Union Find is all you need

Hyungmin Lee

Summary

This report presents an algorithm which make union and find operation for disjoint set efficiently and the weight-based heuristic for algorithm performance improvement.

Table of Contents

Introduction

This report deal with Union Find and the weight-based heuristic.

Union Find is an algorithm which performs Union and Find operation for disjoint set.

(Disjoint set represents intersection of sets is an empty set)

Also, the weight-based heuristic reduces execution count for Union operation.

Discussion

Union Find

Union Find performs Union and Find for disjoint set.

Simply, when intersection of A and B is empty set,

$A \cap B = \emptyset$

Does

$A \cup B$

and

$$\mathrm{find}(x) = \begin{cases} x, & p(x) = x, \\ \mathrm{find}(p(x)), & p(x) \neq x. \end{cases}$$

The weight-based heuristic

The weight-based heuristic starts from the idea that when there are disjoint set A and B, it will be more efficient that union A with B in the situation element number of A is more than B.

For this, we need number of child nodes for root nodes, so this code can be written like this:

```
if elem1 > elem2:
        parent[elem2] += parent[elem1]
        parent[elem1] = parent[elem2]
    else:
        parent[elem1] += parent[elem2]
        parent[elem2] = parent[elem1]
```

The path compress heuristic

The path compress heuristic can be simply told as an algorithm that make child node directly connected to root node.

```
elem1 = find(elem1)
elem2 = find(elem2)

parent[elem1] = parent[elem2]
```

directly connect to root node

```python
def find(elem):
    if elem < 0:
        return elem
    parent[elem] = find(parent[elem])
    return parent[elem]
```

compressed by finding root (if find called next time it will have time complexity of O(1))

Time complexity of Union Find

Time complexity of Union Find is often derived using inverse Ackermann function.

(In case used both the weight-based heuristic and the path compression heuristic to implement)

$$O(\alpha(m, n))$$

Inverse Ackermann function is considered as constant time because increment rate is very low.

so can be expressed like this:

$$O(1)$$

The reason why Time complexity of Union Find is often derived using inverse Ackermann function is

if we use the weight-based heuristic, height of set tree is limited by O(log n).

another reason is the path compress heuristic make child node directly connected to root node so set tree is more flatten.

Implementation

Used the weight-based heuristic.

```python
parent = [-1] * 100000

def find(elem):
    if elem < 0:
        return elem
    parent[elem] = find(parent[elem])
    return parent[elem]

def union(elem1, elem2):
    elem1 = find(elem1)
    elem2 = find(elem2)

    if elem1 > elem2:
        parent[elem2] += parent[elem1]
        parent[elem1] = parent[elem2]
    else:
        parent[elem1] += parent[elem2]
        parent[elem2] = parent[elem1]

union(1, 2)
union(3, 1)
print(find(1))
```

Application of algorithm

Removal of element in data structure like vector costs very expensive. (O(n))

So Union Find can make that process faster.

This is example of an implementation of a stock bid price and ask price matching program using Union Find.

```cpp
#include <bits/stdc++.h>

using namespace std;

vector<int> p, ask;
int n, matches = 0;
int i;

int find(int x){
    return p[x]==x ? x : p[x]=find(p[x]);
```

```
}

void unite(int a,int b){
    a=find(a);
    b=find(b);
    p[a]=b;
}

int main(){
    ios::sync_with_stdio(NULL);
    cin.tie(NULL);

    cin >> n;
    p.resize(n + 1);
    iota(p.begin(), p.end(), 0);
    for (i = 0; i < n; i++) {
        int tmp;
        cin >> tmp;
        ask.push_back(tmp);
    }

    sort(ask.begin(), ask.end());

    for(int i = 0; i < n; i++){
        int bid;
        cin >> bid;

        int pos = upper_bound(ask.begin(), ask.end(), bid) - ask.begin();
        int idx = find(pos);

        if(idx==0)
            continue;
        matches++;
        unite(idx, idx-1);
    }
    cout << matches << '\n';
}
```

Conclusion

Union Find is an efficient algorithm with time complexity of O(1), and can use weight heuristic and path compression heuristic to complement existing algorithms.

Also Union Find can accelerate