I618 Advanced Computer Science II (Part II)

12/21 11:00-12:30
1/ 7 15:10-16:40
1/ 9 9:20-10:50
○1/11 11:00-12:30
1/16 9:20-10:50

Ryuhei Uehara
uehara@jaist.ac.jp
http://www.jaist.ac.jp/~uehara

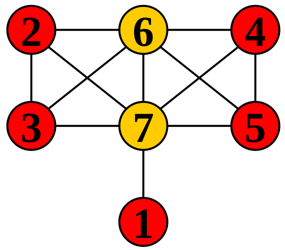
I will give you some report problems on January.

Algorithms on Interval/Chordal Graphs

- Basic problems
  - graph isomorphism;
    - graph isomorphism is *GI-complete* for chordal graphs [Done!]
    - graph isomorphism is *linear time solvable* for interval graphs [Postponed after recognition]
  - graph recognition;
    - chordal graphs can be recognized in linear time
      - LexBFS & MCS
    - interval graphs can be recognized in linear time
      - canonical tree representation
      - multi-sweep LexBFSs
      - modular decomposition

Recognition of a Chordal Graph

- LexBFS and MCS are a kind of "search" algorithms.
  - Both algorithms find *reverse* of a PEO as follows;
  1. put any vertex as v_n;
  2. for each $i=n-1, n-2, \ldots, 1$
     1. find the next vertex and put it as v_i



[Point] How can we find the next vertex?

[MCS] the next vertex v_i has the most numbered neighbors, which is determined by

$$v_i := \max |N(v_i) \cap \{v_{i+1}, v_{i+2}, \ldots, v_n\}|,$$

which is the reason why we call it

"maximum cardinality" search.

(Ties are broken in any way.)

Recognition of a Chordal Graph

- Lexicographically Breadth First Search;

[Definition 8] *Lexicographical ordering* of two strings $X=x_1x_2\dots x_n$ and $Y=y_1y_2\dots y_m$ are defined as follows (usual ordering in dictionary):

$X<Y$ if and only if

1. $\exists\, i\ x_i<y_i$, and $x_j=y_j$ for all $j<i$, or
2. if $x_i=y_i$ for all i in $[1..\min\{n,m\}]$, $X<Y$ if $n<m$ or $Y<X$ if $n>m$

(Otherwise, we have $X=Y$.)

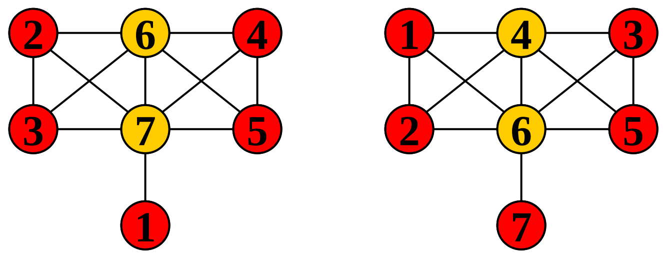
E.g., ε < 0101 < 01010 < 0101<u>1</u> < 01<u>1</u>00 < <u>1</u>

- We can apply the lex. ordering over ordered sets;
  - (1,2,3)<(1,2,3,<u>4</u>)<(1,2,<u>5</u>)<(1,<u>3</u>,4)
  - (3,2,1)<(<u>4</u>,3,1)<(4,3,<u>2</u>,1)<(<u>5</u>,2,1)

Recognition of a Chordal Graph

- LexBFS and MCS are a kind of "search" algorithms.
  - Both algorithms find *reverse* of a PEO as follows;

1. put any vertex as v_n;
2. for each i=n-1, n-2, …, 1
   1. find the next vertex and put it as v_i



[Point] How can we find the next vertex?

[LexBFS] the next vertex v_i is determined by the reverse of the lexicographically ordering of the neighbor sets

$$N(v) \cap \{v_n, v_{n-1}, \ldots, v_{i+1}\},$$

where neighbor sets are ordered in reverse of PEO.

(Ties are broken in any way.)

