Special data structures

Special data structures

- Numbers, sequences, sets, trees etc. are general data types that appear in many applications.
- Many special data structures arise in specific applications.
- Large variety of them – consider a small sample.
- Some data structures required for efficient implementation of specific algorithms.

Heaps

- A heap is a multiset of objects of some type *T* with a strict weak order < defined.
- More restricted than set, no find or erase operation.
- Allows insert and deletion of the maximum element, that is an element x such that $x < y$ is false for all elements y in the heap.
- Also called a priority queue, element with highest priority is deleted first.

Heaps

- Can be implemented using balanced binary search trees.
- Simpler implementation possible using only a vector.
- Although worst-case time is *O(log n)* for both, works faster in practice.

Complete binary tree

- A complete binary tree is a binary tree whose nodes can be numbered *1* to *n* such that
  - Root is numbered *1*.
  - Left child of node *i* is numbered *2i*, if *2i* <= *n*.
  - Right child of node *i* is numbered *2i+1*, if *2i+1* <= *n*.
- Height of tree is *O(log n)*.
- Represented by a vector with i^{th} element corresponding to node numbered *i*.

Heap

- Elements in a heap are stored in the nodes of a complete binary tree, which is just a vector.
- Elements satisfy the heap property

  value in a node is NOT < value in any child.

- Strict weak ordering implies root is NOT < any value (why?)
- Maximum is always at the root.

Insertion

- Heap with n nodes represented by a vector v of size $n+1$, $v[0]$ is not used.
- Add the new element x at the back of the vector.
- If node $(n+1)/2$ (the parent of $n+1$) has a value NOT $< x$, then stop, otherwise
- Shift value in node $(n+1)/2$ to node $n+1$, and try to place x in node $(n+1)/2$.
- Repeat till x is placed, perhaps in the root.

Delete Maximum

- Maximum is always at the root.
- Copy the value x in the last node n to node 1 and delete the last element.
- If x < the value in any of children of the node currently containing x, swap it with the child containing the larger value.
- Repeat for the child node, and continue until there is no child with value < x.

Making a heap

- Heaps can be used for sorting a vector.
- First modify the vector to satisfy heap property.
- Repeatedly use delete maximum to sort.
- Conversion to heap can be done by inserting one element at a time – $O(n \log n)$ time.
- Instead, recursively convert left and right subtrees, and use the procedure in delete maximum for root.
- Takes $O(n)$ time – only $n/2^i$ elements may be compared i times.

Tries

- A data structure for representing sets of strings.
- Assume character set is small.
- Time for insert, find, erase depends only on the length of the string, independent of the size of the set.
- Useful for phone numbers, dictionary etc.

Tries

- Rooted tree in which each non-leaf node has as many subtrees as number of characters, typically 10 (digits) or 26 (letters).
- Character can be used as an index in array.
- Subtree corresponding to a particular character is the trie obtained from the subset of strings that start with this character, after deleting it.

Tries

- To insert, erase, find a string in a trie, start with root node, use the first character to access corresponding subtree and apply the operation using the remaining string to the subtree.
- Every node corresponds to a string, defined by the path from root to the node.
- A boolean value used to indicate whether the node corresponds to a string in the set or not.
- Node corresponding to every prefix of a string in the set.
- Memory requirement is $O(l)$, where l is the sum of lengths of strings in the set, but the constant may be large.

Hashing

- Alternative approach to implementing sets.
- Worst case can be bad, but works better than balanced trees on the average.
- Simple to implement.
- No order relation required on elements, only a mapping to integers.

Hashing

- Extend the idea of bit vectors.
- Subsets of *{0,1,…,n-1}* can be represented by a boolean vector of size *n*.
- Insert, erase, find are all *O(1)* time operations.
- Cannot be used for finite subsets of integers, since there is no bound on possible values of integers.
- Hashing – Map arbitrary values to bounded numbers.

Hashing

- Store elements of the set in an array of size *B*.
- Elements of the array are called buckets, and the array itself is called a hash table.
- Hash function : $h(i) \rightarrow \{0,1,\ldots,B\text{-}1\}$.
- Integer *i* will be stored in bucket *h(i)*.
- A bucket may have to store more than one element.
- Collision occurs when two elements are hashed to the same bucket.

Closed hashing

- A list of elements whose hash value is i is stored in the ith bucket.
- To insert, erase, find an element i, the list in bucket $h(i)$ is searched sequentially.
- If the set has n elements, each bucket will on the average contain n/B elements.
- If B is $\Omega(n)$, this will be $O(1)$, but worst case can be O(n).

Open hashing

- Alternative ways of handling collisions.
- Store actual elements in the array itself and a boolean value to indicate whether a location is empty or not.
- To insert *i*, if location *h(i)* is not empty, start sequentially searching to the right from *h(i)* till an empty location is found, wrap around if needed- called linear probing.
- To handle erase, locations need to be separately marked as deleted.
- Find for *i* may need to keep searching from *h(i)* to the right till either *i* or an empty location is found, cannot stop at a deleted location.

Open hashing

- Linear probing leads to bunching of elements in the array, increasing the search time.
- Alternative ways for searching locations
  - Quadratic probing, search in order $h(i) \pm j^2$.
  - A second hash function, search in order $h(i) + jh_1(i)$.
  - Prime number preferred as bucket size.
- Dynamically increase number of buckets and rehash all elements.

Segment trees

- Special type of binary trees..
- Useful for operations on segments (substrings) of a sequence.
- Easy to implement using only vectors.
- Avoid the use of balanced trees for algorithmic problems.

Segment trees

- Suppose a_0, a_1, …, a_{n-1} is a sequence of elements and assume $n = 2^d$.
- Construct a binary tree whose nodes represent some substrings of the sequence.
- $n/2^i$ nodes at depth d-i, corresponding to substrings of length 2^i.
- The jth node at depth d-i-1 has the $2j$th node at depth d-i as the left child and $(2j+1)$th node as the right child, and represents the concatenation of substrings corresponding to the children.

Segment trees

- Every element in the sequence belongs to *d+1* substrings that correspond to nodes.
- Every substring written as the disjoint union of at most *2d* substrings that correspond to nodes.
- Keeping information for only these substrings may be sufficient to compute it for all substrings.
- Minimum in a substring with updates.
  - Store minimum of corresponding substring in each node.
  - An update may change only O(log n) values.
  - Minimum of any substring obtained by taking minimum of O(log n) values.

Summary

- Many other special data structures.
- Designed for particular applications/ algorithms.
- Improving efficiency is the main goal.
- Worst case analysis does not necessarily indicate actual performance.
- Standard data structures are sufficient in many cases.