Journal of Basic and Applied Engineering Research
Print ISSN: 2350-0077; Online ISSN: 2350-0255; Volume 1, Number 6; October, 2014 pp. 19-21
© Krishi Sanskriti Publications
http://www.krishisanskriti.org/jbaer.html

Maze Generation Using Disjoint-Sets with Stack

Bhawana Singh[1], Swasti Saxena[2], Anil Pandey[3], Pooja Khulbe[4]

[1,2]M.Tech. CSE Department, Bareilly, India
[3]CSE Department, Bareilly, India
[4]M.TechCSE Department, Dehradun, India

Abstract: **An important application of the tree is the representation of sets, where "n" distinct elements are needed to be grouped into number of disjoint sets. This paper defines the application of disjoint-set data structures for generating the maze. It also shows that the maze can be represented as a tree. A new improved algorithm has also been defined for constructing the maze using disjoint-set data structure with the stack.**

1. INTRODUCTION

A relation is defined on a set S if for every pair of elements (a, b), *a,b €S*, *a R b* is either true or false. If *a R b* is true it is said that a is related to b. An Equivalence Relation is a relation R that satisfies three properties:

- Reflexive: *a R a* is true for all *a €S.*
- Symmetric: *a R b* if and only if *b R a.*
- Transitive: *a R b* and *b R c* implies that *a R c.*

Maze is also example of equivalence relation. It is grid-like two-dimensional area of any size, usually rectangular in shape (as shown in figure 1). It consists of cells. A cell is an elementary maze item which is interpreted as a single-site. The maze contains different types of obstacles and may have single or multiple paths from source to destination. It is reflexive as a cell is connected to itself and both symmetric and transitive. For any equivalence relation, denoted ~, the natural problem is to decide for any a and b whether a ~ b.

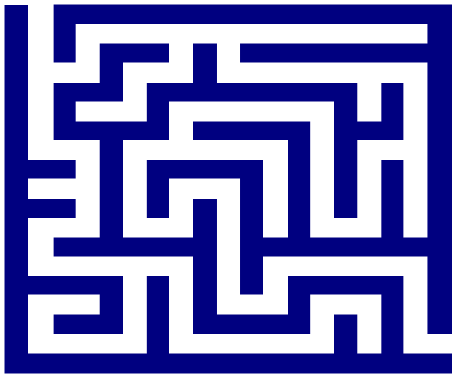

Fig. 1. Representation of Maze

Disjoint-Sets data structure is used to solve equivalence relation problems. The disjoint-sets are also called Partitions. Disjoint sets are sets such that $S_i \cap S_j = \varphi$. Partitions and equivalence relations are reflections of one another. A partition P determines an equivalence relation R on U by defining x R y iff x and y belong to the same set in P. Similarly, if R is an equivalence relation on U, then the set of equivalence classes of R is a partition:

$R[x] = \{y \in U: x \; R \; y\}$
$P = \{R[x]: x \in U\}$

2. DISJOINT-SET DATA STRUCTURE

2.1 Description of traditional Disjoint-set algorithm

The data structure and algorithm is known by three names: Disjoint-Sets, Union-Find and Partition. A disjoint-set data structure maintains a collection $S= \{S_1, S_2, S_k\}$ of disjoint dynamic sets. Each set is identified by a representative, which is some member of the set. If there are two sets S_x and S_y, x is not equal to y, such that S_x= (3, 4, 5, 6, 7) and S_y= (1, 2) then these sets are called disjoint sets as there is no element which is common in both sets. In other words, Disjoint-sets are the data structure for problems requiring equivalence relations, that is, are two elements in the same equivalence class. It requires two major operations:

- Union (x, y) : Modifies the partition by forming the union of the set containing x and the set containing y.
- Find (x, y) : Returns true iff x and y belong to the same set in the partition.