This is a natural ordering if we compute the *reverse* of a PEO, which appears some papers…

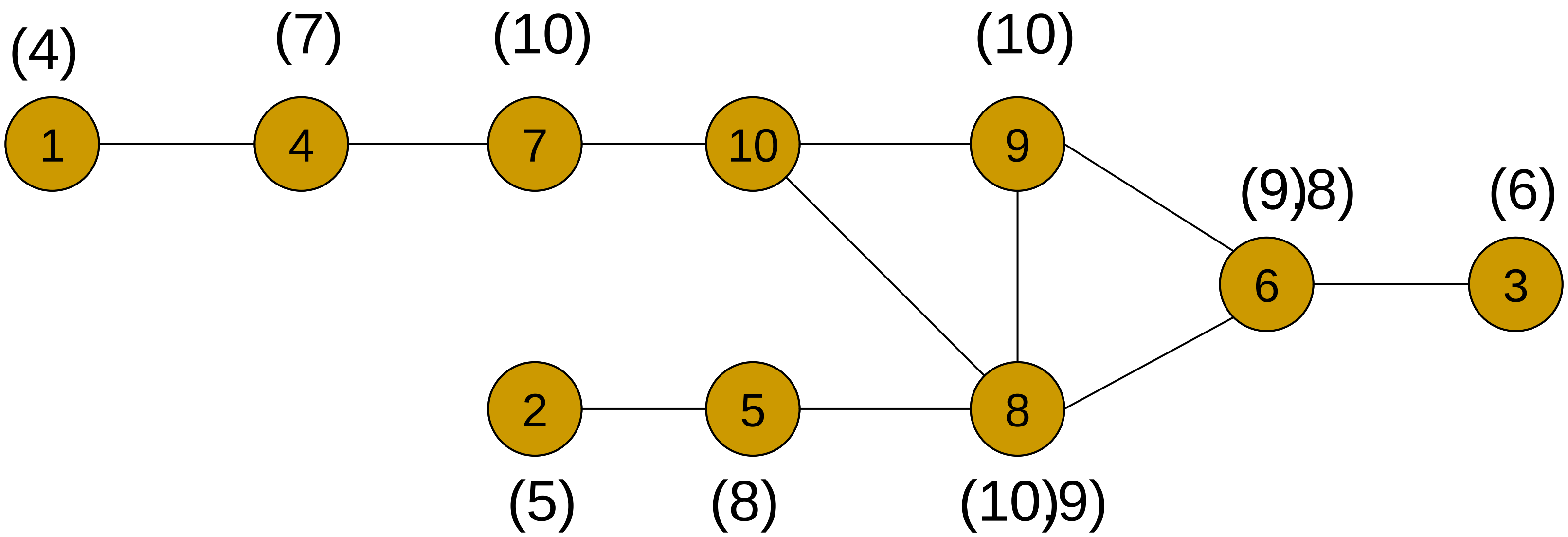
Recognition of a Chordal Graph

[LexBFS] the next vertex v_i is determined by the reverse of the lexicographically ordering of the neighbor sets

$$N(v) \cap \{v_n, v_{n-1}, \ldots, v_{i+1}\},$$

where neighbor sets are ordered in reverse of PEO.

(Ties are broken in any way.)



Recognition of a Chordal Graph

- LexBFS and MCS are a kind of "search" algorithms.

