The Greedy Approach

General idea:

Given a problem with $n$ inputs, we are required to obtain a subset that maximizes or minimizes a given objective function subject to some constraints.

- **Feasible solution** — any subset that satisfies some constraints
- **Optimal solution** — a feasible solution that maximizes or minimizes the objective function
procedure Greedy \((A, n)\)

\[
\text{begin} \\
\text{solution} \leftarrow \emptyset; \\
\text{for } i \leftarrow 1 \text{ to } n \text{ do} \\
\begin{align*}
\text{x} & \leftarrow \text{Select (A)}: \quad \text{// based on the objective} \\
& \quad \text{// function} \\
\text{if Feasible (solution, x),} \\
\text{then solution} & \leftarrow \text{Union (solution, x)};
\end{align*}
\text{end;}
\]

Select: A greedy procedure, based on a given objective function, which selects input from \(A\), removes it and assigns its value to \(x\).

Feasible: A boolean function to decide if \(x\) can be included into solution vector (without violating any given constraint).

About Greedy method

The \(n\) inputs are ordered by some selection procedure which is based on some optimization measures.

It works in stages, considering one input at a time. At each stage, a decision is made regarding whether or not a particular input is in an optimal solution.
1. **Minimum Spanning Tree** (For Undirected Graph)

   *The problem:*
   1) Tree

   A *Tree* is connected graph with no cycles.

   2) Spanning Tree

   A *Spanning Tree* of $G$ is a tree which contains all vertices in $G$.

   Example:

   ![Graph G](image)

   b) Is $G$ a Spanning Tree?

   ![Yes/No](image)

   Key: Yes  
   Key: No

   Note: Connected graph with $n$ vertices and exactly $n - 1$ edges is Spanning Tree.

3) Minimum Spanning Tree

   Assign weight to each edge of $G$, then *Minimum Spanning Tree* is the Spanning Tree with minimum total weight.
Example:

a) Edges have the same weight

\[ G: \]

---

- DFS (Depth First Search)

---

Greedy
- **BFS (Breadth First Search)**

- **DFS**

**b) Edges have different weights**

\[ G: \]

\[ \text{Cost} = 16 + 5 + 10 + 14 + 33 = 78 \]
BFS

\[ \text{Cost} = 16 + 19 + 21 + 5 + 6 = 67 \]

Minimum Spanning Tree (with the least total weight)

\[ \text{Cost} = 16 + 5 + 6 + 11 + 18 = 56 \]

Algorithms:

1) **Prim's Algorithm** (Minimum Spanning Tree)

**Basic idea:**

Start from vertex 1 and let \( T \leftarrow \emptyset \) (\( T \) will contain all edges in the S.T.); the next edge to be included in \( T \) is the minimum cost edge \((u, v)\), s.t. \( u \) is in the tree and \( v \) is not.

**Example:** \( G \)
(Spanning Tree) S.T. \{ 1 \} -

S.T. \{ \}

S.T. \{ \}

S.T. \{ \}

S.T. \{ \}

Cost = 16 + 5 + 6 + 11 + 18 = 56

Minimum Spanning Tree

\( (n - \# \text{ of vertices}, e - \# \text{ of edges}) \)

It takes \( O(n) \) steps. Each step takes \( O(e) \) and \( e \leq n(n-1)/2 \Rightarrow O(n^2) \).

Therefore, it takes \( O(n^2) \) time.

With clever data structure, it can be implemented in \( O(n^2) \).
2) **Kruskal’s Algorithm**

*Basic idea:*

Don’t care if $T$ is a tree or not in the intermediate stage, as long as the including of a new edge will not create a cycle, we include the minimum cost edge.

Example:

**Step 1:** Sort all of edges

- $(1,2) \ 10 \ \checkmark$
- $(3,6) \ 15 \ \checkmark$
- $(4,6) \ 20 \ \checkmark$
- $(2,6) \ 25 \ \checkmark$
- $(1,4) \ 30 \ \times \ reject \ : \ create \ cycle$
- $(3,5) \ 35 \ \checkmark$

**Step 2:** $T$

1. $\{ 1-2 \}$
2. $\{ 1-2, 3-6 \}$
3. $\{ 1-2, 3-6-4 \}$
4. $\{ 1-2, 3-6-4, 5 \}$
How to check:
adding an edge will create a cycle or not?

