Graphs with Isomorphic DFS Spanning Trees

Nika Salia

Academic Advisor: Professor Ervin Gyori

Master Thesis

Department of Mathematics and its Applications

Central European University

Budapest, Hungary

May, 2015.

Acknowledgments

I doubt words can suffice to express my gratitude to my supervisor Professor Ervin Gyõri for everything he has done, among those agreeing to be my supervisor, giving interesting lectures that deepened my knowledge in graph theory and endless support that he gave during these two years.

I am also grateful to Karoly Boroczky and Pál Hegedűs for their constant help and support.

I am thankful to my family (mother and brother) for being with me whenever I needed them regardless of the distance.

I am also thankful to my friends for everything that they have done for me.

Contents

Chapter 1

Introduction

A search in a graph is a methodical exploration of all vertices and edges of the graph. There are many different ways for tracing a graph, which run in linear time. Two most important methods are *Breadth first search* and *Depth first search.* Breadth first search gives an efficient way to compute distances between the vertices. Depth first search is useful for checking many basic connectivity properties, for checking planarity, and also for data flow analysis for compilers. In this work we are going to work with Depth first search algorithm.

The Depth First Search technique is a method of traversing a finite graph. Since the publication of the paper of Hopcroft and Tarjan [8, 9], it is widely recognized as a powerful technique. However, the algorithm was already known in 19th century as a technique for threading mazes. For example, see Lucas' [10] report of Tremaux's work. Another algorithm, which was suggested later by Tarry [11], is just as good for treading mazes and in fact DFS is just a special case of it. There is the additional structure of DFS, which makes it so useful, which will be discussed in following chapter.

Now, we will present Depth First Search algorithm idea from the book [12,

p. 53]. Assume we are given finite undirected graph G. Starting in one of the vertices, we want to "walk" along the edges, from vertex to vertex, visit all the vertices and then halt. We seek an algorithm that will guarantee that we will scan whole graph, and recognize when we are done, without wondering too long in "maze". We allow no pre-planning as by studying the road map before we start our excursion. We must make our decisions, one at a time, since we discover the structure of graph as we scan it. Clearly, we need to leave some markers when we go along, to recognize the fact that we have returned to the place visited before. Let us mark "passages", namely the connections of the edges to vertices. It suffices to use two types of markers: F for the first passage to enter the vertex, and E for any other passage when used to leave the vertex. No marker is ever erased or changed. As we shall prove later the following algorithm will terminate in the original starting vertex, after traversing each edge once in each direction.

Definition 1.0.1. A **subgraph** of a graph G is a graph H such that $V(H) \subset V(G)$ and $E(H) \subset E(G)$.

Definition 1.0.2. A subgraph H of a graph G is a **spanning subgraph** if $V(H) = V(G)$.

Definition 1.0.3. A **spanning tree** of a graph G is a spanning subgraph of G that is a tree.

In 1970, Klaus Wagner ([6] p.50) posed a problem of characterizing connected graphs in which any two spanning trees are isomorphic. In 1971, Bohdan Zelinka [7] published a solution obtained by considering invariants of a tree. In 1973, two different solutions appeared by Fisher and Friess ([2], [3]). Bert L.Harnell ([4], [5]) solved this problem and also gave solution to the problem for graphs with two spanning trees up to isomorphism. Since then many different mathematicians proposed different proofs and variations of this problem.

In this work we are going to discuss various problems, about connected graphs in which any two DFS spanning trees are isomorphic (definition is proposed later in this work). We will see that, this question has several different, interesting variations. We don't discuss Breadth First Search spanning trees because problem becomes less interesting. To be more specific for a fixed graph G and fixed vertex $v \in V(G)$ any BFS spanning tree constructed from v has same set of vertices in each level N_i, the set of vertices at distance i from the root.

1.1 Definitions

For simplification of the language, we will use "graph" meaning a "finite, simple graph", i.e finite graph with no loops or multiple edges.

Definition 1.1.1. A tree T with a distinguished vertex $v \in V(T)$ is called a rooted tree denoted by (T, v).

- T denotes a tree.
- P_n denotes the path with n vertices.
- C_n denotes the cycle with n vertices.
- K_n denotes the complete graph with n vertices.
- $K_{n,m}$ denotes the complete bipartite graph, color classes have size n and m.
- S_n denotes the star, complete bipartite graph $K_{1,n}$, a tree with one internal node and k leaves.

- A graph is called a broom and denoted by B_m^n if it is a path $P_n(v_1, v_2, \cdots, v_n)$ and m hanging leaves on vertex v_n. (see: Figure 1.1)

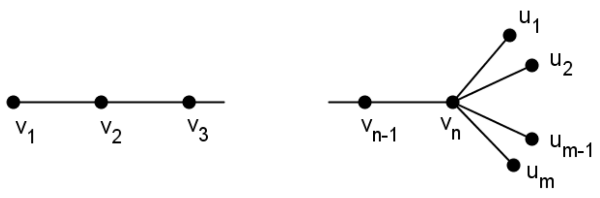

Figure 1.1: Broom

- $G = C((T_1, v_1), (T_2, v_2), \cdots, (T_n, v_n))$ $(n > 2)$ is a graph which contains n disjoint rooted trees (T_i, v_i), $i = 1, 2, , \cdots n$ and edges (v_i, v_{i+1}), $i = 1, 2, , \cdots n$, ("+" operation in modulo n arithmetics). (see: Figure 1.2)

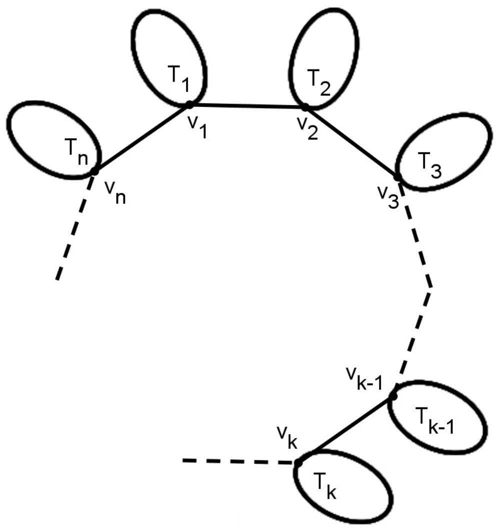

Figure 1.2:
$C((T_1, v_1), (T_2, v_2), \cdots, (T_n, v_n))$

- $G = K_n((T_1, v_1), (T_2, v_2), \cdots, (T_n, v_n)))$ is a graph which contains n disjoint rooted trees (T_i, v_i), $i = 1, 2, , \cdots n$ and edges (v_i, v_j), $i, j = 1, 2, , \cdots n$ and $i \neq j$.

- $G = K_{n,n}((T_1, v_1), (T_2, v_2), \cdots, (T_n, v_n))((T_1^*, v_1^*), (T_2^*, v_2^*), \cdots, (T_n^*, v_n^*))$ is a graph which contains $n+n$ disjoint rooted trees (T_i, v_i) and (T_i^*, v_i), $i = 1, 2, , \cdots n$ and edges (v_i, v_j^*), $i, j = 1, 2, , \cdots n$.

- $\forall v \in V(G)$, $d(v)$ denotes a degree of a vertex v.

- $\forall v, u \in V(G)$, $d_G(v, u)$ denotes the length of shortest path connecting vertex v with vertex u in G.

- A tournament is a digraph, obtained by assigning a direction for each edge in an undirected complete graph

Definition 1.1.2. For a graph G and $\forall v \in G$, DFS(T, v) denotes any spanning tree obtained by DFS algorithm, with starting vertex v.

Definition 1.1.3. Two graphs G and G' are isomorphic, denoted by $G \simeq G'$, if there exists a bijection ϕ between $V(G)$ and $V(G')$ such that any two vertices $v, u \in V(G)$ are adjacent in G if and only if $\phi(v)$ and $\phi(u)$ are adjacent in G'

Definition 1.1.4. Two rooted trees (T_1, v_1) and (T_2, v_2) are root-isomorphic, denoted by $(T_1, v_1) \simeq (T_1, v_1)$, if there exists a bijection ϕ between $V(T_1)$ and $V(T_2)$ such that $\phi(v_1) = v_2$ and any two vertices $v, u \in V(T_1)$ are adjacent in T_1 if and only if $\phi(v)$ and $\phi(u)$ are adjacent in T_2.

For a graph G, $G - e$ denotes a graph with vertices $V(G)$ and edges $E(G) - \{e\}$.

For a digraph G, $e = (v, u) \in E(G)$, we mean an edge which connects vertex v with vertex u and an edge is directed from v to u .

Definition 1.1.5. A directed graph G (digraph) is strongly connected graph if for all $v, u \in V(G)$, there is a directed path from vertex v to vertex u .

Strong orientation of an undirected graph is an assignment of a direction to each edge, that makes graph strongly connected graph.

Definition 1.1.6. For a graph G a vertex $v \in V(G)$ is central if $\forall v' \in V(G)$

$$\max_{u \in V(G)} d(v,u) \leq \max_{u \in V(G)} d(v',u)$$

Definition 1.1.7. For a tree T, the central element is the set of central vertices .

Theorem 1.1.1. In any tree T, the central element is either a single vertex or a pair of adjacent vertices.

