A New Operation for Two R-trees to Efficiently Migrate Spatial Objects

Jeang-Kuo Chen*
Jen-Wei Huang
jkchen@cyut.edu.tw, s9114628@cyut.edu.tw

ABSTRACT

In the past, several papers discussed the access of individual object in R-tree, but rare mentioned how to access several objects simultaneously. In this paper, we propose a new operation, called spatial-migration, for R-tree. The function of this operation is to combine one group of objects into another group of objects according to a special relationship between the two groups of objects. That is, more than one spatial object in one R-tree is migrated to another R-tree at the same time. If objects are migrated one by one, several R-tree nodes may overflow or underflow repeatedly. The database performance may decrease because the R-tree may be reconstructed again and again. When many single-object insertions/deletions are replaced by a single multiple objects insertion/deletion, many redundant node-splits and/or MBR-adjustments can be omitted. When a node overflows due to the insertion of many objects, we once generate enough nodes to contain all the objects inserted into the node. Each node at most has only one node-split and/or MBR-adjustment. Therefore, the proposed spatial-migration operation can efficiently migrate objects between two R-trees without effecting database performance much.

Keywords: Multiple objects access, Spatial-migration, R-tree

1: Introduction

Spatial databases have been more and more important in many applications such as Geographical Information System (GIS), Computer Aided Design (CAD), etc. Many dynamic indexes of spatial objects have been proposed for speeding up object search. In general, multi-dimensional data can be classified into two types: zero-size objects and non-zero-size objects. The K-D-B tree [11], proposed by Robinson, and the G-tree [8], proposed by Kumar, are index structures for zero-size objects. The Grid files [9], proposed by Nievergelt *et al.*, and the Filter tree [14], proposed by Sevcik and koudas, are index structures for non-zero-size objects. A detailed survey on spatial objects access methods can be found in ”Multidimensional access methods” [3] where the R-tree family is the most popular one [1,2,4,5,6,7,12,13]. In the past, some papers for the R-tree were proposed to improve the access speed [2,4], to reduce dead space [13,14], or to improve storage utilization [1,7]. No paper mentioned how to access several objects simultaneously. In general, the R-tree inserts or deletes an object at a time. However, we may need to move several objects from a database B into another database A for some reasons. If each migration includes only one object, we should need much time to finish the job. The database performance will be influenced obviously because the corresponding R-trees of databases A and B also need to be reconstructed again and again.

Suppose that there is a particular relationship, such as overlap, between the objects of databases A and B. Sometimes, some objects in database B must be migrated to database A because of the overlap relationship. For example, database A stores the data of community while database B stores the data of park. If a part has to be combined to a community which overlaps the park, then some objects in database B must be taken out to insert into database A. The traditional operation is to take out park objects from database B one by one and then to insert them into database A one by one. It should take much time to execute the operation of searching, inserting, and deleting objects repeatedly. Especially, node-split or MBR-adjustment may occur again and again in the R-tree for database A. Some leaf nodes in the R-tree for database B may underflow and all objects in these nodes must be reinserted. It makes the R-tree be reconstructed again and again. Hence, the database performance may decrease dramatically. To speed up the migration, we propose a new operation, called *spatial-migration*, for R-tree. The proposed operation can efficiently find and move the related objects from one R-tree into another R-tree. To erase unnecessary node-splits and/or MBR-adjustment, we in advance compute enough extra leaf nodes for the inserted objects. Each node in the R-tree at most has only one node-split or MBR-adjustment.

2: Previous Work

2.1: R-tree

An R-tree [4] is a height-balanced tree similar to a B-tree with index records in the leaf nodes containing pointers to data objects. The B-tree stores one-dimensional data of character or number, while the R-tree keeps two (or more) -dimensional data of spatial objects. There is no order relationship between R-tree nodes. Each node is composed of several entries. Each entry includes a minimal bound rectangle (MBR) and a pointer. The format of an entry in a leaf node is (*I*, *obj-id*). The *I* is an MBR and *obj-id* is a pointer to address a spatial object. The format of an entry in a non-leaf node is (*I, child-pointer*). As an MBR, the *I* covers all the rectangles of the lower node entries and the *child-pointer* is the address of a lower node. We assume that *M* is the maximum number of entries in one node and *m* is the minimum number of entries in a node. An R-tree satisfies the following properties. The root has at least two children unless it is also a leaf node. Each non-root node has the number of children between *m* and *M*. All leaves appear on the same level.