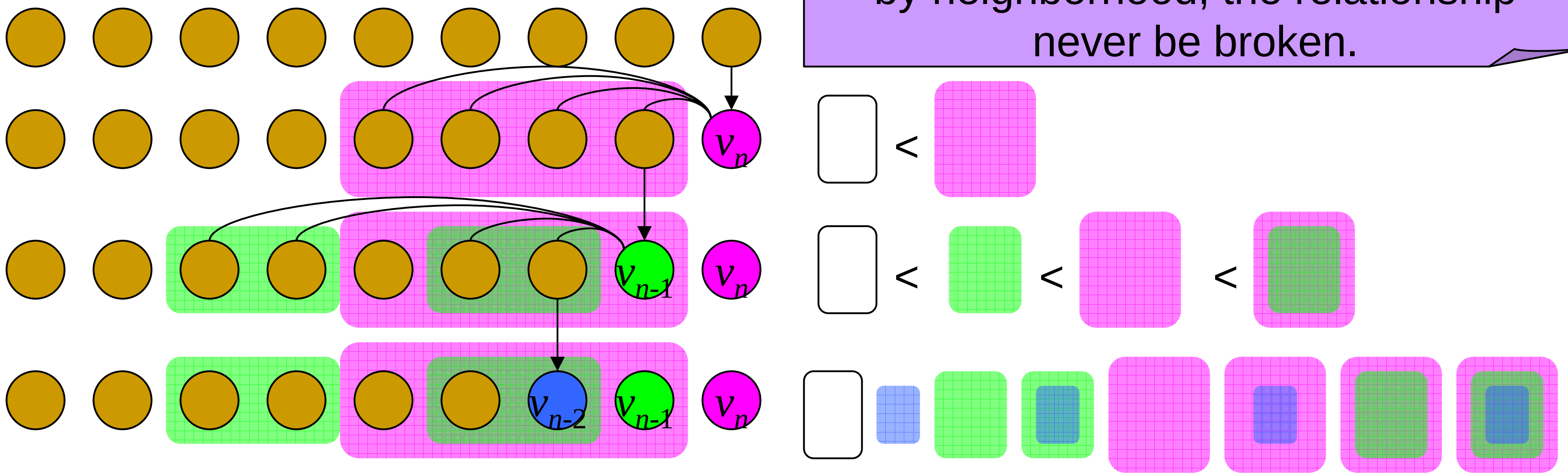
[LexBFS] the next vertex v_i is determined by the reverse of the lexicographically ordering of the neighbor sets

$$N(v) \cap \{v_n, v_{n-1}, \ldots, v_{i+1}\},$$

where neighbor sets are ordered in reverse of PEO.

[Natural explanation]

Once we divide a set into two subsets by neighborhood, the relationship never be broken.



Implementation is easy by a priority queue.

Recognition of a Chordal Graph

- LexBFS and MCS are a kind of "search" algorithms.

[Theorem 12] Let $G=(V,E)$ be any graph. Then we can determine if G is chordal or not in $O(|V|+|E|)$ time and space.

To prove Theorem 12, we need two lemmas;

[Lemma 2] Let G be any chordal graph. Then

1. output of LexBFS is a PEO of G, and
2. output of MCS is a PEO of G.

[Lemma 3] Let $v_1, v_2, \ldots, v_n$ be any ordering over V. Then we can determine if it is a PEO or not in linear time.

(Proof of Lemma 3) Omitted; check the papers!

Recognition of a Chordal Graph

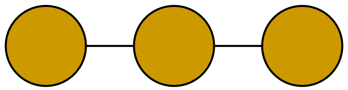
- LexBFS and MCS are a kind of "search" algorithms.
  We only show a part of proofs briefly…

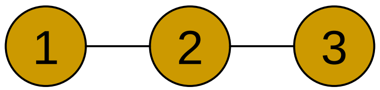
[Lemma 2] Let G be any chordal graph. Then

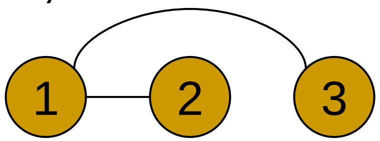
1. output of LexBFS is a PEO of G.

[Note before proof] Not necessarily all vertex orderings of a chordal graph are PEO.

[Example 2]

For a chordal graph  ,



is a PEO, but 

is not a PEO.

Recognition of a Chordal Graph

- LexBFS and MCS are a kind of "search" algorithms.
  We only show a part of proofs briefly…

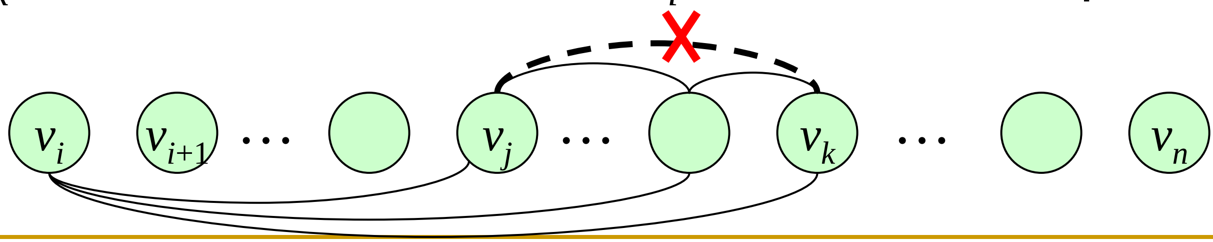
[Lemma 2] Let G be any chordal graph. Then

1. output of LexBFS is a PEO of G.