Proof. Let us define sequence of trees $T_1 = T$ and T_{i+1} is a tree obtained from T_i just deleting leaves and all edges incident to them. Sooner or later, we will get T_{n+1} an empty graph where T_n is nonempty. This means T_n is a vertex or K_2. Since longest path from every vertex is path connecting this vertex with a leaf, it is clear that with this operation centeral element of a tree is not changing. This means, that central element of the tree T is the same as the central element of T_n. Hence theorem is proved.

■

Chapter 2

Depth First Search Spanning Trees

In this chapter, first we are going to present Tremaux's Depth First Search algorithm and then prove that it works. Then we will show modified DFS algorithm by Hopcroft and Tarjan. Also we will present DFS algorithm for digraphs and finally we will discuss several interesting properties of DFS spanning trees, which will be used in following chapters.

Algorithm 2.0.1. Tremaux's DFS Algorithm [12, p.53]:

1. $v \leftarrow r$. ($r \in V(G)$ is the root, any vertex from which we start traversing the graph G);

2. If there are no unmarked passages in v, go to the Step 4;

3. Find any unmarked passage, mark it with marker E and traverse the edge to its other endpoint u. If u has any marked passages (i.e. it is not a new vertex) mark the passage, though which u has just been entered with marker E, traverse the edge backwards to v, and go to

Step 2. If u has no marked passage, i.e. it is a new vertex, mark the passage through which u has been entered with marker F, $v \leftarrow u$ and go to the Step 2;

4. If there is no passage marked with marker F, halt. (We are in the root r and traversing of the graph is complete);

5. Use the passage marked with marker F, traverse the edge to its other endpoint u, $v \leftarrow u$ and go to the Step 2;

Lemma 2.0.1. Tremaux's algorithm never allows an edge to be traversed twice in the same direction [12, p.54]

Proof. If the passage is used as an exit (entering an edge), then either it is being marked E in the process, and thus the edge is never traversed again in this direction, or the passage is already marked F. It remains to be shown that no passage marked F is ever reused for entering the edge.

Let $e = (u, v) \in E(G)$ be the first edge to be traversed twice in the same direction, from u to v. The passage of e, at u, mast be labeled F. Since r(the root) has no passage marked F, $u \neq r$. Vertex u has been left $d(u) + 1$ times; once though each of the passages marked E and twice though e. Thus, u must have been entered $d(u) + 1$ times and some edge has been used twice to enter u, before e is used for the second time. A contradiction. □

An immediate corollary of Lemma 2.0.2 is that the process described by Tremaux's algorithm will always terminate. Clearly it can only terminate in r, since every other visited vertex has an F passage. Therefore, all we need to prove is that upon termination the whole graph has been scanned.

Lemma 2.0.2. [12, p.55] Upon termination of Tremaux's algorithm each edge of the graph has been traversed once in each direction.

Proof. Let us state the proposition differently: For every vertex all its incident edges have been traversed in both directions.

First, consider the start vertex r. Since the algorithm has terminated, all its incident edges have been traversed from r outward. Thus, r has been left $d(r)$ times and since we end up in r, it has also been entered $d(r)$ times. However by Lemma 2.0.2 no edge is traversed more then once in the same direction. Therefore, all the edges incident to r have been traversed once in each direction.

Assume now that S is the set of vertices for which the statement, that each of their incident edges has been traversed once in each direction, holds. Assume $V \neq S$. By the connectivity of the graph G there must be edges connecting vertices of S with $V - S$. All these edges have been traversed once in each direction. Let $e = (v, u) \in E(G)$ be the first edge to be traversed from $v \in S$ to $u \in (V - S)$. Clearly, u's passage, corresponding to e, is marked F. Since this passage has been entered, all other passages must have been marked E. Thus, each of u's incident edges has been traversed outward. The search has not started in u and not ended in u. Therefore, u has been entered $d(u)$ times, and each of it's incident edges has been traversed inwards. A contradiction, since u belongs to S. □

The Hopcroft and Tarjan version of DFS is essentially the same as Termaux's except that they number vertices from 1 to $n = |V(G)|$ in the order in which they are discovered. This is not necessary, as we have seen, for scanning the graph, but the numbering is useful for solving different problems (you can see in following chapters). Let us denote number (Time when v was discovered) of vertex v by $k(v)$. Also, now instead of marking passages, we will mark edges as "used" and remember for each vertex v, other then root r, the vertex $f(v)$ from which v has been discovered.

Now we are ready to present the algorithm.

Algorithm 2.0.2. The Hopcroft and Tarjan version of DFS algorithm [12, p.56]:

1. Mark all the edges "unused", For every $v \in V(G)$, $k(v) \leftarrow 0$, $i \leftarrow 0$. Choose a root $v \leftarrow r$;

2. $i \leftarrow i + 1$ and $k(v) \leftarrow i$;

3. If v has no unused incident edges, go to the step 5;

4. Choose an unused incident edge $e = (v, u) \in E(G)$. Mark e "used", If $k(u) \neq 0$ go to the step 3, Otherwise $k(u) = 0$ then $f(u) \leftarrow v$, $v \leftarrow u$ and go to the step 2;

5. If $k(v) = 1$ then halt;

6. $v \leftarrow f(v)$ and go to the step 3;

Since this algorithm is simple variation of the previous algorithm, we won't prove that algorith will terminate at some point and whole connected graph G will be scanned. Also it's trivial to admit that whole algorithm is of time complexity $O(E)$, namely, linear in the size of the graph.

After applying DFS Algorithm to the a connected graph G let us consider the set of directed edges $E' = \{(f(v), v) | \forall v \in V(G),\ v \neq r\}$.

Lemma 2.0.3. [12, p.56] The digraph $T = (V(G), E')$ defined above is a directed spanning tree with root r [12, p.56]

Proof. Clearly $d_{in}(r) = 0$ and $d_{in}(v) = 1\ \forall v \in V(G),\ v \neq r$. Hence If we take any directed path $P = (v_1, v_2, v_2, v_3, \cdots, v_k)$ then $v_i = f(v_{i+1}) \Longrightarrow k(v_i) < k(v_{i+1}) \Longrightarrow \forall i > j\ k(v_i) > k(v_j)$ hence any directed path P is simple. Since $\forall v \in V(G)$ $v \neq r$ we have $f(v)$, we can lengthen P. $P_1 = (f(v_1), v_1, v_2, v_2, v_3, \cdots, v_k))$, $P_2 = (f(f(v_1)), f(v_1), v_1, v_2, v_2, v_3, \cdots, v_k)$ and so on, this process can't be continuous, so at some point we will get r the root, hence theorem is proved. □

Lemma 2.0.4. If $\forall e = (v, u) \in (E(G) - E')$ then

- v is an ancestor of u;

  or

- v is an descendant of u;

Proof. Without loss of generality we can assume that $k(v) < k(u)$. So, when DFS algorithm entered in vertex v, vertex u was not explored yet and because of $e = (v, u) \in E(G)$ algorithm wont backtrack from v without exploring u. Hence v is an ancestor of u. □

Remark 1. Let us call all the edges from E' tree edges and all other edges back edges.

Algorithm 2.0.3. DFS algorithm for digraphs [12, p.63]:

1. Mark all the edges "unused", For every $v \in V(G)$, $k(v) \leftarrow 0$, $i \leftarrow 0$, choose a root $v \leftarrow r$ and $i \leftarrow 0$;

2. $i \leftarrow i + 1$ and $k(v) \leftarrow i$;

3. If v has no unused incident edges, go to the step 5;

4. Choose an unused incident directed edge $e = (v, u) \in E(G)$. Mark e "used", If $k(u) \neq 0$ go to the step 3, Otherwise if $k(u) = 0$ then $f(u) \leftarrow v$, $v \leftarrow u$ and go to the step 2;

5. If $f(v)$ is defined then $v \leftarrow f(v)$ and go to the step 3;

6. (f(v) is undefined), If there is a vertex u for which $k(u) = 0$ then let $v \leftarrow u$ (New root) and go to the step 2;

7. Halt;

The structure which results from the DFS traversing of digraph isn't as simple as it was in case of undirected ones. Instead of two types of edges we will have four types of them

- A directed edge $e = (v, u)$ is a tree edge if it was used to reach vertex u from vertex v;
- A directed edge $e = (v, u)$ is a forward edge if when it was scanned for the first time $k(u) > k(v)$;
- A directed edge $e = (v, u)$ is a back edge if when it was scanned for the first time $k(u) < k(v)$ and u is an ancestor of v;
- A directed edge $e = (v, u)$ is a cross edge if when it was scanned for the first time, $k(u) < k(v)$ but u isn't an ancestor of v;

Assume DFS was performed on a digraph G, and let the set of tree edges be E'. The graph $(V(G), E')$ is a forest (not a tree), union of disjoint directed trees.

Remark 2. If $e = (v, u) \in E(G)$ and $k(u) > k(v)$ then u is a descendant of v.

2.1 Applications of Depth First Search

There are dozens of examples where we can see that DFS algorithm is useful . Contemporary computer science uses DFS algorithm's different modifications nearly in every algorithm, which deals with graphs and not only with them. To show that DFS algorithm is useful we present several famous examples with DFS algorithm. Also it should be admitted that DFS can be used as a prof technique.

- **Detecting cycle in a graph:** a graph has a cycle if and only if we have a back edge during DFS algorithm.

- **Topological sort of digraph vertices** is a linear ordering of vertices, such that for every directed edge (u, v), u comes before v in the ordering.

  Idea of topological sort is trivial:

  Digraph has a topological sort of vertices, if and only if there is no directed cycle. We can check this with above mentioned DFS algorithm.

