AMO - Advanced Modeling and Optimization, Volume 10, Number 1, 2008

An Optimal Algorithm to Find Maximum and Minimum Height Spanning Trees on Cactus Graphs

Kalyani Das and Madhumangal Pal

Department of Applied Mathematics with Oceanology and Computer Programming,
Vidyasagar University, Midnapore – 721 102, India.
e-mail: mmpalvu@gmail.com

Abstract.

A cactus graph is a connected graph in which every block is either an edge or a cycle. An optimal algorithm is presented here to find the maximum and the minimum height spanning trees on cactus graphs in $O(n)$ time, where n is the total number of vertices of the graph. The cactus graph has many applications in real life problems, specially in radio communication system.

Keywords: Design of algorithms, analysis of algorithms, spanning tree, Euler tour, cactus graph.

AMS Subject Classifications: 68Q22, 68Q25, 68R10.

1 Introduction

Let $G = (V, E)$ be a finite, connected, undirected, simple graph of n vertices and m edges, where V is the set of vertices and E is the set of edges. A tree is a connected graph without any circuits. A tree T is said to be a spanning tree of a connected graph G if T is a subgraph of G and T contains all vertices of G.

A vertex v is called a *cut-vertex* if removal of v and all edges incident to v disconnect the graph. A *non-separable graph* is a connected graph which has no cut-vertex and a *block* means a maximum non-separable sub-graph. A block is a *cyclic block* or simply *cycle* in which every vertex is of degree two.

A *cactus graph* is a connected graph in which every block is either an edge or a cycle.

A *path* of a graph G is an alternating sequence of distinct vertices and edges beginning and ending with vertices. The *length of a path* is the number of edges in the path. The *longest path* and *shortest path* are the paths from the vertex u to the vertex v if there exist no other path from u to v with higher length and lower length respectively. We use $lp(u, v)$ and $\rho(u, v)$ to denote these paths. The *longest distance* $ld(u, v)$ and *distance* $d(u, v)$ between two vertices u and v are the length $lp(u, v)$ and $\rho(u, v)$ in G if such paths exist.

Note that $ld(u, u) = 0$, $ld(u, v) = ld(v, u)$ and $ld(u, v) \leq ld(u, w) + ld(w, v)$.

Also $d(u, u) = 0$, $d(u, v) = d(v, u)$ and $d(u, v) \leq d(u, w) + d(w, v)$.

The *elongation* of a vertex u in a graph G is the longest distance from vertex u to a vertex furthest from u *i.e.*, $el(u) = max\{ld(u, v) : v \in V\}$. Vertex v is said to be a *furthest vertex* of u if $ld(u, v) = el(u)$.

The *eccentricity* of a vertex u in a graph G is the longest distance from the vertex u to a vertex furthest from u i.e, $e(u) = max\{d(u, v) : v \in V\}$.

In a tree, a vertex v is said to be at *level* l if v is at a distance l from the root. The *height* of a tree is the maximum level which is occurred in the tree.

A graph may have more than one spanning tree. The height of a spanning tree T of a graph G is denoted by $H(T, G)$. A *maximum height spanning tree* is a spanning tree whose height is maximum among all spanning trees of a graph. The height of the maximum height spanning tree of a graph G is denoted by $H_{max}(G) = max\{el(u) : u \in V\}$.

Suppose v be the vertex for which $H_{max}(G)$ is attained and v' its furthest vertex, then the longest path *i.e.*, $lp(v, v')$ is called as *maximum height path* (v, v') and denoted by $MHP(v, v')$.

A *minimum height spanning tree* is a spanning tree whose height is minimum among all spanning tree of a graph. The height of the minimum height spanning tree of a graph G is denoted by $H_{min}(G) = min\{e(u) : u \in V\}$. The vertex x for which $H_{min}(G) = e(x)$ is called the *center* of G.

To illustrate the problem we consider a cactus graph of Figure 1.

Some related works are discussed here: In [8], a spanning tree of maximal weight and bounded radius is determined from a complete non-oriented graph $G = (V, E)$ with vertex set V and edge set E with edge weight in $O(n^2)$ time, n is the total number of vertices in G.

In [9], the minimum spanning tree problem is considered for a graph with n vertices and m edges. They introduced randomized search heuristics to find minimum spanning tree in polynomial time with out employing global techniques of greedy algorithms.