[Proof (Sketch)] To derive contradictions, assume that LexBFS outputs a vertex ordering $v_1, v_2, \ldots, v_n$ which is *not* a PEO for a *chordal* graph G.

Then there is a *non*-simplicial vertex v_i in $G[\{v_i,v_{i+1},\ldots,v_n\}]$.

Thus $N(v_i) \cap \{v_{i+1},\ldots,v_n\}$ contains two non-adjacent vertices v_j and v_k. We take the *maximum* v_i and *maximum* pair in $N(v_i)$.



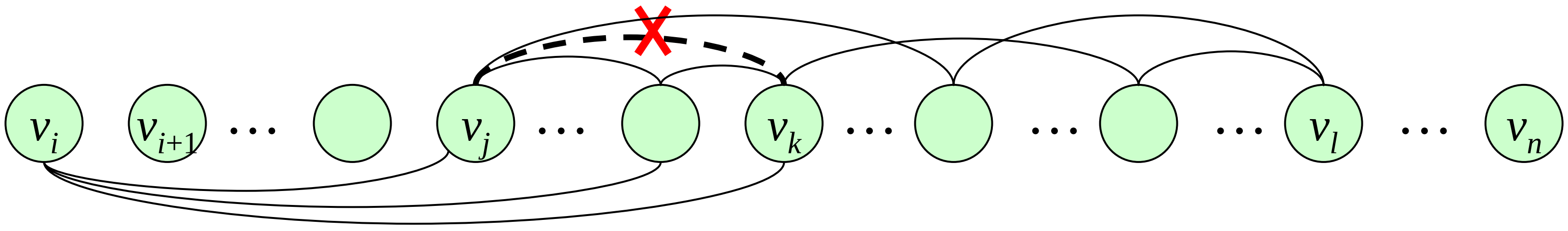
Recognition of a Chordal Graph

- LexBFS and MCS are a kind of "search" algorithms.

[Lemma 2] For any chordal graph G, an output of LexBFS is a PEO of G.

[Proof (Sketch)] In LexBFS, except v_n, each v is added into the ordering by a "*precedessor*" u; v is added because v is in $N(u)$.

Thus, from v_j and v_k, we repeat to find precedessors until we meet the (first) common vertex v_l.



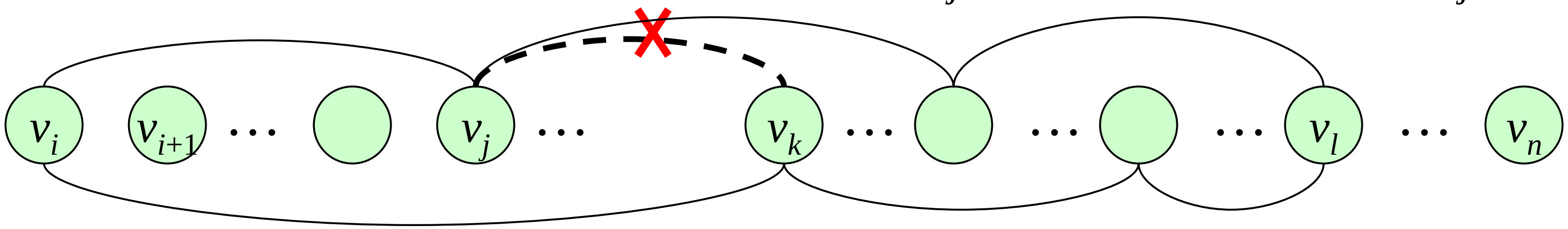
Then, we have a cycle $(v_i,v_j,\ldots,v_l,\ldots,v_k,v_i)$ of length at least 4 with $\{v_j,v_k\} \notin E$.