  If a digraph has no directed cycle then find a vertex with zero in degree, run DFS algorithm from it and for each vertex save time of backtracking it. Output vertices in reverse order of backtracking time. From digraph DFS tree properties, it is obvious that resulting order is topological sort of vertices.

- **To test if a graph is bipartite or not:** traverse graph with DFS algorithm and color each vertex with opposite color of it's parent. For each non DFS tree edge, check if it doesn't link two vertices of the same color. Root can be any color.

- **Solving puzzles with only solution** for example mazes.

- **Proving Redei's theorem:** every tournament contains a Hamiltonian path. Order vertices by decreasing of finishing time. It is obvious that, this will be a Hamiltonian path, from digraph DFS tree properties

Theorem 2.1.1. (Robbins' theorem) A graph G can be oriented to obtain a strongly connected digraph, if and only if G is 2-edge-connected (G has no bridge).

Proof. It is obvious that if a graph G has a bridge then there is no strongly connected orientation of G. Let us prove another side of the theorem.

If a graph G is simple and 2-edge-connected, we are going to find strongly connected orientation of it. Fix any vertex $v \in V(G)$ and construct rooted DFS spanning tree (T, v). Direct all edges of (T, v) from the parent to the child. Direct all other edges from the descendant to the ancestor (By DFS tree property all edges of $E(G) - E(T)$ connect descendants with ancestors). Let us define this directed graph G' formally:

$V(G') = V(G)$;

$E(G') = \{(u, u') | (u, u') \in E(T)$ and $k(u) < k(v)$ or

$(u, u') \in E(G) - E(T)$ and $k(u) > k(v)\}$;

We have left to prove that G' is strongly connected graph. This means that $\forall u_1, u_2 \in V(G')$, we should find P directed path from vertex u_1 to vertex u_2. It is enough to prove that $\forall u \in V(G')$ there are paths from vertex v (root of the DFS tree) to vertex u and from vertex u to vertex v. It is clear that $\forall u \in V(G')$ there is path from vertex v to vertex u. Let us prove that there is path from vertex u to vertex v with assuming contradiction. Assume u is a vertex with minimum distance from root v in the tree T from which there is no path towards vertex v. Let us denote with P a path of a tree T from the root v to the vertex u and with T' subtree of a tree T with root u. If there is an edge from some vertex $u' \in V(T')$ to some vertex $v' in V(P) - u$ then there is path from u to v because there is path from u to u', from u' there is an edge to v' and because of $d_T(v,v') < d_T(v,u)$ from assumption, there is path from vertex v' to v. If there is no edge from some vertex $u' \in V(T')$ to some vertex $v' \in V(P) - u$ then the edge $(u, f(u))$ which connects a vertex u with it's parent is a bridge, a contradiction. Hence Theorem is proved.

■

DFS algorithm is used to find **Nonseparable Components of directed graph.** Whole algorithm is from the book [12, p.57].

First we will define nonseparable components of a graph, then define low-point function for modifying DFS algorithm for our needs. Then we will prove several lemmas and present an algorithm.

A connected graph G is said to have separation vertex $v \in V(G)$, if $\exists u_1, u_2 \in V(G)$, $u_1 \neq v$ and $u_2 \neq v$, such that all paths connecting vertex u_1 with vertex u_2 pass trough v. In this case we also say v separates u_1 from u_2. A graph which has a separation vertex is called separable and one which has none is called Nonseparable.

Let $V' \subseteq V(G)$, the induced subgraph $G'(V', E')$ is called a nonseparable component, if G' is nonseparable and if for every larger V'' $(V' \subset V'' \subseteq V(G))$ the induced subgraph $G''(V'', E'')$ is separable.

If a graph $G(V,E)$ contains no separation vertex, clearly the whole G is a nonseparable component. However, if v is a separating vertex then $V - \{v\}$ can be partitioned into $V_1, V_2, \cdots, V_k$ such that $V_1 \cup V_2 \cup \cdots \cup V_k = V - \{v\}$ and $V_i \cap V_j = \emptyset$ $i \neq j$. Two vertices u_1, u_2 are in same V_j if and only if there is path connecting them, which doesn't contain v. (It is obvious that this relation is reflexive, symmetric and transitive hence this relation is equivalence relation). Now we can consider each of the subgraphs induced by $V_j \cup \{v\}$ and continue to partition it into the smaller parts with same way.

Finally we will get desired partition.

Now we are ready to discuss how we can find nonseparable components of directed graph.

Let define the low-point of v in DFS tree, $L(v)$ be the least number $k(u)$, of a vertex u which can be reached from v via directed path consisting of a tree edges followed by at most one back edge (possible empty). Clearly $L(v) \leq k(v)$ because we can use an empty path from v to itself. Obviously, if a non-empty path is used then its last edge is back edge, because directed path of tree edges leads to vertices with higher value of $k(.)$.

Now, we are ready to prove some lemmas

Lemma 2.1.1. G is a graph whose verteces are already numbered by DFS hence we have function $k : V(G) \longrightarrow \mathbb{N}$ then If $(u, v) \in E(G)$ is a tree edge with corresponding direction $(k(u) < k(v))$, $k(u) > 1$ and $L(v) \geq k(u)$ then u is separating vertex of G

Proof. Let P be the path from the root r to u in DFS spanning tree, including r and not including u, and T' be the subtree of DFS spanning tree rooted at v, including v. By Lemma 2.0.4 there is no edge connecting any vertex from $V(T')$ to any vertex $V(G) - V(T') \cup \{u\} \cup V(P)$. Clearly, $\forall v' \in V(P)$ $k(v') < k(v)$ and there is no edge connecting $\forall t \in V(T')$ to v' because of lemma hypothesis $L(v) \geq k(u)$. Hence u is a vertex, separating $V(P)$ from $V(T')$ and is therefore a separating vertex. □

Lemma 2.1.2. G is a graph whose vertices are already numbered by DFS hence we have function $k : V(G) \longrightarrow \mathbb{N}$ then If $k(u) > 1$ and u is separating vertex then there exists a tree edge $(u, v) \in E(G)$ with corresponding direction $(k(u) < k(v))$, such that $L(v) \geq k(u)$.

Proof. Since u is separating vertex so there is partition $V = \{u\} \cup V_1 \cup V_2 \cdots \cup V_m$ $m > 1$ s.t. $\forall i \neq j$ any path from a vertex of V_i to a vertex of V_j pass trough u. Let us assume that the root of this DFS tree is $r \in V_1$. Then take first tree edge (u, v) with corresponding direction $(k(u) < k(v))$ such that $v \notin V_1$ (without loss of generality we can assume $v \in V_2$). Since there are no edges connecting V_2 with $V - V_2 - \{u\}$, so $L(v) \geq k(u)$. □

Lemma 2.1.3. *G is a graph whose vertices are already numbered by DFS, then root r is separating vertex if and only if it's degree in DFS spanning tree is more than one.*

Proof. Clearly, if r is separating, it should have degree more than one (precisely it will have degree equal to number of partition classes); If r has degree more than one then there are $u_1, u_2 \in N_{DFS}(r)$ and by Lemma 2.0.4 there is no edge connecting vertices from subtree rooted at u_1 with vertices from subtree rooted at u_2. Hence v is separating u_1 from u_2, therefore it is a separating vertex. □

The remaining problem is the way of calculating $L(.)$ function efficiently. It is possible to calculate that $L(v)$ by the time we backtrack vertex v. Calculating $L(v)$ when vertex v is a leaf in DFS spanning tree is easy

$$L(v) = Min\{k(u) | u = v \text{ or } (v, u) \text{ is a back edge}\}$$

For all vertex v not a leaf in DFS spanning tree, we can calculate $L(v)$ by the time when we backtrack vertex v, $L(v) = \text{Min}(k(v), L(u)) \forall u \in V(G)$ where (v, u) is a tree edge with corresponding direction.

From Lemmas 3.0.8, 3.0.9, 3.0.10 and functions $k(.)$ and $L(.)$ we can find separating vertices. If we delete separating vertices from DFS spanning tree connected components will be non-separable components of the graph.

Now we are ready to present the algorithm.

Algorithm 2.1.1. Algorithm for finding non-separable components

```
1. Mark all edges as "unused",
   $\forall v \in V(G)$, $k(v) \leftarrow 0$,
   $i \leftarrow 0$ and fix some $v \in V(G)$,
   Let $S$ be an empty sequence;

2. $i \leftarrow i + 1$, $k(v) \leftarrow i$, $L(v) \leftarrow k(v)$,
   Put v in sequence $S$ from the end;

3. If $v$ has no unused incident edge go to step 5;

4. Choose an unused incident edge of vertex $v$ $e = (v, u)$, mark $e$ 'used'.
   If $k(u) \neq 0$ then $L(v) \leftarrow Min\{L(v), k(u)\}$ and go to step 3,
   Otherwise if $k(u) = 0$ then $f(u) \leftarrow v$, $v \leftarrow u$, and go to step 2;

5. If $k(f(v)) = 1$ go to step 7;
   Else if $k(f(v)) \neq 1$ then if $L(v) < k(f(v))$ then $L(f(v)) \leftarrow Min\{L(f(v)), L(v)\}$
   and go to step 6;
   Else if $k(f(v)) \neq 1$ and $L(v) \geq k(f(v))$ then $f(v)$ is separating vertex.
   All vertices from $S$ down to and including $v$ are now removed from $S$.
   This set with $f(v)$ , forms a nonseparable component;

6. $v \leftarrow f(v)$ go to step 3;

7. All vertices in $S$ down to and including $v$ are now removed from $S$,
   they form with root a nonseparable component;

8. If root has no unused incident edge then halt;

9. Root is separating vertex. Let $v \leftarrow$ root and go to step 4;
```

Time complexity of this algorithm is $O(|E|)$ because each edge is scanned exactly once in each direction and number of operations per edge is bounded by constant.

