A Linear-Time Algorithm for the Feasibility of Pebble Motion on Trees

Article *in* Algorithmica · March 1999

DOI: 10.1007/PL00009259 · Source: DBLP

4 authors, including:

Vincenzo Auletta
Università degli Studi di Salerno
88 PUBLICATIONS **1,039** CITATIONS
SEE PROFILE

Angelo Monti
Sapienza University of Rome
80 PUBLICATIONS **1,377** CITATIONS
SEE PROFILE

Mimmo Parente
Università degli Studi di Salerno
78 PUBLICATIONS **743** CITATIONS
SEE PROFILE

A Linear Time Algorithm for the Feasibility of Pebble Motion on Trees

Vincenzo Auletta[*] Angelo Monti[†] Mimmo Parente[*] Pino Persiano[*]

January 5, 1998

Contact Author:
Mimmo Parente
Dipartimento di Informatica ed Applicazioni
Università di Salerno
84081 Baronissi (ITALY)
e-mail: parente@dia.unisa.it
tel: +39-89-965244
tel: +39-89-965272

[*]Dipartimento di Informatica ed Applicazioni, "R.M. Capocelli", Università di Salerno, 84081 Baronissi (Italy). e-mail: {auletta,parente,giuper}@dia.unisa.it.

[†]Dipartimento di Scienze dell'Informazione, Università di Roma 1,"La Sapienza", Roma (Italy). e-mail: monti@dsi.uniroma1.it.

Abstract

We consider the following pebble motion problem. We are given a tree T with n vertices and two arrangements $\mathcal{R}$ and $\mathcal{S}$ of $k < n$ distinct pebbles numbered $1, \cdots, k$ on distinct vertices of the tree. Pebbles can move along edges of T provided that at any given time at most one pebble is travelling along an edge and each vertex of T contains at most one pebble. We are asked the following question:

Is arrangement $\mathcal{S}$ reachable from $\mathcal{R}$?

We present an algorithm that on input two arrangements of k pebbles on a tree with n vertices decides in time $O(n)$ whether the two arrangements are reachable from one another. We also give an algorithm that, on input two reachable configurations, return a sequence of moves that transforms one configuration into the other.

The pebble motion problem on trees has various applications including memory management in distributed systems, robot motion planning, and deflection routing.

1 Introduction

In this paper we consider the following pebble motion problem which we call the *Pebble Motion problem on a Tree* (the PMT problem, in short). Let T be a tree with n vertices and with $k < n$ distinct pebbles numbered $1, \cdots, k$ on distinct vertices. A move consists in transferring a pebble from its current position to an adjacent unoccupied vertex. We ask the following question: Is a given arrangement of pebbles reachable from another by executing a sequence of moves?

The main contribution of this paper is a *linear-time* algorithm for deciding whether two arrangements of pebbles on a tree are reachable from one another.

The PMT problem is, in a broad sense, a generalization of the famous "15 puzzle" due to Sam Loyd [4] and has applications to several fields. We can think of the pebbles as indivisible packets of data that need to be moved from one site to another of a wide-area communication network without exceeding the capacities of the communication buffers of each site. Another application of the PMT problem is in the field of motion planning of independent robots. Here, each pebble represents a robot that has to be moved from one vertex to another of a system of tracks.

Due to its wide applicability, the problem of pebble motion has been studied in various contexts. Kornhauser, Miller, and Spirakis [3] proved that the decision problem for general graphs is decidable in polynomial time improving on an early work of Wilson [8] that considered only biconnected graph and the case $k = n-1$. Recently, Papadimitriou *et al.* [5], motivated by problems in robot motion planning, considered the following problem. There are a distinguished pebble (called the robot) that has to be moved from its position to a destination position and $k - 1$ pebbles (called the obstacles) to which no destination is assigned and may end-up in any vertex of the graph. It is easy to see that this problem has always a solution on biconnected graphs, and in [5] a simple feasibility criteria for trees was given. Several optimality problems (i.e., where we ask the shortest sequence of moves to reach one arrangement from another) relative to pebble motion problem have also been studied and are believed to be computationally intractable. Goldreich proved that computing the shortest sequence of moves that solves an instance of a generalization of Loyd's famous 15-puzzle (this corresponds to the case in which $k = n - 1$) is NP-complete. This was later improved to the case in which the graph is a grid by Ratner and Warmuth [6]. The negative results about the optimality versions of pebble problems provide further motivations to the study of the feasibility problem. Indeed, one would like to know whether there exists a solution to a given pebble motion problem before embarking in the costly computation of the optimal sequence of moves. Moreover, it is often the case that algorithms to test feasibility of pebble problems can be modified so to give in output a legal (although sub-optimal) sequence of moves.

The feasibility problem for the PMT on general graphs has been studied by Kornhauser *et al.* [3]. There, it is shown that the feasibility problem for PMT can be solved in polynomial time, even though no accurate analysis of the running time was provided. The search for a faster algorithm for this important problem was the main motivation of our research. This paper gives the first linear (and thus optimal) algorithm for a non trivial class of graphs. Moreover we remark that in several applications (for example, movement of data in a network or the planning of robot moves) it is likely that the underlining network (the backbone of a wide area network or a set of tracks on which the robots move) is indeed a tree.

The next theorems present the main results of this paper.

Theorem 1 (Informal statement) *It is possible to decide in time $O(n)$ if a given arrangement of pebbles on a tree with n vertices can be transformed into another arrangement.*

Theorem 2 (Informal statement) *It is possible to compute a sequence of pebble moves that transforms one arrangement of pebbles into another on a tree with n vertices in time $O(n+l)$, where l is the length of the sequence of moves given in output.*

We next give an informal description of the linear-time algorithm for deciding feasibility of the PMT problem. We divide the presentation of the algorithm into two main conceptual steps.

1. **Reducing to permutations.** In Section 3, we show that the feasibility of the PMT problem is equivalent to the feasibility problem of the *Pebble Permutation problem on a Tree* (the PPT problem). In the PPT problem, each pebble has to be moved to the original position of another pebble. In other words, the pebbles have to be permuted among themselves and thus an instance J of the PPT problem is simply a permutation.

   The reduction consists in a linear time algorithm that on input an instance I of the PMT problem outputs an instance J of the PPT problem which is feasible if and only if I is feasible.

2. **Reducing to exchanges.** In Section 4, we reduce the feasibility problem for the PPT problem to the problem of feasibility of exchanges between pairs of pebbles. Indeed we have that a permutation Π (representing an instance of the PPT problem) is feasible if and only if the exchanges $(i, \Pi(i))$ are feasible for $i = 1, \cdots, k$. An exchange (i, j) consists in exchanging the positions of the i-th and j-th pebble and moving back to their original position all other pebbles which have been moved for realizing the exchange.

   In Section 5, we give a linear-time algorithm for simultaneously checking that, for a given permutation Π, all exchanges are feasible.

Extensions. In Section 6 we discuss some extensions of our main result that we briefly review here.

Our algorithm also provides a sequence of moves (a plan) that solves the input instance in case this is feasible. The plan has length $O(k^2(n-k))$. For the classical case of $k = n-1$ (corresponding to the extension to trees of the "15"-puzzle), we show that all feasible instances can be solved by a plan with $O(n)$ moves.

2 Basic definitions and notation

The *Pebble Motion problem on a Tree* (in short the PMT problem) consists in deciding whether there exists a sequence of moves on a given tree T that takes each of k pebbles numbered $1, 2, \cdots, k$ from its source vertex to its destination vertex. No two pebbles start from the same source vertex. A *move* consists in taking a pebble from its current position to one of the adjacent vertices that is not occupied by another pebble. Thus, at any given time, each vertex of T hosts at most one pebble. A vertex hosting a pebble is said to be *occupied* while a vertex not hosting any pebble is said to be a *hole*.

We define a *configuration* on T as a sequence $\mathcal{U} = \langle u_1, u_2, \ldots u_k \rangle$ of vertices where u_i is the vertex hosting the i-th pebble. An *instance* of the PMT is described by a tree T and two configurations: the starting (or source) configuration $\mathcal{S} = \langle s_1, \cdots, s_k \rangle$ and the final (or destination) configuration $\mathcal{D} = \langle d_1, \cdots, d_k \rangle$.

A *plan* P is a sequence of moves. A plan P of l moves applied to a configuration $\mathcal{U}$ results in a series of configurations $\mathcal{U}^0, \cdots, \mathcal{U}^l$, where $\mathcal{U}^0 = \mathcal{U}$ and configurations $\mathcal{U}^i$ and $\mathcal{U}^{i+1}$ differ exactly for the position of the pebble that has been moved by the i-th move of P. We say that a plan P solves, or realizes, an instance $I = (T, \mathcal{S}, \mathcal{D})$ if applying P to $\mathcal{S}$ we obtain a sequence of configurations leading to $\mathcal{D}$.

An instance is *feasible* if there exists a plan that realizes it. The *feasibility problem* of PMT consists in deciding if for a given instance I there exists a plan solving I.

Given a move m which takes a pebble from vertex v_1 to vertex v_2, define $rev(m)$ as the move which takes a pebble from v_2 to v_1. Given a sequence of moves $H = (m_1, m_2, \ldots, m_h)$, define $rev(H)$ as the sequence $(rev(m_h), \ldots, rev(m_2), rev(m_1))$.