In [10], the authors find a spanning tree T that minimizes $D_T = Max_{(i,j)\in E} d_T(i, j)$ where $d_T(i, j)$ is the distance between i and j in a graph $G = (V, E)$. The minimum restricted diameter

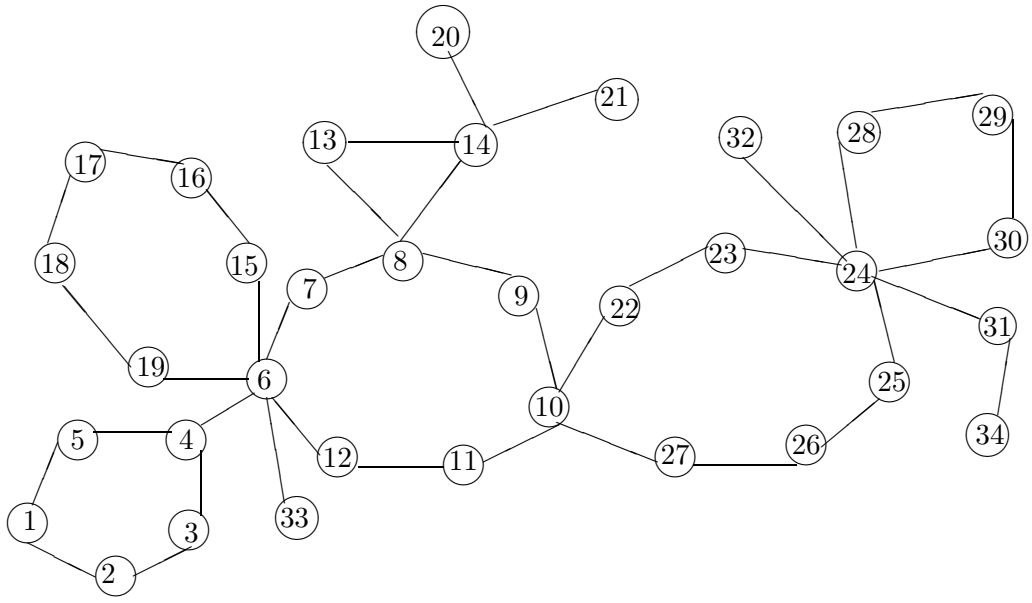

Figure 1: A cactus graph G.

spanning tree problem is to find spanning tree T such that the restricted diameter is minimized. It is solved in $O(\log n)$ time.

In [11], the minimum diameter spanning tree problem on graphs with non-negative edge lengths is determined which is equivalent for finding shortest paths tree from absolute 1-center problem of the general graph is solvable in $O(mn + n^2 \log n)$ time [12].

In our problem, we find the maximum height spanning tree by finding the elongation and the $MHP(u, v)$. Also we find the minimum height spanning tree by finding the eccentricity and the radius of the graph G.

In the following section we construct a tree T_{BC} whose nodes are the blocks of G and edges are defined between two nodes if they are adjacent blocks *i.e.*, they have at least one common vertex of the graph G.

2 Construction of the tree T_{BC}

As described in [7] the blocks as well as cut vertices of a graph G can be determined by applying DFS technique. Using this technique we obtain all blocks and cut vertices of the cactus graph $G = (V, E)$. Let the blocks be B_1, B_2, B_3,..., B_N and the cut vertices be C_1, C_2, $C_3, \ldots, C_R$ where N is the total number of blocks and R is the total number of cut vertices.

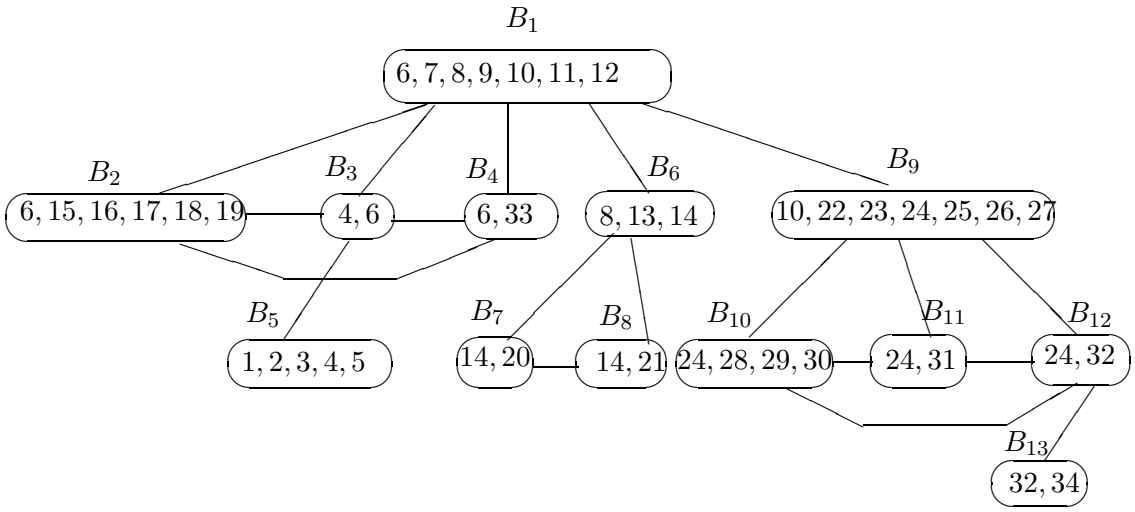

Figure 2: The intermediate graph G' of G.