Algorithm for finding strongly-connected Components [12, p.64]:

Let G be a digraph. Let us define an equivalence relation $\sim$ on $V(G)$. For $x, y \in V(G)$ we say $x \sim y$ if and only if there is directed path from x to y and also there is a directed path from y to x in G. Clearly $\sim$ is an equivalence relation hence it partitions $V(G)$ into equivalence classes. These equivalence classes are called the stronglyconnected components of G.

Let $C_1, C_2, \cdots, C_m$ be the stronglyconnected components of a digraph $G(V, E)$. Let us define the superstructure of G, $G'(V', E')$

$$V' = \{C_1, C_2, \cdots, C_m\}$$

$$E' = \{(C_i, C_j) \text{ ordered pair } | i \neq j, (x, y) \in E(G), x \in C_i \text{ and } y \in C_j\}$$

Graph G' is free of directed cycle, because if it has a directed cycle, all stronglyconnected components on it should have been in one component. Thus, there must be at least one sink, C_k, $(d_{out}(C_k) = 0)$ in G.

Let v is the first vertex visited by DFS algorithm from C_k. All vertices of C_k are reachable from v hence no retracting from v is attempted until all the vertices of C_k are numbered. They all get numbers greater than $k(v)$ and since there are no edges leading out of C_k in G no vertices will be visited outside of C_k from the time of v is numbered until we retract from v. Hence all these vertices are in C_k.

For finding such v we first define again lowpoint then present algorithm and finally prove why this algorithm works.

$L(v) =$ the least number $k(u)$, of a vertex u which can be reached from v via a directed path containing of tree edges (possibly empty), followed by one back edge or a cross edge, provided u belongs to the same strongly connected component.

Algorithm 2.1.2. Algorithm for finding strongly-connected components

1. Mark all edges "unused",

   $\forall v \in V(G)$, $k(v) \leftarrow 0$, $f(v) \leftarrow$"undefined",

   Let S be an empty sequence,

   $i \leftarrow 0$, $v \leftarrow r$ where r is any fixed vertex of $V(G)$, (r is the root);

2. $i \leftarrow i + 1$, $k(v) \leftarrow i$, $L(v) \leftarrow i$ and put v in S

3. If there are no unused incident edges from v go to step 7;

4. Choose an unused directed edge $e = (v, u)$. Mark e "used".

   If $k(u) = 0$ then $f(u) \leftarrow v$, $v \leftarrow u$ and go to step 2.

5. If $k(u) > k(v)$ (e is a forward edge) go to step 3.

   Otherwise if $k(u) \leq k(v)$ if u is not in S (u and v are from different strongly-connected component) go to step 3;

6. $k(u) < k(v)$ and both vertices are in same component, then $L(v) \leftarrow$ Min$(L(v), k(u))$ and go to step 3;

7. If $k(u) = k(v)$ then delete all the vertices from S down to and including v, these vertices form a strongly-connected component.

8. If $f(v)$ is defined then $L(f(v)) \leftarrow Min(L(f(v)), L(v))$, $v \leftarrow f(v)$ and go to step 3;

9. else if $f(v)$ is undefined then find $u \in V(g)$ such that $k(u) = 0$ then let $v \leftarrow u$ and go to step 2;

10. Halt!

Lemma 2.1.4. Let v be the first vertex for which, in step 7, $L(v) = k(v)$. Then all vertices from S down to and including r form a strongly-connected component of G. (This component is one of the sinks)

Proof. All vertices in S, on top of v, have been numbered after v, and since no backtracking from v has been attempted yet, these vertices are the descendants of v; i.e. are reachable from v via tree edge.

Now, we want to show that if u is a descendant of v then v is reachable from u. We have already backtracked from u, but since v is the first vertex for which equality occurs in step 7, $L(u) < k(u)$. Thus, a vertex u', $k(u') = L(u)$, is reachable from u. Also, $k(u') \geq k(v)$, since by step 8 $L(v) \leq L(u)$, $(k(u') = L(u) \geq L(v) = k(v))$ hence u' is a descendant of v. If $u \neq r$ then we can repeat the argument again to find a lower numbered descendant of v reachable from u', and therefore from u. This argument can be repeated until v is found to be reachable from u.

So far we have shown that all vertices in S, on top of v and including v, belong to the same component. It remains to show that no additional vertices belong to same component.

When the equality $L(v) = k(v)$ is discovered, all the vertices with numbers higher then $k(v)$ are on top of v in S. Thus, if there are any additional vertices in component, at least some of them must have been discovered already, and all of those are numbers lower that $k(v)$. There must be at least one (back or cross) edge (x, y) such that $k(x) \geq k(v) > k(y) > 0$ and x, v and y are in the same component. Since no removal from S has taken place yet, $L(x) < L(y)$

ant therefore $L(v) \leq L(x) \leq k(y) \leq k(v)$ a contradiction $\square$

If v is the first vertex for which in step 7 $L(v) = k(v)$, then by Lemma 2.1.4 and it's proof, a component C has been numbered and all its elements are descendants of v, up to now at most one edge from a vertex outside C into C has been used to change the lowpoint value of a vertex outside C. At this point all vertices of C are removed and therefore none of the edges entering C can change a lowpoint value anymore. Actually, this is equivalent to removal of C and all its incident edges from G before DFS algorithm starts. Thus, when equality in Step 7 will occur again, Lemma 2.1.4 is effective again. This proves the validity of the algorithm.

Chapter 3

Graphs with Isomorphic DFS Spanning Trees

In this chapter we deal with various problems related with graphs with isomorphic spanning trees. First, we start with the problem of isomorphic spanning trees, the question is what graphs have spanning trees such that all of them are isomorphic to each other and present proof by P. D. Vestergaard 1989 [1]. Then we consider same question but with only DFS spanning trees. Since the set of DFS spanning trees of a graph is subset of the set with spanning trees of a graph, class of graphs which is solution for new problem will increase but question is how big is this set. Unfortunately, this problem seems to be too complicated and we couldn't find this class of graphs but we found interesting class of examples and propose a conjecture. Next discussed problems in this chapter is also related with previous ones with slight difference, we fix a tree and find all graphs with all DFS spanning trees isomorphic to this tree. Finally, we discuss same problem but with rooted DFS trees.

Theorem 3.0.2 (P. D. Vestergaard 1989 [1])**.** Connected graph G with all isomorphic spanning trees are

- T a tree;
- $C_{2k+1}(T, T, \cdots, T)$;
- $C_{2k}(T_1, T_2, \cdots, T_1, T_2)$;

Proof. Clearly, these graphs are simple, connected and they have all isomorphic spanning trees.

We should prove that these graphs are the only ones with isomorphic spanning trees. Let us assume that G is simple, connected and has no cycle, hence G is a tree and we already know that all trees are solution of this problem.

Let us assume that a simple, connected graph G with all isomorphic spanning trees, has exactly **one cycle**, then $G = C_k((T_1, v_1), (T_2, v_2), \cdots, (T_k, v_k))$.

All operations in this proof will be held in modulo k arithmetics. Let us name edges of C_k

$$e_i = (v_i, v_{i+1})$$

Remark: Spanning trees of the graph G are only $G - e$ trees, $\forall e \in C_k$

Lemma 3.0.5. Every longest path P in $G = C_k((T_1, v_1), (T_2, v_2), \cdots, (T_k, v_k))$

- Contains all edges of C_k but one;

  Or

- Contains all edges of C_k but two e and e', these two edges have common vertex v and the vertex v is diametrically opposite of central element of trees $G - e$ and $G - e'$;

Proof. First we can conclude that $E(C_k) - E(P)$ is C_k or path. Then if $|E(C) - E(P) > 1|$ then we have $\{e_i, e_{i+1}\} \in (E(C) - E(P))$ (two adjacent edges with common vertex v_{i+1}). Because of P is longest path of trees $G - e_i$

and $G - e_{i+1}$, the central element of P, $G - e_i$ and $G - e_{i+1}$ are all same, let's call it c. From theorem statement $G - e_i$ and $G - e_{i+1}$ are isomorphic and we know that central element is an isomorphism invariant, so

$$\{d_{G-e_i}(c, v) | v \in V(G)\} = \{d_{G-e_{i+1}}(c, v) | v \in V(G)\}$$

Easily can be seen that $\{d_{G-e_i}(c, v) | v \in V(G) - T_{i+1}\} = \{d_{G-e_{i+1}}(c, v) | v \in V(G) - T_{i+1}\} \Longrightarrow$

$$\{d_{G-e_i}(c, v) | v \in T_{i+1}\} = \{d_{G-e_{i+1}}(c, v) | v \in T_{i+1}\}$$

Also for some constant number *const* and $\forall v \in T_i$ we have $d_{G-e_i}(c, v) = const + d_{G-e_{i+1}}(c, v)$. From last two equation, we can conclude that $const = 0$. Hence k is even and v_{i+1} has diametrically opposite vertex $v_{i+1+\frac{k}{2}} \Longrightarrow v_{i+1+\frac{k}{2}} = c$. If for path P there exists another adjacent pair of edges in $E(C) - E(P)$, then they will point out another place for c. Hence

$$|E(C) - E(P)| \leq 2$$

$\square$

Assume that every longest path of G contains exactly $\mathbf{k-1}$ edges of $\mathbf{C_k}$ then first, we will prove following lemma:

