A Study of Disjoint Set Union in Programming Competitions

Zijie Shen, Ruixiang Li, Junping Shi*
The School of Computer Science and Engineering, Jishou University, Jishou, Hunan, China
**Corresponding Author.*

Abstract: Disjoint Set Union (DSU) is a tree data structure, which is used to effectively deal with the problems of merging and querying disjoint sets. The DSU algorithm can be used to merge sets and query to which set the node belongs. The DSU algorithm is usually implemented using arrays and tree structures, but there are also methods that use hash tables. The DSU algorithm has a wide range of applications in the connectivity of graph theory, social network and image processing. It helps researchers better understand and analyze set operations, provides a basis for subsequent work, and promotes the solution and optimization of various problems in the field. At the same time, in the programming competition of college students, the use of DSU is more frequent, usually the use of set query is more, resulting in high time complexity and unable to solve the problem quickly. Therefore, path compression strategy is introduced to significantly improve the efficiency of set query. Finally, the application of path compression in union search is introduced in the form of programming competition.

Keywords: Disjoint Set Union; Tree Data Structure; Merging; Querying; Path Compression Strategy

1. Introduction

In the college programming competition, especially the International College Programming Competition (ICPC) [1], Disjoint Set Union (DSU) as a key data structure [2] has attracted much attention. Known as the Olympics of programming competitions for college students, ICPC highlights students' talent in algorithmic [3] problem solving, often involving DSU algorithms in data structure. The DSU algorithm was first proposed in 1964 by Bernard A. Galler and Michael J. Fischer for managing relations between equivalence classes [4]. DSU is a classical data structure, which is widely used in computer science [5] to solve set merging and query problems. The basic idea is to partition elements into a number of DSU and be able to merge and query these sets efficiently. With the DSU algorithm, we can quickly determine whether two elements belong to the same set and combine two sets into a single set. The application fields of DSU algorithm cover graph theory [6], network connection [7], social network analysis [8], image processing [9], database and many other fields. Became an integral part of computer science.

In the ICPC competition, when it is generally involved to determine whether two nodes are in a set [10], most cases need to use the DSU to solve. However, the time complexity [11] of the ordinary DSU algorithm is too high, so that the ordinary DSU algorithm cannot pass in the actual competition. Therefore, we consider to improve the time complexity of the DSU algorithm to reduce the time complexity of the algorithm, so as to solve the problems in the competition smoothly. The improved DSU algorithm can quickly query whether two nodes are in the same set.

2. The Study of DSU

2.1 Definition of DSU

DSU is a data structure for processing set merging and querying, which is widely used to solve set partition and join problems. In DSU, each element is assigned a unique identifier, usually an integer, to represent a set. These sets can be dynamically merged together to form a larger set, while allowing the query of whether two elements belong to the same set. The core idea of DSU is to maintain a set of disjoint sets and provide efficient methods to merge sets as well as to query the set to which an element belongs. In this way, DSU facilitates the solution of various practical

Copyright @ STEMM Institute Press

problems, such as network connectivity problems, connectivity problems in graph theory, and data association in databases. The flexibility and efficiency of DSU make it an important tool in algorithms and data structures, and it is widely used in various aspects of computer science.

2.2 Principle of DSU

DSU is an efficient data structure whose principle is to dynamically represent the union of multiple sets by maintaining a tree structure that is constantly updated. This data structure not only provides efficient query function, but also supports efficient dynamic update operation. The core idea of DSU is to group the elements of each set and assign a representative element to each group. These representative elements are organized into a tree structure, where each node represents a representative element of a set. By iteratively searching the representative elements in the tree, the set to which any element belongs can be quickly determined. In addition, DSU supports the Union operation, which merges two sets to form a larger set by joining their representative elements. This process involves adjusting the tree structure to ensure data consistency and efficiency. DSU has many advantages. First, it is able to quickly query the set to which an element belongs by directly accessing the node that represents the element. Second, DSU supports dynamic update operations, which means that elements can be added or removed without rebuilding the entire data structure. In addition, DSU is space efficient because it only stores representative elements of each set instead of storing all elements. Due to these advantages, DSU is widely used in many fields. Figure 1 is the merging process of the DSU.

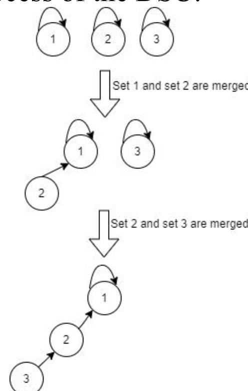

Figure 1. DSU Merging Process

2.3 Implementation of DSU

Assume that there are currently n sets and use the data p to represent each set. In order to facilitate the subsequent operations, these n sets need to be preprocessed first. The main purpose of preprocessing is to determine a representative element for each set and ensure that this representative element is an element of the set itself. With this preprocessing, we can query and update collections more efficiently. Specifically, for each set, we choose itself as its representative element. The preprocessing of n sets is necessary to facilitate the subsequent query and update operations and to improve the efficiency of the overall operation. The C++ code is shown in Figure 2:

```
for (int i = 1; i <= n; i++) {
    p[i] = i;
}
```

Figure 2. Preprocess n Sets

DSU has two basic operations: set query and set merge.

