXAMPLE.



NOTE.

1. From now on, 'unrooted labeled tree' is called shortly 'tree'.
2. When we draw a 'rooted labeled tree', usually draw the root at the top.

Tree III

Definition

T_n = the set of unrooted labeled trees with n vertices

Theorem (Cayley formula)

$$|T_n| = n^{n-2}$$

PROOF. This is proved by Prüfer Algorithm or Prüfer Code.

Forest I

Definition (Forest)

Forest = simple graph, no cycles (not necessarily connected)

NOTE. Forest is consisting of trees. Because each connected component is tree.

Definition

- Labeled Forest = forest, consisting of labeled trees
- Rooted Forest = forest, consisting of rooted trees

KAIST

Forest II

Definition

F_n = the set of rooted labeled forests on n vertices

Theorem

$$|F_n| = (n+1)^{n-1}$$

PROOF. It's clear. Because we get the rooted forest naturally from the unrooted tree by deleting the maximum vertex. In this way, the number of vertices is decreasing by 1. Then

$$|T_{n+1}| = |F_n| .$$

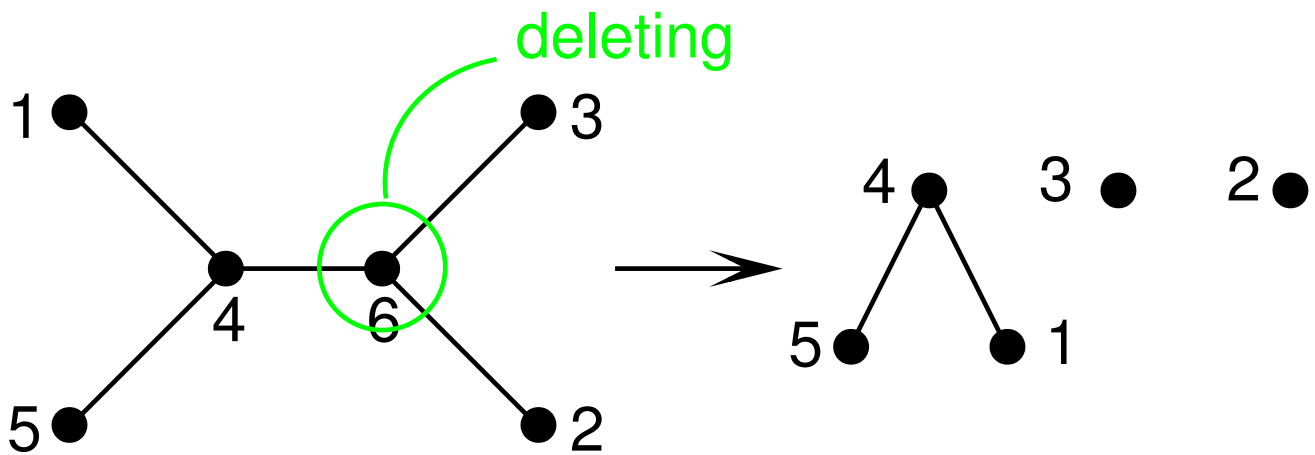
KAIST

Forest III

NOTE.

1. From now on, 'rooted labeled forest' is called shortly 'forest'.
2. We can change a tree to the forest by deleting the maximum vertex in the tree.

Example



Unrooted tree $\longrightarrow$ Rooted forest

Inversion in tree

Definition

- Inversion in tree = ordered pair (i,j) s.t. $i > j$ and j is a descendant of i
- $\mathrm{inv}(T)$ = the # of inversions in T
- $\mathrm{inv}(T:v)$ = the # of inversions in T of form (v,x) where $v > x$ and x is a descendant of v

REMARK. $\mathrm{inv}(T) = \sum_v \mathrm{inv}(T:v)$.

NOTE. j is a descendant of i
= i lies on the unique path from the root to j

Inversion in forest

Definition

- Inversion in forest = ordered pair (i,j) s.t. $i > j$ and j is a descendant of i
- $\mathrm{inv}(F)$ = the # of inversions in F
- $\mathrm{inv}(F:v)$ = the # of inversions in F of form (v,x) where $v > x$ and x is a descendant of v

REMARK. $\mathrm{inv}(F) = \sum_v \mathrm{inv}(F:v)$.

NOTE. This is well-defined since a forest is consisting of trees.

KAIST

Leader in tree and forest

Definition

- Leader in tree T = the vertex v where $\mathrm{inv}(T : v) = 0$
  = the smallest vertex among its all descendants.
- $\mathrm{lead}(T)$ = the # of all leaders in T

NOTE. By definition, all leaf is leader.

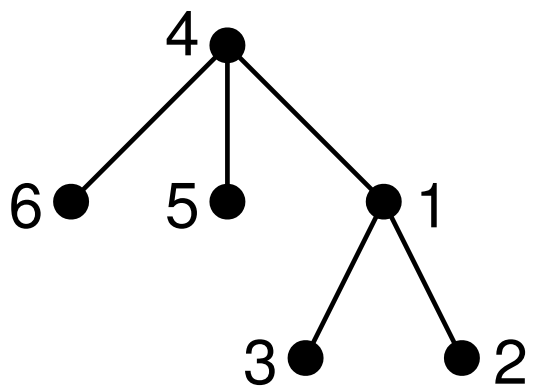
Definition

- Leader in forest F = the vertex v where $\mathrm{inv}(F : v) = 0$
  = the smallest vertex among its all descendants.
- $\mathrm{lead}(F)$ = the # of all leaders in F

KAIST

Example

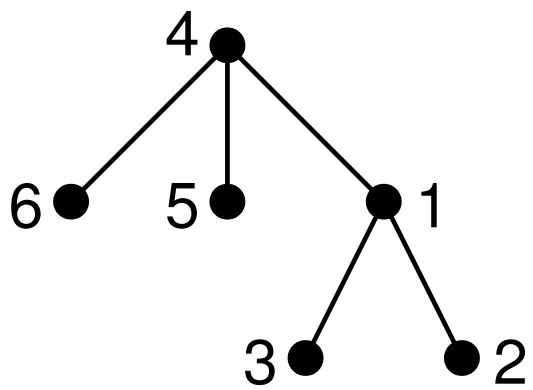
Look at this rooted tree T with root 4.



- $(4,3)$, $(4,1)$, $(4,3)$ are inversions of T
- All vertices are leaders of tree T except 4.
- $\mathrm{inv}(T) = 3$ and $\mathrm{lead}(T) = 5$

Example

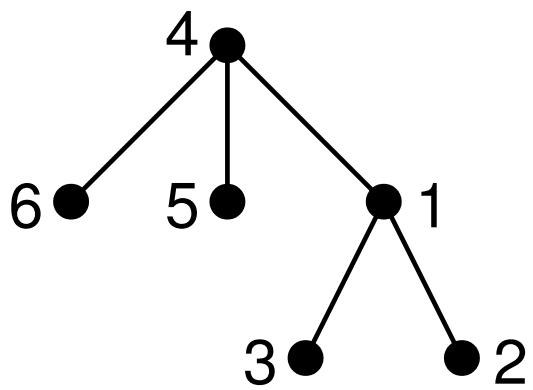
Look at this rooted tree T with root 4.



- $(4,3)$, $(4,1)$, $(4,3)$ are inversions of T
- All vertices are leaders of tree T except 4.
- $\mathrm{inv}(T) = 3$ and $\mathrm{lead}(T) = 5$

Example

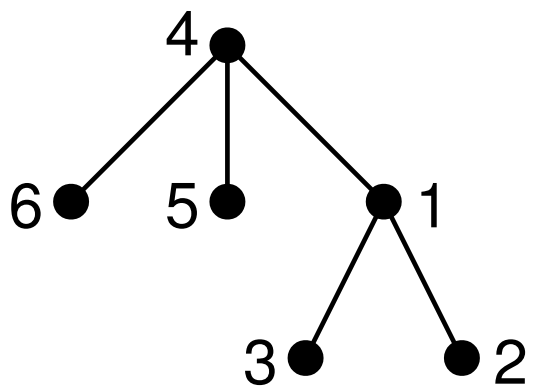
Look at this rooted tree T with root 4.



- $(4,3)$, $(4,1)$, $(4,3)$ are inversions of T
- All vertices are leaders of tree T except 4.
- $\mathrm{inv}(T) = 3$ and $\mathrm{lead}(T) = 5$

Example

Look at this rooted tree T with root 4.