Recognition of a Chordal Graph

- LexBFS and MCS are a kind of "search" algorithms.

[Lemma 2] For any chordal graph G, an output of LexBFS is a PEO of G.

[Proof (Sketch)] We have a cycle $(v_i, v_j, \ldots, v_l, \ldots, v_k, v_i)$ with $\{v_j, v_k\} \notin E$.



Since G is chordal, v_i has to have a neighbor $v_{l'}$ between v_j and v_k. Then, *with careful analysis of LexBFS and maximality of taking the vertices*, we have to have $\{v_i, v_l\} \in E$, and we conclude $v_j < v_i$ or $v_k < v_i$, which is a contradiction. □

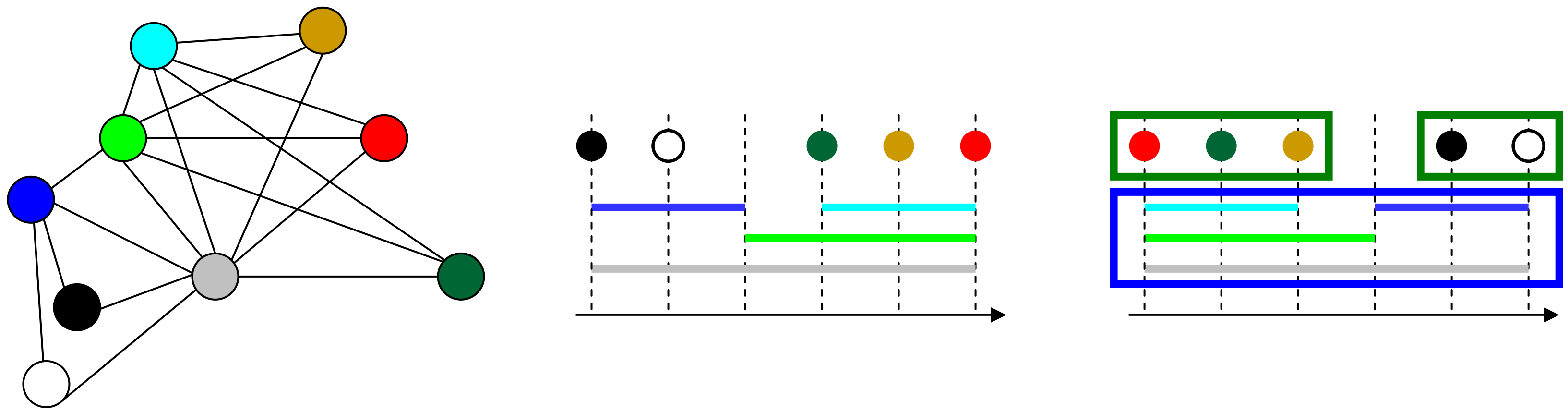
Algorithms on Interval Graphs

- Graph recognitions of interval graphs
  - based on canonical tree representation
    - which construct the *tree representation*
    - using the tree, we can solve *graph isomorphism* in linear time.
  - based on multi-sweep LexBFSs
    - which try to *embed* given graph into a *specific interval representation*
    - tie breaking rule of LexBFS is very important
  - based on modular decomposition
    - which decompose given graph into disjoint components which are called *modular*

Algorithms on Interval Graphs

- Canonical Tree representation of an interval graph
- Basic idea comes from simple observation…

[Observation 2] For an interval graph G, there are several distinct compact interval representations.



intervals can be ordered in arbitrary ordering

intervals can be ordered in “forward” or “backward.”

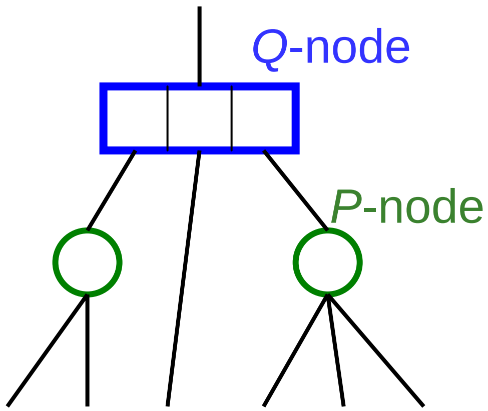
Algorithms on Interval Graphs

- Canonical Tree representation of an interval graph

[Definition 9] A *PQ*-tree consists of two kinds of nodes, called *P*-nodes and *Q*-nodes.

