Data Structures

• What is a "data structure"?

• Operations:
  • Initialize
  • Insert
  • Delete
  • Search
  • Min/max
  • Successor/Predecessor
  • Merge
Arrays

• Sequence of "indexible" locations

\[
\begin{array}{ccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\end{array}
\]

• Unordered:
  • O(1) to add
  • O(n) to search
  • O(n) for min/max

• Ordered:
  • O(n) to add
  • O(log n) to (binary) search
  • O(1) for min/max
Stacks

- LIFO (last-in first-out)

- Operations: push/pop (O(1) each)

- Can not access "middle"

- Analogy: trays at Cafeteria

- Applications:
  - Compiling / parsing
  - Dynamic binding
  - Recursion
  - Web surfing
Queues

- FIFO (first-in first-out)

- Operations: push/pop (O(1) each)
- Can not access "middle"
- Analogy: line at your Bank
- Applications:
  - Scheduling
  - Operating systems
  - Simulations
  - Networks
Linked Lists

• Successor pointers

• Types:
  • Singly linked
  • Doubly linked
  • Circular

• Operations:
  • Add: O(1) time
  • Search: O(n) time
  • Delete: O(1) time (if known)
Trees

- Parent/children pointers

- Binary/N-ary

- Ordered/unordered

- Height-balanced:
  - AVL
  - B-trees
  - Red-black
  - $O(\log n)$ worst-case time
Tree Traversals

- **pre-order:**  1) process node  
  2) visit children  
  ⇒  c  b  a  e  d  f

- **post-order:**  1) visit children  
  2) process node  
  ⇒  a  b  d  f  e  c

- **in-order:**  1) visit left-child  
  2) process node  
  3) visit right-child  
  ⇒  a  b  c  d  e  f
Heaps

- A tree where all of a node’s children have smaller “keys”
- Can be implemented as a binary tree
- Can be implemented as an array

Operations:
- Find max: $O(1)$ time
- Add: $O(\log n)$ time
- Delete: $O(\log n)$ time
- Search: $O(n)$ time
Hash Tables

- Direct access
- Hash function
- Collision resolution:
  - Chaining
  - Linear probing
  - Double hashing
- Universal hashing
- $O(1)$ average access
- $O(n)$ worst-case access

Q: How can worst-case access time be improved to $O(\log n)$?
Fact: almost half of all CPU cycles are spent on sorting!!

- Input: array $X[1..n]$ of integers
  Output: sorted array

- Decision tree model

**Thm:** Sorting takes $\Omega(n \log n)$ time

**Pf:** $n!$ different permutations

$\Rightarrow$ decision tree has $n!$ leaves

$\Rightarrow$ tree height is: $\log(n!)$

$> \log((n/e)^n)$

$= \Omega(n \log n)$
Sort Properties

• Worst case?
• Average case?
• In practice?
• Input distribution?
• Randomized?
• Stability?
• In-Situ?
• Stack depth?
• Internal vs. external?
• **Bubble Sort:**

For k=1 to n
   For i=1 to n-1
      If X[i+1]>X[i]
         Then Swap(X,i,i+1)

⇒ $\Theta(n^2)$ time

• **Insertion Sort:**

For i=1 to n-1
   For j=i+1 to n
      If X[j]>X[i] Then Swap(X,i,j)

⇒ $\Theta(n^2)$ time
• **Quicksort:**

Quicksort(X,i,j)

  If i<j Then p=Partition(X,i,j)

    QuickSort(X,i,p)

    QuickSort(X,p+1,j)

⇒O(n log n) time (ave-case)

• C.A.R. Hoare, 1962
• **Good news:** usually best in practice
• **Bad news:** worst-case O(n²) time
• Usually avoids worst-case
• Only beats O(n²) sorts for n>40
• **Merge Sort:**

MergeSort(X,i,j)

if i<j then \[ m = \lfloor (i+j)/2 \rfloor \]

MergeSort(X,i,m)

MergeSort(X,m+1,j)