- $(4,3)$, $(4,1)$, $(4,3)$ are inversions of T
- All vertices are leaders of tree T except 4.
- $\mathrm{inv}(T) = 3$ and $\mathrm{lead}(T) = 5$

Remark

In [6, Seo & S], we proved combinatorially by RP-code

$$\sum_{T \in T_n} u^{\mathrm{lead}(T)} c^{\deg_T(1)} = uP_{n-1}(1, u, cu)$$

where $P_n(a, b, c) = c \sum_{i=1}^{n-1} (ia + (n-i)b + c)$. Also, we know that each number of leaders and vertices is decreased by 1 when unrooted tree is changed to a forest by deleting the minimum vertex '1'. Hence we deduce that

$$\sum_{F \in F_n} u^{\mathrm{lead}(F)} c^{\mathrm{tree}(F)} = P_n(1, u, cu)$$

where tree means the number of trees in a forest.

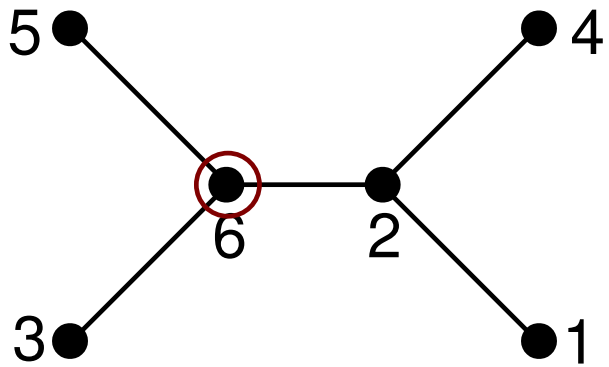
Prüfer order I

- We think about the order of the vertices in trees and forests.
- In making Prüfer Code from a tree, we remove the smallest leaves. In this time, we naturally get the order of deleting vertices. The first vertex is the smallest leaf and the last vertex is the maximum vertex (root) in unrooted tree. We will call it ‘Prüfer order’.
- RP-order means ‘Reverse Prüfer order’, that is, really the reverse order of Prüfer order. Trivially, the first in RP-order is the maximum vertex (root) in unrooted tree.

KAIST

Prüfer order II

We know RP-order of vertex in tree by traveling toward to the largest unvisiting vertex from maximum vertex (root) in an unrooted tree (or the root in rooted tree).



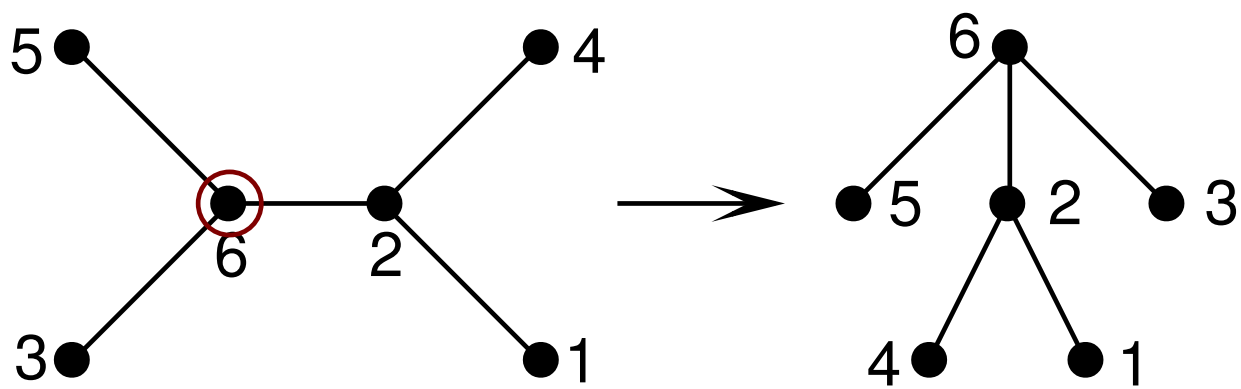
Prüfer order : $1 \to 3 \to 4 \to 2 \to 5 \to 6$

RP-order : $6 \to 5 \to 2 \to 4 \to 3 \to 1$

KAIST

The method by drawing tree

- We want to fix the shape of tree in one way.
- Draw the root at the top. We consider the maximum as the root in an unrooted tree.
- If we draw children, put children from earliest to latest RP-order among sibling from left to right.
- It seems that the shape of tree is figured as a rooted ordered tree after drawing.



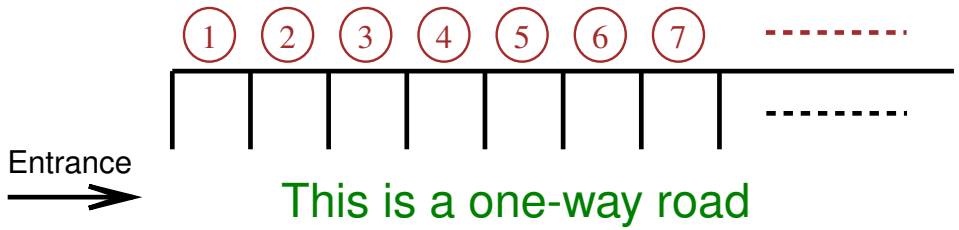
KAIST

Outline

KAIST

Parking Algorithm I

1. There are infinitely many parking spaces whose entrance is at the left.



2. This parking space is an one-way road from left to right and do not allow that car is going back.
3. Each car has its favorite parking space.
4. Two or more cars cannot be parked at one parking space.
5. Cars can be parked one by one from the first car to last car.
6. When a car is parked, a car drives at its parking space. And then, attempt to be parked at its favorite parking space. If the space is empty, the car is parked. Otherwise, attempt again at the next parking space. Repeat this process until the parks.

KAIST

Parking Algorithm II

Given a sequence $(p_1, p_2, \ldots, p_n)$ of length n, where p_i means the favorite parking space of the i-th car, we will park n cars into parking spaces by previous rules. This method is called a Parking Algorithm and the notation PA is used as the function name.

EXAMPLE. Given a sequence $(4, 3, 3, 1, 5)$, 5 cars are parked by the Parking Algorithm as following.

Parking Space	①	②	③	④	⑤	⑥	⑦	---------
Cars' Number	4	$\emptyset$	2	1	3	5	$\emptyset$	---------

We get a infinite sequence

$$PA(4, 3, 3, 1, 5) = (4, \emptyset, 2, 1, 3, 5, \emptyset, \emptyset, \ldots).$$

Parking Function I

Definition (Parking Function)

- If every car can be parked at one of first n parking spaces, the sequence $P = (p_1, p_2, \ldots, p_n)$ used in Parking Algorithm is called a parking function.
- PF_n = the set of parking functions with length n

Theorem

$$|PF_n| = (n+1)^{n-1}$$

KAIST

Parking Function II

Another Definition

Given a sequence $P = (p_1, p_2, \ldots, p_n)$, the followings are equivalent.

1. P is a parking function.
2. $PA(P)$ can be considered as a permutation of length n.
3. For all $k \leqslant n$, $\#\{i \mid p_i \leqslant k\} \geqslant k$.
4. If $p'_1 \leqslant p'_2 \leqslant \cdots \leqslant p'_n$ is the rearrangement of P, then $p'_k \leqslant k$ for all k.

Jump in parking function

Definition

- Jump in parking function = the attempt to park the next space because of a non-empty parking space
- $\mathrm{jump}(P : c)$ = the # of the jumps only to park car c
  = $Q_c - P_c$ where $Q = PA(P)^{-1}$
- $\mathrm{jump}(P)$ = the # of the jumps to park all cars
  = $\sum_c \mathrm{jump}(P : c)$

NOTE.

$$\mathrm{jump}(P) = \sum_c \mathrm{jump}(P : c) = \sum_c Q_c - P_c = \binom{n+1}{2} - |P|$$

KAIST

Lucky in parking function

Definition

- Lucky car in parking function P
  = the car c where $\mathrm{jump}(P : c) = 0$
  = the car which is parked at its favorite parking space
- $\mathrm{lucky}(P)$ = the # of all lucky cars in P

Example

Let $P = (2)(4)(2)(1)(3)$ be a parking function. After parking,

$$PA(P) = 4\ 1\ 3\ 2\ 5$$

Q_c	(1)	(2)	(3)	(4)	(5)
c	4	1	3	2	5
P_c	(1)	(2)	(2)	(4)	(3)
$Q_c - P_c$	0	0	1	0	2

- $\mathrm{jump}(P:3) = 1$ and $\mathrm{jump}(P:5) = 2$
- Lucky cars are 1, 2 and 4.
- $\mathrm{lucky}(P) = 3$ and $\mathrm{jump}(P) = 3$