- The children of a *P*-node are ordered in arbitrary way.
- The children of a *Q*-node are ordered in forward or backward.

[Theorem 13] For any interval graph G, its all affirmative compact interval representations can be represented by one *PQ*-tree, where each leaf corresponds to a maximal cliques in the interval graph.

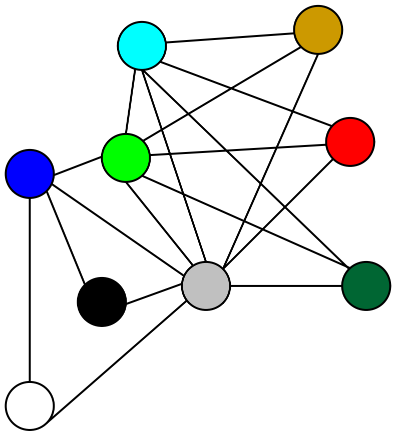


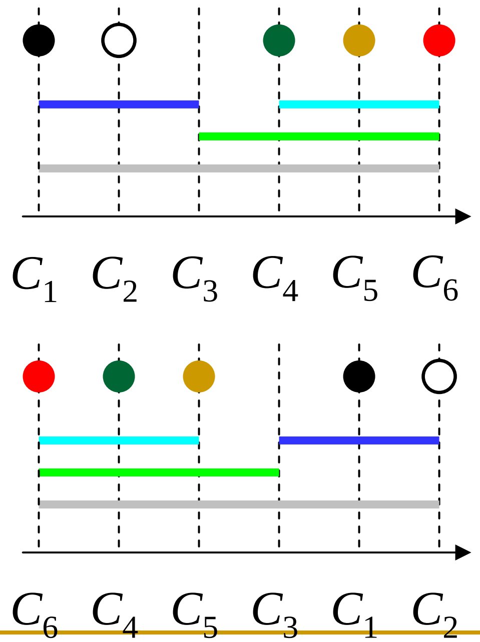
([Theorem 3] Each integer point corresponds to a maximal clique on a compact interval representation…)

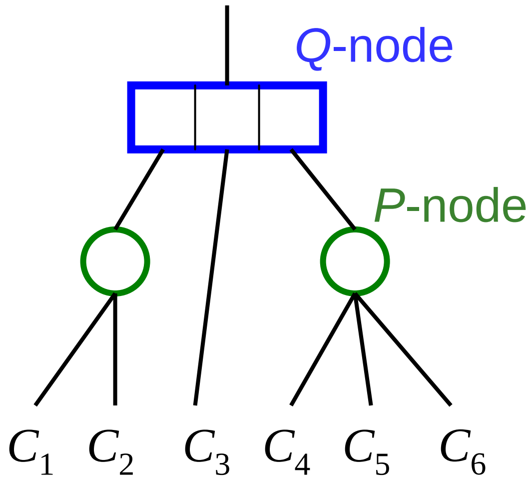
Algorithms on Interval Graphs

- Canonical Tree representation of an interval graph

[Theorem 13] For any interval graph G, its all affirmative compact interval representations can be represented by one *PQ*-tree, where each leaf corresponds to a maximal cliques in the interval graph.