Merge(X,i,m,j)

\[ T(n) = 2 \ T(n/2) + n \]

\[ \Rightarrow \Theta(n \log n) \text{ time} \]

• **Heap Sort:**

InitHeap

For i=1 to n HeapInsert(X(i))

For i=1 to n M=HeapMax

Print(M)

HeapDelete(M)

\[ \Rightarrow \Theta(n \log n) \text{ time} \]
• **Counting Sort:**

Assumes integers in small range 1..k

For i = 1 to k C[i] = 0
For i = 1 to k C[X[i]]++
For i = 1 to k
   If C[i] > 0 Then print(i) C[i] times

⇒ \( \Theta(n) \) time (worst-case)

• **Radix Sort:**

Assumes d digits in range 1..k

For i = 1 to d StableSort(X on digit i)

⇒ \( O(dn + kd) \) time (worst-case)
• **Bucket Sort:**

Assumes uniform inputs in range 0..1

For i=1 to n

Insert $X[i]$ into Bucket $\lfloor n \cdot X[i] \rfloor$

For i=1 to n \textbf{Sort} Bucket i

Concat contents of Buckets 1 thru n

$\Rightarrow O(n)$ time (expected)

$O(\textbf{Sort})$ time (worst)
Order Statistics

- **_exact** comparison count
- Minimum element

\[ k = X[1] \]

For \( i = 2 \) to \( n \)

If \( X[i] < k \) Then \( k = X[i] \)

\( \Rightarrow \) \( n-1 \) comparisons

**Thm:** Min requires \( n-1 \) comparisons.

**Proof:**
• **Min and Max:**

  (a) Compare all pairs  
  (b) Find Min of min’s of all pairs  
  (c) Find Max of max’s of all pairs

⇒ \( n/2 + n/2 + n/2 = 3n/2 \) comparisons

**Thm:** Min&Max require 3\(n/2\) comparisons.  
**Pf:** Represent known info by four sets:

<table>
<thead>
<tr>
<th>Unknown</th>
<th>Not Min</th>
<th>Not Max</th>
<th>Neither</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
</tr>
</tbody>
</table>

Initial: \(n\) 0 0 0  
Final: 0 1 1 n-2

Track movement of elements between sets.
Effect of comparisons:

<table>
<thead>
<tr>
<th>Origin</th>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;</td>
<td>&gt;</td>
</tr>
<tr>
<td>A&amp;A</td>
<td>C&amp;B</td>
</tr>
<tr>
<td>A&amp;B</td>
<td>C&amp;B</td>
</tr>
<tr>
<td>A&amp;C</td>
<td>C&amp;D</td>
</tr>
<tr>
<td>A&amp;D</td>
<td>C&amp;D</td>
</tr>
<tr>
<td>B&amp;B</td>
<td>D&amp;B</td>
</tr>
<tr>
<td>B&amp;C</td>
<td>D&amp;D</td>
</tr>
<tr>
<td>B&amp;D</td>
<td>D&amp;D</td>
</tr>
<tr>
<td>C&amp;C</td>
<td>C&amp;D</td>
</tr>
<tr>
<td>C&amp;D</td>
<td>C&amp;D</td>
</tr>
<tr>
<td>D&amp;D</td>
<td>D&amp;D</td>
</tr>
</tbody>
</table>

- Going from A to D forces passing through B or C
- "Emptying" A into B&C takes n/2 comparisons (1)
- "Almost emptying" B takes n/2-1 comparisons (2)
- "Almost emptying" C takes n/2-1 comparisons (3)
- Other moves will not reach the "final state" faster
- Total comparisons required: 3n/2-2
Problem: Find Max and next-to-Max using least # of comparisons.
Selection

- Not harder than median-finding (why?)
- Randomized $i^{th}$-Selection
  (return the $i^{th}$-largest element in $X[p..r]$)

Select($X, p, r, i$)
  If $p = r$ Then Return($X[p]$)
  $q =$ RandomPartition($X, p, r$)
  $k = q - p + 1$
  If $i \leq k$ Then Return(Select($X, p, q, i$))
    Else Return(Select($X, q + 1, r, i - k$))

