Union Find - Recitation 11

January 9, 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* $A[x] \geq sizeA[y]$ **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)

Union By Rank.