The blocks and cut vertices of the cactus graph shown in Figure 1 are respectively $\{B_1 = (6, 7, 8, 9, 10, 11, 12), B_2 = (6, 15, 16, 17, 18, 19), B_3 = (4, 6), B_4 = (6, 33), B_5 = (1, 2, 3, 4, 5), B_6 = (8, 13, 14), B_7 = (14, 20), B_8 = (14, 21), B_9 = (10, 22, 23, 24, 25, 26, 27), B_{10} = (24, 28, 29, 30), B11 = (24, 31), B12 = (24, 32), B_{13} = (32, 34)\}$.

Now we have in a position to construct the tree T_{BC}. Before constructing the tree we define *an intermediate graph* G' whose vertices are the blocks of G and an edge is defined between two blocks if they are adjacent blocks of G .

i.e., $G' = (V', E')$ where $V' = \{B_1, B_2, \ldots, B_N\}$

and $E' = \{(B_i, B_j) : i \neq j, i, j = 1, 2, \ldots, N,\ B_i$ and B_j are adjacent blocks $\}$.

The graph G' for the graph G of Figure 1 is shown in Figure 2.

Two properties of the graph G' are described below.

Lemma 1 *In G' there exists no cycle of length more than 3.*

Lemma 2 *The three vertices of G' forming a triangle must have a common cut vertex of G.*

Now the tree T_{BC} is constructed from G' as follows:

We discard some suitable edges from G' in such a way that the resultant graph becomes a tree. The procedure for such reduction is given below:

Let us take any arbitrary vertex of G', containing at least two cut-vertices of G, as root of the tree T_{BC} and mark it. All the adjacent vertices of this root are taken as children of level one and

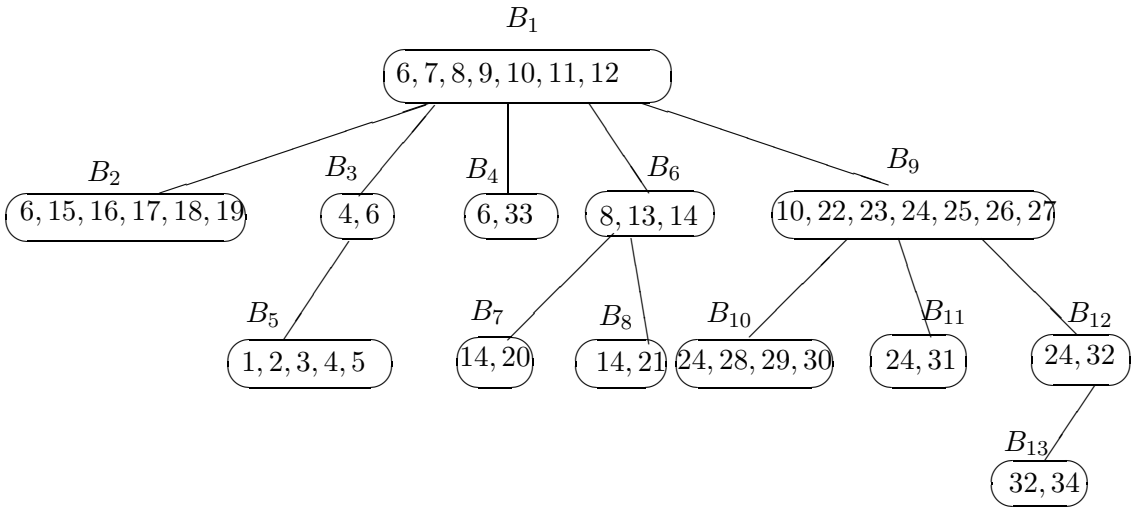

Figure 3: The tree T_{BC} of the graph G

are marked. If there are edges between the vertices of this level, then those edges are discarded. Each vertices of level one is considered one by one to find the vertices which are adjacent to them but unmarked. These vertices are taken as children of the corresponding vertices of level one and are placed at level two. These children at level two are marked and if there be any edge between them then they are discarded. This process is continued until all the vertices are marked.

Thus the tree $T_{BC} = (V', E'')$ where $V' = \{B_1, B_2, \ldots, B_N\}$ and $E'' \subset E'$ is obtained.

For convenience, we refer the vertices of T_{BC} as nodes.

We note that each node of this tree is a block of the graph $G = (V, E)$.

