Greedy Methods

**EASY KNAPSACK**

**Instance:** \( n \) items where item \( i \) has weight \( w_i \) and cost \( c_i \) and an integer \( W \).

**Problem:** Determine a fraction \( f_i \) of item \( i \), so that

\[
\sum_{i=1}^{n} f_i \cdot w_i \leq W \quad \text{and} \quad \sum_{i=1}^{n} f_i \cdot c_i \quad \text{is maximum.}
\]

**Greedy Solution:** Take as much of each item as possible in order of decreasing cost/weight ratio.

The greedy solution has the form:

\[(1,1,1,1,\ldots,1,1,g_k,0,0,0,\ldots,0,0)\] where \( 0 < g_k \leq 1 \).

**Claim:** The greedy solution is always optimal.

**Proof:** Let \( (f_1, f_2, \ldots, f_n) \) be an optimal solution.

**Case 1:** \( \forall 1 \leq i \leq n, \ g_i \leq f_i \)

\[W \geq \sum_{i=1}^{n} f_i \cdot w_i = \sum_{i=1}^{k} f_i \cdot w_i + \sum_{i=k+1}^{n} f_i \cdot w_i = \sum_{i=1}^{k} g_i \cdot w_i + (f_i - g_i) \cdot w_i + \sum_{i=k+1}^{n} f_i \cdot w_i = W + (f_i - g_i) \cdot w_i + \sum_{i=k+1}^{n} f_i \cdot w_i \]

\[\Rightarrow f_i = g_i \quad \text{and} \quad \forall k+1 \leq i \leq n, \ f_i = 0 \quad \Rightarrow \quad \forall 1 \leq i \leq n, \ f_i = g_i\]
ACTIVITY SELECTION

Instance: A set $S$ of $n$ activities with start and end times, $(s_i, e_i)$.

Problem: Select a largest set of non-overlapping activities.

Greedy Choice: Select the activity with earliest end time and discard any activities that conflict with it.
Activities are sorted by non-decreasing end time.

**Input:** Start \( S[1:n] \) and end \( E[1:n] \) times.

**Output:** \( A[1:n] \) \( A[i]=1 \) if activity \( i \) is selected else 0

\[
i=1
\]

while (\( i \leq n \))

\[
A[i]\leftarrow 1; \quad t\leftarrow E[i] \\
i\leftarrow i+1
\]

while (\( i \leq n \) and \( S[i] < t \))

\[
A[i]\leftarrow 0 \\
i\leftarrow i+1
\]

endwhile

endwhile

---

**Claim:** The greedy choice yields an optimal solution to the activity selection problem.

**Proof:** By induction on \( n=|S| \).

**Base:** \( n=0 \). Then the only solution is \( \emptyset \). So the greedy solution (\( \emptyset \)) is optimal.

**Step:** \( n>0 \). Let \( G \) be the set of activities selected by the greedy solution and let \( A \) be an optimal solution.

Let \( i \) be the first activity in \( A \). Let \( A'=A\setminus\{i\} \).

Every activity in \( A' \) starts after \( e[i] \geq e[1] \).

Let \( S' \) be the subset of activities that start after \( e[1] \).

\( G'=G\setminus\{1\} \) is an optimal solution to \( S' \) by induction.

\( A' \) is a solution to \( S' \) since every activity in \( A' \) starts after \( e[i]\geq e[1] \).

\( |G'|\geq |A'| \) since \( G' \) is an optimal solution to \( S' \).

\( |G|\geq |G'|+1 \geq |A'|+1 = |A| \), so \( G \) is an optimal solution to \( S \).  

QED
To show greedy is optimal:

**Method 1:** Characterize the “Greedy Solution”.
Show any optimal solution is no better than the “greedy solution.”

**Method 2:** Characterize the “Greedy Choice” and remaining subproblem.
Show that there is an optimal solution made up of the greedy choice and an optimal solution to the remaining subproblem.

**EASY KNAPSACK**

**ACTIVITY SELECTION, MINIMUM SPANNING TREE, HUFFMAN CODING**

**MINIMUM SPANNING TREE**

Given a graph, \( G = (V,E) \) with edge weights \( w(e) \), find a minimum weight spanning tree of \( G \).

\[
V = \{v_1,v_2,\ldots,v_n\} \quad E = \{e_1,e_2,\ldots,e_m\}
\]

\( T \subseteq E \) induces a spanning tree of \( G \),
if the graph \( (V,T) \) is connected and acyclic.