2.2: Spatial Join

The concept of spatial join has been applied to several access methods such as spatial-merge join [10]. Brinkhoff *et. al.* [2] proposed five methods with the depth-first search to perform spatial join for R-tree. Each node must be checked to determine whether it overlaps with other nodes or not. Therefore, the five methods accomplish only local optimization. Later, Hung *et. al.* [5,6] proposed the method Breadth-First R-tree Join (BFRJ) with the feature of global optimization for the optimization of memory, buffer management, and the order of overlap data within non-leaf nodes. BFRJ adopts the breadth-first search to traverse nodes from the top of two R-trees level-by-level to leaf nodes. BFRJ uses search pruning to reduce the number of node-pair checking. BFRJ also uses several tables, called intermediate join indexes (IJI), to save pairs of overlap nodes. Each entry in an IJI includes two fields to record two overlap nodes belonging to different R-trees. Search pruning is performed by using IJI to implement join computation. Thus, the number of node-pair checking is reduced.

3: Spatial-migration operation

In this section, we describe the spatial-migration operation how to migrate objects from the combined R-tree to the combining R-tree at the same time. Assume that there are two groups of spatial objects associated with the combining R-tree R and the combined R-tree S, as shown in Figures 1 and 2, respectively. Figure 3 shows the overlap situation of the two groups of spatial objects. The spatial-migration operation is composed of two phases. The first phase is to find the related objects and the relevant leaf nodes in the two R-trees. Then, take out these related objects from S to insert into the relevant leaf nodes in R. The second phase is to delete these inserted objects from S.

3.1: The first phase of spatial-migration operation

We must first find the overlap objects that belong to R or S. With the concept of spatial-join [6], we can finish the above requirement without IJI tables. Instead, we use a queue, called Spatial Join Queue (SJQ), and a data structure, called Overlap Pair of Nodes (OPN), to save the overlap pair-nodes. SJQ is used to save OPN records. An OPN is composed of four fields. The parentR and parentS fields denote a pair of overlap parent nodes that belong to R and S, respectively. The childR and childS fields represent a pair of overlap child nodes or objects that belong to parentR and parentS, respectively.

Our spatial-join action starts from the roots of the two R-trees with level-by-level to check whether each pair of nodes at each level overlaps or not. If two nodes have an overlap, they are stored as an OPN record to SJQ. Next the first OPN record in SJQ is taken to check whether the children of the two overlap nodes have an overlap or not. If yes, an OPN record for the overlap pair of children is stored to SJQ. A table called PATH is used to record each ancestor node (at each level) of each object (in R) which overlaps objects in S. These ancestor nodes can be referenced when node-split and/or MBR-adjustment propagate upward. Each row, called a branch, in PATH is composed of two fields. The field nodeP denotes a certain node of R which overlaps a certain node N in S. The field nodeC denotes a certain child node or object of nodeP which overlaps a child node or object of N. The process of fetching an OPN record, examining overlap nodes, producing new OPN records, and storing OPN records to SJQ is repeated until all related nodes of the two R-trees are checked. Figure 4 shows the results of SJQ and PATH after spatial-join.

3.1.1: Delete the invalid OPN records in SJQ

Now, several OPN records are stored in SJQ. However, some of these OPN records are invalid because of the following two cases. First, the same object may overlap different objects in different leaf nodes at the same time. The same object may be repeatedly inserted into different leaf nodes. Second, the same object may overlap different objects in the same leaf node at the same time. The same object may be repeatedly inserted into the same leaf node. Therefore, these invalid OPN records in SJQ must be erased.

To erase the invalid OPN records in case 1, only one of the different leaf nodes must be determined. The determined leaf node is the one that contains an object which has the largest overlap area with the combined object. Resolve ties by choosing the one with fewer entries. The remaining OPN records in case 1 must be deleted after the determined leaf node is found. For example, object s9 overlaps object r11 in leaf node R4 and object r17 in leaf node R6, respectively, as shown in Figure 4. We must decide that s9 should be inserted into R4 or R6. Since the overlap area of objects r17 and s9 is larger than that of objects r11 and s9, as shows in Figure 3, the determined leaf node is R6. In level-3 SJQ of Figure 4, the 11th OPN record is retained while the 1st OPN record is deleted. To erase the invalid OPN records in case 2, all the OPN records, except one, must be deleted if these records have the same values of parentR and childS. For example, the 2nd to 4th OPN records in Figure 4 indicate that object s4 will be inserted into leaf node R7 three times. Thus, only one OPN record is kept, others must be deleted. The same way is also applied to records 5th to 7th, records 8th to 12th. The final result of SJQ is show in Figure 5.