KAIST

Example

Let $P = (2)(4)(2)(1)(3)$ be a parking function. After parking,

$$PA(P) = 4\ 1\ 3\ 2\ 5$$

Q_c	(1)	(2)	(3)	(4)	(5)
c	4	1	3	2	5
P_c	(1)	(2)	(2)	(4)	(3)
$Q_c - P_c$	0	0	1	0	2

- $\mathrm{jump}(P:3) = 1$ and $\mathrm{jump}(P:5) = 2$
- Lucky cars are 1, 2 and 4.
- $\mathrm{lucky}(P) = 3$ and $\mathrm{jump}(P) = 3$

KAIST

Example

Let $P = (2)(4)(2)(1)(3)$ be a parking function. After parking,

$$PA(P) = 4\ 1\ 3\ 2\ 5$$

Q_c	(1)	(2)	(3)	(4)	(5)
c	4	1	3	2	5
P_c	(1)	(2)	(2)	(4)	(3)
$Q_c - P_c$	0	0	1	0	2

- $\mathrm{jump}(P:3) = 1$ and $\mathrm{jump}(P:5) = 2$
- Lucky cars are 1, 2 and 4.
- $\mathrm{lucky}(P) = 3$ and $\mathrm{jump}(P) = 3$

KAIST

Example

Let $P = \textcircled{2}\textcircled{4}\textcircled{2}\textcircled{1}\textcircled{3}$ be a parking function. After parking,

$$PA(P) = 4\ 1\ 3\ 2\ 5$$

Q_c	①	②	③	④	⑤
c	4	1	3	2	5
P_c	①	②	②	④	③
$Q_c - P_c$	0	0	1	0	2

- $\mathrm{jump}(P:3) = 1$ and $\mathrm{jump}(P:5) = 2$
- Lucky cars are 1, 2 and 4.
- $\mathrm{lucky}(P) = 3$ and $\mathrm{jump}(P) = 3$

KAIST

Example

Let $P = \textcircled{2}\textcircled{4}\textcircled{2}\textcircled{1}\textcircled{3}$ be a parking function. After parking,

$$PA(P) = 4\ 1\ 3\ 2\ 5$$

Q_c	①	②	③	④	⑤
c	4	1	3	2	5
P_c	①	②	②	④	③
$Q_c - P_c$	0	0	1	0	2

- $\mathrm{jump}(P:3) = 1$ and $\mathrm{jump}(P:5) = 2$
- Lucky cars are 1, 2 and 4.
- $\mathrm{lucky}(P) = 3$ and $\mathrm{jump}(P) = 3$

KAIST

Example

Let $P = \textcircled{2}\textcircled{4}\textcircled{2}\textcircled{1}\textcircled{3}$ be a parking function. After parking,

$$PA(P) = 4\ 1\ 3\ 2\ 5$$

Q_c	①	②	③	④	⑤
c	4	1	3	2	5
P_c	①	②	②	④	③
$Q_c - P_c$	0	0	1	0	2

- $\mathrm{jump}(P:3) = 1$ and $\mathrm{jump}(P:5) = 2$
- Lucky cars are 1, 2 and 4.
- $\mathrm{lucky}(P) = 3$ and $\mathrm{jump}(P) = 3$

