

7 Dynamic Data Structures

Why do we need dynamic data structures?

- large data sets to be searched

- symbol tables

- online data rather than static data

Size of the data structure?

- array doubling

- expandable in increments

- fixed size, but much smaller than the full data set

7.1 Hash Tables

Good for inverting large sets of keys without sorting

- Good for present/absent tests

- Distinct from, but not unrelated to, hashing for cryptography.

7.1.1 Closed Address Hashing

7.1.2 Open Address Hashing

rehashing

- linear probing

- quadratic probing

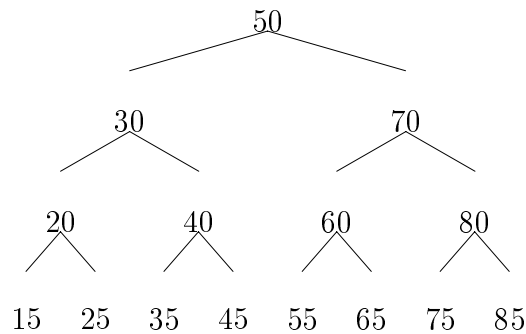
- deletions

- multiplicative congruential random number generation

7.2 Search Trees and Online Structures

Definition 7.1. *A binary tree in which the nodes have **keys** from an ordered set has the **binary search tree property** if the key at each node is greater than all the keys in its left subtree and less than or equal to all keys in its right subtree. In this case the tree is called a **binary search tree**.*

It would be nice to be able to create balanced trees in an online situation, but we can't always do that.



If we had to do dynamic insertion, we could wind up with this.



7.3 The Union Find Algorithm

Definition 7.2. *An equivalence relation is a binary relation R on a set S that is*

- *symmetric: $aRb \implies bRa$*
- *reflexive: aRa is true*
- *transitive: aRb and $bRc \implies aRc$*

Examples:

Equality of elements

Membership in a set

Membership in a cycle/cliq/connected component in a graph

The **Union-Find** algorithm is an algorithm for constructing a data structure with online data for which testing the equivalence relation is “fast.”

We start with a graph (G, E) of nodes G and edges E . Each node starts as the root of its own tree.

The **find** operation chases pointers from a node until it finds the root.

If we issue a **find** for two nodes g and h , and we chase pointers to different roots, and yet these two nodes are supposed to be equivalent, then we do a **union** operation to join the two trees together into one.

7.3.1 Example: Set Membership

Let’s assume that we have sets $\{A, B, C, D, E, F, G, H\}$ and $\{J, K, L\}$ to represent in a data structure. If we issue “edges” $AB, BC, CD, DE, EF, FG, GH, JK, KL$ in order, then we get the structures:

A *B* *C* *D* *E* *F* *G* *H* *I* *J* *K* *L*
| | | | | | | | | | |

then

A *C* *D* *E* *F* *G* *H* *I* *J* *K* *L*
| | | | | | | | | |

B

then

A *D* *E* *F* *G* *H* *I* *J* *K* *L*
| | | | | | | | |

B
|

C

then

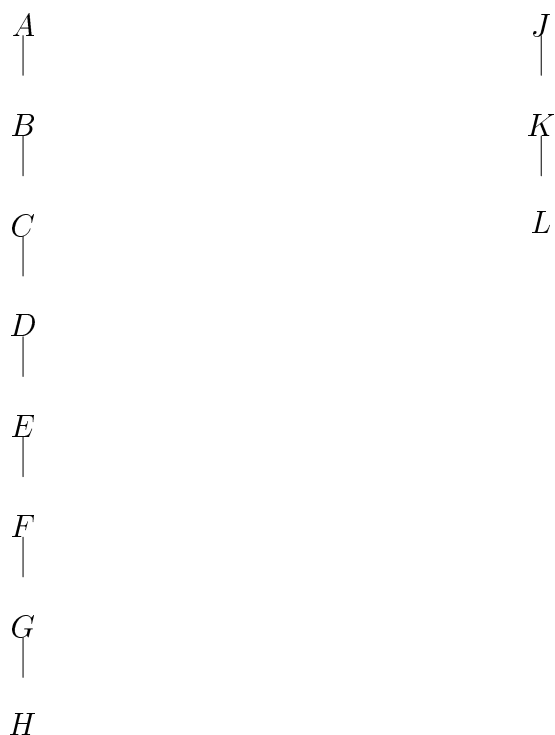
A *E* *F* *G* *H* *I* *J* *K* *L*
| | | | | | | |

B
|

C
|

D

and eventually

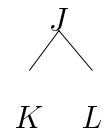
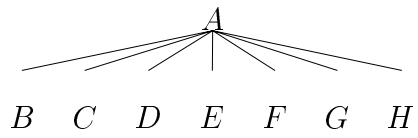


This “works” but is clearly inefficient. We chase more pointers than is perhaps necessary to get from G to A . If instead we presented “edges” $AB, CD, EF, GH, AC, EG, AE, JK, JL$, then we might get



which would have better depth searching (“find”) characteristics. And if searching was a major priority, and this was a set operation so that member-

ship in the tree was the highest priority issue, then probably a flat structure would be best:



7.3.2 Example: Cycles in graphs

Now consider a graph with nodes

$$\{A, B, C, D, E, F, G, H\}$$

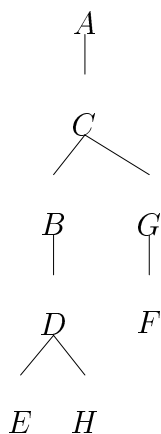
and edges

$$\{AC, BC, BD, CD, CG, DE, DG, DH, FG, GH\}.$$

This graph has cycles with nodes $BCD, CDG, CGH, GCDG, CDGH, BCDGH$.

We can generate a **spanning set of cycles** such that the **sums of cycles** comprise all the cycles by using a union-find algorithm.

If we present the edges as above, and do no path compression, then the final union-find tree for this connected component of the graph will be as follows.



As we add edges, when we try to add CD , this would produce a cycle (we can tell this because the “find” will return the same root for the C node and the D node. Instead of adding the edge, we write out the cycle and

discard the edge. Similarly, when we try to add the DG edge, we will get the cycle $BCDG$, and when we try to add the GH edge, we will get the cycle $BCDGH$.

This gives a set of three cycles. There are three sums of three cycles, and so six cycles in all.

Issues: We want low depth, but sometimes we also want the structure of the graph to be preserved. It is relevant to consider the number of nodes and number of edges, as well as the time to build the forest of trees compared with the time to search the forest of trees. And many variations exist on pointer jumping techniques for improving the “find” process.