The parent of the node B_i in the tree T_{BC} will be denoted by *Parent(B_i)*. The tree T_{BC} constructed from G' is given in Figure 3.

3 Euler Tour

Euler tour produces an array of nodes. The tour proceeds with a visit to the root and there after visits to the children of the root one by one from left to right returning each time to the root using tree edges in both directions. Algorithm GEN-COMP-NEXT of Chen et al. [2] implements this Euler tour on a tree starting from the root. The input to the algorithm is the tree represented by a 'parent of' relation with explicit ordering of the children. The output of

$i:$	1	2	3	4	5	6	7	8	9	10	11	12	13
$S(i):$	$(B_1)_1$	$(B_2)_1$	$(B_1)_2$	$(B_3)_1$	$(B_5)_1$	$(B_3)_2$	$(B_1)_3$	$(B_4)_1$	$(B_1)_4$	$(B_6)_1$	$(B_7)_1$	$(B_6)_2$	$(B_8)_1$
$i:$	14	15	16	17	18	19	20	21	22	23	24	25	
$S(i):$	$(B_6)_3$	$(B_1)_5$	$(B_9)_1$	$(B_{10})_1$	$(B_9)_2$	$(B_{11})_1$	$(B_9)_3$	$(B_{12})_1$	$(B_{13})_1$	$(B_{12})_2$	$(B_9)_4$	$(B_1)_6$	

Table 1: The sequence of nodes obtained from Euler tour.

the algorithm is the tour starting from the root of the tree and ending also at the root. The tour is represented by an array $S(1:2N-1)$ that stores information connected to the visits during the tour. The element $S(i)$ of the array S is a record consisting of two fields, one of which, denoted by $S(i).node$, is the node visited during the ith visit while the other, denoted by $S(i).subscript$ is the number of times the node $S(i).node$ is visited during the first i visits of the tour. Two fields of an element of S are written together using the notation $(node)_{subscript}$.

Also, we consider an array $f(j)$ which stores the total number of occurrence of the block $B_j, j = 1, 2, 3, \ldots, N$ in the array $S(i), i = 1, 2, 3, \ldots, 2N-1$. Thus $f(j)$ represents the number of visits of the block B_j in the Euler tour, *i.e.*, $f(j)$ is the maximum subscript of B_j in the array $S(i)$.

The array S for the graph of Figure 1 is shown in Table 1.

For each j, $j = 1, 2, \ldots, N$, $(B_j)_{f(j)}$ occurs only once in the array $S(i)$ and before $(B_j)_{f(j)}$ all of $(B_j)_1, (B_j)_2, \ldots, (B_j)_{f(j)-1}$ occur in order of increasing subscripts of B_j.

The order in which $(B_j)_{f(j)}$, $j = 1, 2, \ldots, N$, occurs in the array $S(i)$ are first noted and corresponding nodes are considered in the same order for determining the dominating set of G. This process implies that nodes come first for consideration before the consideration of their parent.

The following important lemma is proved in [6].

Lemma 3 *If $S(i).subscript = 1$ and $S(i+1).subscript \neq 1$, then $S(i).node$ is a leaf node of the tree.*

4 Computation of elongation

In this section, we introduce a relationship between furthest vertex u of an arbitrary vertex s and furthest vertex v of u in the cactus graph G. Ghosh et al. [3] proposed an algorithm to find the diameter of a tree by using shortest path between u and v. In our problem, we find the longest path between u and v where they contain in the blocks which are the nodes of the tree

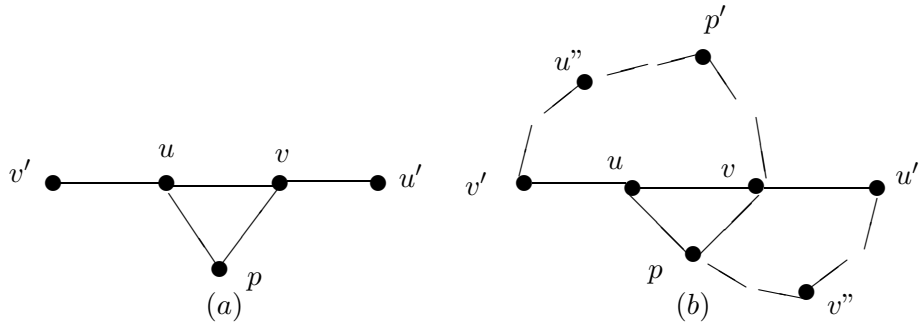

Figure 4: (*a*) Two paths $lp(u, u')$ and $lp(v, v')$ intersect, (*b*) Two paths $lp(u, u')$ and $lp(v, v')$ do not intersect.

obtained from G. Here u belongs to root node of the tree. Let $FV(u)$ denote that the set of all furthest vertices from u in G.