KAIST

Outline

KAIST

Diagram of φ and φ^{-1}

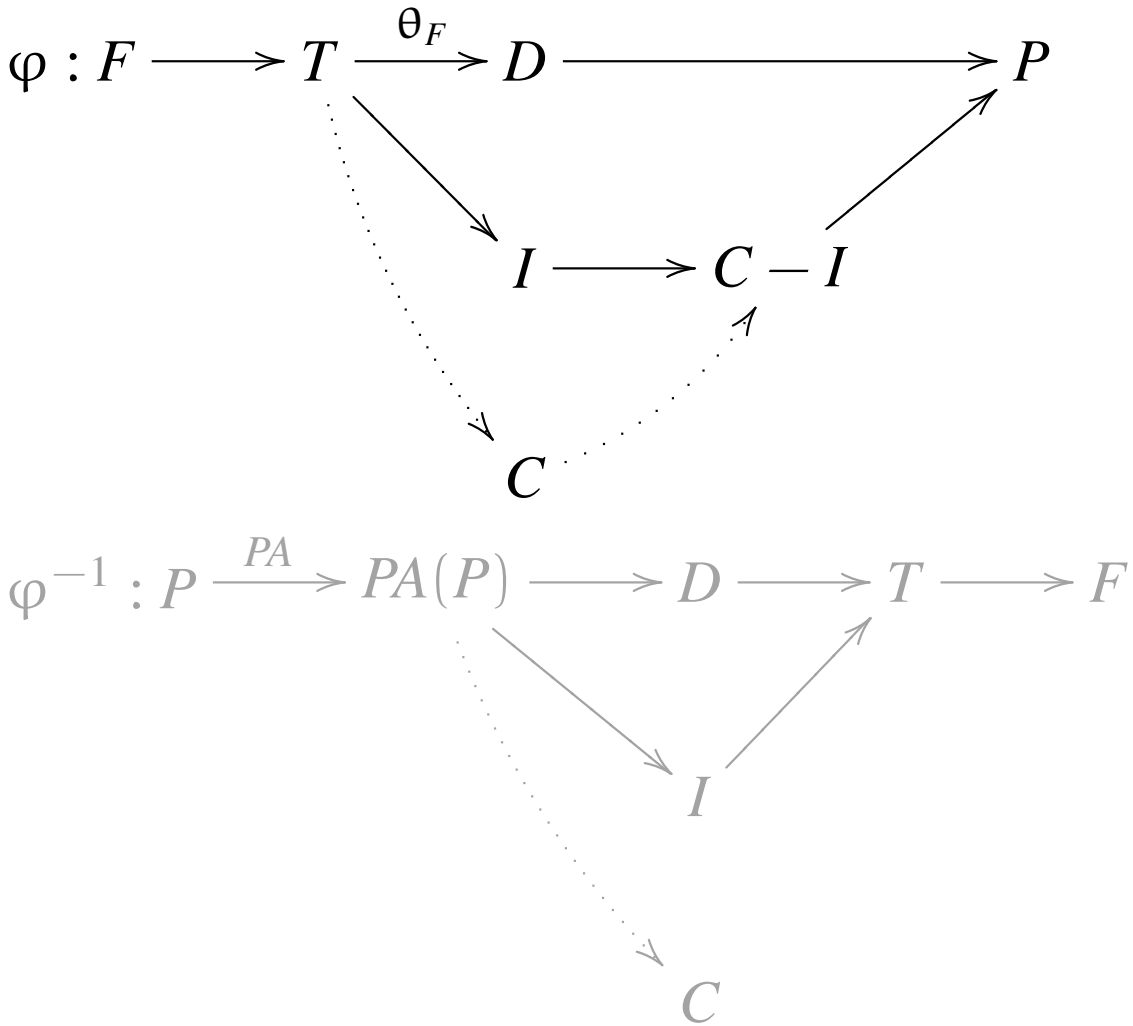
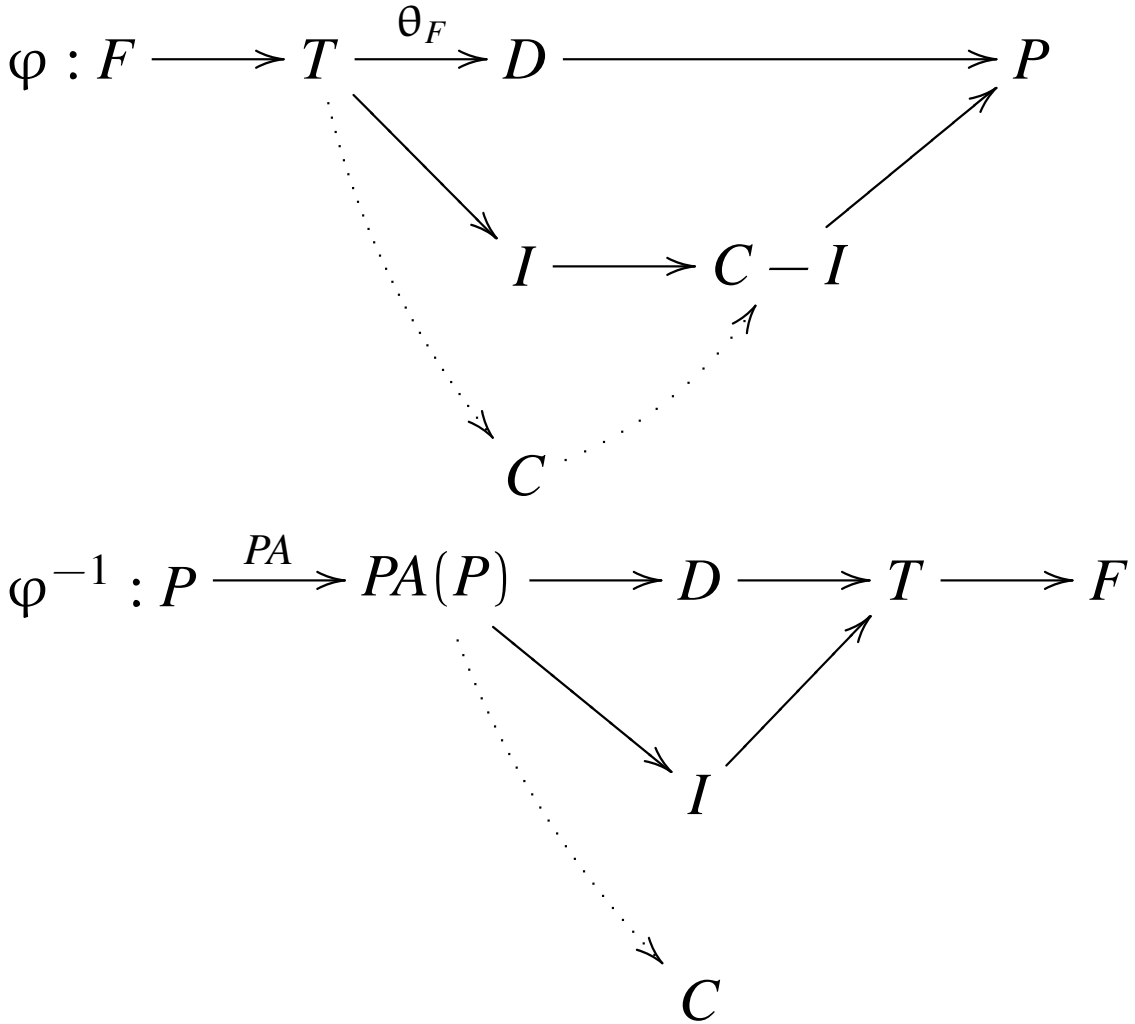
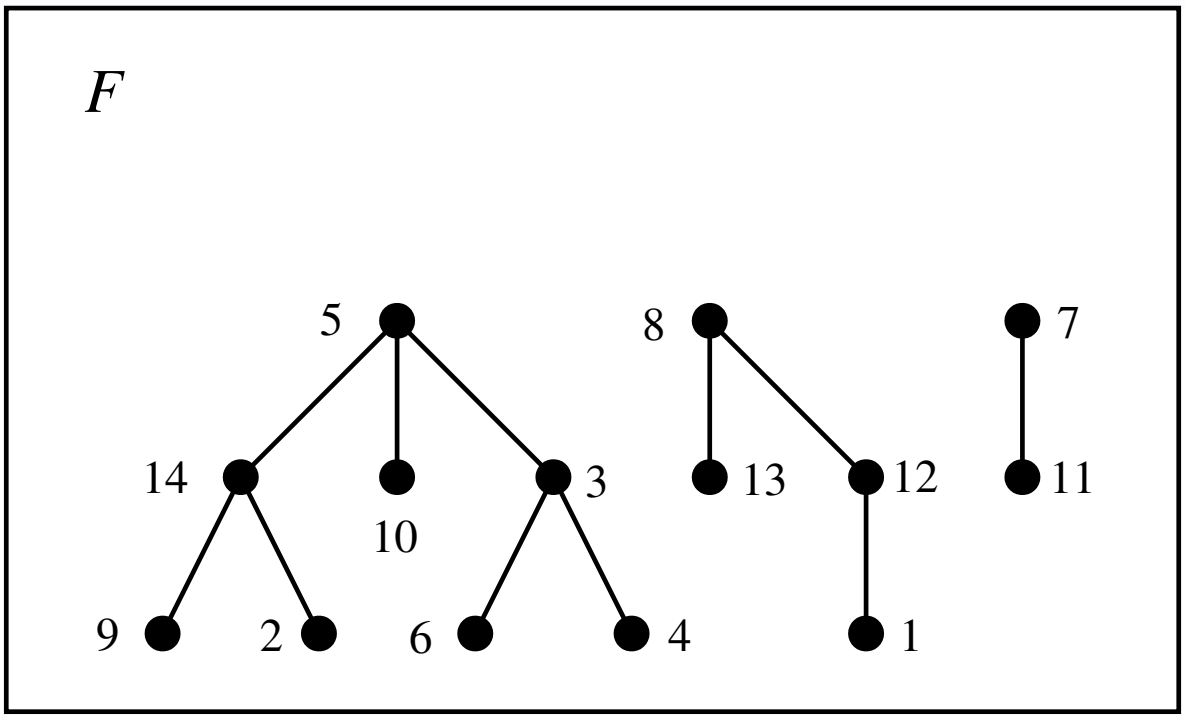
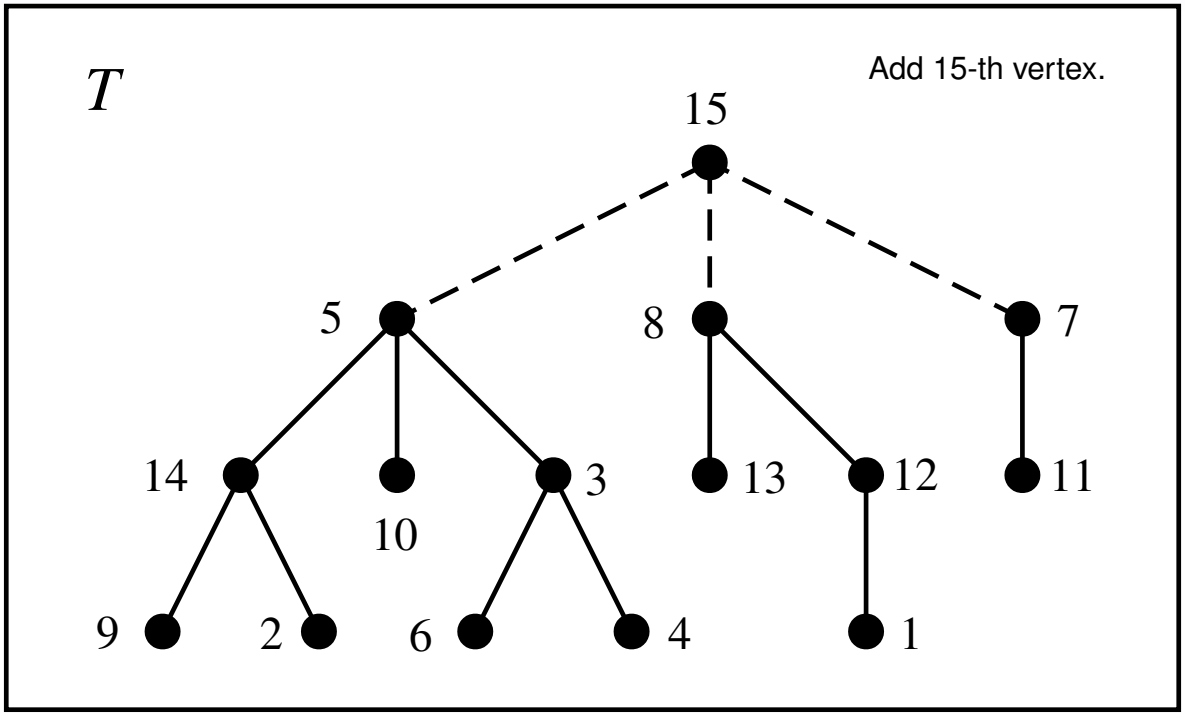