3.1.2: Prepare enough leaf nodes for inserted objects

It is possible to insert a lot of objects into one leaf node for the spatial-migration operation. The leaf node may split many times if many objects are inserted one by one. The more the number of node-split makes the less the performance of R-tree. To erase redundant node-splits, we prepare enough extra leaf nodes to contain the inserted objects at the same time. The number of prepared leaf nodes can be computed as follows. Suppose that the maximum number and the possession number of entries in a leaf node N are M_R and M_{RC}, respectively, in R-tree R while the number of objects to be inserted into N is M_{SC}. There are two conditions can be considered. First, if $M_R \geq M_{RC} + M_{SC}$ then all the M_{SC} can be inserted into N directly. No extra leaf node is needed. Second, if $M_R < M_{RC} + M_{SC}$ then one or more extra leaf nodes are needed. The number of enough leaf nodes is $X=\lceil (M_{RC} + M_{SC})/M_R \rceil$. According to the node-split technique [4], the M_{RC}+M_{SC} objects must be divided into X groups to distribute these objects to the X leaf nodes, respectively. These X leaf nodes later should be inserted into the parent node P of N. If P also overflows, the same idea is applied to P until one of P's ancestors does not overflow. After node-split process, the MBR of the leaf nodes must be adjusted in its parent node and the adjustment must be propagated upward until the root. Figure 6 shows the final R-tree R after the insertion of the objects in R-tree S.

3.2: The second phase of spatial-migration operation

Finally, the related objects should be deleted from S after they are successfully inserted into R. If a leaf node in S is underflow, the remainder objects in this leaf node must be reinserted [4]. When many objects are deleted once, the number of underflow nodes and reinsertion times will increase. The corresponding R-tree needs to be reconstructed again and again. The deletion performance will be influenced obviously. The deletion of original R-tree [4] may not be suitable when deleting large number of objects once. We use a merge method to deal with the objects in underflow leaf nodes to achieve optimal results for reconstructing an R-tree. Composed of four steps, the merge method is described as follows.

In the first step, we delete the corresponding objects in S according to the objects identified by the childS values from SJQ. In the second step, reset PATH and record the paths of all nodes and objects of S to PATH level-by-level. The third step is to reconstruct S for all the objects instead of reinserting the remainder objects. We assume that the number of all objects in S is N_{ro} and the maximum number of entries of a leaf node in S is M_s. The least number of desired leaf nodes is $Y=\lceil N_{ro}/M_s \rceil$. The objects in S must be divided into Y groups with the following strategy. Add an object to the group whose covering rectangle will have to be enlarged least to accommodate the object. Resolve ties by adding the object to the group with smaller area. The fourth step is to adjust node's MBRs ascending from leaf nodes to the root.

An example is illustrated as follows. First, we retrieve the records in SJQ to find that six objects, s4, s5, s9, s10, s11, and s12 should be deleted from S. After the deletion, we traverse S level-by-level to record all the branches of all the paths to all nodes and objects in S to PATH. The result is shown in Figure 7. Now, N_{ro} is 3 (objects s6, s7, and s8) and we need $Y=\lceil N_{ro}/M_s \rceil = \lceil 3/4 \rceil = 1$ leaf node to contain these objects. The objects s6, s7, and s8 are inserted into the same node as shown in Figure 8. Final, the R-tree S has only one node, the root, as shown in Figure 9.

4: Conclusion

In tradition, the data access operations of R-tree including search, insert, delete, and update, aim at a single object. For real applications, the user may need to process large number of objects simultaneously. It is necessary to develop a special operation to process several objects at the same time. This is the motivation for us to propose the spatial-migration operation to combine two groups of objects together. The spatial-migration operation has some characteristics as follows. The OPN structure can be extended dynamically to keep information for user's demands such as adding a field to record the overlap area value of two overlap objects. Traditional object insertion may lead to many times of node-splits and MBR-adjustments that decrease the database performance. We solve the problem by preparing enough extra leaf nodes to contain all inserted objects once. Each node has only one node split and/or MBR-adjustment. In general, a leaf node in R-tree may be unable to meet the minimum number of entries after some objects are deleted from that leaf node. All objects in an under-flow node must be reinserted. Reinsertion makes database performance degrade. Therefore, our method deletes all objects from related leaf nodes once. All objects in the R-tree are redistributed. Our method avoids the R-tree shorten after some objects are deleted from the R-tree but recover again after some objects are inserted into the R-tree.