Two paths intersect if they have at least one common vertex.

Lemma 4 *If u' and v' are furthest vertex of u and v respectively then $lp(u, u')$ intersect $lp(v, v')$.*

Proof: Let us consider a graph G_1 of Figure 4(a). Here u, u' are furthest vertices and v, v' are furthest vertices.

Suppose $lp(u, u')$ and $lp(v, v')$ do not intersect. Then there must exist vertices u'' and p' in $lp(v, v')$ so that the $ld(v, v')$ remains same. Similarly, there must exist a vertex v'' in $lp(u, u')$ so that $ld(u, u')$ remains same. This is shown in Figure 4(b).

But the graph in Figure 4(b) does not represent a cactus graph. Hence our assumption that $lp(u, u')$ and $lp(v, v')$ do not intersect is not correct.

There fore $lp(u, u')$ and $lp(v, v')$ intersect. □

Lemma 5 *If $u \in FV(s)$ and $v \in FV(u)$ then $ld(s, u) \leq ld(u, v)$.*

Proof: Assume on the contrary that $ld(s, u) > ld(u, v)$. Since $ld(u, s) = ld(s, u)$ hence $ld(u, s) > ld(u, v)$ which implies that s is furthest vertex of u than v. But it is given that v is a furthest vertex of u. It is possible only when $ld(s, u) \leq ld(u, v)$. Thus our assumption is wrong. Hence $ld(s, u) \leq ld(u, v)$. □

Lemma 6 *For any arbitrary $MHP(u, u')$ in G and for any arbitrary $v \in V$ either $u \in FV(v)$ or $u' \in FV(v)$.*

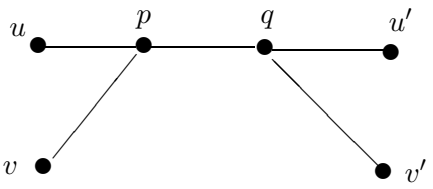

Figure 5: The paths $lp(u, u')$ and $lp(v, v')$ intersect

Proof: Let $v' \in V$ be the furthest vertex of v. We have to show that $ld(v, v') = ld(v, u)$ or $ld(v, v') = ld(v, u')$. From Lemma 4 we see that $lp(v, v')$ and $lp(u, u')$ intersect. This is shown in Figure 5. Let $lp(p, q) = lp(v, v') \cap lp(u, u')$.

Now $p \in lp(u, q)$ and since u' is furthest vertex of u, $lp(q, u') \geq lp(q, v')$. Similarly $lp(q, v') \geq lp(q, u')$ as v' is furthest vertex of v.

From the above inequality it is seen that $lp(q, v') = lp(q, u')$ *i.e*, $lp(v, v') = lp(v, u')$.

In a similar manner we can prove that $lp(v, v') = lp(v, u)$. □

In the following algorithm we compute a furthest vertex of an arbitrary vertex $s \in V$. Also we find the elongation of s using this algorithm.

Algorithm $FDV(s)$

Input: The cactus graph $G = (V, E)$ and an arbitrary vertex s.

Output: The furthest vertex from s and $el(s)$.

Step 1: Compute the blocks and cut vertices of G and construct a tree T_{BC} where the root node contains the vertex s.

Step 2: Apply Euler tour on T_{BC} and store the output in the array $S(1 : 2N - 1)$, N is the total number of nodes of T_{BC}.

Step 3: Compute $f(j)$ which stores total number of occurrences of the node B_j in the array $S, j = 1, 2, \ldots, N$.

Step 4: Note the order in which $(B_j)_{f(j)}$, $j = 1, 2, \ldots, N$ occurs in the array $S(i)$.

Step 5: Consider the nodes B_j one by one following the order of Step 4 and let a be the cut vertex of B_j and Parent(B_j). Now the earning longest distance, denoted by eld(a) is computed as follows:
(i) If $f(j) = 1$, *i.e.*, for a leaf node B_j, $eld(a) = |B_j| - 1$ and $FV(a)$ is the set of adjacent vertices of a in B_j.
(ii) If $f(j) \neq 1$, *i.e.*, for an interior and root node B_j. Compute $eld(a)$ and $eld(s)$ respectively as
$eld(a) = \text{Max } \{|B_j| - 1, eld(v_k) + ld(a, v_k)\}$
$eld(s) = \text{Max } \{|B_j| - 1, eld(v_k) + ld(s, v_k)\}$
$v_k's$ are the cut vertices of B_j other than a or s where $j = 1, 2, \ldots, N$ and $k < |B_j|$.