From now on we let T denote a given tree with n vertices and let k denote the number of pebbles. Without loss of generality, we assume that T is a rooted tree and there is a pebble in the root vertex. For each vertex u we denote by T_u the subtree of T rooted in u and by $F(u)$ the forest obtained from T by deleting u. Moreover, for any forest F and any vertex v belonging to F we denote by $T(F, v)$ the tree of F containing v and by $C(F, v)$ the set of the remaining trees of F (see Figure 1). We let $u \rightsquigarrow v$ consist of the internal vertices of the unique path in T from u to v (i.e., all the vertices of the path from u to v excluding u and v).

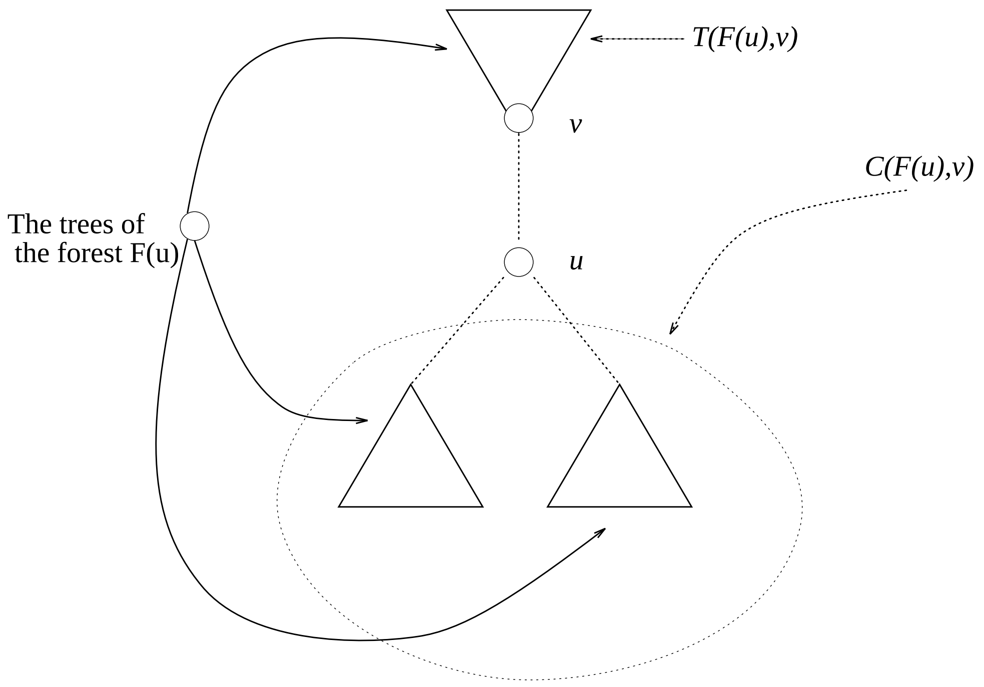

Figure 1: The forests $F(u)$, $C(F(u), v)$, and the tree $T(F(u), v)$.

In the rest of the paper, we use uppercase letters for configurations of pebbles and, given a configuration $\mathcal{U}$, we denote the i–th element of $\mathcal{U}$ by u_i, that is the position of pebble i in $\mathcal{U}$.

2.1 The 1 pebble problem

In this Section we discuss a related pebble problem that we call the *1-pebble problem* and prove a simple property of plans solving this problem that will be used in the proof of Lemma 6. This property is a down-sized version of a lemma appearing without proof in [5]. For sake of self containment we give a proof of the lemma.

An instance of the 1-pebble problem consists in a configuration of k pebbles, each located in a vertex of a tree, such that $k-1$ pebbles are identical and one is distinguished: it is required to move the distinguished pebble from its initial position s to a given vertex t. No restriction is imposed on the final destination of the remaining pebbles. Thus an instance of the 1 Pebble problem is described by a triple $(\mathcal{U}, s, t)$.

For a plan P that solves the instance $(\mathcal{U}, s, t)$ of the 1 Pebble problem, we denote by R the set of vertices visited by the distinguished pebble and denote by $\overline{R}$ the set of the remaining vertices of the tree.

Definition 1 *Let P be a plan that solves the instance $(\mathcal{U}, s, t)$ of the 1 Pebble problem. We say that P is a* simple plan *if and only if the following conditions hold:*

1. *P does not contain any move from vertices of $\overline{R}$ to vertices of R;*
2. *the distinguished pebble visits in $T(F(s), t)$ only vertices in $s \rightsquigarrow t$ or adjacent to these;*
3. *each move from a vertex of $R \cap T(F(s), t)$ to a vertex of $\overline{R} \cap T(F(s), t)$ occurs before the distinguished pebble starts moving;*
4. *P does not contain any sequence of moves that take a pebble from a vertex in $C(F(s), t)$ to a vertex of $T(F(s), t)$.*

Next Lemma shows that for each instance of the 1-Pebble problem there exists a simple plan that solves it.

Lemma 1 *Each feasible instance $H = (\mathcal{U}, s, t)$ of the 1-Pebble problem admits a simple plan.*

Proof: Let Q be a plan that solves the instance H and assume that Q is not a simple plan. We show how to modify Q so to obtain a simple plan P that solves H.

We start by considering all the moves from a vertex $w \in \overline{R}$ to vertex $z \in R$. If no other move of the plan takes a pebble to w then the move from w to z is deleted. Suppose, instead, that w is subsequently occupied by a pebble moved from w' to w. If w' belongs to R then, since w and w' are adjacent, it must be the case that $w' = z$. Then, the two moves from w to z and back are deleted from Q. Instead if w' belongs to $\overline{R}$ we have to distinguish two cases. If, after this move, w' remains empty then the moves from w to z and from w' to w are deleted from Q. If w' is then occupied by a pebble then this pebble must have been moved from a vertex of $\overline{R}$ as all vertices adjacent to w' belong to $\overline{R}$ and the same argument is iterated until we get a vertex that remains empty. After deleting all these moves we have a plan that moves the distinguished pebble from s to t and satisfies the first property of the simple plans.

To ensure the second property we observe that, obviously, all the vertices of $s \rightsquigarrow t$ are in R. However, it is possible that the distinguished pebble visits vertices outside of $s \rightsquigarrow t$. This happens to let some pebble that is ahead of the distinguished pebble to slide behind the distinguished pebble. Observe though that for this it is sufficient that the distinguished pebble moves to a vertex adjacent to $s \rightsquigarrow t$ (or to a vertex of $C(F(s), t)$ with degree at least 3) and let the other pebbles pass along the path from s to t. Therefore, we can delete from Q all the moves of the distinguished pebble to vertices of $T(F(s), t)$ that do not belong to $s \rightsquigarrow t$ nor are adjacent to this path. Now we have constructed a plan that still moves the distinguished pebble from s to t but enjoys the first two properties of a simple plan.

Suppose, now, that Q moves l pebbles originally placed at vertices of $R \cap T(F(s), t)$ to l vertices of $\overline{R} \cap T(F(s), t)$. We notice that none of the paths connecting the origins and the destinations of these sequences contain vertex s. Then, it is possible to obtain a new plan that contains l sequences of moves, each taking a pebble from a vertex of $R \cap T(F(s), t)$ to a vertex of $\overline{R} \cap T(F(s), t)$ that is not occupied in the starting configuration. Moreover, these sequences are executed at the beginning of the plan, before the distinguished pebble starts moving. The other pebbles on R, instead, are moved to the same places in $R \cup C(F(s), t)$ where pebbles were moved by Q.

To prove the last property consider vertices $v_1, v_2 \in C(F(s), t)$ and $w_1, w_2 \in T(F(s), t)$ such that Q contains two sequences of moves from v_1 to w_1 and from w_2 to v_2, respectively. We construct a new plan that contains a sequence of moves from v_1 to v_2 and a sequence from w_2 to w_1. All the other pebbles are moved as in Q. We observe that both the paths from v_1 to v_2 and from w_2 to w_1 do not contain vertex s and thus the sequences of moves can be executed at the beginning of the plan, before the distinguished pebble starts moving. We notice that this plan is still a feasible plan for H. In fact, the net effect of the two sequences is to reach a configuration where two pebbles are in v_2 and w_1 while v_1 and w_2 are unoccupied. This configuration is equal to the configuration reached by Q except for a permutation of the pebbles. However, as the pebbles are indistinguishable, we can say that the new plan solves the same instances that Q solves.