Lemma 3.0.6. $\forall i, j \in 1, 2, \cdots, k$

$$h(T_i) = h(T_j)$$

Proof. Assume contradiction, then $\exists i$ for which $h(T_i) > h(T_{i+1})$. From theorem assumption, the length of a longest path of $G - e_i$ is equal to length of a longest path of G, so a longest path of $G - e_i$ is a longest path of G, hence it contains all edges of C_k except e_i and has length $h(T_i) + h(T_{i+1}) + k - 1$. Since all longest paths contain exactly $k - 1$ edges from C_k so the length of any path starting in T_i and ending in T_{i+2} is less then $h(T_i)+h(T_{i+1})+k-1$. Hence $h(T_{i+2}) \leq h(T_{i+1})$. By the same argument, a longest path of $G-e_{i+1}$ has length $h(T_{i+1})+h(T_{i+2})+k-1$ but $h(T_{i+1})+h(T_{i+2})+k-1 < h(T_i)+h(T_{i+1})+k-1$ a contradiction. Therefore $h(T_i)=h(T_j)$ $\forall i,j \in 1,2,\cdots,k$. □

If k is odd, then $\forall i \in \{1,2,\cdots,k\}$ from Lemma 3.0.5 v_i is central element of the tree $G-e_{i+(k-1)/2}$. The intersection of all longest paths in this tree, will contain $E(C_k)-e_{i+(k-1)/2}$. Because of central element and intersection of all longest paths are isomorphism invariants, the isomorphism ϕ, which exists by statement of the theorem, between trees $G-e_{i+(k-1)/2}$ and $G-e_{i+1+(k-1)/2}$, will map the vertex v_i to the vertex v_{i+1} and $E(C_k)-e_{i+(k-1)/2}$ to $E(C_k)-e_{i+1+(k-1)/2} \Longrightarrow$ so T_i is isomorphic to T_{i+1} $\forall i \in \{1,2,\cdots,k\}$. Hence theorem is proved if every longest path of G contains exactly $k-1$ edges of C_k and k is odd.

If k is even, $\forall i \in \{1,2,\cdots,k\}$ from Lemma 3.0.5 e_i is central element of the tree $G-e_{i+\frac{k}{2}}$. Intersection of all longest paths in this tree will contain $E(C_k)-e_{i+\frac{k}{2}}$. The isomorphism ϕ from the tree $G-e_{i+\frac{k}{2}}$ to the tree $G-e_{i+1+\frac{k}{2}}$ maps e_i to e_{i+1}. If $\phi(v_i)=v_{i+1}$ and $\phi(v_{i+1})=v_{i+2}$ then we deduce $T_i \simeq T_{i+1}$ and $T_{i+1} \simeq T_{i+2}$; If $\phi(v_i)=v_{i+2}$ and $\phi(v_{i+1})=v_{i+1}$ then we deduce $T_i \simeq T_{i+2}$. These are solutions from statement of this theorem. Hence Theorem is proved if every longest path of G contains exactly $k-1$ edges of C_k and k is even.

If there exists a longest path P of G containing $\mathbf{k-2}$ edges of $\mathbf{C_k}$ then assume without loss of generality $E(C_k)-E(P)=\{e_1,e_2\}$. By Lemma 3.0.5, central element of P path is diametrically opposite of v_2. So $h(T_1)=h(T_3)=h$ and $h(T_2)<h$ and $h(T_4)<h$ because of P is one of the longest paths.

Lemma 3.0.7. *k is even and $\forall T_j$ with odd subscript have same height h and all other trees have height less then h*

Proof. If in tree $G - e_4$ all longest paths contain $k - 1$ edges from C_k then $h(T_5) \geq h$ because of $h(T_4) \leq h-1$. On the other hand, $h(T_5) < h$ otherwise, a longest path starting in T_3 and ending in T_5 would be a longest paths in the tree $G - e_4$ with $k - 2$ edges from C_k, a contradiction. Hence tree $G - e_4$ has a longest paths containing $k-2$ edges from C_k. Hence two missing edges are e_3, e_4 or e_4, e_5. In the second case if we use Lemma 3.0.5 then $h(T_6) = h(T_4)$ and because of $h(T_4) \leq h - 1$ we got a contradiction because longest path of the tree $G - e_4$ will be shorter than path P. Therefore, the two missing edges are e_3, e_4 hence $h(T_1) = h(T_3) = h(T_5) = h$ from Lemma 3.0.5. We can continue this argument around C_k. If k were odd then $h(T_k) = h$ and a longest path starting in T_1 and ending in T_k would be longer then P a contradiction. So $2|k$ and $h(T_j) = h \; \forall \text{odd } j \in 1, 2, \cdots, k$. The proof of lemma is complete. □

From last Lemma and Lemma 3.0.5, the central element of the longest path P is $v_{2+\frac{k}{2}}$ hence this is central element of trees $G - e_1$ and $G - e_2$. The two vertices belonging to the intersection of all longest paths and each having distance $\frac{k}{2} - 1$ to the central element is an isomorphism invariant, so for the isomorphism ϕ between trees $G - e_1$ and $G - e_2$

$$\phi(\{v_1, v_3\}) = \{v_1, v_3\}$$

and since v_1 has different degree in $G - e_1$ and $G - e_2$ we have $\phi(v_1) = v_3$ and $\phi(v_3) = v_1$ so rooted trees (T_1, v_1) and (T_3, v_3) are isomorphic. By the same argument around C_k, we get

$$(T_{2j+1}, v_{2j+1}) \simeq (T_{2i+1}, v_{2j+1})$$

To prove $(T_{2j}, v_{2j}) \simeq (T_{2i}, v_{2j})$, we consider two cases $\mathbf{k \equiv 0 \text{ or } 2 (mod\ 4)}$.

If $\mathbf{k \equiv 0 (mod\ 4)}$ then in the tree $G - e_1$ the rooted tree, attached to the central vertex outside from intersection of all longest paths, is $(T_{2+\frac{k}{2}}, v_{2+\frac{k}{2}})$.

In the tree $G - e_3$ the rooted tree, attached to the central vertex outside from intersection of all longest paths, is $(T_{4+\frac{k}{2}}, v_{4+\frac{k}{2}})$. Since the rooted tree, attached to the central vertex outside from intersection of all longest paths is an isomorphism invariant and $G - e_1$ is isomorphic to $G - e_3$, we get

$$(T_{2+\frac{k}{2}}, v_{2+\frac{k}{2}}) \simeq (T_{4+\frac{k}{2}}, v_{4+\frac{k}{2}})$$

By the same argument around C_k we get

$$(T_{2j}, v_{2j}) \simeq (T_{2i}, v_{2j})$$

So theorem is proved if $k \equiv 0(mod\ 4)$.

If $\mathbf{k \equiv 2(mod\ 4)}$ then consider trees $G - e_1$ and $G - e_2$ and isomorphism function ϕ between of them. Central element, intersection of all longest paths and the two vertices belonging to the intersection of all longest paths with fixed distance from the central vertex are an isomorphism invariants. So we have $\phi(\{v_1, v_3\}) = \{v_1, v_3\}$ and $\phi(\{v_{1+\frac{k}{2}}, v_{3+\frac{k}{2}}\}) = \{v_{1+\frac{k}{2}}, v_{3+\frac{k}{2}}\}$ obviously $d_{G-e_1}(v_1) < d_{G-e_2}(v_1)$ hence $\phi(v_1) = v_3 \implies \phi(v_{1+\frac{k}{2}}) = v_{3+\frac{k}{2}} \implies$ it implies that

$$(T_{1+\frac{k}{2}}, v_{1+\frac{k}{2}}) \simeq (T_{3+\frac{k}{2}}, v_{3+\frac{k}{2}})$$

By using same argument around C_k we will get

$$(T_{2j}, v_{2j}) \simeq (T_{2i}, v_{2j})$$

So we have proved that, all graphs with maximum one cycle, are only ones which are stated in the theorem.

If a graph G has more then one cycle, then we can delete several edges, so that we will maintain connectivity of the graph and we will get new connected simple graph G' with exactly two chordless cycles and all isomorphic spanning trees. There are three types of such graph G'

- G' has two disjoint chordless cycle $C_1 = (u_1, u_2, \cdots, u_{t_1})$ and $C_2 = (u'_1, u'_2, \cdots, u'_{t'_1})$;

- G' has two chordless cycle $C_1 = (u, u_2, \cdots, u_{t_1})$ and $C_2 = (u, u'_2, \cdots, u'_{t_1})$ with exactly one common vertex u;

- G' has two chordless cycle $C_1 = (v_1, v_2, \cdots, v_l, u_1, u_2, \cdots, u_{t_1})$ and $C_2 = (v_1, v_2, \cdots, v_l, u'_1, u'_2, \cdots, u'_{t_2})$

Proof of first two cases are just the same, that's why we will prove only first case. There exists path from C_1 to C_2 and without loss of generality, we can assume that there is path $P = (u_1, v_2, v_3 \cdots, v_t, u'_1)$ such that $V(P) \cap V(C_1) = \{u_1\}$ and $V(P) \cap V(C_2) = \{u'_1\}$. Knowing solution for graphs with one cycle, we can conclude that number of vertices in u_1's connected component after deleting vertex u'_1 is equal to number of vertices of a tree hanging on u'_3 (without vertex u'_3) and number of vertices in u'_1's connected component after deleting vertex u_1 is equal to number of vertices of a tree hanging on u_3 (without vertex u_3) which is clearly a contradiction. Hence there is no graph with all isomorphic spanning trees of first two type.