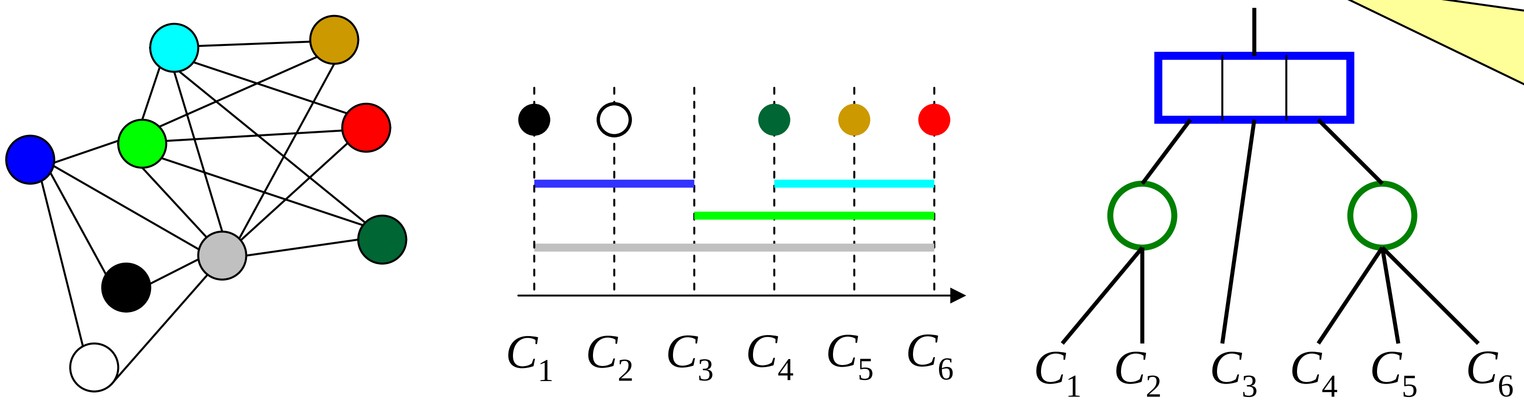


Each vertex has to appear in consecutive cliques.

Algorithms on Interval Graphs

- Canonical Tree representation of an interval graph

[Theorem 14] A graph G is an interval graph if and only if it has a unique *PQ*-tree for its maximal cliques.



Each vertex has to appear in consecutive cliques.

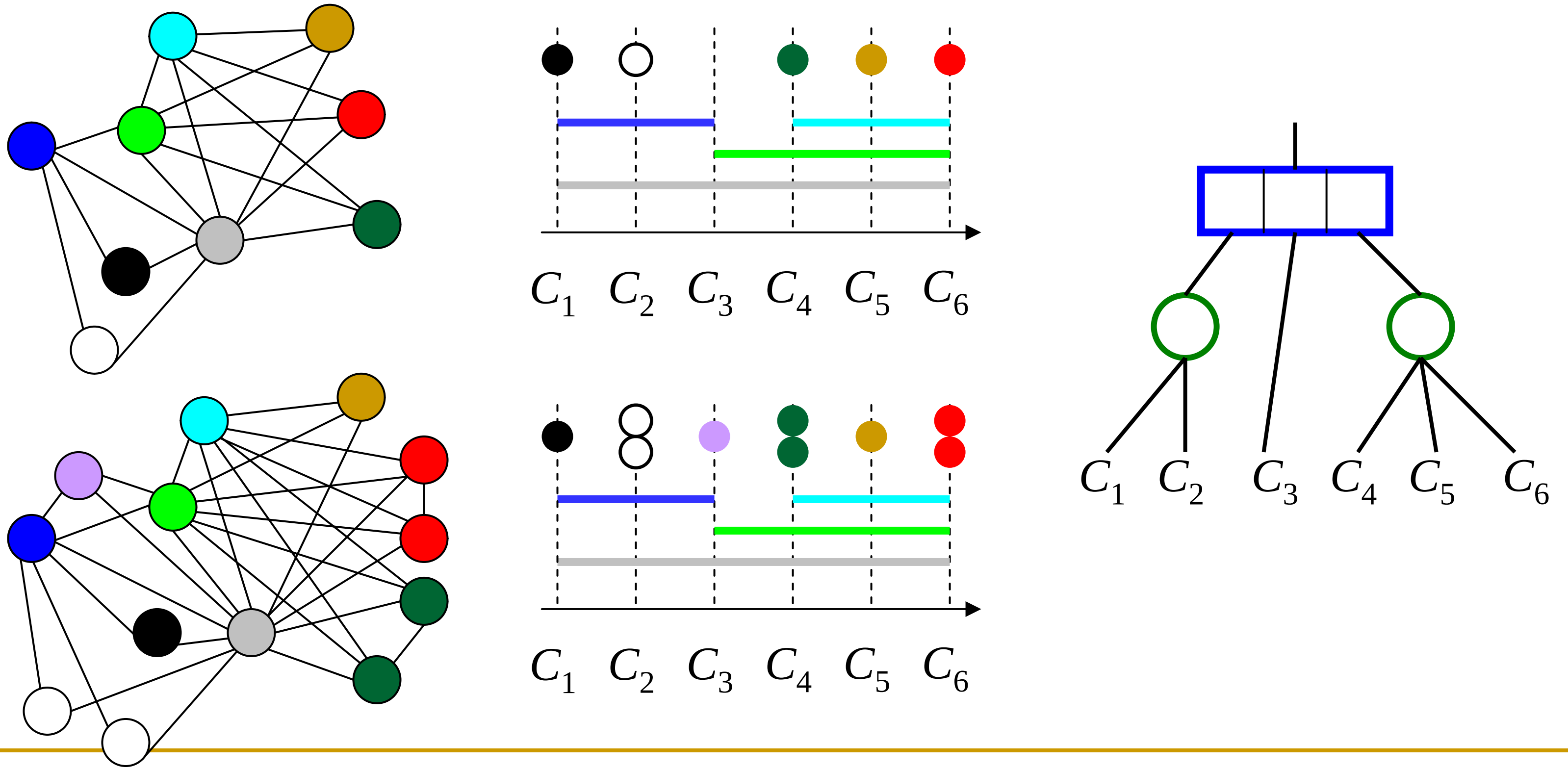
[Theorem 15] [Booth, Lueker 1976] For an interval graph G, its *PQ*-tree can be constructed in linear time.