Suppose, now, that Q contains no sequence of moves that take a pebble from $T(F(s), t)$ to $C(F(s), t)$. In this case it is possible to obtain a new plan for H where the distinguished pebble does not visit any vertex of $C(F(s), t)$ (remember that the distinguished pebble has to visit vertices of $C(F(s), t)$ only if some pebbles must be moved out of $T(F(s), t)$). By the first property, as in this new plan the distinguished pebble does not enter in $C(F(s), t)$ we obtain that there is no sequence of moves in the plan that take a pebble from $C(F(s), t)$ to $T(F(s), t)$. ■

3 The Pebble Permutation Problem: algorithm *Reduce*.

In this section we present a linear time reduction algorithm *Reduce* from the PMT problem to the simpler *Pebble Permutation Problem on a Tree.*

The *Pebble Permutation Problem on a Tree* (PPT problem, in short) is a special case of the PMT problem where the set of source vertices and the set of destination vertices coincide. Thus the problem consists in permuting the pebbles and an instance of the PPT problem is described by a tree T, a starting configuration $\mathcal{S} = \langle s_1, \cdots, s_k \rangle$ and a permutation Π on k objects. The destination of the i-th pebble that starts at s_i is the vertex $d_i = s_{\Pi(i)}$. In the following when T and $\mathcal{S}$ are clear from the context, we will speak of a plan of moves which realizes or solves the permutation Π, meaning that it solves the instance $(T, \mathcal{S}, \Pi)$.

Given an instance $I = (T, \mathcal{S}, \mathcal{D})$ of the PMT problem, the algorithm *Reduce* returns an instance $J = (T, \mathcal{S}', \Pi)$ of the PPT problem such that J is feasible if and only if I is feasible. The computation of algorithm *Reduce* can be divided into two main phases:

Phase I. Construct a feasible instance $I' = (T, \mathcal{S}, \langle d_{i_1}, \cdots, d_{i_k} \rangle)$ of the PMT problem.

Phase II. Let Π be the permutation defined as $\Pi(j) = l$ if and only if $d_{i_j} = d_l$. Return the instance $J = (T, \langle d_{i_1}, \cdots, d_{i_k} \rangle, \Pi)$ of the PPT problem.

We now prove that, provided that I' is feasible, instance J computed by *Reduce* is feasible if and only if I is feasible. In the next subsection we show an algorithms for computing I', prove that it takes $O(n)$ time and that the instance I' produced is indeed feasible.

Lemma 2 *Let I, I', and J be as above. Then, I is feasible if and only if J is feasible.*

Proof: Let P be a plan for I'. If the instance J of the PPT problem is feasible then a plan for the instance I of the PMT problem is obtained by concatenating P and the plan for J. On the other side, if I is feasible, then a plan for J is given by inverting the moves of P and executing them in reverse order and then by executing the plan for I. ■

3.1 Phase I of Algorithm *Reduce*

The algorithm computes a matching $((s_1, d_{i_1}), \cdots, (s_k, d_{i_k}))$ of source and destination vertices of I that naturally defines the instance I'.

The matching is computed in a bottom-up way starting from the leaves of T. To this aim, the algorithm constructs two lists for each vertex v of the tree: a list of source vertices O_v and a list of destination vertices D_v. Roughly speaking, the list O_v contains two kinds of elements: source vertices that belong to T_v and to which no destination has been assigned yet; and pairs (u_1, u_2) of vertices belonging to T_v and constituting a match of one source and one destination vertex. Instead, D_v is the list of all the destination vertices of the subtree T_v to which no source has been assigned yet. In proving the correctness of the algorithm, we will use terms as front and tail of the list and expression as "vertex v is ahead of vertex u" or "vertex v is behind vertex u" with the obvious meaning. The algorithm uses a global counter ℓ to assign a label to each pair of the matching; the counter is not actually needed to compute the matching between source and destination vertices, but it is used to show that the resulting instance I' is feasible.

The lists relative to a leaf v are computed as follows. If v is both a source and a destination vertex, then the algorithm produces the triple (v, v, ℓ), increments the counter ℓ, and sets O_v and D_v equal to the empty list. If v is neither a source nor a destination then, again, both O_v and D_v are set equal to the empty list and no triplet is produced. Finally, if v is a source (resp., a destination) the list O_v (resp., D_v) contains v, the list D_v (resp., O_v) is empty, and no triplet is produced. Notice that, for the leaf v, at most one of the two lists is nonempty and, as we shall see, this invariant is maintained throughout the execution of the algorithm.

Now, consider an internal vertex v, let $v_1, \cdots, v_m$ be its children, and let O_{v_i} and D_{v_i} be the lists relative to vertex v_i, for $i = 1, \cdots, m$. Call O the list obtained by concatenating the lists O_{v_i} and D the list obtained by concatenating the lists D_{v_i}.

Consider the front of the list O.

1. If the front of O consists of a single vertex u_1, then the algorithm extracts u_1 and the vertex u_2 at the front of D, produces the triplet (u_1, u_2, ℓ), and increments ℓ.

2. If the front of O is a pair (u_1, u_2) then the algorithm produces the triplet (u_1, u_2, ℓ) and increments ℓ;

Figure 2: The steps of the main loop of *Reduce*.

If v is a source vertex then v is inserted in front of the list O. If v is a destination vertex then we have to distinguish two cases.

Case I. If O contains at least one single vertex, then the closest such vertex to the tail of O, call it u, is replaced with the pair (u, v).

Case II. If the list O contains no single elements, then v is appended to the list D.

Once O and D have been constructed, the algorithm executes the two steps of the main loop (see Figure 2) until O is empty or its front element is a single vertex and D is empty. At the end of the loop O_v is set equal to O and D_v is set equal to D.

First of all, we state the following easy to prove properties of the algorithm.

Lemma 3 *For each vertex v let O and D be the lists constructed by the algorithm Reduce for v. The following properties hold:*

1. *For each source vertex u of O, it holds that all the source vertices along $u \rightsquigarrow v$ that belong to O are ahead of u.*

2. *For each destination vertex u of D, it holds that all the destination vertices of $u \rightsquigarrow v$ that belong to D are behind u.*

3. *At most one of the lists O_v and D_v is non empty.*

In other words, property 1 above says that when a source vertex u gets to the front of O, then no other source vertex lying on $u \rightsquigarrow v$ is still in O. Property 2 instead states that when a destination vertex u gets to the front of D, then no other destination vertex descendant of u is still in D.

To show that I' is feasible, we consider the plan *Plan* obtained by taking the triplets computed from the reduction algorithm in the order specified by the counter field and executing for each triplet (u_1, u_2, ℓ) the moves for moving a pebble from the source vertex u_1 to the destination vertex u_2. We claim that *Plan* realizes I'. The proof of this claim is based on the following two lemmas.

Lemma 4 *Let v be a vertex of T and suppose that the pair (u_1, u_2) is at the front of O. Then each source vertex s of T_{u_2} that is in O is part of a pair (s, d) and u_2 does not belong to $s \rightsquigarrow d$.*

Proof: Each source vertex s of T_{u_2} that is behind (u_1, u_2) in O must be part of a pair, for otherwise the algorithm would have made a pair consisting of s and u_2. As the pair (s, d) has been created before the pair (u_1, u_2), it must be the case that s and d are descendants of the same child of u_2 and thus u_2 does not lie on $s \rightsquigarrow d$. ■

The correctness of the plan follows from the following property of the triplets computed by *Reduce*.

Lemma 5 *For each triplet (u_1, u_2, ℓ) computed by Reduce it holds that all source vertices on $u_1 \rightsquigarrow u_2$ belong to a triplet with a counter field less than ℓ and all destination vertices on $u_1 \rightsquigarrow u_2$ belong to a triplet with a counter field greater than ℓ.*

Proof: We first prove that all the source vertices on $u_1 \rightsquigarrow u_2$ belong to triplets with a counter field less than ℓ.

Let v be the vertex that the algorithm *Reduce* is visiting when the triplet (u_1, u_2, ℓ) is output and let O and D the lists constructed by the algorithm for v. We notice that if a source vertex s of T_v is not in O at the moment (u_1, u_2, ℓ) is produced, then s belongs to one of the triplets already output and this triplet has a counter field less than ℓ.

We have to distinguish two cases depending on whether the triplet (u_1, u_2, ℓ) is produced at step 1 or at step 2 of the main loop.

Suppose that (u_1, u_2, ℓ) is produced during step 1 of the main loop. Then, by Property 3 of Lemma 3, u_1 and u_2 come from two different subtrees, T_1 and T_2, rooted at the children v_1 and v_2 of v and D_{v_1} and O_{v_2} are empty. This means that no source vertex of T_2 is in O and thus all source vertices of T_2 belong to triplets with a counter field less than ℓ. Moreover, by Property 1 of Lemma 3, when u_1 is extracted from O all the source vertices in $u_1 \rightsquigarrow v$ are not in O. Thus we obtain that all the source vertices belonging to $u_1 \rightsquigarrow u_2$ are in triplets with a counter field less than ℓ.

Suppose now that (u_1, u_2, ℓ) is given in output during step 2. In this case u_2 belongs to $u_1 \rightsquigarrow v$. By Property 1 of Lemma 3, at the moment the triplet is produced there is no source vertex belonging to $u_1 \rightsquigarrow v$ in O. Thus, all the source vertices in $u_1 \rightsquigarrow u_2$ are assigned to triplets with a counter field less than ℓ.

To prove the second part of the lemma we show that if the algorithm *Reduce* outputs a triplet (u_1, u_2, ℓ) then, for all the triplets (s, d, k) that will be subsequently produced by *Reduce* (and thus $k > \ell$) it cannot be the case that u_2 belongs to $s \rightsquigarrow d$. We proceed by contradiction and assume that u_2 belongs to $s \rightsquigarrow d$ and thus at least one of s and d must be in T_{u_2}. As before we have to distinguish two cases, depending on whether (u_1, u_2, ℓ) is produced at step 1 or at step 2 of the main loop.