Suppose that G' has two chordless cycle $C_1 = (v_1, v_2, \cdots, v_l, u_1, u_2, \cdots, u_{t_1})$ and $C_2 = (v_1, v_2, \cdots, v_l, u'_1, u'_2, \cdots, u'_{t_2})$. If l is odd then in graph with one cycle $G' - (v_1, u_{t_1})$ hanging trees on vertices v_1 and v_l should be isomorphic and also in graph $G' - (v_l, u'_1)$, which cannot happen at the same time so we got a contradiction else if l is even then number of vertices on the tree hanging on vertex u_1 should be equal to the number of vertices hanging on v_1 in graphs $G' - (v_1, u_{t_1})$ and $G' - (v_l, u'_1)$ which is also impossible to happen either at the same time , a contradiction again and the theorem is proved.

■

Now, we consider just DFS spanning trees, but the question is the same.

What graphs have DFS spanning trees such that all of them are isomorphic to each other (we are not interested what vertex is the root of DFS spanning tree in this case). First it is easy to see that T, C_n, K_n and $K_{n,n}$ are in solution class. After considering the class of graphs whose spanning trees are isomorphic to each other (previous problem), we can try to hang disjoint trees on each vertex but question is how?

- Hanging trees on vertices of T is trivial you can hang anything on each vertex, still it will stay as a tree;
- Hanging trees on vertices of C_n is also clear now because the set of $C_n((T_1, v_1), (T_2, v_2), \cdots, (T_n, v_n))$'s DFS spanning trees is same as just spanning trees. So we know solution for this case $C_{2k+1}(T, T, \cdots, T)$ and $C_{2k}(T_1, T_2, \cdots, T_1, T_2)$;
- Hanging trees on vertices of K_n isn't also very complicated after we have seen solution of previous problem, because the set of

  $C_n((T_1, v_1), (T_2, v_2), \cdots, (T_n, v_n))$'s DFS spanning trees is just union of spanning trees of $C_n((T_{\sigma(1)}, v_{\sigma(1)}), (T_{\sigma(2)}, v_{\sigma(2)}), \cdots, (T_{\sigma(n)}, v_{\sigma(n)}))$ for every permutation σ. and we get some solution in this case $K_n(T, T, \cdots, T)$;
- Hanging trees on vertices of $K_{n.n}$ will give us also solutions $K_k(T, T, \cdots, T)$ and $K_{k,k}(T, T, \cdots, T)(T', T', \cdots, T')$. (From previous case this is just trivial if we consider that we can't take just any permutation σ. σ should be such permutation which permutes odd numbers to odd and even numbers to even);

We couldn't prove that there is no other graph such that all DFS trees are isomorphic to each other, so we propose a conjecture.

Conjecture 3.0.1. Connected graph G with all isomorphic DFS spanning trees are

- T a tree;
- $C_{2k+1}(T, T, \cdots, T)$;
- $C_{2k}(T_1, T_2, \cdots, T_1, T_2)$;
- $K_k(T, T, \cdots, T)$;
- $K_{k,k}(T, T, \cdots, T)(T', T', \cdots, T')$;

Now, we will fix a tree (”interesting” ones) such as P_n, S_n, B_m^{n-m} and find all graphs whose DFS spanning trees are isomorphic to each other.

Theorem 3.0.3. Simple, connected graph G with all DFS spanning trees paths are only

- P_n
- C_n
- K_n
- $K_{\frac{n}{2},\frac{n}{2}}$

Proof. If $\exists v_0 \in V(G), d(v_0) = 1$ construct DFS path from v_0

$$P_1 = (v_0, v_1, \cdots, v_{n-1})$$

Start constructing DFS path from v_{k-1} for all $k = 2, 3, \cdots, n-2$ and first take the subpath of P_1 towards v_0, $P_2' = (v_{k-1}, v_{k-2}, \cdots, v_1, v_0)$, because of $d(v_0) = 1$ DFS algorithm will start backtracking and hence all DFS spanning trees are paths

$$(v_i, v_j) \notin E(G) \text{ if } i < k-1 \text{ and } j > k-1$$

From last we can conclude that $\mathbf{G} = \mathbf{P_n}$ this is a solution.

If $\forall v \in V(G), d(v) > 1$ then construct DFS path from any $v_0 \in V(G)$

$$P = (v_0, v_1, \cdots, v_{n-1})$$

Start constructing DFS path from v_1 and first take sub-path of P towards v_{n-1}, $P' = (v_1, v_2, \cdots, v_{n-1})$ then, DFS algorithm shouldn't start backtracking because of $d(v_0) > 1$ and if it does then DFS spanning tree won't be a path. Hence $(v_0, v_{n-1}) \in E(G)$ which means that we have Hamilton cycle $C = (v_0, v_1, \cdots, v_{n-1})$ (see: Figure 3.1 (a))

From now on, all operations will be modulo n operations in this proof. It is obvious that $\mathbf{G} = \mathbf{C_n}$ is a solution. If $G \neq C_n$ then

$$E(G) - E(C) \neq \emptyset \tag{3.1}$$

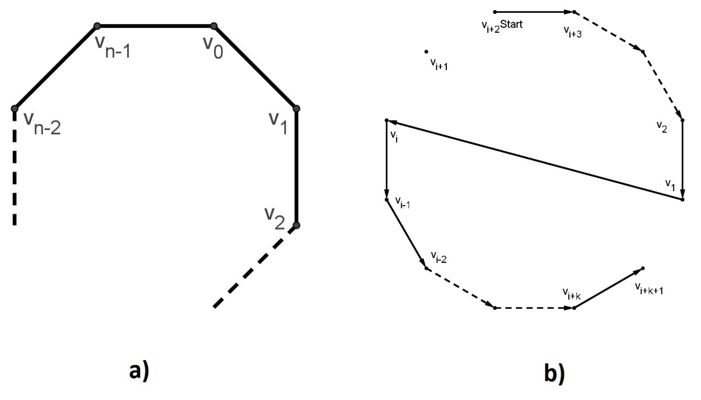

Figure 3.1: $C = (v_0, v_1, \cdots, v_{n-1})$

Lemma 3.0.8. If $(v_i, v_{i+k}) \in E(G)$ then $\forall i'$ $(v_{i'}, v_{i'+k}) \in E(G)$

Proof. Start constructing DFS spanning path (see: Figure 3.1 (b))

$$v_{i+2}, v_{i+3}, \cdots, v_{i+k}, v_i, v_{i-1}, v_{i-2}, \cdots, v_{i+k+1}$$

There is left only one vertex v_{i+1} and if $(v_{i+1}, v_{i+1+k}) \notin E(G)$ then after backtracking , v_{i+1} will be hung somewhere in the middle of this path because of $d(v_{i+1}) > 1$, a contradiction so $(\mathbf{v_{i+1}}, \mathbf{v_{i+1+k}}) \in \mathbf{E}(\mathbf{G})$. By the same argument around C_n, Lemma 3.0.8 will be proved. □

Lemma 3.0.9. $\forall i$ $(v_i, v_{i+3}) \in E(G)$

Proof. To prove this lemma it is sufficient to prove for $i = 0$ because of lemma 3.0.8.

From (3.1) and Lemma 3.0.8 $\exists k$ for which $(v_1, v_{1+k}) \in E(G)$ Also $(v_3, v_{3+k}) \in E(G)$ Start constructing DFS spanning path

$$v_4, v_5, \cdots, v_{1+k}, v_1, v_2, v_{2+k}, v_{3+k}, \cdots, v_0$$

There is left only one vertex v_3 and if $(v_0, v_3) \notin E(G)$ then after backtracking, v_3 will be hung somewhere in the middle of this path because of $d(v_3) > 1$, a contradiction so $(v_0, v_3) \in E(G)$ □

Lemma 3.0.10. If $(v_i, v_{i+k}) \in E(G)$ then $(v_i, v_{i+k+2}) \in E(G)$

Proof. From lemmas 3.0.8 $(v_1, v_{1+k}) \in E(G)$ and $(v_2, v_{2+k}) \in E(G)$

Start constructing DFS spanning path

$$v_2, v_1, v_{1+k}, v_k, v_{k-1}, \cdots, v_3, v_{3+k}, v_{4+k}, \cdots, v_{n-1}, v_0$$

There is left only one vertex v_{2+k} and if $(v_0, v_{k+2}) \notin E(G)$ then after backtracking, v_{2+k} will be hung somewhere in the middle of this path because of $d(v_{2+k}) > 1$, a contradiction so $(v_0, v_{k+2}) \in E(G)$. By Lemma 3.0.8 we will have $(v_i, v_{i+k+2}) \in E(G)$ □

From Lemmas 3.0.8,3.0.9,3.0.10

$$\forall m \in \mathbb{N} \ (v_i, v_{i+2m+1}) \in E(G) \tag{3.2}$$

From (3.2), if n is odd then $\mathbf{G} = \mathbf{K_n}$ and this is a solution. If n is even G contains $K_{\frac{n}{2},\frac{n}{2}}$ from (3.2) and if G has no other edge $\mathbf{G} = \mathbf{K_{\frac{n}{2},\frac{n}{2}}}$ and this is a solution. If there is an other edge $(v_i, v_{i+2k}) \in E(G)$ in G from lemma 3.0.8 $\exists k \ (v_0, v_{2k}) \in E(G)$ using lemma 3.0.10 $(v_0, v_{2k+(n+2-2k)}) = (v_0, v_2) \in E(G)$ now we can conclude that $\mathbf{G} = \mathbf{K_n}$ from lemmas 3.0.8 and 3.0.10

■

Another interesting, but simple case is to find all graphs where all DFS spanning trees are stars.