A simple and clever way to implement these algorithms is using a tree model. Assume that the universe is an initial segment of non-negative integers U = {0, 1, . . . , n−1}. Let v be a vector of integer elements with size n. Let v[x] hold the parent of the element x, and denote roots (element without parents) with value -1. The set of elements in a tree represents a set in the partition of U. Thus v contains the information defining a forest, a collection of trees, and trees in the forest represent the sets in the partition. The structure is initialized to the partition of singleton sets P = {{0}, {1}, . . . {n − 1}}, which means every element is in a tree of one node by itself. This is represented by v[x] = −1 for all x. Using this representation, the two subsidiary operations on the data structure are defined as follows:

- Union (x, y) = Link(Root(x),Root(y))
- Find (x, y) = (Root(x) == Root(y))
- Link (root1, root2): Modifies the partition by merging the two roots
- Root (x) : Returns the root of the tree containing x

2.2 Shortcoming of the traditional Union-Find Algorithm

The algorithm takes the large amount of time in generating the maze. So it is not applicable when number coordinates increases. Hence construction of large mazes is time-consuming. As the maze finds an application in gaming such puzzle solving etc. so there is a need to construct the maze in lesser time.

3. NEW IMPROVED DISJOINT-SET ALGORITHM

3.1 Proposed Algorithm

Step 1: Initialize a grid of r x c squares, where r denotes the number of rows and c denotes the number of columns.

```
Initialize( int r, int c)
{
maze = new int [r*c];
for (int e = 0; e < r*c; e++)
maze[e] = e;
}
```

Step 2: Start with the entire grid subdivided into squares.

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14
15	16	17	18	19

Fig. 2. Grid of 4x5 square

Step 3: Represent each square as a separate disjoint set.

Example-
{0} {1} {2} {3} {4} {5} {6} {7} {8} {9} {10} {11} {12} {13} {14} {15} {16} {17} {18} {19}

Step 4: Randomly choose a cell and mark it as a current cell.
Step 5: Initialize number of visited cells equal to one.
Step 6: Repeat the following algorithm-

1. Find the cells adjacent to the current cell-
   a) Let c denote the number of columns, r denotes the number of rows and i denote the cell.
   b) If (i<c) then there is no cell above the cell i.
   c) If (i>=(r*c-c)) then there is no cell below the cell i.
   d) If (i%c==0) then there is no cell to the left of cell i.
   e) If ((i+1) %c==0) then there is no cell to the right of cell i.

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14
15	16	17	18	19

0	1	i-c	3	4
5	i-1	i	i+1	9
10	11	i+c	13	14
15	16	17	18	19

Fig. 3. Adjacent cells

2. Store the adjacent cells in an array.
3. Initialize a stack of capacity r x c and mark its top equal to -1.
4. If the number of adjacent cells are greater than zero, then-
5. Randomly choose one of the adjacent cells.
   a) Check whether the two cells are disjoint. That is, if Find(x)! = Find(y), where x and y are two cells, then they are said to be disjoint.

```
int find( int i )
{
return maze[i];
 }
```

If sets are disjoint then knock the wall = union the sets.

```
void UnionSets( int i, int j )
{
        if(isadjacent(i,j)==TRUE)
{
rooti=find(i);
rootj=find(j);
for (int k=0; k<r*c; k++)
if (maze[k] == rootj)
maze[k] = rooti;
}
}
```

d) Push the current cell on the stack and mark the unioned cell as current cell.
e) Increment the number of visited cells.

Else pop the cell from the stack.

3.2 Working of algorithm

First enter the number of rows and columns and initialize a grid of row*columns. Each cell of the grid is represented as a separate disjoint set. Now randomly choose one of the cells and mark it as a current cell and mark the number of visited cells as 1. Inside a loop find the number of cells adjacent to the current cell and store them in array. Also initialize a stack and mark its top as empty. Check if the adjacent cells exist then randomly choose one of them. Check whether the current cell and the chosen adjacent cell lie in the same set or not. If they are disjoint then union the two cells or knock the wall between them. Now push the current cell in the stack and mark the chosen adjacent cell as the current cell. Also increment the number of visited cells. Otherwise if the number of adjacent cells does not exist then pop the topmost cell from the stack and continue with the loop checking for its adjacent cells.

3.3 Analysis of new algorithm

This new algorithm has improved the time complexity. It can be seen from the following table that this algorithm has improved the working of traditional Disjoint-Set algorithm.