Diagram of φ and φ^{-1}

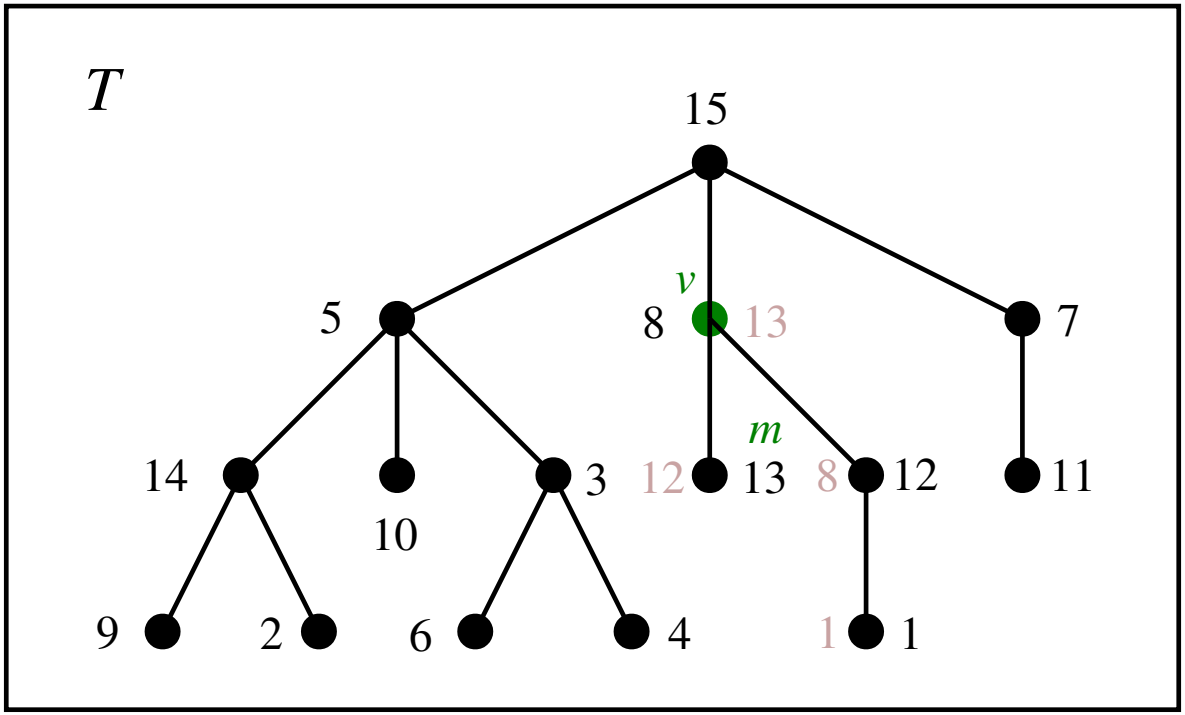




We consider $F \in F_{14}$ for example. Of course, F is drawn in the method we decide.



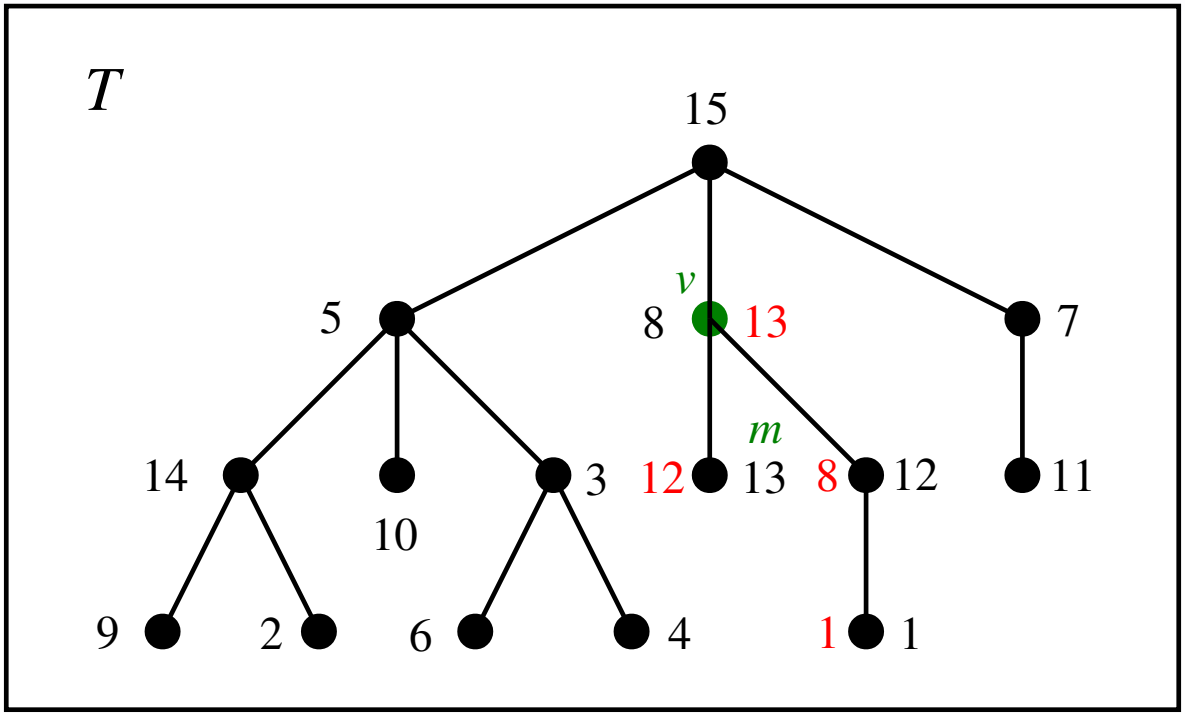
Add the vertex 15 at the top and change the forest F to the tree T.



for all $v \in V$ do

1. find the maximum label m on descendants of v.
2. label m on v.
3. rearrange the other labels in descendants of v by order-preserving.