If Maintain a set for each group
(initially each node represents a set)
Ex: set1 set2 set3

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

\[\therefore \text{ new edge } 2 \quad 6 \]

from different groups \(\Rightarrow\) no cycle created

Data structure to store sets so that:

i. The group number can be easily found, and

ii. Two sets can be easily merged

---

**Kruskal’s algorithm**

While (T contains fewer than n-1 edges) and (E \(\neq \emptyset\) ) do
Begin
Choose an edge (v,w) from E of lowest cost;
Delete (v,w) from E;
If (v,w) does not create a cycle in T
then add (v,w) to T
else discard (v,w);
End;

With clever data structure, it can be implemented in \(O(e \log e)\).
So, complexity of Kruskal is \( O(e \log e) \)

\[ e \leq \frac{n(n-1)}{2} \implies \log e \leq \log n^2 = 2\log n \]  
\[ \implies O(e \log e) = O(e \log n) \]

3) Comparing Prim’s Algorithm with Kruskal’s Algorithm

i. Prim’s complexity is \( O(n^2) \)

ii. Kruskal’s complexity is \( O(e \log n) \)

if \( G \) is a complete (dense) graph, \n    Kruskal’s complexity is \( O(n^2 \log n) \)

if \( G \) is a sparse graph, \n    Kruskal’s complexity is \( O(n \log n) \).

2. Dijkstra’s Algorithm for Single-Source Shortest Paths

The problem: Given directed graph \( G = (V, E) \),
  a weight for each edge in \( G \),
  a source node \( v_0 \),

Goal: determine the (length of) shortest paths from \( v_0 \) to all the
  remaining vertices in \( G \)

Def: Length of the path: Sum of the weight of the edges

Observation:

- May have more than 1 paths between \( w \) and \( x \) (\( y \) and \( z \))
- But each individual path must be minimal length
- (in order to form an overall shortest path from \( V_0 \) to \( V_i \))
Notation

cost adjacency matrix $Cost$, $\forall 1 \leq a, b \leq |V|$

$Cost(a, b) = \begin{cases} 
\text{cost from vertex } i \text{ to vertex } j & \text{if there is a edge} \\
0 & \text{if } a = b \\
\infty & \text{otherwise}
\end{cases}$

$s(w) = \begin{cases} 
1 & \text{if shortest path } (v_0, w) \text{ is defined} \\
0 & \text{otherwise}
\end{cases}$

$Dist(j)$ $\forall j$ in the vertex set $V$

$= \text{the length of the shortest path from } v_0 \text{ to } j$

$From(j) = i$ if $i$ is the predecessor of $j$ along the shortest path from $v_0$ to $j$

Example:

![Graph example](image)

a) Cost adjacent matrix

$$
\begin{bmatrix}
0 & 50 & 10 & \infty & 45 & \infty \\
\infty & 0 & 15 & \infty & 10 & \infty \\
20 & \infty & 0 & 15 & \infty & \infty \\
\infty & 20 & \infty & 0 & 35 & \infty \\
\infty & \infty & \infty & 30 & 0 & \infty \\
\infty & \infty & \infty & \infty & 3 & \infty
\end{bmatrix}
$$
b) Steps in Dijkstra’s Algorithm

1. $\text{Dist} (v_0) = 0$, $\text{From} (v_0) = v_0$
2. $\text{Dist} (v_2) = 10$, $\text{From} (v_2) = v_0$

3. $\text{Dist} (v_3) = 25$, $\text{From} (v_3) = v_2$
4. $\text{Dist} (v_1) = 45$, $\text{From} (v_1) = v_3$
5. Dist \( (v_4) = 45 \), From \( (v_4) = v_0 \)  