Consider first the case when the triplet is produced at step 1. However, when (u_1, u_2, ℓ) is produced there is no source vertex of T_{u_2} in O and, since u_2 is the front element of D, by Property 2 of Lemma 3 there is no other destination vertex of T_{u_2} in D. Thus, no such pair (s, d) exists.

Consider, now, the case when the triplet is produced at step 2. In this case u_2 belongs to $u_1 \rightsquigarrow v$. By Property 3 of Lemma 3, since O_{u_2} contains the pair (u_1, u_2), then D_{u_2} is empty. Therefore, no destination vertex of T_{u_2} is in D_{u_2} and thus in D. Moreover, all the destinations of T_{u_2} that are in O are already assigned to a source that belongs to the same subtree. Thus, there is no destination vertex d in T_{u_2} that is in a triplet (s, d, k) where $s \rightsquigarrow d$ contains u_2 and $k > \ell$. The lemma follows by observing that, by Lemma 4, when (u_1, u_2) is extracted from O each source vertex s of T_{u_2} that is in O is matched to a destination vertex d that is a descendant of the same child of u_2 as s. ■

We conclude with the following theorem.

Theorem 3 *Algorithm Reduce, on input an instance I of the PMT problem, computes in time $O(n)$ an instance J of the PPT problem that is feasible if and only if I is feasible.*

Proof: By Lemma 2, to prove that the reduction algorithm works correctly it is sufficient to show that *Plan* realizes the instance I' computed in the phase I of the algorithm. By Lemma 5 when *Plan* moves a pebble from u_1 to u_2 the path R from u_1 to u_2 does not contain any pebble. In fact, all source vertices on R have no pebbles, as they have already been moved, and no pebble has already been moved to a destination vertex on R. Therefore *Plan* is a feasible plan.

An analysis of the running time of the algorithm is derived from the observations that each of the k source and destination vertices is inserted into and removed from the lists exactly once and, aside from insertion and extraction operations, the algorithm spends constant time for each vertex. ■

4 Reducing permutations to exchanges

In this section, unless otherwise specified, T is the tree where the pebbles move and $\mathcal{S}$ is the starting configuration of the pebbles.

We say that a permutation Π is feasible and can be applied to $\mathcal{S}$ meaning that there exists a plan that realizes the instance $(T, \mathcal{S}, \Pi)$ of the PPT problem.

The algorithm for testing feasibility of an instance of the PPT problem is based on the concept of a *good decomposition* of a permutation. It is well known that any permutation can be described as the product of exchanges of two elements. A good decomposition of a permutation Π is a sequence of exchanges whose product realizes Π and such that each exchange brings at least one element to its final position.

More formally, we define the distance *diff*$(\mathcal{U}, \mathcal{V})$ of two configurations $\mathcal{U}$ and $\mathcal{V}$ as the number of pebbles that are at different vertices in $\mathcal{U}$ and $\mathcal{V}$ and have the following definition.

Definition 2 *Let Π be a permutation on k objects, $(\pi_1, \pi_2, \cdots, \pi_l)$ a decomposition of Π into exchanges and, for $j = 1, 2 \cdots, l$, $\mathcal{U}^j$ the configuration obtained by applying the permutation $\Pi_j = \pi_1 \circ \pi_2 \circ \cdots \circ \pi_j$ to the configuration $\mathcal{S}$. We say that $(\pi_1, \pi_2, \cdots, \pi_l)$ is a* good decomposition *of Π if and only if for each j it holds that*

$$\text{diff}(\mathcal{U}^j, \Pi(\mathcal{S})) < \text{diff}(\mathcal{U}^{j-1}, \Pi(\mathcal{S})).$$

We say that a good decomposition $(\pi_1, \pi_2, \cdots, \pi_l)$ of Π is feasible *if and only if for each j the permutation Π_j can be applied to $\mathcal{S}$.*

Two configurations $\mathcal{U}$ and $\mathcal{V}$ are *reachable* from each other if there exists a permutation Π such that $\mathcal{V} = \Pi(\mathcal{U})$ and the instance $(T, \mathcal{U}, \Pi)$ of the PPT problem is feasible. In particular, when Π consists only of the exchange of pebbles i and j we say that vertices u_i and u_j are *equivalent* with respect to $\mathcal{U}$, meaning that there is a plan that starting from the configuration $\mathcal{U}$ exchanges pebbles i and j leaving all the other pebbles in their original positions. In the sequel, we simply say that two vertices are equivalent every time the starting configuration is clear from the context.

We start by proving (Lemma 6 and Corollary 1) that for each feasible instance $I = (T, \mathcal{S}, \Pi)$ of the PPT problem there exists i such that vertex s_i and vertex $s_{\Pi(i)}$ are equivalent. Based on this, in Theorem 4, we prove that I is feasible if and only if in the configuration $\mathcal{S}$ it is possible to exchange the pebbles at vertices s_i and $s_{\Pi(i)}$, for $1 \le i \le k$, while leaving all the other pebbles in their original position. We would like to stress here that when we say that the other pebbles are left in their original positions this does not mean that they are not moved at all, but rather that at the end of the sequence of moves they are back to their original position.

The following property of the trees is easy to prove.

Fact 1 *Let $v_0, v_1, \ldots, v_{l-1}$ be distinct vertices of a tree T and for $i = 0, 1, \ldots, l-1$ let p_i be the path from v_i to $v_{(i+1) \bmod l}$. Then there exists $0 \le i \le l-1$ such that p_i and $p_{(i+1) \bmod k}$ share at least one edge.*

Next Lemma shows that if it is possible to move one pebble from u to v and another from v to w then the two pebbles can be exchanged provided that the two paths have an edge in common. An obvious case is when w is equal to u. Also, it is very easy to see that the lemma holds when u, v and w lie on a path and w is between u and v. On the other hand, it is easy to see that if the two paths do not intersect then the two pebbles cannot necessarily be exchanged: the tree is a path and v is between u and w.

Lemma 6 *Let $\mathcal{U}$ be any configuration and suppose that vertices u, v, w are such that the paths from u to v and from v to w are not edge-disjoint. If there exists a sequence of moves P that moves a pebble from u to v and a pebble from v to w then vertices u and v are equivalent with respect to $\mathcal{U}$.*

Proof: Without loss of generality, call the pebble located at u pebble 1 and the pebble located at v pebble 2. Consider the first occurrence in P of a configuration $\mathcal{S} = \langle s_1, \ldots, s_k \rangle$ such that the path from the position s_1 of pebble 1 to its destination v and the path from the position s_2 of pebble 2 to its destination w are edge disjoint and denote by Q the sequence of moves which reaches configuration $\mathcal{S}$ from $\mathcal{U}$.

The plan P' to exchange pebbles 1 and 2 has the following 3-step structure:

1. execute Q to reach configuration $\mathcal{S}$;
2. execute a sequence of moves to reach a configuration where the pebbles 1 and 2 are exchanged (and all the other pebbles are in the same position as in $\mathcal{S}$);
3. execute $rev(Q)$.

It can be easily seen that P' exchanges pebbles 1 and 2 while all the other pebbles are in their original positions. It remains to show how step 2 is performed.

In the configuration of P just preceding $\mathcal{S}$, at least one pebble, say pebble 2, is at a vertex y of $u \rightsquigarrow v$ and the last move of Q takes pebble 2 from y to s_2. Let z and x be the vertices adjacent to y along $s_1 \rightsquigarrow y$ and $y \rightsquigarrow v$, respectively (see Figure 3). We show now that it is possible to reach from $\mathcal{S}$ a configuration in which pebble 1 is at vertex x, pebble 2 is at vertex s_2 and y and z are holes and then observe that in this configuration vertices s_2 and x are equivalent. Observe that, as the last move of Q takes pebble 2 from y to

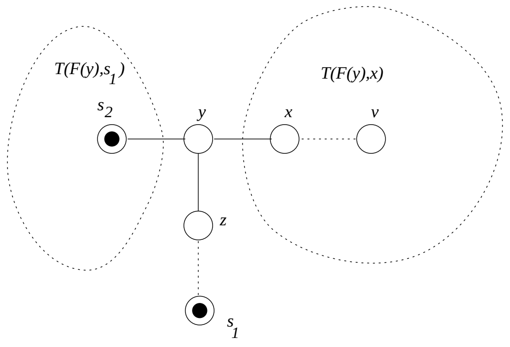

Figure 3: The paths from s_1 to s_2 and from v to s_2.

s_2, in configuration $\mathcal{S}$ vertex y is a hole. Moreover, if in $\mathcal{S}$ there is a pebble at x, then it has been moved to x after pebble 2 has reached y; it follows that in $T(F(y), x)$ there is at least one hole and hence x can be emptied without occupying y. Call $\mathcal{C}$ the configuration reached.