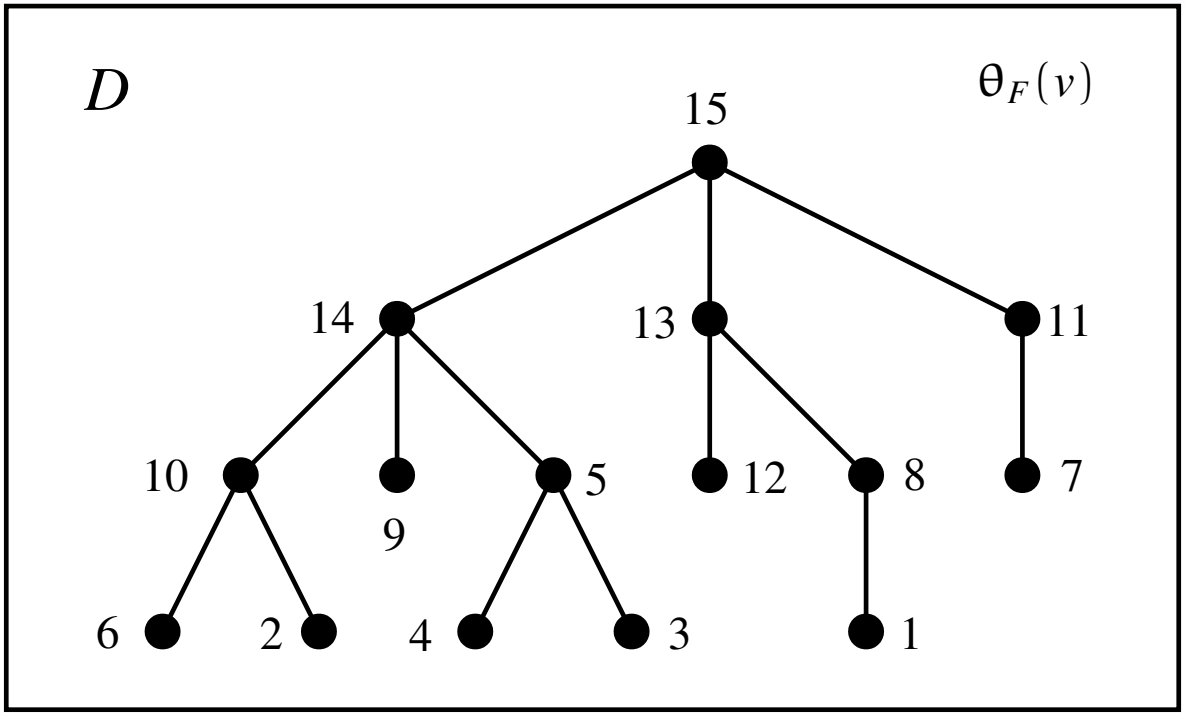
end do



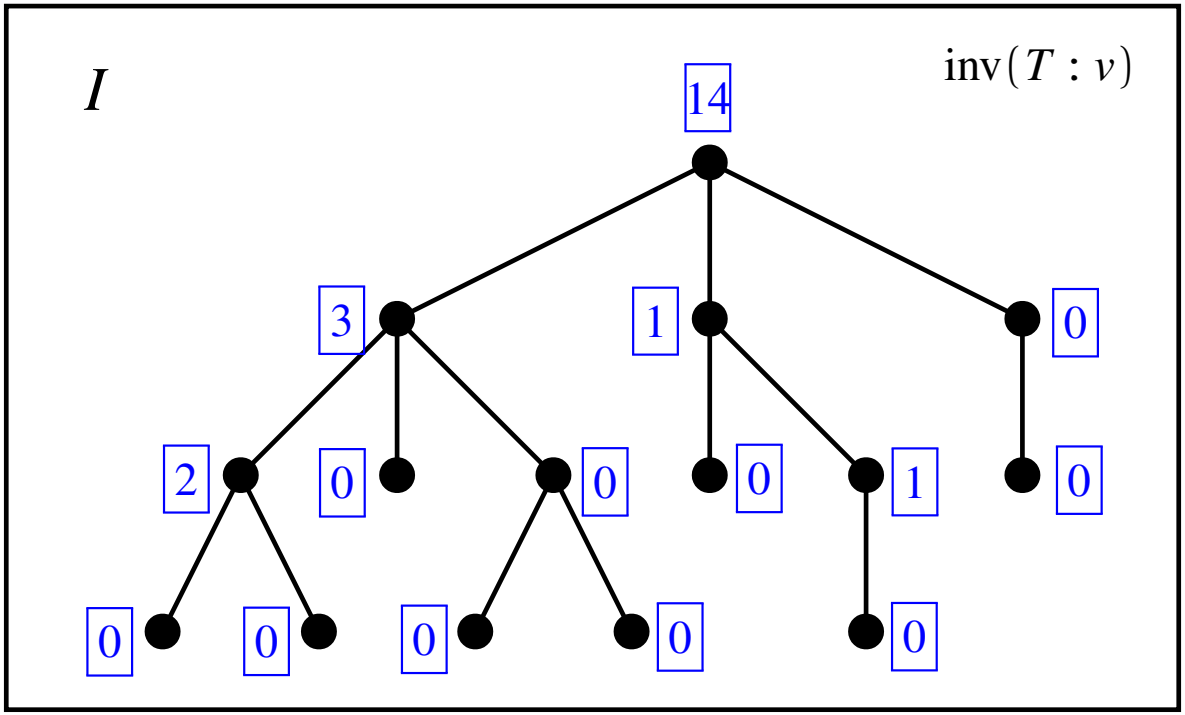
for all $v \in V$ do

1. find the maximum label m on descendants of v.
2. label m on v.
3. rearrange the other labels in descendants of v by order-preserving.

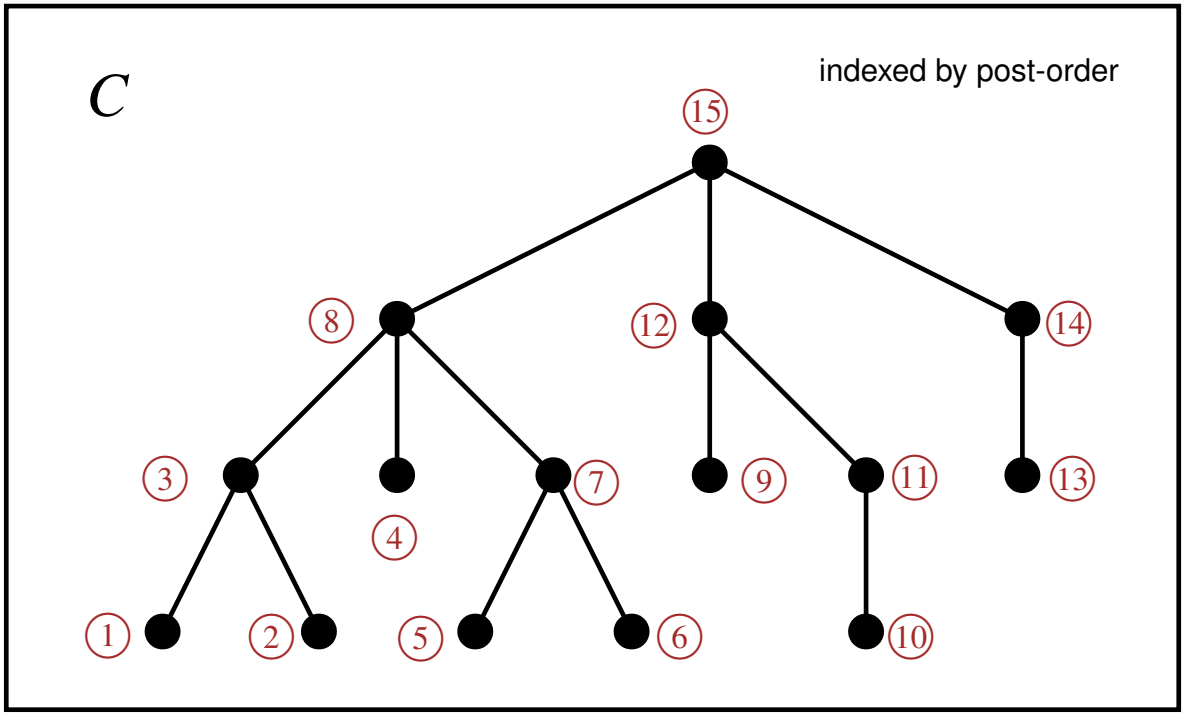
end do



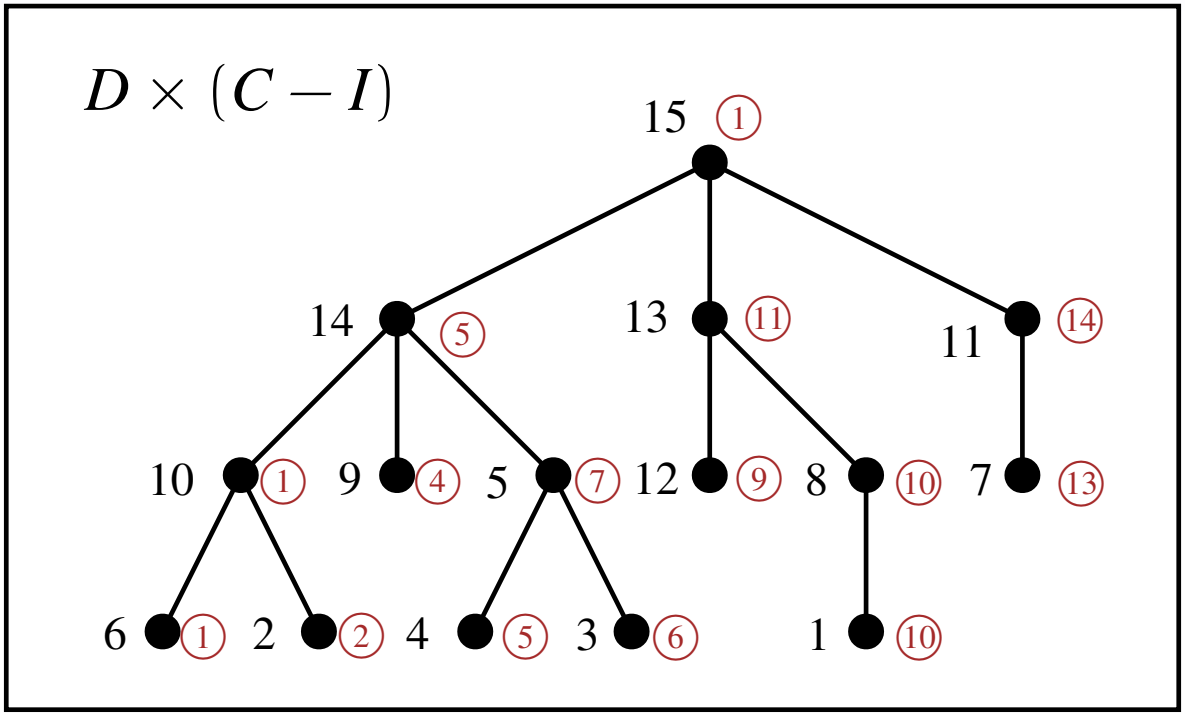
The decreasing tree is made after the process for every vertex. But we cannot remake the original tree T from only tree D. So, we need another tree induced from the unused information of T.



And then, label $\mathrm{inv}(T:v)$ on vertex v. In order to distinguish it from other labels, we use the box (or blue color). And then, the trees D and I can produce the original tree T.



Label the vertices indexed by post-order which is indicated by circle (or brown color). The tree C is determined by only the *underlying graph*, that is, its tree structure. This is the reason why we define the method we draw the tree.



The plain labels are induced by D.
The circled labels are induced by C subtracted by I.