Step 6: Obtain a sequence of nodes through which $el(s)$ is determined from step 5. In this sequence for a node B_j if
(i) $f(j) \neq 1$ and $eld(a) = |B_j| - 1$, then $FV(s) = FV(a)$, otherwise
(ii) $FV(s) = FV(a)$ for a node B_j in which $f(j) = 1$, a is a cut-vertex of B_j and Parent(B_j).

Step 7: Find $el(s)$ from the relation $el(s) = eld(s)$.

end $FDV(s)$

The blocks and cut vertices of any graph can be computed in $O(m + n)$ time [7]. For cactus graph $m = O(n)$, hence Step 1 of Algorithm FDV(s) takes $O(n)$ time. As the array S is obtained by applying Euler's tour on the tree T_{BC}, Step 2 takes $O(n)$ time. Step 3 takes only $O(n)$. Step 5 can be perform by comparing $f(j)$ with 1 for $j = 1, 2, \ldots, n$, so this step takes only $O(n)$ time. Obviously, Step 4 and Step 6 takes $O(n)$ time. Hence the total time complexity of Algorithm $FDV(s)$ is $O(n)$. Here all the arrays are of size $O(n)$. So the space complexity is also of $O(n)$.

5 Determination of maximum height spanning tree

In this section, we describe an algorithm to find the tree with maximum height. According to Lemma 6, if $u \in FV(s)$ then u is an end point of $MHP(u, v)$, $v \in FV(u)$. Thus again we construct a tree where the root node contains u. Then we find $el(u)$ and corresponding $FV(u)$ by applying the algorithm $FDV(u)$. Hence the length of $MHP(u, v) = el(u), v \in FV(u)$.

Algorithm MXHST
Input: The cactus graph $G = (V, E)$.
Output: Maximum height spanning tree $HMAX(G) = (V, E'), E' \subset E$.

Step 1: Select arbitrarily a vertex s and find the vertex $u \in FV(s)$ using algorithm $FDV(s)$.
Step 2: Find a vertex $v \in FV(u)$ using algorithm $FDV(u)$.
Step 3: Take u as root of the spanning tree $HMAX(G)$ and delete one edge say e from each cycle such that $e \notin lp(u, v)$,

end MXHST

Lemma 7 *The tree $HMAX(G)$ obtained from the algorithm MXHST is the tree with maximum height.*

Proof: In algorithm MXHST we arbitrarily select a vertex s, then find the furthest vertex u using algorithm $FDV(s)$. Also find the furthest vertex v of u using $FDV(u)$. Thus we get the $MHP(u, v)$. Lemma 5 supports that there is no longest path other than $lp(u, v)$ in the graph G. Hence the height which is equal to the length of $MHP(u, v)$ obtained from the algorithm MXHST is maximum. □

Theorem 1 *The spanning tree obtained from the algorithm MXHST is computed in $O(n)$ time.*

Proof: Step 1 and Step 2 of algorithm MXHST determine the furthest vertices of s and u by using algorithms $FDV(s)$ and $FDV(u)$. Those algorithms take $O(n)$ time. In Step 3 the deletion of edges from all cycles can be performed in $O(n)$ time as the numbers of blocks are less than the number of vertices n. Hence the algorithm MXHST is computed in $O(n)$ time. Again, the space complexity of the algorithms $FDV(s)$ and $FDV(u)$ is $O(n)$. Hence the space complexity of the algorithm is also of $O(n)$. □

6 Minimum height spanning tree in cactus graph

In order to find the minimum height spanning tree we have to find the center of G first. Wang et al. [5] find the center of the cactus graph in linear time.

Here we construct a tree having the center x as root and delete one edge from each cyclic block so that $e(x)$ becomes the height of the tree.

6.1 Deletion of edge from a cyclic block with m vertices

Let B_j be a cyclic block where $B_j = \{v_1, v_2, \ldots, v_m\}$ and v_1 is the cut vertex of B_j and Parent(B_j).

If m is even, delete the edge $(v_{\frac{m}{2}}, v_{\frac{m}{2}+1})$ or $(v_{\frac{m}{2}+1}, v_{\frac{m}{2}+2}) \notin \rho(x, u)$ or any one of them if both does not belongs to $\rho(x, u)$.