References

[1] N. Beckmann, H.P., Kriegel, R. Schneider, B. Seeger, "The R*-tree: An Efficient and Robust Access Method for Points and Rectangles," Proc. ACM SIGMOD Int. Conf. on Management of Data, Atlantic City, NJ, pp.322-331, 1990.

[2] T. Brinkhoff, H.P., Kriegel and B. Seeger, "Efficient processing of spatial joins using R-trees," in: *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pp.237-246, 1993.

[3] V. Gaede and O. Gunther, "Multidimensional access methods," ACM Computing Surveys, pp.170–231, 1997.

[4] A. Guttman, "R-trees: a dynamic index structure for spatial searching," in: Proceedings of the ACM SIGMOD, pp.47-57, 1984.

[5] Y.W. Hung, N. Jing, and E.A. Rundesteiner, "Spatial Joins Using R-trees: Breadth-First Traversal with Global Optimizations," in: *Proc. 23rd Int. Conf. on VLDB*, pp.396-405, 1997.

[6] Y.W. Hung, N. Jing and E.A. Rundesteiner, "BFRJ: global optimization of spatial joins using R-trees," Dept. of Computer Science, Worcester Polytechnic Institute, Tech. Report WPI-CS-TR-97-5, January, 1997.

[7] P.W. Huang, P.L. Lin, H.Y. Lin, "Optimizing storage utilization in R-tree dynamic index structure for spatial databases," Journal of Systems and Software of Elsevier Science, 55(3), pp.291-299, 2001.

[8] A. Kumar, "G-tree: A new data structure for organizing multidimensional data," IEEE Trans. Knowl. Data Eng. 6(2), pp.341–347, 1994

[9] J. Nievergelt, H. Hinterberger and K.C. Sevcik, "The grid file: An adaptable, symmetric multikey file structure," ACM Trans. Database Syst. 9(1), pp.38–71, 1984.

[10] J.M. Patel, and D.J. DeWitt, "Partition Based Spatial-Merge Join," in: Proc. ACM SIGMOD Int. Conf. on Mangement of Data, pp.259-270, 1996.

[11] J.T. Robinson, "The K-D-B-tree: A search structure for large multidimensional dynamic indexes," In Proceedings of the ACM SIGMOD International Conference on Management of Data, pp.10–18, 1981.

[12] N. Roussopoulos, D. Leifker, "Direct spatial search on pictorial databases using packed R-trees," In: Proceedings of the ACM SIGMOD, pp.17-31, 1985.

[13] T.Sellis, N. Roussopoulos, C. Faloutsos, "The R+-Tree: A Dynamic Index for Multi-Dimensional Objects," Proc. 13th Int. Conf. on Very Large Databases, Brighton, England, pp.507-518, 1987.

[14] K. Sevcik and D N. Koudas, "Filter trees for managing spatial data over a range of size granularities," In Proceedings of the 22th International Conference on Very Large Data Bases (Bombay), pp.16–27, 1996.

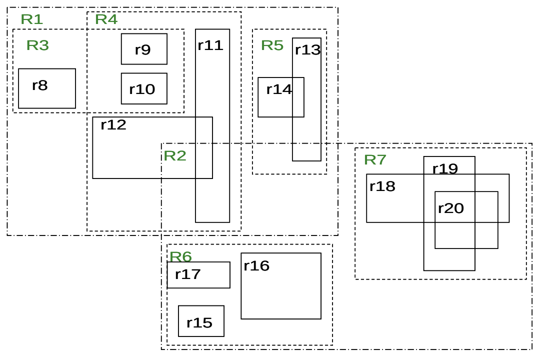


Figure 1(a) The spatial objects indexed by R-tree R.

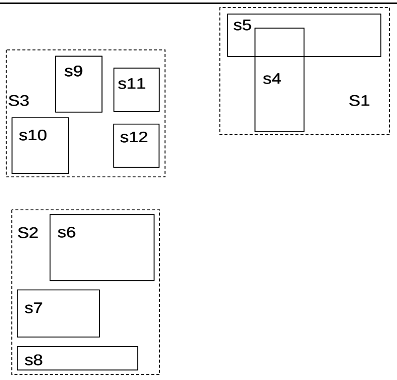


Figure 2(a) The spatial objects indexed by R-tree S.