Theorem 3.0.4. Simple, connected graph G with all DFS spanning trees stars S_{n-1} are

- S_{n-1}
- K_3

Proof. If $n \leq 3$ then $G \in \{S_0, S_1, S_2, S_3, K_3\}$ is trivial to check.

If $n > 3$ construct DFS spanning star S_{n-1} from any $u \in V(G)$. Let us assume that central vertex is v and there are $n-1$ leaves $v_1, v_2, \cdots, v_{n-1}$ and if $G \neq S_{n-1}$ then $\exists v_i, v_j, v_k$ pairwise different vertices such that $(v_i, v_j) \in E(G)$. Start constructing DFS spanning tree v_k, v, v_i, $v_j \cdots$ we got path of length 3 which is contradiction because every path in a star has length less than 3.

■

Now, we consider a generalization of paths and stars, brooms. We wont to find all graphs where all DFS spanning trees are Brooms.

Theorem 3.0.5. Simple, connected graph G such that all DFS spanning trees are brooms B_k^{n-k} where $1 < k < n-2$ and $n > 6$ are only

- B_k^{n-k}

Remark 3. If $n \leq 6$ then except of brooms there are solutions which you can see on figure 3.2

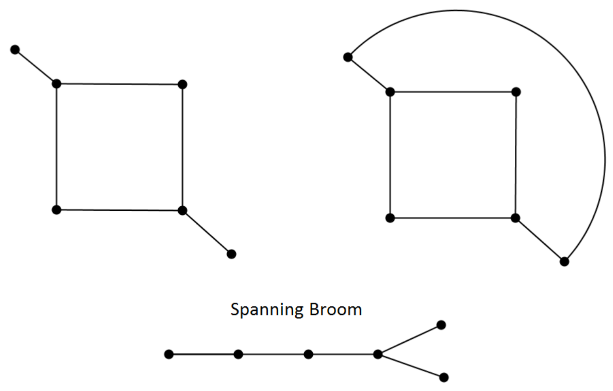

Figure 3.2: Broom

Proof. Construct any DFS spanning broom B from any vertex, $V(B) = \{v_1, \cdots, v_{n-k}, u_1, \cdots, u_k\}$ and $E(B) = \{(v_i, v_{i+1}) | i = 1, 2, \cdots, n-k-1\} \cup \{(v_{n-k}, u_i) | i = 1, 2, \cdots, k.\}$.

Lemma 3.0.11. For every DFS spanning broom B of G

$$(u_i, u_j) \notin E(G)$$

Proof. Let us assume by contradiction $(u_i, u_j) \in E(G)$, then start constructing DFS spanning broom $v_1, v_2, v_3, \cdots v_{n-k}, u_i, u_j \cdots$ we got path of length $n-k+1$ but B_k^{n-k}'s longest path's length is only $n-k$, a contradiction. So $(u_i, u_j) \notin E(G)$. $\square$

Lemma 3.0.12. For every DFS spanning broom B of G

$$d_G(v_1) = 1$$

Proof. If $(v_1, u_i) \in E(G)$, then start constructing DFS spanning broom $u_i, v_1, v_2, v_3, \cdots, v_{n-k}, u_j$ $\forall j \neq i$ we got path of length $n - k + 1$ but B_k^{n-k}'s longest path's length is only $n - k$, hence a contradiction. So $(v_1, u_i) \notin E(G)$. Start constructing DFS spanning broom $v_2, v_3, \cdots, v_{n-k}, u_1$ from Lemma 3.0.11 and $(v_1, u_i) \notin E(G)$, DFS algorithm will backtrack one step and hang all $u_2, u_3, \cdots, u_k$ vertices on v_{n-k}. Now place for v_1 is strictly defined, it should be hanged on vertex v_2, hence DFS algorithm should start backtracking till v_2, it is easy to conclude that

$$d_G(v_1) = 1$$

□

Lemma 3.0.13. For every DFS spanning broom B of G

$$d_G(u_i) = 1$$

Proof. If $\forall j \in \{1, 2, \cdots, k\}$ $(v_2, u_j) \notin E(G)$, then start constructing DFS spanning broom $u_1, v_{n-k}, v_{n-k-1}, \cdots, v_2, v_1$ since $d_G(v_1) = 1$ from Lemma 3.0.12 and from assumption, DFS algorithm will start backtracking and make two backward steps. Hence all u_j should be hung on v_{n-k}. So $d_G(u_i) = 1$ for $i \neq 1$. With same argument, we can prove the same for u_1 just we need to start constructing DFS spanning broom from vertex u_2.

If $\exists j \in \{1, 2, \cdots, k\}$ for which $(v_2, u_j) \in E(G)$ then we distinguish three cases for $n - k$

Case 1: $n - k > 4$

If $n-k \geq 5$ then start constructing DFS spanning broom $v_4, v_3, v_2, u_j, v_{n-k} \cdots$ since $d_G(v_1) = 1$ from Lemma 3.0.12, v_1 vertex finally will be hanged on v_2

a contradiction ($d_{Broom}(v_2) > 2$ and in vertex v_2 starts two disjoint paths of length more then one)

Case 2: $n - k = 4$

If $n - k = 4$ then $m > 2$ because of problem hypothesis. Without loss of generality let us assume that $j = 1$ and start constructing DFS spanning broom u_2, v_4, u_1, v_2, v_1 since $d_G(v_1) = 1$ from Lemma 3.0.12, DFS algorithm will backtrack with one step and hung v_3 on vertex v_2, so all u_i where $i > 2$ should be hung on vertex v_2. So we have edges $(v_2, u_i) \in E(G)$ where $i > 2$. With same argument we can prove that $(v_2, u_2) \in E(G)$ so we have

$$(v_2, u_i) \in E(G)\ \forall i$$

Let's start constructing DFS spanning broom, u_1, v_2, v_1 since $d_G(v_1) = 1$ from Lemma 3.0.12, DFS algorithm will backtrack with one step and continue from vertex v_2. v_2, u_2, v_4, v_3 but this is a contradiction because degree of vertex v_2 in spanning tree is 3 already but all u_i's $i > 2$ will be hung on vertices v_4 or v_3.

Case 3: $n - k < 3$

If $n - k \leq 3 \implies n - k = 3$ then the length of longest path in broom is equal to 3 but if we start constructing DFS spanning tree $v_1, v_2, u_j, v_3, u_{i\neq j}$ we will get path of length, 4 a contradiction.

$\square$

We have proved that $d_G(v_1) = 1$ and $d_G(u_i) = 1$ and vertex v_1 is hanging on vertex v_2 and vertices u_i are hanging on vertex v_{n-k}, so DFS spanning tree of a graph G', induced subgraph of G on vertices $\{v_1, \cdots, v_{n-k}\}$, should be path with starting and ending vertices v_1 and v_{n-k}, from Theorem 3.0.3 we know all graphs with all DFS spanning trees paths, so from checking those graphs, we will get G' is a path. Hence the theorem is proved.

■

Our question for DFS trees will be very different from problem conjecture 3.0.1, if we consider rooted DFS spanning trees. Recall a definition,

Definition 3.0.1. Two rooted trees (T_1, v_1) and (T_2, v_2) are root-isomorphic, denoted by $(T_1, v_1) \simeq (T_1, v_1)$, if there exists a bijection ϕ between $V(T_1)$ and $V(T_2)$ such that $\phi(v_1) = v_2$ and any two vertices $v, u \in V(T_1)$ are adjacent in T_1 if and only if $\phi(v)$ and $\phi(u)$ are adjacent in T_2.

Theorem 3.0.6. Simple, connected graphs with one isomorphic class of rooted DFS spanning trees are

- C_n
- K_n
- $K_{\frac{n}{2},\frac{n}{2}}$

Proof. Assume G is a graph with all rooted DFS trees isomorphic to (T, v_0) (This rooted tree is one of rooted DFS tree of G). Find longest path of T starting in vertex v_0

$$P = (v_0, v_1, v_2, \cdots, v_m)$$

Length of longest path $m+1$ is invariant for all trees. If $\exists u \in V(G) - V(P)$ and $(v_0, u) \in E(G)$ then start constructing DFS spanning tree from vertex u and then take path P from vertex v_0, new tree will have different length longest path started in the root (length will be $m+2$).$\Longrightarrow$ there is no such u and $d_T(v_0) = 1$.

Let us find minimum $k < m$ such that $d_T(v_k) > 2$ then $\exists u \in V(G) - V(P)$ such that $(u, v_k) \in T(G)$ (This k is invariant for all rooted DFS trees of G too). Start constructing DFS spanning tree from vertex v_1 taking path P

towards u, ($P = (v_1, v_2, \cdots, v_k, u)$). From DFS tree property the only vertices connected with u or it's descendants can be $v_0, v_1, \cdots, v_k$ or u's descendants. After tracing all descendants of u DFS tree will start backtracking and will come back to u. $v_1, \cdots, v_k$ and u's all descendants are visited already and $(v_0, u) \notin E(G)$. Hence DFS algorithm will continue backtracking and on v_k will hung v_{k+1} and u these means minimum $t < n$ such that $d_{T'}(v'_k) > 2$ is different from k which is contradiction. $\Longrightarrow \mathbf{T} = \mathbf{P}$ (T is a path) .