We observe that the problem of moving pebble 1 from s_1 to x can be seen as an instance of the 1 Pebble problem. Thus, by Lemma 1, there exists a simple plan *Simple* that, starting from $\mathcal{C}$, moves pebble 1 from s_1 to x. This is not enough for our purpose as *Simple* might move pebble 2 away from s_2 and move some pebble to z (y is empty when the pebble 1 reaches x). We now show how to modify *Simple* so to ensure that when pebble 1 reaches x pebble 2 is at s_2 and z is empty.

The only reason for pebble 2 to be moved from s_2 is to let pebbles originally at vertices visited by pebble 1 on his trip from s_1 to x enter $T(F(y), s_2)$. Observe though that, by the properties of the simple plans, these pebbles are moved before pebble 1 leaves s_1 for the first time. We can thus modify *Simple* so that pebble 2 is first moved to x, then all moves of pebbles to $T(F(y), s_2)$ are performed and then pebble 2 is moved back to s_2. However, we need to make sure that pebble 1 does not need to visit s_2 as in the modified *Simple* plan s_2 is now occupied. Pebble 1 needs to move to s_2 only if, upon its arrival to y, vertex x is occupied by a pebble (wlog let it be pebble 3) moved to x during the plan, and pebble 3 has to be moved to a vertex previously visited by pebble 1. Therefore, in the original plan *Simple* pebble 2 is moved to a vertex a to let 1 visit s_2. We modify *Simple* by replacing the move of pebble 2 from s_2 to a with the move of pebble 3 from x to a.

Let us now show how to ensure that z is empty when pebble 1 reaches x. If z is not empty, then z must be occupied by a pebble, say pebble 3, that was moved from x to z while 1 was standing at a vertex t adjacent to y other than x, s_2 and z. In this case we modify *Simple* by moving pebble 3 to t while pebble 1 is at z and then moving pebble 1 from z to y and then from y to x. ■

The next corollary proves that for each feasible permutation Π, there exists an i such that the pebbles i and $\Pi(i)$ can be exchanged, leaving all other pebbles at their original position.

Corollary 1 *Let $I = (T, \mathcal{S}, \Pi)$ be a feasible instance of PPT such that Π is not the identity permutation. Then there exists i, $1 \leq i \leq k$, such that $i \neq \Pi(i)$ and the instance $I' = (T, \mathcal{S}, (i, \Pi(i)))$ is feasible.*

Proof: Consider any cycle $(s_{i_0}, \cdots, s_{i_{l-1}})$ of the permutation Π. By Fact 1, there exists $0 \leq j \leq l-1$ such that the paths from s_{i_j} to $s_{i_{j+1}}$ and from $s_{i_{j+1}}$ to $s_{i_{j+2}}$ share at least one edge. From Lemma 6 it follows that s_{i_j} and $s_{i_{j+1}}$ are equivalent. Hence the instance $I' = (T, \mathcal{S}, (i_j, \Pi(i_j)))$ is feasible. ■

Corollary 2 *For each feasible instance $I = (T, \mathcal{S}, \Pi)$, the permutation Π admits a feasible good decomposition $(\pi_1, \pi_2, \cdots, \pi_l)$.*

Proof: We construct a feasible good decomposition of Π by iterating the following process for $i = 0, 1, 2 \cdots$ until we have an instance I_i with Π_i equal to identity permutation. Consider the feasible instance $I_i = (T, \mathcal{U}^i, \Pi_i)$ (initially, $i = 0$, $I_0 = I$ and thus $\mathcal{U}_0 = \mathcal{S}$ and $\Pi_i = \Pi$). By Corollary 1 there exists h_i such that the pebbles at vertices $u^i_{h_i}$ and $u^i_{\Pi_i(h_1)}$ are equivalent. Then set $\pi_i = (h_i, \Pi_i(h_i))$ and let $\mathcal{U}^{i+1}$ be the configuration obtained by exchanging pebbles h_i and $\Pi_i(h_i)$ in $\mathcal{U}^i$. Instance I_{i+1} is defined as $(T, \mathcal{U}^{i+1}, \Pi_{i+1})$, where $\Pi_{i+1} = (\pi_1 \cdots \pi_i)^{-1}\Pi$. I_{i+1} is clearly feasible and $\mathit{diff}(\mathcal{U}^{i+1}, \Pi(\mathcal{S})) < \mathit{diff}(\mathcal{U}^i, \Pi(\mathcal{S}))$. ■

Finally, we can state the main result of the section that reduces the problem of deciding the feasibility of a permutation Π to the problem of deciding the feasibility of each exchange $(i, \Pi(i))$.

Theorem 4 *The instance $I = (T, \mathcal{S}, \Pi)$ is feasible if and only if the exchanges $(i, \Pi(i))$, $i = 1, 2, \cdots, k$ are feasible.*

Proof: **only if.** We observe that the equivalence of two vertices with respect to a configuration does not depend on the identity of the pebbles that occupy the vertices. Thus if two vertices are equivalent with respect to $\mathcal{U}$ they are still equivalent with respect to any configuration that can be obtained from $\mathcal{U}$ by permuting some of the pebbles.

Now let $(i_1, j_1) \cdots (i_{k-1}, j_{k-1})$ be a good decomposition of Π. To show that $(i, \Pi(i))$ is feasible with respect to $\mathcal{S}$ (i.e., that vertices s_i and $s_{\Pi(i)}$ are equivalent with respect to $\mathcal{S}$) we observe that, if $j_h = \Pi(i)$, the plan that executes the following exchanges is feasible by the observation above and exchanges pebble i and pebble $\Pi(i)$

$$(i_1, j_1) \cdots (i_{h-1}, j_{h-1})(i_h, j_h)(i_{h-1}, j_{h-1}) \cdots (i_1, j_1).$$

if. Suppose Π can be decomposed in cycles $C_1, \cdots, C_m$. We show that I is feasible by giving a plan for each of the cycles $C_1, \cdots, C_m$. Suppose $C_1 = (i_0, \cdots, i_{l-1})$. By hypothesis, the exchange $H_j = (i_j, i_{j+1 \bmod l})$ is feasible and denote by P_j a plan for realizing H_j, $j = 0, 1, \cdots, l-1$. Then, as it is easily seen, the cycle C_1 is realized by the plan obtained by concatenating the plans P_j's. The same reasoning applies also to the other cycles. ■

5 The Algorithm for the PPT Problem

In this section we give a linear time algorithm *Mark* that, on input an instance $I = (T, \mathcal{S}, \Pi)$ of the PPT problem, partitions the vertices of $\mathcal{S}$ into equivalence classes with two vertices being equivalent if and only if the corresponding pebbles are exchangeable with respect to $\mathcal{S}$. By Theorem 4, this is enough to decide the feasibility of the instance I. Throughout this section, we will assume that $k \leq n - 2$, for otherwise no two pebbles can be exchanged and I is feasible if and only if Π is the identical permutation.

In the sequel we call a vertex a *branch* vertex if it has degree greater than 2. For each vertex s of $\mathcal{S}$ and for each subtree $R \in F(s)$ we denote by $\mathit{holes}(R)$ the number of holes in R and denote by $d(R)$ the minimum distance in T between s and a branch vertex not belonging to R (possibly s itself). Obviously, if there are no branch vertices outside of R then the tree obtained from T by removing R is a path and $d(R)$ is equal to the length of this line. Moreover, we define $seen(R)$ as the set of vertices v of $\mathcal{S}$ that belong to R and such that $s \rightsquigarrow v$ contains no pebble (see Figure 4). The union of $seen(R)$ for all the subtrees R of $F(s)$ is denoted by $seen(s)$. The values $seen(R), holes(R)$ and $d(R)$, for each vertex s of $\mathcal{S}$ and for each subtree R of $F(s)$ are computed in time $O(n)$ during a preprocessing phase.

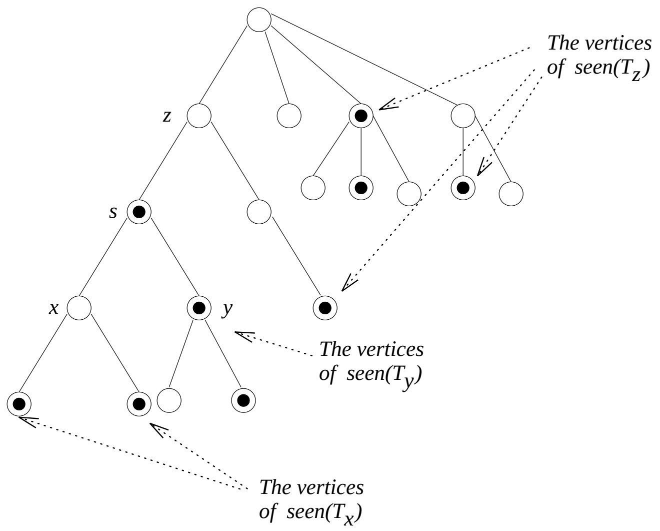
The vertices of seen(T_z)
z
s
x
y
The vertices of seen(T_y)
The vertices of seen(T_x)

Figure 4: The sets that constitute $seen(s)$.

The algorithm *Mark* puts the root vertex in an equivalence class by itself and then traverses the tree by levels. Each time a vertex s of $\mathcal{S}$ is visited, *Mark* assigns each vertex of $seen(s)$ that is also a descendant of s to an equivalence class (notice that when *Mark* arrives at s, the vertices in $seen(s)$ that are not descendant of s and s itself have already been labeled).