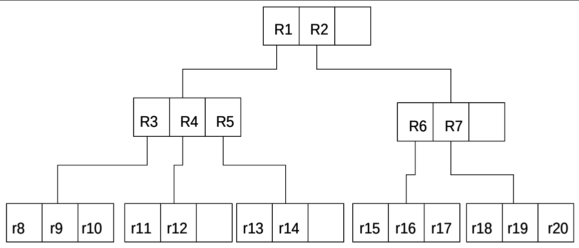


Figure 1(b) The structure of R-tree R.

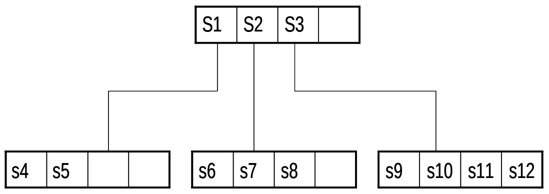


Figure 2 (b) The structure of R-tree S.

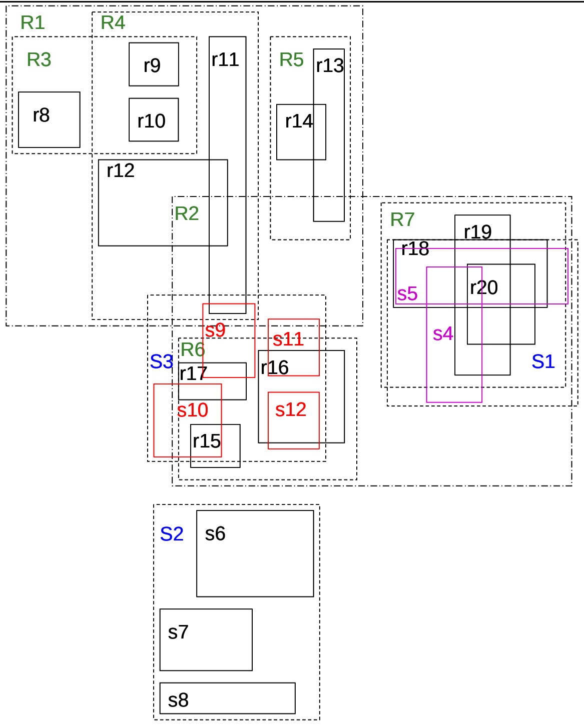


Figure 3 The overlap condition of the two groups of spatial objects.

Level 1 SJQ

parentR	childR	parentS	childS
Null	R1	Null	S3
Null	R2	Null	S1
Null	R2	Null	S3

PATH

nodeP	nodeC
Null	R1
Null	R2

Level 2 SJQ

parentR	childR	parentS	childS
R1	R4	S3	s9
R2	R7	S1	s4
R2	R7	S1	s5
R2	R6	S3	s9
R2	R6	S3	s10
R2	R6	S3	s11
R2	R6	S3	s12

PATH

nodeP	nodeC
Null	R1
Null	R2
R1	R4
R2	R7
R2	R6

Level 3 SJQ

parentR	childR	parentS	childS
R4	r11	S3	s9
R7	r18	S1	s4
R7	r19	S1	s4
R7	r20	S1	s4
R7	r18	S1	s5
R7	r19	S1	s5
R7	r20	S1	s5
R6	r15	S3	s10
R6	r16	S3	s11
R6	r16	S3	s12
R6	r17	S3	s9
R6	r17	S3	s10

PATH

nodeP	nodeC
Null	R1
Null	R2
R1	R4
R2	R7
R2	R6
R4	r12
R7	r18
R7	r19
R7	r20
R6	r15
R6	r16
R6	r17

Figure 4 The contents of SJQ and PATH at each level.

parentR	childR	parentS	childS
R7	r18	S1	s4
R7	r18	S1	s5
R6	r15	S3	s10
R6	r16	S3	s11
R6	r16	S3	s12
R6	r17	S3	s9

Figure 5 The SJQ contents after deleting 6 invalid OPN records.

