Union Find - Recitation 11

January 10, 2023

1 Union Find.

We have mentioned that for finding efficently the minimal spanning tree using kruskal, one has to answer quickly about wheter a pair of vertices v, u share the same connactivity commponent. In this recitation we will presnt a datastructure that will allow us both quering the belonging of given item and mergging groups at efficent time cost.

The problem define as follow. Given n items $x_1 ... x_n$ we would like to maintain the parttion of them into disjointss sets by supporting the following operations:

1. Make-Set(x) create an empty set whose only member is x. We could assume that this operation can be called over x only once.
2. Union(x, y) merge the set which contains x with the one which contains y.
3. Find-Set(x) returns a pointer to the set holding x.

Notice that the navie immpleamntion using pointres array, A, defined to store at place i a pointer to the set containing x can perform the Find-Set operation at $O(1)$. The bottle neck of that immplemntion is that the mergging will require from us to run over the whole itmes and changes their corresponding pointer at A one by one. Namly, a running time cost of $\Theta(n)$ time. Let's review a diffrent approch:

Linked Lists Immplemntation. One way to have a non-trival improvmeant is to associate for each set a linked list storing all the elemnents belonging to the set. Each node of those linked lists contains, additilly to it's value and it's sibling pointer, also a pointer for the list itself (the set). Consider again the mergging operation. it's clear that having those lists allow us to uinfind sets by iterating and updating only the elememnts belong to them. Still one more trick is needed for achiving a good running cost.

Uinon(x, y)

if *size* $A[x] \geq$ *size*$A[y]$ **then**
    size $A[x] \leftarrow$ size $A[x]$ + size $A[y]$
    for $z \in A[y]$ **do**
        $A[z] \leftarrow A[x]$
    $A[x] \leftarrow A[x] \cup A[y]$ // $O(1)$ concatenation of linked lists.
else
    Union (y, x)

Clearly, executing the above over sets at linear size require at least linear time. Let's anlayze what happens when mergging n times. As we have allredy seen at graphs, runtime can be measured by counting the total number of operations that each item/vertex do along the whole running. So we can ask our selfes how many times does an item change his location and his set pointer. Assume that at the time when x were changed $A[x]$ containd (before the mergging) t elemenets then immdetly after that $A[x]$ will store at least $2t$ elements. In otherwords

$$\text{size}A^{(t+1)}[x] \leftarrow \text{size}A^{(t)}[x] + \text{size}A^{(t)}[y] \geq 2A^{(t)}[x]$$

Union By Rank.

Path Compression. Let's analyses the cost of queries m times by counting the edges on which the algorithm went over. Let's denote by $\text{find}(v^{(t)})$ the query which requestef at time t and let $P^{(t)} = v, v_2..v_k$ be the vertices path on which the algorithm climbe from v up to his root. Now, observes that by comperssing the path the ranks of the verticis in P must be distincit. Now consdier any parttion of the line into buckets $B_i = [b_i, b_{i+1}]$. $\sum_v \sum_{B_i} \sum_{v \to u \ r[v]=r[u]} 1 \leq \sum_v \sum_B |B|$

$$\begin{aligned}
T(n,m) &= \text{ direact parent move } + \text{ climbing moves } = \\
&= \text{ direact parent move } + \text{ stage exchange } + \text{ inner stage } = \\
&\leq m + m \cdot |\mathcal{B}| + \sum_{B \in \mathcal{B}} \text{ steps inside B} \\
&\leq m + m \cdot |\mathcal{B}| + \sum_{B \in \mathcal{B}} \sum_{u \in B} \text{ steps inside B started at } u \\
&\leq m + m \cdot |\mathcal{B}| + \sum_{B \in \mathcal{B}} \sum_{u \in B} |B| \\
&\leq m + m \cdot |\mathcal{B}| + \sum_{B \in \mathcal{B}} \frac{n}{\min B} |B|
\end{aligned}$$