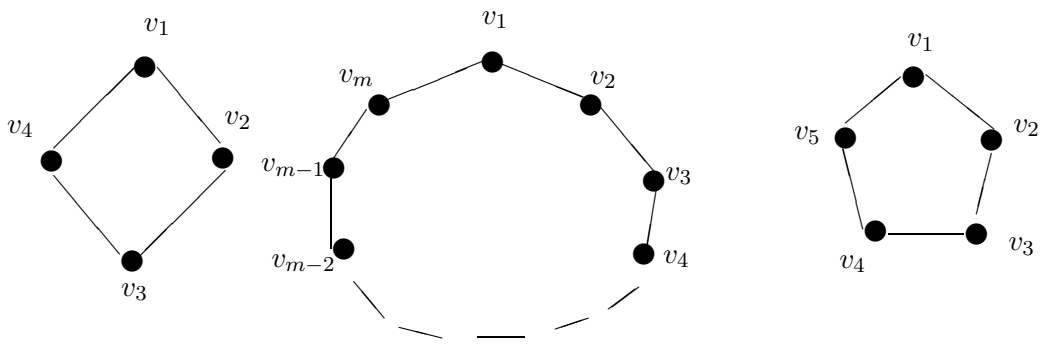

Figure 6: cyclic block with m vertices,m being even and odd.

If m is odd, delete the edge $(v_{\frac{m+1}{2}}, v_{\frac{m+1}{2}+1}) \notin \rho(x,u)$.
For the root block we take x as v_1.

Algorithm MNHST
Input: The cactus graph $G = (V, E)$.
Output: Minimum height spanning tree $HMIN(G) = (V, E''), E'' \subset E$.

Step 1: Find the center of G say x and the corresponding shortest path $\rho(x,u)$ and eccentricity $e(x)$.
Step 2: Determine the cut vertices and blocks and construct a tree T_{BC} in which the root node contains the vertex x as described in Section 3.
Step 3: For each node B_j find cut vertex of B_j and Parent(B_j). Also determine the number of vertices m_j in each node B_j. If m_j is even or odd delete one edge from each cycle as descried in Section 6.1.
end MNHST

Lemma 8 *The tree $HMIN(G)$ obtained from the algorithm MNHST is the tree with minimum height.*

Proof: In algorithm MNHST we find the center x of G so that the eccentricity of x is minimum among all the vertices of G. Also we make a tree with the blocks where the root contains the vertex x and delete the edge from each block in such a way that the height of the tree does not exceed the length of $e(x)$. Hence the tree $HMIN(G)$ obtained from the algorithm MNHST is of minimum height. □

Theorem 2 *The spanning tree obtained from the algorithm MNHST is computed in $O(n)$ time.*