We have just proved that all rooted DFS Spanning trees are rooted paths hence solution of this problem is subset of solution of theorem 3.0.3. After checking P_n, C_n, K_n, $K_{\frac{n}{2},\frac{n}{2}}$ graphs we will get solution.

Remark: If $n = 2$ then P_2 is also solution but $P_2 = K_2$

■

Open Problem 3.0.1. Find all graph G and and a vertex $v \in V(G)$ for which all DFS(T, v) trees are isomorphic to each other.

Open Problem 3.0.2. Find all graph G and set $S \subseteq V(G)$ for which DFS(T, v) are in same isomorphism class, for $\forall v \in S$

Remark 4. If we can solve open problem 3.0.1, then we can solve 3.0.2 and vice versa.

Proof. Let a graph G with n vertices and a set $S \subset V(G)$ be a solution of Open Problem 3.0.2. We can produce solution for Open Problem 3.0.1, graph G' with $n+1$ vertices and a vertex v . Add a vertex v to the vertex set of G and add edges (v, v') where $v' \in S$. $V(G') = V(G) \cup \{v\}$ and $E(G') = E(G) \cup \{(v, v') : v' \in S\}$.$G'$ is a solution of Open Problem 3.0.1

Similarly if we have a graph G with n vertices and a vertex v, a solution of Open Problem 3.0.1, we can produce solution for Open Problem 3.0.2, graph G' with $n-1$ vertices and a set S . Delete a vertex v from vertex set of G and take S all neighbors of v. Formally $S = \{v' : (v, v') \in E(G)\}$, $V(G') = V(G) - v$ and $E(G') = E(G) - \{(v, v') : (v, v') \in E(G)\}$. New graph G' with set S is a solution of Open Problem 3.0.2.

From the last two paragraphs we can conclude that there is bijection between solutions. And bijection function is defined well. All solutions of Open Problem 3.0.2 can be produced from solutions of Open Problem 3.0.1 and vice versa.

■

We couldn't solve above mentioned open problems, but we found several interesting solutions and patterns.

On the Figure 3.3 you can see a solution of Open Problem 3.0.2. The set of roots is vertices labeled with stars.(remark: the set of roots can be any subset of that set also).

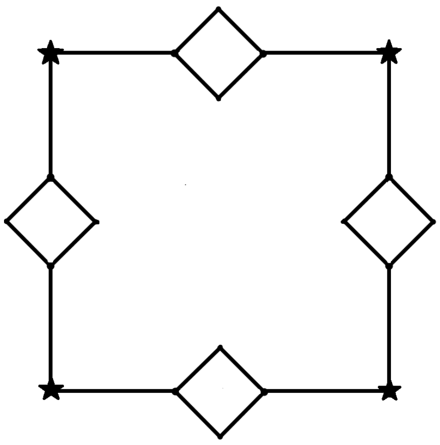

Figure 3.3: Solution of f Open Problem 3.0.2

On the Figure 3.4 you can see a solution of Open Problem 3.0.2. The set of roots is vertices labeled with big black dots or vertices labeled with "X".(remark: the set of roots can be any subset of those set also).

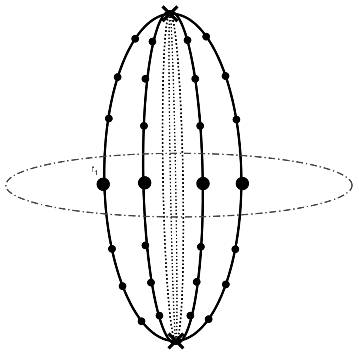

Figure 3.4: Solution of f Open Problem 3.0.2

On the Figure 3.5 you can see a solution of Open Problem 3.0.1. The root is a vertex labeled with big black dot. This solution is derived from solution on the Figure 3.3. From, this we can see pattern how to construct solution of Open Problem 3.0.1 from solution of Open Problem 3.0.2.

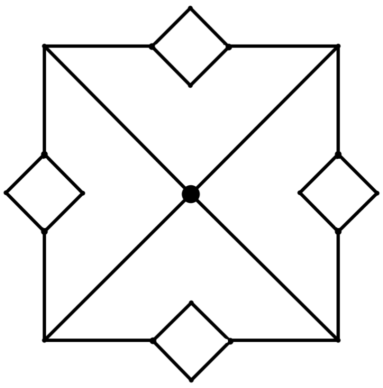

Figure 3.5: Solution of f Open Problem 3.0.1

On the Figure 3.6 you can see a solution of Open Problem 3.0.1. The root is a vertex labeled with black star. This solution is derived from solutions on the Figure 3.3 and the Figure 3.4. From, this solution we can see pattern how to construct bigger solutions from already know solutions for Open Problem 3.0.1

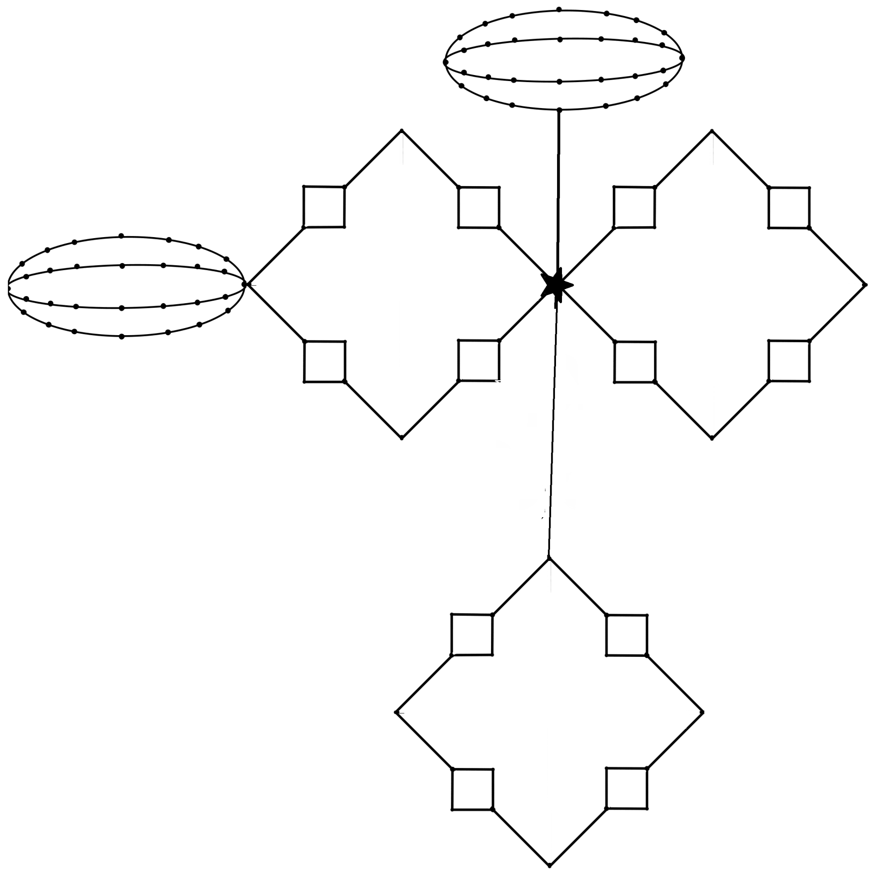

Figure 3.6: Solution of f Open Problem 3.0.1

Bibliography

[1] P.D. Vestergaard, *Finite and Infinite Graphs Whose Spanning Trees are Pairwise Isomorphic.* Annals of Discrete mathematics 41 (1989) 421-436. Elsevier Science Publishers B.V. (North-Holland)

[2] R. Fischer, *Uber Graphen mit Isomorphen Geuisten.* Monatsh. f. Math. 77 (1973), 24-30.

[3] L. Friess, *Graphen, worin je zwei Geruste Isomorph sind.* Math. Ann. 204 (1973), 65-71.

[4] B. L. Hartnell, *On Graphs with exactly two Iomorphism Classes of Spanning Trees.* Utilitas Mathematica 6 (1974), 121-137.

[5] B. L. Hartnell, *The Characterization of those Graphs whose Spanning Trees can be Partitioned into Two Isomorphism Classes.* Ph.D. Thesis, Faculty of Mathematics, University of Waterloo (1974).

[6] K. Wagner, *Graphentheorie.* Bibliographisches Institut, Mannheim (1970).

[7] B. Zelinka, *Grafu, jejichz vsechny kostry jsou spolu isomorfni, "Graphs, all of whose Spanning Trees are Isomorphic to each other."* Cas.pro Pest. Mat. 06 (1971), 33-40.

[8] J. Hopcroft, R.Tarjan, "Algorithm 447: Efficient Algorithms for Graph Manipulation" Comm. ACM, Vol. 16, 1973, pp. 372-378.

[9] R.Tarjan, "Depth First Search and linear Graph Algorithms", SIAM J.Comput, Vol. 1, 1972, pp. 146-160

[10] E. Lucas, "Recreations Mathematiques", Paris, 1882

[11] G. Tarry,"Le Probleme des Labyrinthes ", Nouvelles Ann. de Math., Vol. 14, 1895, page 187.

[12] S. Even, "Graph Algorithms", Pitman 1979.