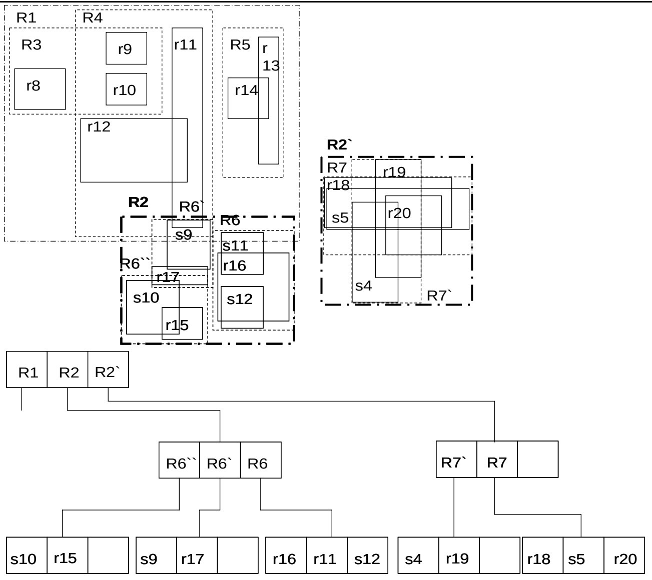
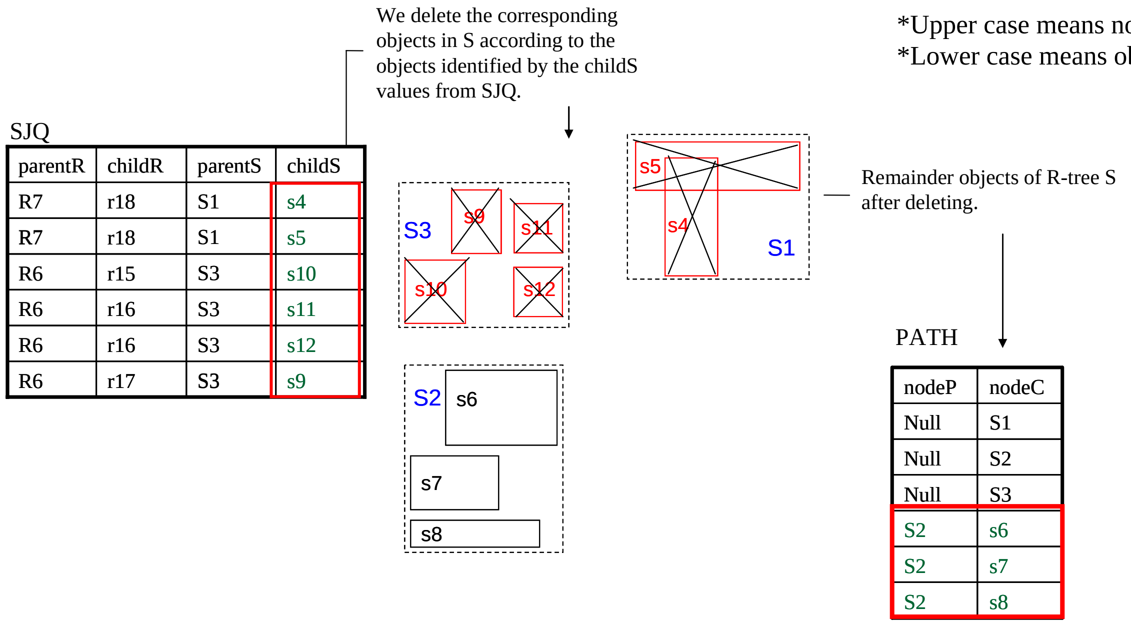


Figure 6 The final result R-tree R after inserting object.

*Upper case means node
*Lower case means object



We delete the corresponding objects in S according to the objects identified by the childS values from SJQ.

Remainder objects of R-tree S after deleting.

SJQ

parentR	childR	parentS	childS
R7	r18	S1	s4
R7	r18	S1	s5
R6	r15	S3	s10
R6	r16	S3	s11
R6	r16	S3	s12
R6	r17	S3	s9

PATH

nodeP	nodeC
Null	S1
Null	S2
Null	S3
S2	s6
S2	s7
S2	s8

Figure 7 The PATH contents after deleting objects.

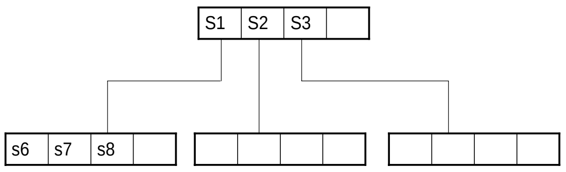


Figure 8 The structure of R-tree S after deleting objects.

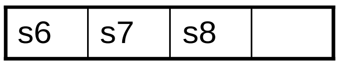


Figure 9 The final R-tree S after restructuring S.