6. Dist \( (5) = \infty \)

c) Shortest paths from source \( v_0 \)

\[
\begin{align*}
    v_0 &\rightarrow v_2 \rightarrow v_3 \rightarrow v_1 & 45 \\
    v_0 &\rightarrow v_2 & 10 \\
    v_0 &\rightarrow v_2 \rightarrow v_3 & 25 \\
    v_0 &\rightarrow v_4 & 45 \\
    v_0 &\rightarrow v_5 & \infty
\end{align*}
\]
**Dijkstra's algorithm:**

```plaintext
procedure Dijkstra (Cost, n, v, Dist, From)
    // Cost, n, v are input, Dist, From are output
    begin
    for i ← 1 to n do
        s(i) ← 0;
        Dist(i) ← Cost(v, i);
        From(i) ← v;
        s(v) ← 1;
    for num ← 1 to (n – 1) do
        choose u s.t. s(u) = 0 and Dist(u) is minimum;
        s(u) ← 1;
        for all w with s(w) = 0 do
            if (Dist(u) + Cost(u, w) < Dist(w))
                Dist(w) ← Dist(u) + Cost(u, w);
                From(w) ← u;
    end;
```

**Ex:**

```
Ex:

a) Cost adjacent matrix

<table>
<thead>
<tr>
<th></th>
<th>01</th>
<th>02</th>
<th>03</th>
<th>04</th>
<th>05</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>50</td>
<td>30</td>
<td>100</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>∞</td>
<td>0</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>3</td>
<td>∞</td>
<td>5</td>
<td>0</td>
<td>50</td>
<td>∞</td>
</tr>
<tr>
<td>4</td>
<td>∞</td>
<td>20</td>
<td>∞</td>
<td>0</td>
<td>∞</td>
</tr>
<tr>
<td>5</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>10</td>
<td>0</td>
</tr>
</tbody>
</table>
```

**Greedy**
b) Steps in Dijkstra’s algorithm

1. $\text{Dist}(1) = 0$, $\text{From}(1) = 1$
2. $\text{Dist}(5) = 10$, $\text{From}(5) = 1$

3. $\text{Dist}(4) = 20$, $\text{From}(4) = 5$
4. $\text{Dist}(3) = 30$, $\text{From}(3) = 1$
5. $\text{Dist}(2) = 35$, $\text{From}(2) = 3$

Shortest paths from source 1:
- $1 \rightarrow 3 \rightarrow 2$: 35
- $1 \rightarrow 3$: 30
- $1 \rightarrow 5 \rightarrow 4$: 20
- $1 \rightarrow 5$: 10
3. Optimal Storage on Tapes

The problem:

Given \( n \) programs to be stored on tape, the lengths of these \( n \) programs are \( l_1, l_2, \ldots, l_n \) respectively. Suppose the programs are stored in the order of \( i_1, i_2, \ldots, i_n \).

Let \( t_j \) be the time to retrieve program \( i_j \).

Assume that the tape is initially positioned at the beginning.

\( t_j \) is proportional to the sum of all lengths of programs stored in front of the program \( i_j \).

Ex:

<table>
<thead>
<tr>
<th>Order</th>
<th>Total Retrieval Time</th>
<th>MRT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5+(5+10)+(5+10+3)=38</td>
<td>38/3</td>
</tr>
<tr>
<td>2</td>
<td>5+(5+3)+(5+3+10)=31</td>
<td>31/3</td>
</tr>
<tr>
<td>3</td>
<td>10+(10+5)+(10+5+3)=43</td>
<td>43/3</td>
</tr>
<tr>
<td>4</td>
<td>10+(10+3)+(10+3+5)=41</td>
<td>41/3</td>
</tr>
<tr>
<td>5</td>
<td>3+(3+5)+(3+5+10)=29</td>
<td>29/3</td>
</tr>
<tr>
<td>6</td>
<td>3+(3+10)+(3+10+5)=34</td>
<td>34/3</td>
</tr>
</tbody>
</table>

Note: The problem can be solved using greedy strategy, just always let the shortest program goes first.

( Can simply get the right order by using any sorting algorithm)
**Analysis:**

Try all combinations: \( O(n!) \)

Shortest-length-First Greedy method: \( O(n \log n) \)

**Shortest-length-First Greedy method:**
Sort the programs s.t. \( l_1 \leq l_2 \leq \ldots \leq l_n \)
and call this ordering \( L \).