In the set query operation, the user can input an element x, and DSU will quickly determine which set this element belongs to, that is, determine the representative element of the set to which it belongs. This fast query capability makes DSU very useful in application scenarios where the set to which an element belongs needs to be queried frequently. The C++ code is shown in Figure 3:

```
int find(int x) {
    if (p[x] == x) {
        return x;
    }
    return find(p[x]);
}
```

Figure 3. Set Querying

In addition to query operations, DSU also provides set merging capabilities. In the set merge operation, the user can merge two or more sets into a new set. DSU achieves this operation by adjusting its internal data structure to maintain data consistency and efficiency. The merge operation may involve the adjustment of the tree structure, the join of nodes and other operations to ensure that the merged set can be queried and updated quickly. The C++ code is shown in Figure 4:

Copyright @ STEMM Institute Press

```
void union(int x, int y) {
    p[find(x)] = find(y);
}
```

Figure 4. Set Merging

2.4 Improvement of DSU

The optimization method of path compression is used to optimize the union search set, which aims to improve the efficiency of the search operation. Path compression is a technique that optimizes the lookup process by connecting nodes directly to their root node. Specifically, when searching for the root node, we directly connect the current node as well as all its parents along the path to the root node, thus reducing the height of the tree to the minimum and improving the speed and efficiency of the subsequent lookup operation. Figure 5 shows the optimized merging process.

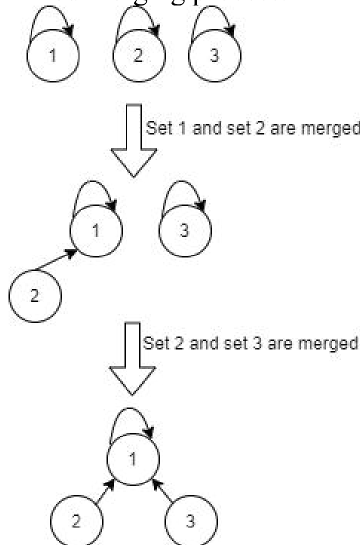

Figure 5. DSU Merging Process after Optimization

In this implementation, each node maintains a reference to its parent node, and when searching for the root node, we walk up the node's parent pointer until we find the root node, and connect all nodes along the way to the root node directly, which can significantly reduce the time complexity of subsequent lookup operations. Figure 6 is the c++ code for finding the root node to which x belongs after optimization.

```
int find(int x) {
    if (p[x] != x) {
        p[x] = find(p[x]);
    }
    return p[x];
}
```

Figure 6. Set Querying After Optimization

Through path compression optimization and lookup set, merging and lookup operations can be performed more efficiently, which is suitable for solving various practical problems, such as graph theory, network connection and so on. Path compression is one of the commonly used optimization techniques in the union and Find set algorithm, which improves the performance and reliability of the algorithm, and is widely used in practical projects.

3. Practical Application

3.1 Description of the Title

Taking Luogu problem P1551 as an example.

If a family has an excessively large number of members, determining whether two individuals are relatives can be quite challenging. Now, given a family relationship graph, the task is to determine whether any two given individuals are relatives.

Specification: If x and y are relatives, and y and z are relatives, then x and z are also relatives. If x and y are relatives, then all relatives of x are also relatives of y, and vice versa – all relatives of y are also relatives of x.

Input:

First line: Three integers n, m, p ($n, m, p \le 5000$), representing the number of people n, the number of relative relationships m, and the number of queries about relatives p.

The following m lines: Each line contains two numbers Mi, Mj, where $1 \le Mi, Mj \le n$, indicating that Mi and Mj are relatives.

Next p lines: Each line contains two numbers Pi, Pj, querying whether Pi and Pj are relatives.

Output:

p lines, each line containing either "Yes" or "No", indicating the answer to the ith query as either "having" or "not having" a relative relationship.

3.2 Problem Analysis

Each person can be viewed as a vertex in the graph, and when there is a relative relationship between two people, the relationship forms an edge connecting the two vertices. In this way, when given a set of relative relations, we naturally obtain a graph model consisting of n vertices and m edges. It is worth noting that the connected branches in the graph represent the set of individuals with relatives.

Given the transitivity of kinship relations, we

Copyright @ STEMM Institute Press

know that any two vertices within the same connected branch are relatives. This means that, for a given query, we only need to check whether two vertices are in the same connected component.