Call z the parent of s, T_0 the tree of $F(s)$ containing z and $T_1, T_2, \cdots T_d$ the trees rooted at the children of s. The algorithm *Mark*, given in Figure 5, labels each vertex v of $seen(T_i)$, for $i = 1, 2, \cdots, d$, according to the following rules. All vertices equivalent to s are labelled with $\ell(s)$. Lemma 8 gives a necessary and sufficient condition for a vertex $v \in seen(s)$ to be equivalent to s. This condition can be tested in constant time.

The labelling of the vertices of $seen(s)$ that turn out not to be equivalent to s (we call them the *bad* vertices) is based on the following observations, proved in Lemmas 9–12:

Observation 1. All bad vertices of $seen(T_i)$ are equivalent to each other.

Observation 2. If some vertex of $seen(s)$ is bad, then at most two trees of $F(s)$ are non-full, and at most one of these trees has more than one hole.

Observation 3. Let $u \in seen(T_i)$ and $v \in seen(T_j)$, with $i \neq j$, be two bad vertices. Vertices u and v are equivalent if and only if there exists a tree T_h, $h \neq i, j$ such that $holes(T_h) \geq 2$.

Observation 4. Suppose that $v \in seen(T_i)$ is bad and is not equivalent to any vertex of $seen(s)$ outside T_i. Then, all vertices equivalent to v belong to T_i.

By observation 1, all bad vertices of $seen(s)$ that belong to the same tree T_i are equivalent to each other and by observation 4 there is no need to check these bad vertices for equivalence to vertices outside T_i that do not belong to $seen(s)$. Thus all it is left to do is to check whether bad vertices from different trees are equivalent to each other. By observation 2 there can be at most two trees with at least one hole and we distinguish three cases:

Case I. There is only one non-full tree T_j with at least two holes (remember there are at least two holes in the whole tree T).

By observation 3 all bad vertices outside T_j belong to the same equivalence class, while the bad vertices of T_j belong to their own equivalence class. Therefore if $j \neq 0$, then the bad vertices outside of T_j are all equivalent to z otherwise they are in a new equivalence class.

Case II. There are two non-full trees T_i and T_j with one hole each.

In this case, as we shall see in Lemma 8, the bad vertices belong to the trees T_i and T_j and two new equivalence classes are created: one containing the bad vertices of T_i and the other containing the bad vertices of T_j. Obviously, if $i = 0$ or $j = 0$ then the corresponding equivalence class already exists and needs not to be created.

Case III. There are two non-full trees: T_i with one hole and T_j with at least two holes.

If s has degree 2 then we are in the same case as before and the bad vertices in T_i and T_j are in two different equivalence classes.

Otherwise, only T_j contains bad vertices and they are in an equivalence class by themselves.

The only non trivial step in algorithm *Mark* is the test of equivalence of s and v performed at line 3. Nonetheless, as we shall see in Lemma 8, this test can be performed in constant time.

The next subsections are organized in the following way: in Subsection 5.1 we give necessary and sufficient conditions for a vertex of $seen(s)$ to be equivalent to s; these conditions take into account information about the topology of the tree and the configuration of the pebbles that can be tested in constant time by using tables computed during the preprocessing phase. In Subsection 5.2, instead, we prove the four observations above mentioned.

5.1 Testing the equivalence of s and a vertex of $seen(s)$

Consider vertices s and $v \in seen(s)$. In this subsection we prove that s and v are equivalent if and only if there exists a branch vertex that can be reached by the pebbles at s and v. We start with the following easy Lemma that we state without proof. An example is given in Figure 6.

Lemma 7 *Let u and v be two vertices of T and let $\mathcal{U}$ be a configuration of pebbles such that the only holes are on vertices of $u \leadsto v$. Then if all vertices of $u \leadsto v$, except possibly for the vertices adjacent to u and v, have degree 2 then u and v are not equivalent with respect to $\mathcal{U}$.*

```
01. Put all vertices of $seen(s)$ that are equivalent to $s$ in the same equivalence class as $s$.
02. Let $w_1, \cdots, w_m$ be the bad vertices of $seen(s)$ that do not belong to $T_0$;
03. if there is a unique non-full tree $T_j \in F(s)$ then
04.     if $j \neq 0$ then
05.         put all bad vertices that are not in $T_j$ in the equivalence class of $z$;
06.         put all bad vertices of $T_j$ in a new equivalence class;
07.         return;
08.     if $j = 0$ then
09.         put all bad vertices that are not in $T_0$ in a new equivalence class;
10.         return;
11. if there are two trees $T_i, T_j \in F(s)$ that have exactly one hole then
12.     put all bad vertices of $T_i$ in a new equivalence class unless $i = 0$;
13.     put all bad vertices of $T_j$ in a new equivalence class (unless $j = 0$);
14.     return;
15. if there are two non-full trees $T_i, T_j \in F(s)$ and $T_j$ has more than one hole then
16.     put all bad vertices of $T_j$ in a new equivalence class unless $j = 0$;
17.     return;
```

Figure 5: A description of algorithm *Mark*.

We notice that the condition that the branch vertices are adjacent to u and v is crucial. Indeed if some vertex of $u \rightsquigarrow v$ not adjacent to u or v is a branch vertex, it is easy to see that u and v are equivalent.

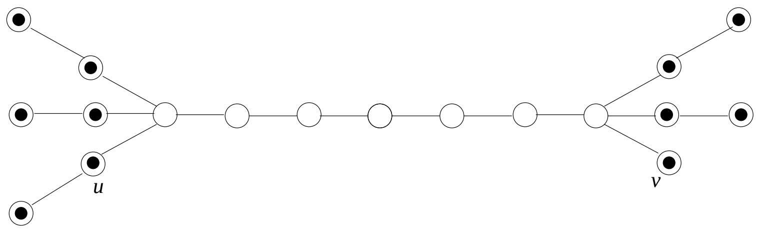

Figure 6: A configuration $\mathcal{U}$ as in Lemma 7.

In the next corollary we show that the number of holes necessary for the exchange of two pebbles depends on the distance from branch vertices. In Figure 7, we give an example of a configuration where the exchange is not possible because the number of holes is too small compared to the distance from the branch vertices.

Corollary 3 *Let $\mathcal{U}$ be a configuration on the tree T and suppose that all vertices of $u \rightsquigarrow v$ have degree 2 and do not contain pebbles. Let A be the forest consisting of the trees of $F(u)$ that do not contain v and, similarly, let B be the forest consisting of the trees of $F(v)$ that do not contain u. Then, if for all branch* *vertices $w_A \in A$ and $w_B \in B$ it holds that*

$$holes(A) \le dist(u, w_A) + 1 \quad \text{and } holes(B) \le dist(v, w_B) + 1$$

then u and v are not equivalent with respect to $\mathcal{U}$.

Proof: The proof is by contradiction. Assume that a plan P exists that exchanges the pebbles of u and v in the configuration $\mathcal{U}$. Let z_A be a vertex of A at distance $holes(A)$ from u and z_B a vertex of B at distance $holes(B)$ from v. Since by hypothesis the closest branch vertex of A is at distance $\ge holes(A) - 1$ from u, none of the vertices of $z_A \leadsto u$, except possibly the one adjacent to z_A, is a branch vertex. Similarly for $z_B \leadsto v$. Therefore the only vertices of $z_A \leadsto z_B$ that can be branch vertices are the ones adjacent to z_A and z_B.

Consider the configuration $\mathcal{V}$ obtained from $\mathcal{U}$ by performing the following plan Q: all pebbles that lie on z_A and on vertices of $z_A \leadsto u$ are moved to vertices of A that are outside of the path from z_A to u; move the pebble from u to z_A; execute similar moves for the pebbles that lie on the path from v to z_B. Observe that in $\mathcal{V}$, z_A and z_B satisfy the hypothesis of Lemma 7 ($z_A \leadsto z_B$ is empty and the only branch vertices are adjacent to z_A and z_B) and thus are not equivalent with respect to $\mathcal{V}$. On the other hand the following plan exchanges the pebbles of z_A and z_B thus yielding a contradiction:

1) execute the moves of $rev(Q)$ to reach configuration $\mathcal{U}$;

2) execute P and exchange pebbles in u and v;

3) execute Q. ■

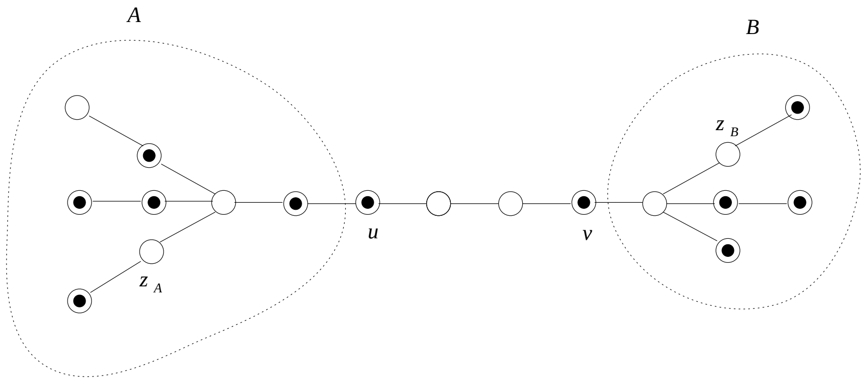

Figure 7: A configuration $\mathcal{U}$ as in Corollary 3.