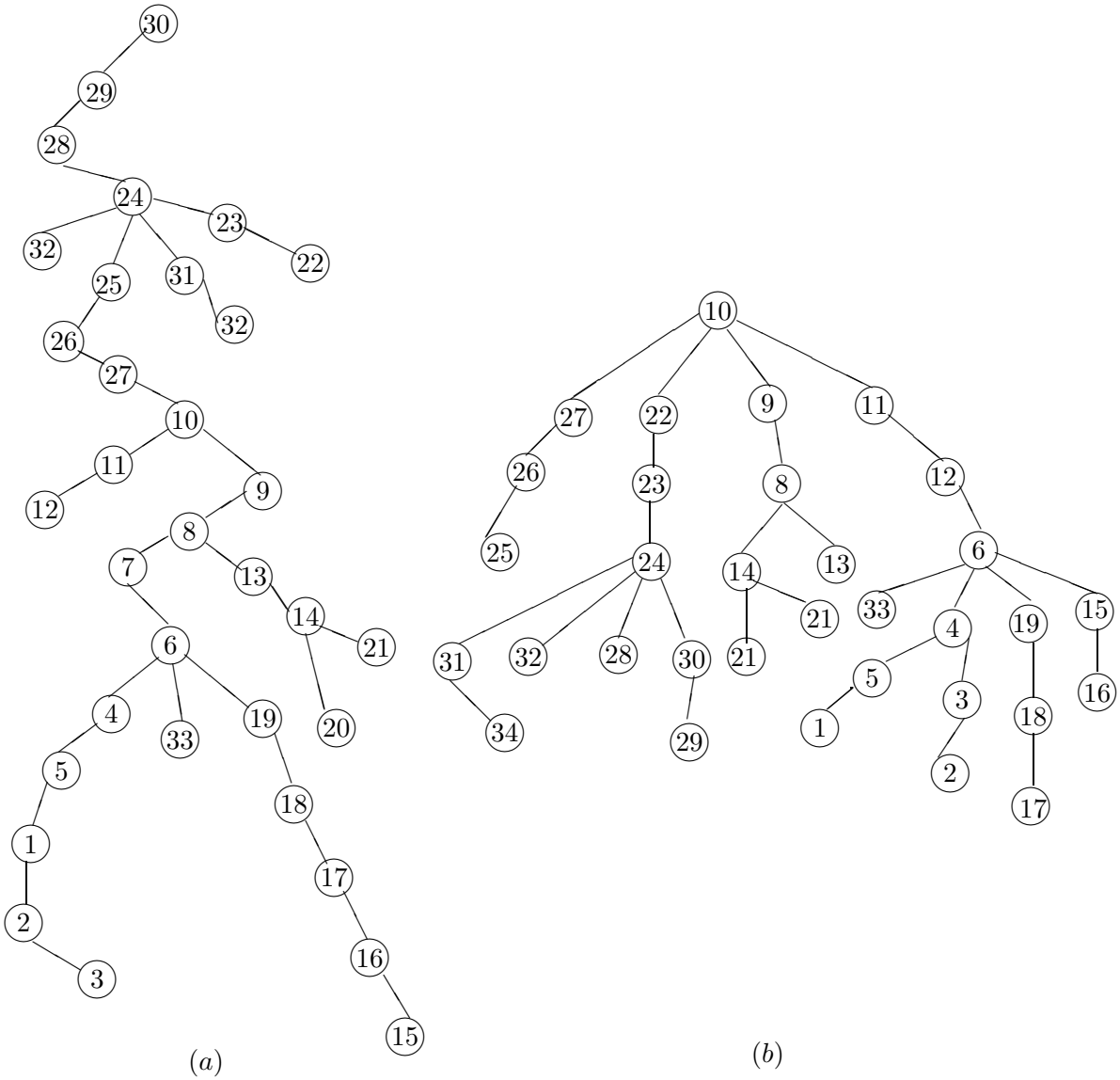

Figure 7: (a) Spanning tree with maximum height, (b) Spanning tree with minimum height of the graph of Figure 1.

Proof: In the algorithm MNHST Step 1 takes $O(n)$ time as in [5], the center of a cactus graph is computed in linear time. The blocks and cut vertices of any graph can be computed in $O(m+n)$ time [7] and the tree T_{BC} is also found in $O(n)$, hence Step 2 takes $O(n)$ time. Step 3 is computed in $O(n)$ time as number of nodes are less than the number of vertices of G and every node is taken once to find the number of vertices and deletion of edge. Hence the complexity of the algorithm MNHST is $O(n)$. □

6.2 An illustration

We illustrate the problem for the cactus graph of Figure 1.

Suppose the arbitrary vertex is 7. Then $FV(7) = \{3, 5, 15, 19, 28, 30\}$ and $el(7) = 11$ using algorithm $FDV(7)$. Now take any vertex say, 30 from $FV(7)$. Then $FV(30) = \{3, 5, 15, 19\}$ and $el(30) = 16$ using algorithm $FDV(30)$, *i.e.*, maximum height of the spanning tree is 16.

The center of the graph is 10 and $e(10) = 6$, *i.e.*, minimum height of the spanning tree is 6.

The trees are given in the Figure 7.

References

[1] Chen, C. C. Y., and Das, S. K., Breadth-first traversal of trees and integer sorting in parallel, *Information Processing Letters,* 41 (1992) 39-49.

[2] Chen, C. C. Y., Das, S. K., and Akl, S. G., A unified approach to parallel depth-first traversals of general trees, *Information Processing Letters,* 41 (1991) 49-55.

[3] Ghosh, S. K. and Maheswari, A., An optimal parallel algorithm for computing furthest neighbors in a tree, *Information Processing Letters* 44 (1992) 155-160.

[4] Koontz, W. L. G., Economic evaluation of loop feeder relief alternatives, *Bell System Technical J.,* 59 (1980) 277-281.

[5] Lan, Y. -F., Wang, Y. -L. and Suzuki, H., A linear-time algorithm for solving the center problem on weighted cactus graphs, *Information Processing Letters,* 71 (1999) 205-212.

[6] Pal, M., and Bhattacharjee, G. P., An optimal parallel algorithm for all-pairs shortest paths on unweighted interval graphs, *Nordic Journal of Computing,* 4 (1997) 342-356.

[7] Reingold, E. M., Nivergent, J and Deo, N., Combinatorial Algorithms : Theory and Practice, (Prentice Hall, Inc., Englewood Chiffs, New Jersy, 1977).

[8] Scrdjukov, A.I., On finding a maximum spanning tree of bounded radius, *Discrete Applied Mathematics,* 114 (2001) 249-253.

[9] Neumann, F. and Wegener, I., Randomized local search, evolutionary algorithms and the minimum spanning tree problem, *Theoretical Computer Science* , 2007, Article in Press.

[10] Hassin, R. and Levin, A., Minimum restricted diameter spanning tree, *Discrete Applied Mathematics,* 137 (2004) 343-357.

[11] Hassin, R. and Tamir, A., On the minimum diameter spanning tree problems, *Information Processing Letters,* 53(2) (1995) 109-111, 1995.

[12] Kariv, O. and Hakimi, S. L., An algorithmic approach to network location Problems, Part 1: The p-center, *SIAM J. Appl. Math*, 37 (1979) 513-537.