Union-Find Algorithm | Set 2 (Union By Rank and Path Compression)

In the previous post, we introduced *union find algorithm* and used it to detect cycle in a graph. We used following *union()* and *find()* operations for subsets.

```
// Naive implementation of find
int find(int parent[], int i)
{
    if (parent[i] == -1)
        return i;
    return find(parent, parent[i]);
}

// Naive implementation of union()
void Union(int parent[], int x, int y)
{
    int xset = find(parent, x);
    int yset = find(parent, y);
    parent[xset] = yset;
}
```

Run on IDE

The above *union()* and *find()* are naive and the worst case time complexity is linear. The trees created to represent subsets can be skewed and can become like a linked list. Following is an example worst case scenario.

```
Let there be 4 elements 0, 1, 2, 3

Initially, all elements are single element subsets.
0 1 2 3

Do Union(0, 1)
   1   2   3
  /
 0

Do Union(1, 2)
     2   3
    /
   1
```

```
 /
0

Do Union(2, 3)
         3
        /
      2
     /
   1
 /
0
```

The above operations can be optimized to *O(Log n)* in worst case. The idea is to always attach smaller depth tree under the root of the deeper tree. This technique is called ***union by rank***. The term *rank* is preferred instead of height because if path compression technique (we have discussed it below) is used, then *rank* is not always equal to height. Also, size (in place of height) of trees can also be used as *rank*. Using size as *rank* also yields worst case time complexity as O(Logn) (See this for proof)

```
Let us see the above example with union by rank
Initially, all elements are single element subsets.
0 1 2 3

Do Union(0, 1)
   1   2   3
  /
 0

Do Union(1, 2)
   1    3
 /  \
0    2

Do Union(2, 3)
    1
 /  |  \
0   2   3
```

The second optimization to naive method is ***Path Compression***. The idea is to flatten the tree when *find()* is called. When *find()* is called for an element x, root of the tree is returned. The *find()* operation traverses up from x to find root. The idea of path compression is to make the found root as parent of x so that we don't have to traverse all intermediate nodes again. If x is root of a subtree, then path (to root) from all nodes under x also compresses.

```
Let the subset {0, 1, .. 9} be represented as below and find() is called
for element 3.
              9
         /    |    \
        4     5      6
     /     \        /  \
```

```
    0        3     7    8
            /  \
           1    2

When find() is called for 3, we traverse up and find 9 as representative
of this subset. With path compression, we also make 3 as the child of 9 so
that when find() is called next time for 1, 2 or 3, the path to root is reduced.

               9
          /    /  \    \
         4    5    6     3
      /           /  \   /  \
     0           7    8  1   2
```

The two techniques complement each other. The time complexity of each operation becomes even smaller than O(Logn). In fact, amortized time complexity effectively becomes small constant.

Following is union by rank and path compression based implementation to find a cycle in a graph.

```
// A union by rank and path compression based program to detect cycle in a graph
#include <stdio.h>
#include <stdlib.h>

// a structure to represent an edge in the graph
struct Edge
{
    int src, dest;
};

// a structure to represent a graph
struct Graph
{
    // V-> Number of vertices, E-> Number of edges
    int V, E;

    // graph is represented as an array of edges
    struct Edge* edge;
};

struct subset
{
    int parent;
    int rank;
};

// Creates a graph with V vertices and E edges
struct Graph* createGraph(int V, int E)
{
    struct Graph* graph = (struct Graph*) malloc( sizeof(struct Graph) );
    graph->V = V;
    graph->E = E;

    graph->edge = (struct Edge*) malloc( graph->E * sizeof( struct Edge ) );

    return graph;
}

// A utility function to find set of an element i
// (uses path compression technique)
int find(struct subset subsets[], int i)
{
    // find root and make root as parent of i (path compression)
    if (subsets[i].parent != i)
        subsets[i].parent = find(subsets, subsets[i].parent);

    return subsets[i].parent;
```

```
}

// A function that does union of two sets of x and y
// (uses union by rank)
void Union(struct subset subsets[], int x, int y)
{
    int xroot = find(subsets, x);
    int yroot = find(subsets, y);

    // Attach smaller rank tree under root of high rank tree
    // (Union by Rank)
    if (subsets[xroot].rank < subsets[yroot].rank)
        subsets[xroot].parent = yroot;
    else if (subsets[xroot].rank > subsets[yroot].rank)
        subsets[yroot].parent = xroot;

    // If ranks are same, then make one as root and increment
    // its rank by one
    else
    {
        subsets[yroot].parent = xroot;
        subsets[xroot].rank++;
    }
}

// The main function to check whether a given graph contains cycle or not
int isCycle( struct Graph* graph )
{
    int V = graph->V;
    int E = graph->E;

    // Allocate memory for creating V sets
    struct subset *subsets =
        (struct subset*) malloc( V * sizeof(struct subset) );

    for (int v = 0; v < V; ++v)
    {
        subsets[v].parent = v;
        subsets[v].rank = 0;
    }

    // Iterate through all edges of graph, find sets of both
    // vertices of every edge, if sets are same, then there is
    // cycle in graph.
    for(int e = 0; e < E; ++e)
    {
        int x = find(subsets, graph->edge[e].src);
        int y = find(subsets, graph->edge[e].dest);

        if (x == y)
            return 1;

        Union(subsets, x, y);
    }
    return 0;
}

// Driver program to test above functions
int main()
{
    /* Let us create the following graph
         0
        |  \
        |    \
        1-----2 */

    int V = 3, E = 3;
    struct Graph* graph = createGraph(V, E);

    // add edge 0-1
    graph->edge[0].src = 0;
    graph->edge[0].dest = 1;
```

```
    // add edge 1-2
    graph->edge[1].src = 1;
    graph->edge[1].dest = 2;

    // add edge 0-2
    graph->edge[2].src = 0;
    graph->edge[2].dest = 2;

    if (isCycle(graph))
        printf( "Graph contains cycle" );
    else
        printf( "Graph doesn't contain cycle" );

    return 0;
}
```