Using a traditional approach to this problem, one might first build a complete graph for these n vertices and m edges, and then look for connected components to make a judgment. However, this method not only requires a large amount of storage space to store m edges, but also is not efficient using regular traversal algorithms.

To solve this problem more efficiently, we can use the DSU method. Specifically, a separate set is created for each person, and these sets initially contain only that individual, indicating that no one initially knows whether they are related or not. Whenever a new relative relation is provided, we merge the two sets. In this way, we can obtain the current set relation in real time.

When we need to run a query, we simply check whether two elements of the current result belong to the same set. Since the DSU method maintains an efficient data structure during the merging process, this query operation can be done in constant time.

In general, by using DSU, we can process kinship queries more efficiently, especially when dealing with large datasets. This method not only saves the storage space, but also improves the query efficiency, making it possible to judge the relative relationship quickly and accurately in dynamic environment.

4. Conclusion

With the continuous development of programming competitions, more and more attention is paid to the in-depth study of algorithms, and the demand for algorithm efficiency is also increasing. In the competition, players are faced with a variety of complex problems, which often require efficient algorithms to solve. Therefore, the study of data structures and algorithms has become the core content of programming competitions.

Data structure and algorithm are not only the key to solve the problem, but also an important standard to measure a contestant's programming ability and thinking level. In programming competitions, participants need to be familiar with a variety of common data structures and algorithms, and be able to flexibly use them to solve practical problems. As an efficient data structure for processing set merging and set query, DSU shows excellent performance in solving some specific problems. DSU is a data structure designed to deal with dynamic set merging and set query problems. By grouping elements into different sets and maintaining a tree structure to represent these sets, DSU provides an efficient way to merge and query sets. By applying the path compression optimization strategy, the DSU can simultaneously update the nodes on the path during the merge process, which makes the subsequent lookup operation faster.

Path compression optimization is a common technique used to speed up the query and merge operations of a merge set. In the traditional union lookup set, when we need to merge two sets, we need to find the root node of both sets. This is an O (log n) operation. But in practice, we usually need to merge the same set multiple times, which leads to a lot of double computations. To solve this problem, we can use path compression optimization.

In DSU, the optimization strategy of path compression is to direct the parent node of a node to the root node, so that the subsequent search operations can directly reach the root node from the parent node, thus avoiding repeated search process. This optimization not only reduces the lookup time, but also makes the whole data structure more compact and improves the space efficiency.

A deep understanding of the principle and implementation details of DSU is important for solving real-world problems and improving algorithm design skills.

Acknowledgments

This work was supported by the 2023 Innovation and Entrepreneurship Training Program for College Students in Hunan Province of China under Grant (number S202310531040).

References

[1] Yonghui Wu and Jiande Wang. Algorithm Design Practice for Collegiate Programming Contests and Education. Routledge, 2018, 706-706.
[2] Clifford A. Shaffer. A Practical Introduction to Data Structures and Algorithm Analysis. Prentice Hall PTR,

 Copyright @ STEMM Institute Press

2000, 512-512.

[3] G. J. Chaitin. Algorithmic information theory. IBM Journal of Research and Development, 1977, 21(4):350-359.

[4] Galler, Bernard A. and Fisher, Michael J. An improved equivalence algorithm. Association for Computing Machinery, 1964, 7(5):301-303.

[5] Eleni Stroulia and Stan Matwin. Advances in Artificial Intelligence. Proceedings of the 14th Biennial Conference of the Canadian Society on Computational Studies of Intelligence, Berlin, Heidelberg, 2001.

[6] Chen, Qingyun, Laekhanukit, Bundit, Liao Chao et al. Survivable Network Design Revisited: Group-Connectivity. 2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS), 2022, 278-289.

[7] Chen X, Wang Y, Dong H, et al. Network Representation Learning Based On Random Walk Of Connection Number. International journal of innovative computing, information and control, 2022, 18(3):883-900.

[8] Hwang J H, Shin H U. Effects of Job Search Behavior Patterns on the Employment of Persons with Disabilities in Korea through Social Network Analysis. Journal of rehabilitation, 2023, 89(2):50-57.

[9] Patterson, C. L. and Buechler, Guenther. Digital image processing at the Aerospace Corporation. Computer, 1974, 7(5):46-52.

[10] Gundala, Laxmi Amulya and Spezzano, Francesca. A Framework for Predicting Links between Indirectly Interacting Nodes. 2018 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM), 2018, 544-551.

[11] Golumbic, Martin Charles and Shamir, Ron. Complexity and algorithms for reasoning about time: a graph-theoretic approach. Association for Computing Machinery, 2010, 40(5):1108-1133.