$\Rightarrow \ O(n)$ time (ave-case)
Deterministic $i^{th}$-Selection

[Blum, Floyd, Pratt, Rivest, Tarjan; 1973]

- Partition input into $n/5$ groups of 5 each
- Compute median of each group
- Compute median of medians (recursively)
- Compute median of medians (recursively)
- Eliminate $3n/10$ elements & recurse on rest

$$T(n) = T(n/5) + T(7n/10) + O(n)$$
$$= T(2n/10) + T(7n/10) + O(n)$$
$$\leq T(9n/10) + O(n) \text{ since } T(n) = \Omega(n)$$

$\Rightarrow T(n) = O(n)$
Problem: Find in $O(n)$ time the majority element (i.e., occurring $\geq n/2$ times, if any).

a) Using "<", ">", "="

b) Using "=" only (i.e., no "order")
Minimum Spanning Trees
Prim’s MST Algorithm

\[ T = v_0 \]

**Until** \( T \) spans all nodes **do**

- **Select** nodes \( x \in T, \ y \notin T \)
  - w/min cost\((x,y)\)
- **Add** edge \((x,y)\) to \( T \)

**Return** \( T \)

- Time complexity: \( O(E \log E) \)
- Kruskal: \( O(E \log V) \)
- Fibonacci heaps: \( O(E + V \log V) \)
Shortest Paths Trees
Dijkstra’s Single-Source Shortest paths Algorithm

\[ T = v_0 \]

**Until** \( T \) spans all nodes **do**

- **Select** nodes \( x \in T, y \notin T \) w/min cost\((x,y) + \text{dist}(v_0,x)\)
- **Add** edge \((x,y)\) to \( T \)

**Return** \( T \)

- **Time complexity:** \( O(V^2) \)
- **All pairs:** \( O(V^3) \)
Cost-Radius Tradeoffs


Signal delay $\uparrow \Rightarrow$ Performance $\downarrow$

- Source $\rightarrow$ sink pathlength $\propto$ delay
  
  $\Rightarrow$ Avoid long paths

- Capacitive delay / building cost
  
  $\Rightarrow$ Minimize total wirelength
Possible Trees

MST:

SPT:

?
Definitions

Input: pointset with distinguished source

**ptset radius** $R$: max source-sink dist

**tree radius**: max source-sink pathlength
Problem Formulation

Given a pointset $P$, $\varepsilon \geq 0$, find min-cost tree $T$ with $r(T) \leq (1 + \varepsilon) \cdot R$

**Tradeoff:** $\varepsilon$ trades off radius and tree cost

$\varepsilon = 0 \Rightarrow \text{“Shortest Path Tree”}$

$\varepsilon = \infty \Rightarrow \text{Minimum Spanning Tree}$
Arbitrary $\varepsilon \Rightarrow$ hybrid construction

- Unifies Prim and Dijkstra!
Bounded Radius MSTs

**Goal:** cost \( \approx \) cost(MST)

radius \( \approx \) r(SPT)

- Let \( Q = \) MST
- Let \( L \) be tour of MST:
• **Traverse** $L$

• $A = \text{running total of edge costs}$

• **If** $A > \varepsilon \cdot R$ **Then** $A = 0$

$$Q = Q \cup \text{minpath}_G(s,L_i)$$

• **Final routing** tree is $SPT_Q$
\[ \text{dist}_G(s, v_i) \leq R \]

\[ \text{L} = \text{MST tour} \]

\[ \text{minpath}_G(s, v_i) \leq R \]

\[ \text{dist}_T(s, v) \leq \text{dist}_G(s, v_i) + \text{dist}_L(v_i, v) \]

\[ \leq R + \varepsilon \cdot R = (1 + \varepsilon) \cdot R \]

\[ \Rightarrow r(T) \leq (1 + \varepsilon) \cdot R \]
\[
\text{dist}_L(v_i, v_{i+1}) \leq \varepsilon R
\]