Run on IDE

Output:

```
Graph contains cycle
```

Related Articles :

Union-Find Algorithm | Set 1 (Detect Cycle in a an Undirected Graph)

Disjoint Set Data Structures (Java Implementation)

Greedy Algorithms | Set 2 (Kruskal's Minimum Spanning Tree Algorithm)

Job Sequencing Problem | Set 2 (Using Disjoint Set)

References:

http://en.wikipedia.org/wiki/Disjoint-set_data_structure

IITD Video Lecture

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Improved By : DeepeshThakur

Practice Tags : Graph union-find

Article Tags : Graph union-find

Please write to us at contribute@geeksforgeeks.org to report any issue with the above content. Login to Improve this Article

Recommended Posts:

Disjoint Set (Or Union-Find) | Set 1 (Detect Cycle in an Undirected Graph)
Union-Find Algorithm | Set 2 (Union By Rank and Path Compression)
Detect Cycle in a Directed Graph
Detect cycle in an undirected graph
Minimum cost path from source node to destination node via an intermediate node
Print all the cycles in an undirected graph
Dominant Set of a Graph
Print the DFS traversal step-wise (Backtracking also)
Find alphabetical order such that words can be considered sorted
Undirected graph splitting and its application for number pairs

(Login to Rate)

3.5 Average Difficulty : **3.5/5.0**
Based on **74** vote(s)

Basic Easy Medium Hard Expert

Add to TODO List
Mark as DONE

Feedback Add Notes Improve Article

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Share this post!

anlian523 • 12 days ago

But I don't find the operation of Path Compression!There is something wrong i think. In the function Union,after xroot and yroot are found, there must be the operation of Path Compression,for instance:
if(xroot != subsets[x].parent){
subsets[x].parent = xroot
}
if(yroot != subsets[y].parent){
subsets[y.parent = yroot
}
But there are none!

Reply • Share ›

shail dave • 2 months ago

Why does it say "Graph contains cycle" for following:
https://ide.geeksforgeeks.o...

I did following changes:
- Changed V to 4
- Changed graph->edge[1].src = 1 to graph->edge[1].src = 30;

EDIT:

I think this solution has undefined behaviour when value of edge[i].src is greater than number of edges.

Can someone please confirm?

Reply • Share ›

rahul sharma • 5 months ago

THe time complexity of entire program is Logn or nlogn?

Reply • Share ›

Dhiraj Dwivedi ➚ rahul sharma • 3 months ago

Using a disjoint set forest with path compression and union-by-rank, you could get an asymptotic run time of O(E*α(n)) per operation, where α(n) is an extremely slowly-growing function (the Ackerman inverse function) that's essentially 5 for all inputs you could fit into the universe.
https://stackoverflow.com/q...

Reply • Share ›

Priyadarshini Mitra • 6 months ago

If you comment out :
graph->edge[2].src = 0;
graph->edge[2].dest = 2;

ie remove the edge between 0 and 2 it still says graph has cycle. The algorithm is not correct for detecting a cycle.

Reply • Share ›

Rounak Agarwal ➚ Priyadarshini Mitra • 5 months ago

if you are commenting out :
graph->edge[2].src = 0;
graph->edge[2].dest = 2;
you need to change the number of edges from 3 to 2, i.e., make int E=2 in int main()
The algorithm works perfectly fine then and gives the output that 'Graph doesn't contain cycle.'
You can check the working code by following the below link:
https://ide.geeksforgeeks.o...

Reply • Share ›

Shivan • 6 months ago

What is rank here ? I understand path compression but cant seem to figure out how rank is calculated

Reply • Share ›

Tanishq Mittal ➚ Shivan • a month ago

The more the rank, the more deeper the tree is.

Reply • Share ›

Varun Jain • 7 months ago

wouldn't the below code be better. we will be making node with more children the parent. therefore less recursive calls of find() for more children nodes?

```
// Attach smaller rank tree under root of high rank tree
// (Union by Rank)
if (subsets[xroot].rank < subsets[yroot].rank){
subsets[xroot].parent = yroot;
subsets[yroot].rank++;
}
else if (subsets[xroot].rank > subsets[yroot].rank){
subsets[yroot].parent = xroot;
subsets[xroot].rank++;
}
```

Reply • Share ›

Alexander Getman • 7 months ago

What if you use path compression without rank?

Like this:

```
int find(int parent[], int i)
{
if (parent[i] == -1)
return i;
parent[i]=find(parent, parent[i]);
return parent[i];
}
```

This will automatically flatten parents to all nodes "above" the 'i';
Since "union" operation also call "find", our structure will be always flat.

Reply • Share ›

Wenjing Mou ➚ Alexander Getman • 6 months ago

+1,

The rank is unnecessary with path compression since no matter parent is chosen, the next find is called on the parent, the tree will be flattened.

1 ^ v • Reply • Share ›

A computer science portal for geeks

710-B, Advant Navis Business Park,
Sector-142, Noida, Uttar Pradesh - 201305
feedback@geeksforgeeks.org