We are now ready to prove the necessary and sufficient conditions for the equivalence of a vertex s and and vertex $v \in seen(s)$.

Lemma 8 *Let $\mathcal{U}$ be a configuration of pebbles on the tree T and suppose no pebble is in the vertices of $u \leadsto v$. Then vertices u and v are equivalent with respect to $\mathcal{U}$ if and only if at least one of the following conditions is met (see Figure 8).*

Condition 1. *There exists a branch vertex w in $u \leadsto v$, such that, denoted by A_1, A_2 the trees of $F(w)$ containing respectively u and v, both A_1 and A_2 are not full or at least one of the other trees of $F(w)$ is not full.*

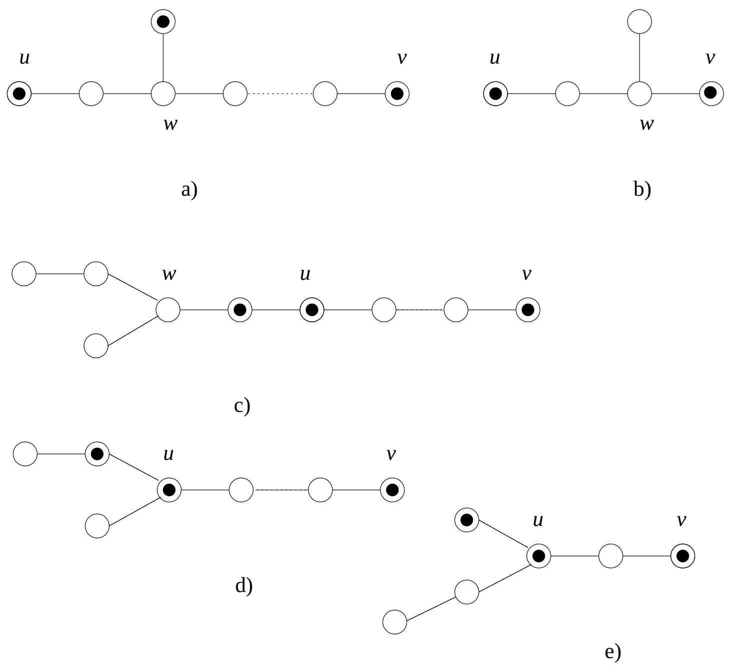

Figure 8: Examples of configurations satisfying at least one of the Conditions of Lemma 8. In a) and b) Condition 1 is satisfied. In c) Condition 2 is satisfied. In d) and e) Condition 4 is satisfied.

Condition 2. *There exists a branch vertex w such that u belongs to $w \rightsquigarrow v$ and, denoted by A_1 the tree of $F(u)$ that contains w, it holds that A_1 has at least $d(A_1)+2$ holes.*

Condition 3. *Obtained from the above condition by exchanging the role of u and v.*

Condition 4. *Vertex u has degree greater than 2 and, denoted with A_1 the tree of $F(u)$ that contains v, we have that at least two trees of $F(u)$ are not full and at least two holes are outside of A_1.*

Condition 5. *Similar to the above condition with v in place of u.*

Proof: We will prove only that the conditions of the lemma are necessary. Their sufficiency can be easily proved by constructing plans that exchange the pebbles in u and v for each of the 5 conditions.

To prove that the conditions are necessary we proceed by contradiction. We assume that pebbles u and v are exchangeable with respect to a configuration $\mathcal{U}$ and that $\mathcal{U}$ does not satisfy any of the conditions of the Lemma.

Let us first consider the case in which no branch vertex lies on $u \rightsquigarrow v$ (and thus Condition 1 is negated) and suppose that u and v are not branch vertices and thus Conditions 4 and 5 are negated). Now, $F(u)$ consists of two trees U_1 and U_2 with $v \in U_1$ and, similarly, $F(v)$ consists of V_1 and V_2 with $u \in V_1$. Notice that U_2 and V_2 might be the empty tree. Since Conditions 2 and 3 are not met then we have that for all branch vertices $w \in U_2$ it holds that

$$holes(U_2) \leq dist(u,w)+1$$

and similarly for all branch vertices $w \in V_2$. Now, we are in the hypothesis of Corollary 3 and thus u and v are not equivalent contradicting the hypothesis.

Next consider the case in which $u \rightsquigarrow v$ does not contain a branch vertex (thus Condition 1 does not hold) and u is a branch vertex (the case in which v is a branch vertex is derived similarly). To negate Condition 4 we have two choices. The first assumes that only one tree T' of $F(w)$ is non full. If T' does not contain v, then u and v must be adjacent. Moreover, since Condition 2 does not hold, then we have

$$holes(T') \leq dist(u, w) + 1$$

where w is the branch of T' closest to u. Therefore it is possible to reach from $\mathcal{U}$ a configuration where the pebble in u is at a vertex w' adjacent to w and all the holes of T' are at vertices of $w' \rightsquigarrow v$. Thus we are in the hypothesis of Corollary 3 and thus w' and u are not equivalent and consequently neither are u and v in $\mathcal{U}$. Contradiction.

On the other hand if T' does contain v, denote by F the forest of trees of $F(v)$ that do not contain u and let w' be branch vertex of F that is closest to v. Since Condition 3 does not hold, we have that

$$holes(F) \leq dist(v, w') + 1.$$

On the other hand, the forest of the trees of $F(u)$ that do not contain v has no hole and thus, again, we are in the hypothesis of Corollary 3. Therefore, u and v are not equivalent and we have a contradiction.

The case in which less than two holes lie outside A_1 is proved using a similar argument.

Suppose now that $u \rightsquigarrow v$ contains some branch vertices. If there are more than 2 branch vertices then at least one of them, w, is not adjacent to u nor to v and, since no pebble is on vertices of $u \rightsquigarrow v$, two of the trees of $F(w)$ are non full. Thus Condition 1 is met contradicting our hypothesis. Suppose now that $u \rightsquigarrow v$ has one branch vertex w and suppose w is adjacent to a branch vertex u . Then all subtrees of $F(w)$ are full, except for the one containing v. Since u and v are exchangeable so are w and v with respect to the configuration $\mathcal{U}'$ in which a pebble has been moved from u to w. On the other hand, w and v satisfy the hypothesis of Corollary 3. To see this, remind that Condition 2 does not hold for v and that all subtrees of $F(w)$, except the one containing v, are full. Therefore they are not equivalent with respect to $\mathcal{U}'$. Contradiction.

Finally, if there are two branch vertices, they are adjacent to u and v and we are in the hypothesis of Lemma 7. Therefore u and v are not equivalent. ■

5.2 Proofs of the observations

Lemma 9 (Observation 1) *All bad vertices of* seen(T_i) *are equivalent each other.*

Proof: We observe that if T_i is full then *seen*(T_i) consists of only the root of T_i and the Lemma trivially holds.

Suppose now that T_i is not full and let u and v be two bad vertices of *seen*(T_i). We notice that the vertices of $u \rightsquigarrow v$ contain at least one branch vertex and none of them contains a pebble. Then if some tree $T_j \in F(s)$, with $j \neq i$, is not full then vertices u and v meet Condition 1 of Lemma 8 with s as branch vertex and are thus equivalent.

On the other hand, if all tree of $F(s)$ other than T_i are full, then $holes(T_i) \geq 2$ (remind that we have assumed that $holes(T) \geq 2$). We observe that, since u is not equivalent to s, by Condition 1 of Lemma 8, for each branch vertex w of $u \rightsquigarrow s$, all trees of $F(w)$ that do not contain u nor s are full and similarly for the branch vertices of $v \rightsquigarrow s$. The two conditions combined together imply that u and v are siblings and all the holes of T_i lie on the path from their parent z to s. Now, Condition 1 of Lemma 8 holds for u and v with z as branch vertex and thus u and v are equivalent. ■

Lemma 10 (Observation 2) *If some vertex of* seen(s) *is bad, then at most two trees of $F(s)$ are non-full, and at most one of these trees has more than one hole.*

Proof: If more than 2 trees of $F(s)$ are non-full then Condition 4 of Lemma 8 is satisfied for s and any vertex of $seen(s)$. Therefore all vertices of $seen(s)$ are equivalent to s.

Suppose now that exactly two trees of $F(s)$ have more than 1 hole. Then, again, Condition 4 of Lemma 8 is satisfied for s and any vertex of $seen(s)$ and thus all vertices of $seen(s)$ are equivalent to s. ■

Lemma 11 (Observation 3) *Let $u \in$ seen(T_i) and $v \in$ seen(T_j), with $i \neq j$, be two bad vertices. Vertices u and v are equivalent if and only if there exists a tree T_h, $h \neq i, j$ such that $holes(T_h) \geq 2$.*

Proof: The if part of the lemma is trivial. If there exists a tree T_h with at least two holes, we can assume without loss of generality, that the two holes are at root of T_h and thus we can move the pebble from s to the root of T_h. Let $\mathcal{U}$ be the configuration reached. We observe that in $\mathcal{U}$ $u \rightsquigarrow v$ does not contain any pebble and it includes the branch vertex s. Since T_h is not full by condition 1 of Lemma 8 u and v are equivalent.