**MST LEMMA**

If \( A \subseteq E \) and \( \exists \) a minimum spanning tree \( T \) with \( A \subseteq T \),

\( V \) can be partitioned into \( S \) and \( V-S \) such that no edge in \( A \) spans \( S \) and \( V-S \),

and \( e \) is an edge of minimum weight between \( S \) and \( V-S \).

then \( A \cup \{e\} \) is also a subset of a minimum spanning tree.

\[
V-S \quad 3 \quad 4 \quad 8 \quad 6
\]

\( S \)

**Proof:**

**Case 1:** \( e \in T \). Then \( A \cup \{e\} \subseteq T \).

**Case 2:** \( e \notin T \). \( e = (u,v) \)

Since \( e \) spans \( S \) and \( V-S \), \( u \in S \) and \( v \in V-S \) (or vice versa).

Then \( T \cup \{e\} \) has a cycle since \( u \) and \( v \) are connected in \( T \).

This cycle must have another edge \( e' \) that spans \( S \) and \( V-S \).

\( e' \notin A \) since it spans \( S \) and \( V-S \) and \( w(e) \leq w(e') \).

Let \( T' = (T \cup \{e\})-\{e'\} \).

\( T' = (T \cup \{e\})-\{e'\} \) is acyclic and connected, \( (T \cup \{e\} \) was connected and had only one cycle containing \( e' \).
Proof: (cont)

So \( T' \) is a spanning tree.

\[
\text{cost}(T') = \text{cost}(T) - w(e') + w(e) \leq \text{cost}(T) \text{ since } w(e) \leq w(e').
\]

So \( T' \) is a minimum spanning tree.

Since \( e' \notin A, \ A \subseteq T - \{e'\} \).

So \( A \cup \{e\} \subseteq (T \cup \{e\}) - \{e'\} = T' \), a minimum spanning tree.

QED

---

**Kruskal MST**

\( A \leftarrow \emptyset \)

for each vertex \( u \)

\( \text{MakeSet}(u) \)

endfor

\( H \leftarrow \text{BuildHeap}(E) \)

while \( |A| < |V| - 1 \)

\( (u,v) \leftarrow \text{ExtractMin}(H) \)

If \( \text{FindSet}(u) \neq \text{FindSet}(v) \)

then \( A \leftarrow A \cup \{(u,v)\} \)

\( \text{Union}(u,v) \)

endwhile

\( \text{BuildHeap } O(m) \)

\( m \text{ ExtractMin's } O(m \log m) \)

\( O(m) \text{ UNION-FIND ops. } O(m \log^* n) \)

Total \( = O(m \log m) \)

---

**Prim MST**

\( A \leftarrow \emptyset \)

\( H \leftarrow \text{MakeHeap}() \)

for each vertex \( u \)

\( \text{key}(u) \leftarrow +\infty; \)

\( n[u] \leftarrow \text{nil} \)

\( \text{Insert}(H, u) \)

endfor

\( \text{DecreaseKey}(r, 0) \)

while \( |A| < |V| - 1 \)

\( u \leftarrow \text{ExtractMin}(H) \)

\( A \leftarrow A \cup \{(u, n[u])\} \)

for each edge \( (u,v) \)

if \( v \in H \text{ and } \text{key}[v] > w(u,v) \)

then \( \text{DecreaseKey}(v, w(u,v)) \)

\( \text{n[v]} \leftarrow u \)

endfor

endwhile

---

**Kruskal’s**

- Use UNION-FIND to maintain connected components

**Prim’s**

- Let \( S \) be the component of a fixed vertex, all others are singletons

- For each vertex not in \( S \) keep track of cheapest edge to \( S \)

---

**Complexity**

<table>
<thead>
<tr>
<th>Operation</th>
<th>Kruskal’s</th>
<th>Prim’s</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n \text{ Insert's} )</td>
<td>( O(lg n) )</td>
<td>( O(1) )</td>
</tr>
<tr>
<td>( m \text{ DecreaseKey's} )</td>
<td>( O(lg n) )</td>
<td>( O(1) )</td>
</tr>
<tr>
<td>( n \text{ ExtractMin’s} )</td>
<td>( O(lg n) )</td>
<td>( O(lg n) )</td>
</tr>
<tr>
<td>Total</td>
<td>( O(m \log n) )</td>
<td>( O(m + n \log n) )</td>
</tr>
</tbody>
</table>