Wrap-up: Kruskal’s Algorithm and Union Find

- **Kruskal’s Algorithm**: Sort the edges by increasing weight and keep adding the next edge that doesn’t complete a cycle.
- **Union Find**: Data structure to maintain connected components of growing spanning tree. Should support the following operations:
  - Find($v$): return name of set containing $v$
  - Union($A$, $B$): merge two sets

\[
\text{for each edge } e \text{ do}
\]
\[
\text{Let } u\text{ and } v\text{ be endpoints of } e
\]
\[
\text{if } \text{find}(u) \neq \text{find}(v) \text{ then } \text{▷ Not in same component?}
\]
\[
T = T \cup \{e\}
\]
\[
\text{Union(\text{find}(u), \text{find}(v))} \text{ ▷ Merge components}
\]
\[
\text{end if}
\]
\[
\text{end for}
\]

Simple Implementation of Union-Find

- Each disjoint set is stored as a linked list of nodes where each node consists of three data items:
  - name of element
  - “label” pointer to label of the set
  - “next” pointer to next node in list
- There are three basic operations:
  - Make-Set($v$): Takes $O(1)$ time to add a single node.
  - Find($v$): Takes $O(1)$ time to follow pointer to label.
  - Union-Set($u$, $v$): $O$ (size of smaller set).
    - Update “next” pointer at end of longer list to point to start of shorter list
    - Update “label” pointers of shorter list to point to label of other list
    - Update auxiliary pointers and size information

Union-Find Analysis

**Theorem**: Consider a sequence of $m$ operations including $n$ Make-Set operations. Total running time is $O(m + n \log n)$.

- Total time from Find and Make-Set: $O(m)$
- Total time from Union: $O(n \log n)$
- Updating next pointers: $O(n)$
- Updating label pointers: $O(n \log n)$ because the label pointer for a node can be updated at most $\log_2 n$ times.

Hence, Kruskal’s algorithm can be implemented in time
\[
O(m \log m) + O(m + n \log n) = O(m \log m)
\]

Other Greedy Problems

- Huffman Coding and data compression
- Minimum Cost Arborescence (e.g., MST in directed graphs)

Algorithm Design Techniques

- Greedy
- Divide and Conquer
- Dynamic Programming
- Network Flows
Comparison

<table>
<thead>
<tr>
<th></th>
<th>Greedy</th>
<th>Divide and Conquer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Formulate problem</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>Design algorithm</td>
<td>easy</td>
<td>hard</td>
</tr>
<tr>
<td>Prove correctness</td>
<td>hard</td>
<td>easy</td>
</tr>
<tr>
<td>Analyze running time</td>
<td>easy</td>
<td>hard</td>
</tr>
</tbody>
</table>

Divide and Conquer: Recipe

- Divide problem into several parts
- Solve each part recursively
- Combine solutions to sub-problems into overall solution

- Common example
  - Problem of size $n \rightarrow$ two parts of size $n/2$.
  - Combine solutions in $O(n)$ time.

Example: Mergesort

```mergesort
MergeSort(Arr)
if length(Arr) ≤ 2 then  # Base case
  Sort however you like, return sorted list.
else
  middle = length(Arr)/2  # Recursive Steps
  L = MergeSort(Arr[0:middle])
  R = MergeSort(Arr[middle:length(Arr)])
  Return Merge(L, R)      # Combine Step
end if
```

Example: Mergesort

```mergesort
MergeSort(Arr)
if length(Arr) ≤ 2 then . # Base case
  Sort however you like, return sorted list.
else
  middle = length(Arr)/2 . # Recursive Steps
  L = MergeSort(Arr[0:middle])
  R = MergeSort(Arr[middle:length(Arr)])
  Return Merge(L, R) . # Combine Step
end if
```

Mergesort Running time

- Base Case: $O(1)$.
- Recursive step: $O(1) + ???$
- Merge step: $O(n)$.

Recurrence Relations

Let $T(n)$ be running time for inputs of length $n$.

$$T(n) \leq 2T(n/2) + cn \quad \text{when } n \geq 2$$

$$T(0), T(1), T(2) \leq c$$

How do we solve for $T(n)$?

Solving recurrences

- Unravel recurrence
- Guess and check

Mergesort runtime: $O(n \log n)$.

Maximum Subsequence Sum (MSS)

**Input**: array $A$ of $n$ numbers

**Find**: value of the largest subsequence sum


(Note: empty subsequence ($j < i$) is allowed and has sum zero)
What is a simple algorithm for MSS?

Remember Homework 1?

MSS(A)
Initialize all entries of $n \times n$ array $B$ to zero
for $i = 1$ to $n$
do
  sum = 0
  for $j = i$ to $n$
do
    sum += $A[j]$
    $B[i, j] = sum$
  end for
endoor
Return maximum entry of $B[i, j]$

Running time? $O(n^2)$. Can we do better?

Divide-and-conquer for MSS

Recursive solution for MSS

Idea:
- Find MSS $L$ in left half of array
- Find MSS $R$ in right half of array
- Find MSS $M$ for sequence that crosses the midpoint
Return $\max(L, R, M)$

MSS(Arr)
if length(Arr) == 1 then  ▶ Base case
  return $\max(A[0], 0)$
end if
mid = length(Arr)/2
L = MSS(Arr[0:mid]), R = MSS(Arr[mid:length(Arr)])  ▶ Recursive Steps
Set sum = 0, $L’ = 0$.
for $i = mid-1$ down to 0 do
  sum += Arr[i], $L’ = \max(L’, sum)$.
endoor
Set sum = 0, $R’ = 0$.
for $i = mid$ up to length(Arr)-1 do
  sum += Arr[i], $R’ = \max(R’, sum)$.
endoor
return $\max(L, R, L’ + R’)$.

More recurrences

- Problem of size $n \rightarrow k$ parts of size $n/2$.
- Combine solutions in $O(n)$ time.

Recurrence

$$T(n) \leq kT(n/2) + cn, \quad T(1) \leq c.$$  

Qualitatively different behavior $k = 1, k = 2$, and $k > 2$.

- If $k = 1$, $T(n) = O(n)$.
- If $k = 2$, $T(n) = O(n \log n)$.
- If $k > 2$, $T(n) = O(n^{\log_2(k)})$. 

MSS running time

- Base Case: $O(1)$.
- Recursive step: $O(1) + ???$
- Merge step: $O(n)$.

Recurrence:

$$T(n) \leq 2T(n/2) + cn, \quad T(1) \leq c.$$  

Solves to $O(n \log n)$ just like Mergesort.