Table 1. Comparison between traditional Disjoint-Set algorithm and new improved algorithm

Original data		Computing Time(ms)	
Number of rows	***Number of columns***	***Traditional Algorithm***	***New algorithm***
4	5	63	47
8	8	187	140
10	10	219	203
20	20	781	735
30	30	1781	1625
50	50	5141	4031

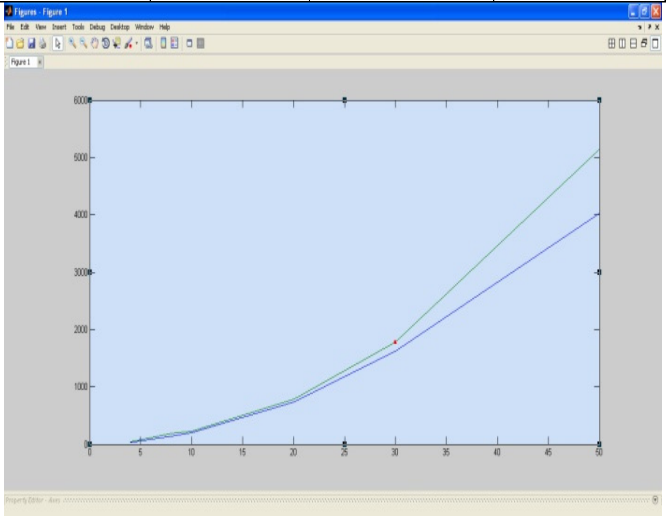

Fig. 4. Comparative study of the two algorithms

This graph depicts the comparative study of the Disjoint-Set algorithm and the new proposed algorithm. The y-axis indicates the time and the x-axis indicates the number of columns. The blue line stands for the new proposed algorithm and the green line stands for the traditional Disjoint-Set algorithm. It can be clearly seen that the computing time for the two algorithms is almost same for small mazes but vary largely for generating large mazes.

4. CONCLUSION

Disjoint-set algorithm had been used for various purposes. This paper has found one of the applications of Disjoint-set algorithm. By using it has improved the running time of the algorithm which was earlier very large. By this the need to generate the large maze in less time can be solved easily.

5. FUTURE SCOPE

This algorithm has used linear data structure and has improved the time complexity of traditional Disjoint-Set algorithm. Linear data structure is easy to implement. Further other methods can be applied for generating the maze which has multiple paths to the dead-end. In this paper disjoint-sets have been used to generate the maze structure that gives a unique solution. This method can also be used for more complex maze structure with different geometrical structure than rectangular structure.

REFERENCES

[1] T. Pasquier, J. Erdogan,"Genetic Algorithm Optimization in Maze Solving Problem", Institut Superieur d'Electronique de Paris

[2] N. S. Choubey, "A-Mazer with Genetic Algorithm", MPSTME, SVKM's NMIMS, Shirpur, Maharashtra, India, International Journal of Computer Applications (0975-8887) 58 (17), November 2012

[3] T. Sukumar, Dr. K. R. Santha, "Maze Based Data Hiding Using Back Tracker Algorithm" Department of IT, SVCE and Department of EEE, SVCE, Anna University, India, International Journal of Engineering Research and Applications (IJERA) ISSN: 2248-9622, 2 (4), July-August 2012, pp. 499-504.

[4] Maze Generation, ece.uwaterloo.ca.

[5] Maze classification, www.astrolog.org/labyrnth/algrithm.htm

[6] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, "Introduction to Algorithms", 2nd Edition, Pearson Education.

[7] Aho, Ullman, Hopcroft, "Design and Analysis of algorithms", Pearson Education.

[8] Union-Find Algorithm, www.cs.princeton.edu /~rs/AlgsDS07/01UnionFind.

[9] Patwary, Manne, "Multi-core Spanning Forest Algorithms using the Disjoint-Set Data Structure", 2012 IEEE 26th International Conference, pp-827- 835.

[10] Patwary, Palsetia, Agrawal, Manne, "A new scalable parallel DBSCAN algorithm using the disjoint-set data structure",2012 IEEE International Conference,pp-1-11.