Greedy may not work but may still be effective

Solving the items in decreasing order of the profit/weight ratio:

\[ y_1, y_2, \ldots, y_{k-1}, y_k \]

\[
\max \left[ \sum_{j=1}^{k-1} w(y_j), w(y_k) \right]
\]

Best possible fractional knapsack

Solution is

\[
\sum_{j=1}^{k-1} \left( w(y_j) + w(y_k) \right) + \left( w(y_k) \right)
\]

\[
\text{OPT of original problem} \leq \text{OPT'} \ (\text{fractional knapsack}) \leq A + B
\]

\[
\Rightarrow \quad \text{OPT} \leq 2 \max \{A, B\}
\]

\[
\Rightarrow \quad \frac{\text{OPT}}{2} \leq \max \{A, B\}
\]

Guarantee:

\[
\frac{\text{Our solution}}{\text{Best solution}}
\]
H.W. : Without the term B, construct a counterexample so that only the term A may not be even 10% of OPT

Matching Problem

\[ Z \]

Greedy can give 50% guarantee

Approximation Algorithm is an important field

Implementation of Kruskal's algorithm

How do you test for cycles?

Observation: If an edge goes across two trees, then we can add it else it creates a cycle
Find: Given a vertex which tree contains this vertex?

- If the endpoints $(u, v)$ belong to different trees, then we must join $T_1, T_2$.

Union Find data structure

Given subsets $S_1, S_2, ..., S_k$, we want to maintain a data structure that supports the following operations:

- $\text{Find}(x)$: return a set $S_i$ such $x \in S_i$.
- $\text{Union}(S'_1, S'_2)$: returns $S'_1 \leftarrow S'_2 \cup S'_3$ (and implicitly destroys $S_2, S_3$).
We want to design a data structure that is efficient for a sequence of union and find operations.

For Kruskal’s algorithm, 2m finds and n-1 unions.

We will focus on disjoint union find possibilities.

1. Each set is a linked list of members.
   - Find could be \( O(n) \)
   - Union \( O(1) \)

2. For every element, we can keep the set identification.

<table>
<thead>
<tr>
<th>( x_1 )</th>
<th>( x_2 )</th>
<th>( x_3 )</th>
<th>( \ldots )</th>
<th>( x_n )</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>5</td>
<td>10</td>
<td>5</td>
<td>1</td>
</tr>
</tbody>
</table>

Find is \( O(1) \) - array look up.

Union: \( O(n) \) - change the labels of the elements involved.

When we do union, let us change the labels of the elements in the smaller set.
The cost of a sequence of at most \( n-1 \) union operations can be changed to the number of label changes over all elements. Let \( \eta(x) \) be the no. of label changes incurred by \( x \) over the entire sequence of \( n-1 \) unions.

Then \( \text{cost of } n-1 \text{ unions} \leq \sum_{x} \eta(x) \)

**Claim** \( \eta(x) \leq \log n \)

For every label change \( x \) is in a set which is at least twice the size of the previous set.

So total cost of \( n-1 \) unions \( \leq n \cdot \log n \)

Total cost of \( m \) finds and \( n-1 \) unions is bounded by \( O(m + n \log n) \)