[Proof (Sketch)] They give incremental algorithm, which has many case analysis with around 20 templates.

Algorithms on Interval Graphs

- Canonical Tree representation of an interval graph

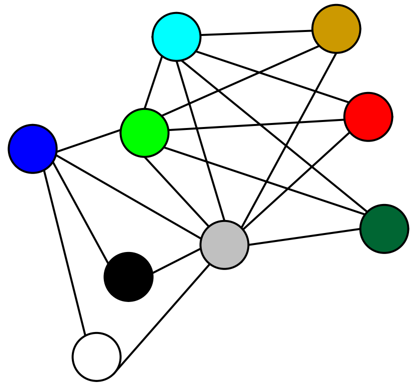
[Note] Any interval graph G has a unique PQ-tree, but a *PQ*-tree can represent *non-isomorphic* interval graphs.

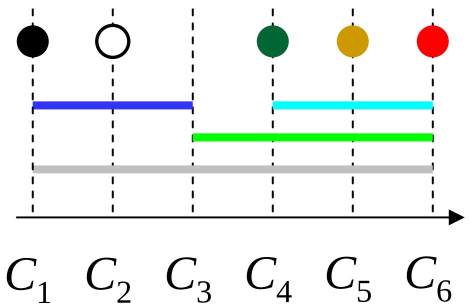


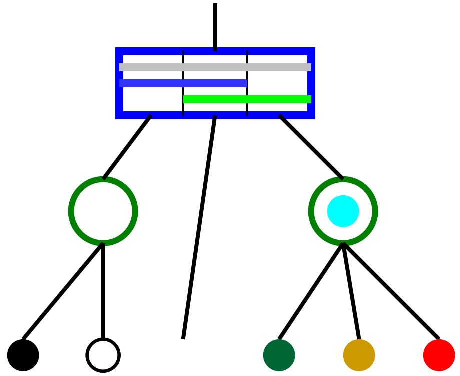
Algorithms on Interval Graphs

- Canonical Tree representation of an interval graph

[Theorem 16] [Lueker, Booth 1979] (1) Any interval graph G has a unique *labeled* PQ-tree, and vice versa.







[Theorem 16] [Lueker, Booth 1979] (2) For any interval graph, its *labeled* *PQ*-tree can be constructed in linear time.

[Corollary 3] The GI problem for interval graphs can be solved in linear time.