And then, we delete the tree structure and sort by plain #.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
⑩	②	⑥	⑤	⑦	①	⑬	⑩	④	①	⑭	⑨	⑪	⑤	①

Below the plain label 15, there is always circle label ①. So, we can omit it, and then second row (circle label) becomes a *parking function* P of length 14.

$$P = ⑩②⑥⑤⑦①⑬⑩④①⑭⑨⑪⑤$$

Because all labels of C are distinct in worst case which means every labels of I is all zero. Note that every permutation is a parking function.

And then, we delete the tree structure and sort by plain #.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
⑩	②	⑥	⑤	⑦	①	⑬	⑩	④	①	⑭	⑨	⑪	⑤	①

Below the plain label 15, there is always circle label ①. So, we can omit it, and then second row (circle label) becomes a *parking function* P of length 14.

$$P = ⑩②⑥⑤⑦①⑬⑩④①⑭⑨⑪⑤$$

Because all labels of C are distinct in worst case which means every labels of I is all zero. Note that every permutation is a parking function.

And then, we delete the tree structure and sort by plain #.

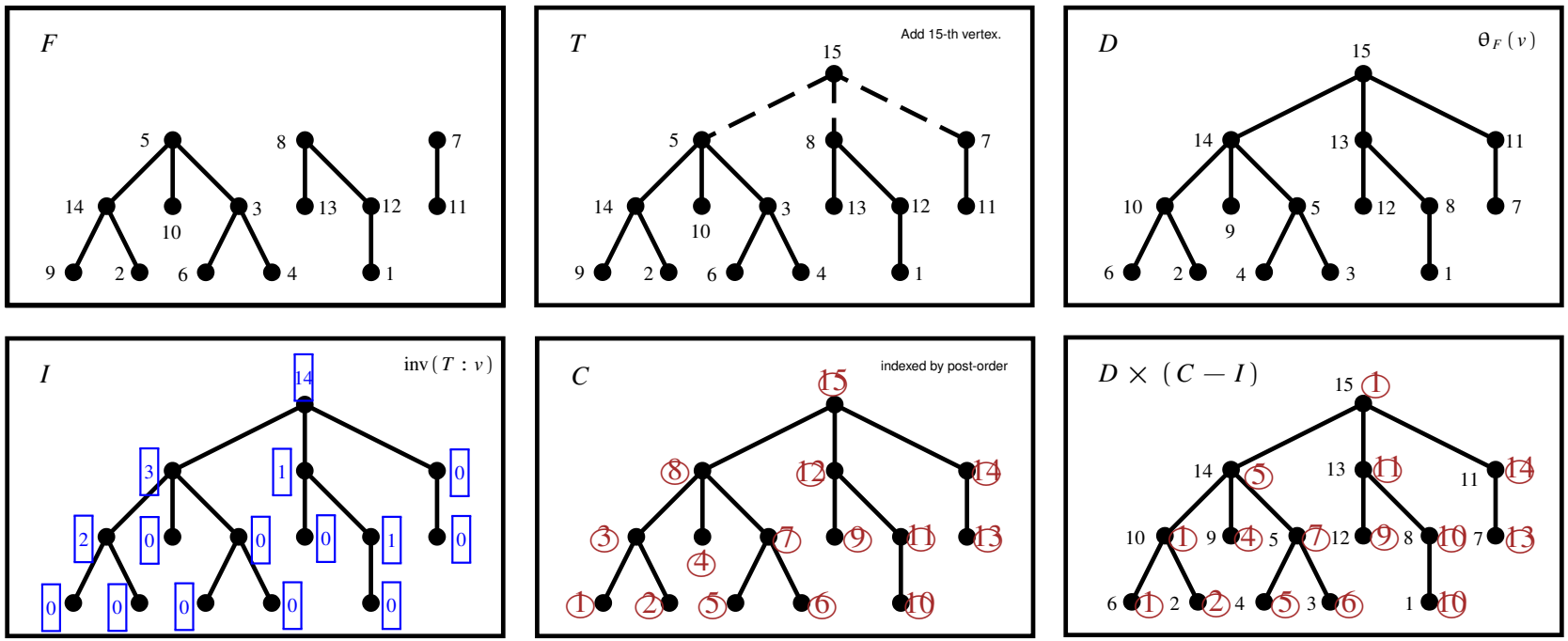
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
⑩	②	⑥	⑤	⑦	①	⑬	⑩	④	①	⑭	⑨	⑪	⑤	①

Below the plain label 15, there is always circle label ①. So, we can omit it, and then second row (circle label) becomes a *parking function* P of length 14.

$$P = ⑩②⑥⑤⑦①⑬⑩④①⑭⑨⑪⑤$$

Because all labels of C are distinct in worst case which means every labels of I is all zero. Note that every permutation is a parking function.

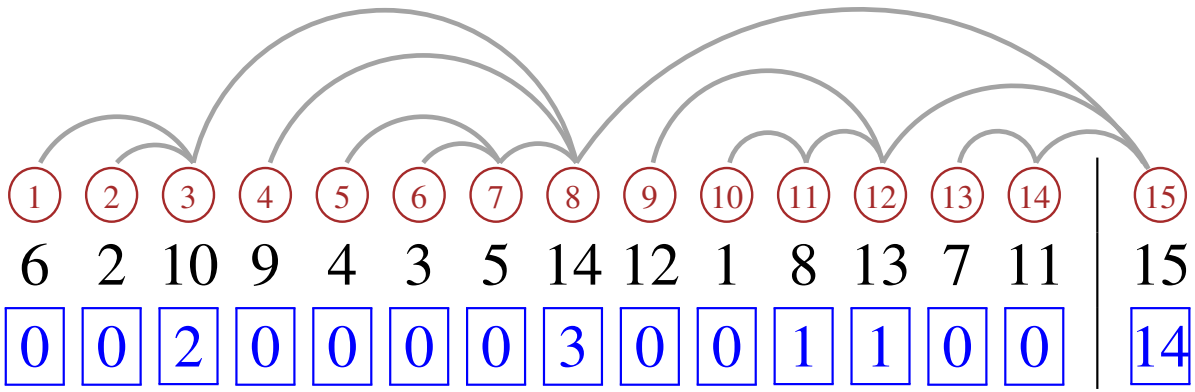
Summary of the map φ



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
P=	10	2	6	5	7	1	13	10	4	1	14	9	11	5	1

Inverse Map of φ

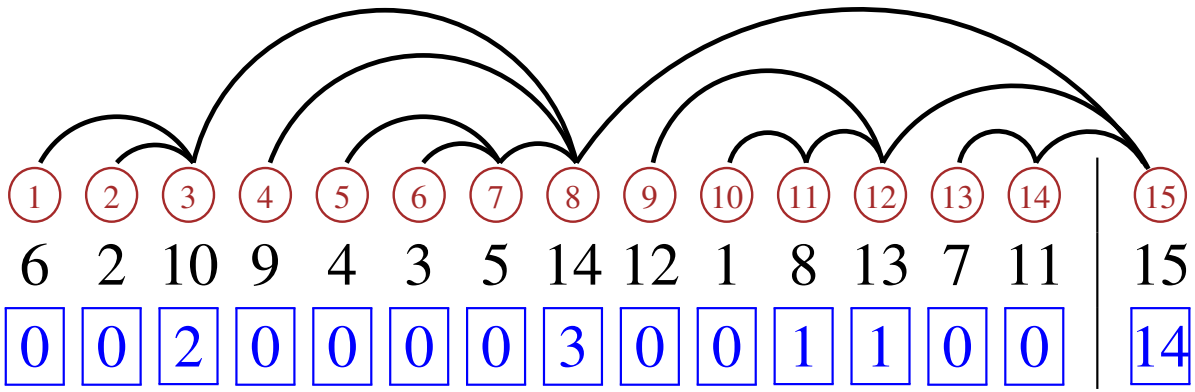
Let $P = (10)(2)(6)(5)(7)(1)(13)(10)(4)(1)(14)(9)(11)(5)$
After adding the (1) at the end, 15 cars is parked as following by the parking algorithm.



We draw a edge between car c and the closest car on its right which is larger than c. If we consider 15 as a root, we can rebuild the tree structure and find trees C, D and I.

Inverse Map of φ

Let $P = (10)(2)(6)(5)(7)(1)(13)(10)(4)(1)(14)(9)(11)(5)$
After adding the (1) at the end, 15 cars is parked as following by the parking algorithm.