\[
\text{minpath}_G(s, v_{i+1}) \leq R
\]

\[
\text{minpath}_G(s, v_i) \leq R
\]

\[
\text{cost}(T) \leq \text{cost}(\text{MST}_G) + \frac{\text{cost}(L)}{\varepsilon \cdot R} \cdot R
\]

\[
= \text{cost}(\text{MST}_G) + \frac{2 \cdot \text{cost}(\text{MST}_G)}{\varepsilon}
\]

\[
= (1 + \frac{2}{\varepsilon}) \cdot \text{cost}(\text{MST}_G)
\]

\[
\Rightarrow \text{cost}(T) \leq (1 + \frac{2}{\varepsilon}) \cdot \text{cost}(\text{MST}_G)
\]
Bounded Radius MST Algorithm

Compute $\text{MST}_G$ and $\text{SPT}_G$

$E' = \text{edges of } \text{MST}_G$

$Q = (V, E')$

$L = \text{depth-first tour of } \text{MST}_G$

$A = 0$

For $i = 2$ to $|L|$

\[ A = A + \text{cost}(L_{i-1}, L_i) \]

If $A > \varepsilon \cdot R$ Then

\[ E' = E' \cup \text{minpath}_G(s, L_i) \]

$A=0$

$T = \text{SPT}_Q$

Input: $G=(V,E)$, source $s$, radius $R$, $0 \leq \varepsilon$

Output: $T$ = routing tree with

\[ \text{cost}(T) \leq \left(1 + \frac{2}{\varepsilon}\right) \cdot \text{cost}(\text{MST}_G) \]

\[ r(T) \leq (1 + \varepsilon) \cdot R \]
Steiner Trees
Bounded Radius Steiner Trees

Given weighted graph $G=(V,E)$, node subset $N$, source $s \in N$, and $0 \leq \varepsilon$, find min-cost tree $T$ spanning $N$, with $r(T) \leq (1+\varepsilon) \cdot r(N)$

- NP-complete
Bounded Radius Steiner Trees

- Can use *any* low-cost spanning tree

- Use [KMB, 1981] to span N (cost $\leq 2 \cdot \text{opt}$)

- Run previous algorithm

$$\Rightarrow \text{cost}(T) \leq 2 \cdot (1 + \frac{2}{\varepsilon}) \cdot \text{opt}$$
Geometry Helps

- Add Steiner points when \( A = 2\varepsilon \cdot R \)

- Use bounds on MST/Steiner ratio

<table>
<thead>
<tr>
<th>Tree type</th>
<th>Graph type</th>
<th>Radius bound</th>
<th>Cost bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>spanning</td>
<td>arbitrary</td>
<td>((1+\varepsilon) \cdot R)</td>
<td>((1+2/\varepsilon) \cdot \text{MST})</td>
</tr>
<tr>
<td>Steiner</td>
<td>arbitrary</td>
<td>((1+\varepsilon) \cdot R)</td>
<td>(2 \cdot (1+2/\varepsilon) \cdot \text{opt})</td>
</tr>
<tr>
<td>Steiner</td>
<td>Manhattan</td>
<td>((1+\varepsilon) \cdot R)</td>
<td>(\frac{3}{2} (1+1/\varepsilon) \cdot \text{opt})</td>
</tr>
<tr>
<td>Steiner</td>
<td>Euclidean</td>
<td>((1+\varepsilon) \cdot R)</td>
<td>(\frac{2}{\sqrt{3}} \cdot (1+1/\varepsilon) \cdot \text{opt})</td>
</tr>
</tbody>
</table>
Experimental Results

The figure shows two graphs. The top graph plots \(\frac{r(T)}{r(MST)}\) against net size. The bottom graph plots \(\frac{\text{cost}(T)}{\text{cost}(MST)}\) against net size. The graphs are labeled with different values of \(\epsilon\), where \(\epsilon = 0.25, 1.00, 2.00\).