Let us prove that if u and v are equivalent then there must exist a tree T_h in $F(s)$, with $h \neq i, j$, that contains two holes. Without loss of generality, call $1, 2$ and 3 the pebbles placed at vertices u, v and s, respectively. Assume that there is only one non full tree T_h in $F(s)$ that does not contain u and v and T_h has only one hole. We will show that in this configuration u and v cannot be equivalent.

Let $\mathcal{U}$ be the configuration reached from $\mathcal{S}$ by moving pebble 3 from s to the root of T_h (which without loss of generality is assumed to be a hole). Obviously, if u and v are equivalent with respect to $\mathcal{S}$ they are also equivalent with respect to $\mathcal{U}$. Therefore, as there are no pebble in $u \rightsquigarrow v$, it must exist a branch vertex x that in configuration $\mathcal{U}$ satisfies some of the conditions of Lemma 8. However, if x is distinct from s, then x would satisfy the same conditions also in $\mathcal{S}$ and, thus, s would be equivalent to at least one of u and v. If x is equal to s the trees T_i and T_j containing u and v must be non full, since all the other trees of $F(s)$ are full (remember that the unique hole of T_h has been occupied by moving pebble 3 in T_h). But, if T_i and T_j are non full in $\mathcal{U}$, by construction they are non full also in $\mathcal{S}$. Then, in $\mathcal{S}$ T_h and T_j are non full and vertex s is equivalent to u, by condition 4 of Lemma 8. ■

Lemma 12 (Observation 4) *Let u be a bad vertex of* seen(T_i) *that is not equivalent to any vertex of* seen(s) *outside T_i. Then no vertex outside T_i is equivalent to u.*

Proof: Suppose that there exists a vertex $v \in T_j$, $j \neq i$, that is equivalent to u. Since by hypothesis u is not equivalent to any vertex of $seen(s)$, v does not belong to $seen(T_j)$. Therefore there are at least two pebbles in $u \rightsquigarrow v$. In fact, one is the pebble in s and at least another pebble is in a vertex w of $s \rightsquigarrow v$ (remember $v \notin seen(s)$).

Without loss of generality assume that pebbles $1, 2, 3$ and 4 are placed at u, v, s and w, respectively.

Let x be a branch vertex of $u \rightsquigarrow v$. If x belongs to $u \rightsquigarrow s$ then all the trees of $F(x)$ that do not contain u and v are full, otherwise, by condition 1 of Lemma 8, u and s would be equivalent. Similarly if x belongs to $w \rightsquigarrow v$ then all the trees of $F(x)$ that do not contain u and v are full, otherwise, by condition 1 of Lemma 8, v and s would be equivalent. If x, instead, belongs to the path from s to w, then there is at most one non full tree in $F(x)$ that does not contain u and v and this tree has only one hole, for otherwise, by conditions $4-5$ of Lemma 8, u would be equivalent to one of s or w.

In sum, we claim that there is at most one hole outside of the trees not containing u and v and, without loss of generality, assume that the hole is a vertex not of $u \rightsquigarrow v$ that is adjacent to the branch vertex. Let $\mathcal{U}$ be the configuration reached from $\mathcal{S}$ by moving either pebble 4 or pebble 3 to this hole (assume pebble 4 was moved). Clearly, if u and v are equivalent with respect to $\mathcal{S}$ they are also equivalent with respect to $\mathcal{U}$. However, since u and v are bad and are equivalent to each other, by Lemma 11 there is a tree in $F(s)$ not containing u and v that has more than one hole. This is a contradiction. ■

We have thus proved the following Theorem.

Theorem 5 *For each instance $I = (T, \mathcal{S}, \Pi)$ of the PPT problem algorithm Mark stops after $O(n)$ steps and partitions the vertices of $\mathcal{S}$ into equivalence classes with two vertices being in the same class if and only if the relative pebbles can be exchanged with respect to $\mathcal{S}$.*

6 Concluding remarks and open problems

6.1 Finding a plan for solving the problem

It can be easily seen that our algorithm can be adapted to output, for each feasible instance, also a plan that solves the instance in which case the algorithm stops in time $O(n + l)$ where l is the length of plan (a better running time can be obtained if a compact representation of the plan is sufficient; we omit details). An analysis of the algorithm shows that the plan output consists of $O(k^2(n-k))$ moves. The worst case is when $k = \Theta(n)$, in which case the solution has $O(n^3)$ moves. In [3], it is claimed that all feasible instances admit an $O(n^2)$-move plan but we were unable to verify the claim. A lower bound of $\Omega(n^2)$ on the length of plans of trees of n vertices can be derived from [1] and thus any algorithm that computes a solution (not necessarely optimal) for a feasible istance of the PMT problem has to take time $\Omega(n^2)$.

We can improve the $O(k^2(n-k))$ bound to $O(n)$ for the length of the plan in the case in which $k = n-1$.

Theorem 6 *Each feasible instance I of the PMT problem with $k = n-1$ pebbles has a plan of length $O(n)$. Moreover, this plan can be computed in time $O(n)$ and is optimal.*

Proof's sketch: The proof is based on the following observation. A pebble in a vertex u can reach its destination v if and only if u and v are adjacent and the subtree $C(F(v), u)$ is not full.

From this it follows that an instance $I = (T, \langle s_1, \cdots s_k \rangle, \langle d_1, \cdots d_k \rangle)$ with $k = n-1$ pebbles is feasible if and only if a) all pebbles are in their destination; or b) there exists an order of the pebbles $l_1, \cdots, l_{k'}$ whose source and destination do not coincide such that $d_{l_i} = s_{l_{i+1}}$ and $d_{l_{k'}}$ is a hole. It can be proved that this plan is the only solution to I and thus is trivially optimal. ■

We notice that for general graph finding the shortest plan for the case $k = n - 1$ is NP-complete.

6.2 Vertices with capacities

In some application it makes sense to assign capacities to vertices so that if vertex v has capacity $c(v)$ then at any given time no more than $c(v)$ pebbles can be located at v. It is possible to modify our algorithm so to handles this case as well in linear time algorithm. We omit the rather trivial but cumbersome details.

6.3 Open problems

Next we list some problems that seems to be worth investigating.

General graphs. First of all, we would like to ask whether our algorithm can be generalized to general graphs thus giving a linear time feasibility algorithm for the pebble motion problem on graphs.

Obstacles. In the problem we have studied all pebbles have a destination. The case in which only one pebble has a destination has been solved in [5]. Nothing is known for the intermediate cases; that is, we have k pebbles and a destination is assigned only for $1 < k_1 < n - 1$ of them and we are asked if it is possible to reach a configuration in which the k_1 pebbles are at their destination and no restriction is imposed on the final destination of the remaining $k - k_1$ pebbles.

Computing a shortest plan. Not much is known about the problem of computing the shortest plan on graphs. From a negative side it has been proved that the problem is NP- complete for $k = n - 1$ even when restricted to the grid [6]. However, the cases $k < n - 1$ have not been studied. Obtaining shortest plan for constant k is trivial but the complexity of the algorithm seems to be exponential in k.

On the positive side, algorithms for computing the shortest plan on trees have been given only by in [5] and [2] that considered the case of one pebble with destination and $k - 1$ pebbles without destination.

References

[1] N. Amato, M. Blum, S. Irani, R. Rubinfeld, *Reversing Trains: A Turn of the Century Sorting Problem*, Journal of Algorithms, 10, 413–428, 1989.

[2] V. Auletta, D. Parente and G. Persiano, *A New Approach to Optimal Planning of Robot Motion on a Tree with Obstacle* , in Proc. of the 4-th European Symposium on Algorithms, (ESA), Barcelona, Spain, September 25–27, 1996.
(A full version is available as: http://www.unisa.it/papers/jrobots.ps.gz).

[3] D. Kornhauser, G. Miller, and P. Spirakis, *Coordinating pebble motion on graphs, the diameter of permutations groups, and applications*, in Proc. of 25-th IEEE Symp. on Found. of Comp. Sc., (FOCS), 241–250, 1984.

[4] S. Loyd, *Mathematical Puzzles of Sam Loyd*. Dover, New York, 1959.

[5] C. Papadimitriou, P. Raghavan, M. Sudan and H. Tamaki, *Motion Planning on a graph*, in Proc. of 35-th IEEE Symp. on Found. of Comp. Sc., (FOCS), 511–520, 1994.

[6] D. Ratner and M. Warmuth, *The* (n^2-1)-*Puzzle and Related Relocation Problems*, Journal of Symbolic Computation, 10, 111–137, 1990.

[7] J.T. Schwartz, M. Sharir and J. Hopcroft, *Planning, Geometry, and Complexity of Robot Motion*, Ablex, Norwood NJ, 1987.

[8] R. M. Wilson, *Graph puzzles, homotopy, and the alternating group*, Journal of Comb. Theory Series B, 16, 86-94, 1974.