We draw a edge between car c and the closest car on its right which is larger than c. If we consider 15 as a root, we can rebuild the tree structure and find trees C, D and I.

Outline

KAIST

Relations of statistics

We analyze the map φ, then we know the followings.

- $\mathrm{inv}(F : v) = \mathrm{jump}(\varphi(F) : \theta_F(v))$
- If v is a root of a tree in F, then $\theta_F(v)$ is a right-to-left maximum in $PA(\varphi(F))$

Corollary

If $P = \varphi(F)$, these facts are induced automatically.

- $\mathrm{inv}(F) = \mathrm{jump}(P)$
- $\mathrm{lead}(F) = \mathrm{lucky}(P)$
- $\mathrm{tree}(F) = \mathrm{critical}(P)$

where $\mathrm{tree}(F)$ is the # of trees in forest F and
$\mathrm{critical}(P)$ is the # of right-to-left maximums in $PA(P)$.

NOTE. The car c is called *critical* if there is no empty parking space on the right of car c after parking car c.

Outline

KAIST

Main Result I

I_n = homogeneous polynomials of degree n

$$I_n(\mathbf{q}) = I_n(q_0, q_1, q_2, \ldots) = \sum_{F \in F_n} q_0^{t_0(F)} q_1^{t_1(F)} q_2^{t_2(F)} \cdots$$

where $t_i(F)$ = the # of vertices v s.t. $\mathrm{inv}(F : v) = i$,
especially, $t_0(F) = \mathrm{lead}(F)$ and $\sum_i t_i(F) = n$.
J_n = homogeneous polynomials of degree n

$$J_n(\mathbf{q}) = J_n(q_0, q_1, q_2, \ldots) = \sum_{P \in PF_n} q_0^{s_0(F)} q_1^{s_1(F)} q_2^{s_2(F)} \cdots$$

where $s_i(F)$ = the # of cars c s.t. $\mathrm{jump}(P : c) = i$,
especially, $s_0(P) = \mathrm{lucky}(P)$ and $\sum_i s_i(F) = n$.

KAIST

Main Result II

Theorem (Main Theorem)

$$I_n(\mathbf{q}) = J_n(\mathbf{q})$$

PROOF. For $P = \varphi(F)$, there is the correspondence θ_F between all vertices v in the forest F and all cars c in the parking function P such that

$$\begin{aligned} \mathrm{inv}(F : v) &= \mathrm{jump}(P : c) \\ &= \mathrm{jump}(\varphi(F) : \theta_F(v)) \end{aligned}$$

Because each labels of tree I means inversions of forest F and jumps of parking function P.

Corollary I

Corollary

$$\sum_{F \in F_n} q^{\mathrm{inv}(F)} u^{\mathrm{lead}(F)} = \sum_{P \in PF_n} q^{\mathrm{jump}(P)} u^{\mathrm{lucky}(P)}$$

PROOF. By theorem, $I_n(u, q, q^2, \ldots) = J_n(u, q, q^2, \ldots)$.

$$\begin{aligned} \mathrm{inv}(F) &= \textstyle\sum_i i \cdot t_i(F) \\ \mathrm{lead}(F) &= t_0(F) \\ \mathrm{jump}(P) &= \textstyle\sum_i i \cdot s_i(P) = \binom{n+1}{2} - |P| \\ \mathrm{lucky}(P) &= s_0(P) \end{aligned}$$

Corollary II

Recall that

$$\sum_{F \in F_n} u^{\mathrm{lead}(F)} c^{\mathrm{tree}(F)} = P_n(1, u, cu).$$

By the map θ_F, the root of each tree in F corresponds to right-to-left maximums in $PA(P)$. So, we have

$$\sum_{F \in F_n} u^{\mathrm{lead}(F)} c^{\mathrm{tree}(F)} = \sum_{P \in PF_n} u^{\mathrm{lucky}(P)} c^{\mathrm{critical}(P)}.$$

Corollary

$$\sum_{P \in PF_n} c^{\mathrm{critical}(P)} u^{\mathrm{lucky}(P)} = P_n(1, u, cu).$$

KAIST

Conclusion

Theorem (Expansion of main theorem)

$$I_n(\mathbf{q} : c) = J_n(\mathbf{q} : c)$$

where

$$I_n(\mathbf{q} : c) = \sum_{F \in F_n} q_0^{t_0(F)} q_1^{t_1(F)} q_2^{t_2(F)} \cdots c^{\mathrm{tree}(F)}$$

$$J_n(\mathbf{q} : c) = \sum_{P \in PF_n} q_0^{s_0(F)} q_1^{s_1(F)} q_2^{s_2(F)} \cdots c^{\mathrm{critical}(P)}$$

REMARK. Also, we get

$$\sum_{F \in F_n} q^{\mathrm{inv}(F)} u^{\mathrm{lead}(F)} c^{\mathrm{tree}(F)} = \sum_{P \in PF_n} q^{\mathrm{jump}(P)} u^{\mathrm{lucky}(P)} c^{\mathrm{critical}(P)}$$

KAIST

Outline

KAIST

Summary

- Forests and Parking Functions have not only the same cardinality, but also many equinumerous statistics.
- There exists the map φ which corresponds simultaneously between statistics inv, lead and tree in forests and statistics jump, lucky and $\mathrm{critical}$ in parking functions.
- There exists not only the correspondence φ of combinatorial objects, but also the correspondence θ between vertices in forests and cars in parking functions in detail.
- Now to conclude, forests and parking functions are very similar and the same structures.

KAIST

Summary

- Forests and Parking Functions have not only the same cardinality, but also many equinumerous statistics.
- There exists the map φ which corresponds simultaneously between statistics inv, lead and tree in forests and statistics jump, lucky and $\mathrm{critical}$ in parking functions.
- There exists not only the correspondence φ of combinatorial objects, but also the correspondence θ between vertices in forests and cars in parking functions in detail.
- Now to conclude, forests and parking functions are very similar and the same structures.

KAIST

Summary

- Forests and Parking Functions have not only the same cardinality, but also many equinumerous statistics.
- There exists the map φ which corresponds simultaneously between statistics inv, lead and tree in forests and statistics jump, lucky and $\mathrm{critical}$ in parking functions.
- There exists not only the correspondence φ of combinatorial objects, but also the correspondence θ between vertices in forests and cars in parking functions in detail.
- Now to conclude, forests and parking functions are very similar and the same structures.

KAIST

Summary

- Forests and Parking Functions have not only the same cardinality, but also many equinumerous statistics.
- There exists the map φ which corresponds simultaneously between statistics inv, lead and tree in forests and statistics jump, lucky and $\mathrm{critical}$ in parking functions.
- There exists not only the correspondence φ of combinatorial objects, but also the correspondence θ between vertices in forests and cars in parking functions in detail.
- Now to conclude, forests and parking functions are very similar and the same structures.

KAIST

Further Study

- Mark D. Haiman found that there exists the duality of inversions in a tree structure by the dimension of group representation.
- No one know yet what the duality of inversions means in a tree structure.
- Can you maybe find a duality of jumps in parking functions?

KAIST

Further Study

- Mark D. Haiman found that there exists the duality of inversions in a tree structure by the dimension of group representation.
- No one know yet what the duality of inversions means in a tree structure.
- Can you maybe find a duality of jumps in parking functions?

Further Study

- Mark D. Haiman found that there exists the duality of inversions in a tree structure by the dimension of group representation.
- No one know yet what the duality of inversions means in a tree structure.
- Can you maybe find a duality of jumps in parking functions?

KAIST

References I

A. Cayley
A theorem on trees
Quart. J. Math. **23**, 376–378, 1889.

H. Prüfer,
Neuer Beweis eines Satzes über Permutationen,
Archiv der Math. (3) **27**, 142–144, 1918.

S. Seo,
A combinatorial proof of Postnikov's identity and a generalized enumeration of labeled trees,
Electron. J. Combin. **11** no. 2, #N3, 2004.

M. Cho, D. Kim, S. Seo and H. Shin,
Colored Prüfer Codes for k-Edge Colored Trees,
Electron. J. Combin. **11** no. 1, #N10, 2004.

KAIST

References II

Ira M. Gessel and S. Seo,
A refinement of Cayley's formula for trees,
arXiv: math.CO/0507497, 2005.

S. Seo and H. Shin,
A Generalized Enumeration of Labeled Trees and Reverse Prüfer Algorithm,
arXiv:math.CO/0601009, 2006.

KAIST

Thank you for listening!

YEUNGNAM UNIVERSITY, KOREA

Email : H.Shin@kaist.ac.kr
Homepage : http://hshin.info