Next is to show that the ordering \( L \) is the best

**Proof by contradiction:**
Suppose Greedy ordering \( L \) is not optimal, then there exists some other permutation \( I \) that is optimal.

\[ I = (i_1, i_2, \ldots, i_n) \]

\[ \exists a < b, \text{ s.t. } l_{i_a} > l_{i_b} \quad \text{(otherwise } I = L) \]

Interchange \( i_a \) and \( i_b \) in and call the new list \( I' \):

\[ I \]

\[ \ldots \quad i_a \quad i_{a+1} \quad i_{a+2} \quad \ldots \quad i_b \quad \ldots \]

\[ I' \]

\[ \ldots \quad i_b \quad i_{a+1} \quad i_{a+2} \quad \ldots \quad i_a \quad \ldots \]

In \( I' \), Program \( i_{a+j} \) will take less \((l_{i_a} - l_{i_b})\) time than in \( I \) to be retrieved.
In fact, each program \( i_{a+1}, \ldots, i_{b-1} \) will take less \((l_{i_a} - l_{i_b})\) time.
For \( i_b \), the retrieval time decreases \( x + l_{i_a} \)
For \( i_{a} \), the retrieval time increases \( x + l_{i_b} \)

\[
\text{totalRT}(I) - \text{totalRT}(I') = (b-a-1)(l_{i_a} - l_{i_b}) + (x + l_{i_b}) - (x + l_{i_a})
\]

\[
= (b-a)(l_{i_a} - l_{i_b}) > 0 \quad \text{Contradiction!!}
\]

Therefore, greedy ordering \( L \) is optimal
4. Knapsack Problem

The problem:
Given a knapsack with a certain capacity M, 
\( n \) objects, are to be put into the knapsack, 
each has a weight \( w_1, w_2, \cdots, w_n \) and 
a profit if put in the knapsack \( P_1, P_2, \cdots, P_n \).

The goal is find \( (x_1, x_2, \cdots, x_n) \) where \( 0 \leq x_i \leq 1 \)
s.t. \( \sum_{i=1}^{n} p_i x_i \) is maximized and \( \sum_{i=1}^{n} w_i x_i \leq M \)

Note: All objects can break into small pieces 
or \( x_i \) can be any fraction between 0 and 1.

Example:
\( n = 3 \)
\( M = 20 \)
\( (w_1, w_2, w_3) = (18,15,10) \)
\( (p_1, p_2, p_3) = (25,24,15) \)

Greedy Strategy#1: Profits are ordered in nonincreasing order \((1,2,3)\)

\( (x_1, x_2, x_3) = (1, \frac{2}{15}, 0) \)

\[ \sum_{i=1}^{3} p_i x_i = 25 \times 1 + 24 \times \frac{2}{15} + 15 \times 0 = 28.2 \]
Greedy Strategy#2: Weights are ordered in nondecreasing order \((3,2,1)\)

\[
(x_1, x_2, x_3) = (0, \frac{2}{3}, 1)
\]

\[
\sum_{i=1}^{3} p_i x_i = 25 \times 0 + 24 \times \frac{2}{3} + 15 \times 1 = 31
\]

Greedy Strategy#3: \(p/w\) are ordered in nonincreasing order \((2,3,1)\)

\[
\begin{align*}
\frac{p_1}{w_1} &= \frac{25}{18} = 1.4 \\
\frac{p_2}{w_2} &= \frac{24}{15} = 1.6 \quad \Rightarrow (2,3,1) \\
\frac{p_3}{w_3} &= \frac{15}{10} = 1.5 \\
(x_1, x_2, x_3) &= (0, \frac{1}{2}) \\
\sum_{i=1}^{3} p_i x_i &= 25 \times 0 + 24 \times 1 + 15 \times \frac{1}{2} = 31.5
\end{align*}
\]

Optimal solution

Analysis:
Sort the \(p/w\), such that \(\frac{p_1}{w_1} \geq \frac{p_2}{w_2} \geq \cdots \geq \frac{p_n}{w_n}\)
Show that the ordering is the best.

Proof by contradiction:
Given some knapsack instance
Suppose the objects are ordered s.t. \(\frac{p_1}{w_1} \geq \frac{p_2}{w_2} \geq \cdots \geq \frac{p_n}{w_n}\)

let the greedy solution be \(X = (x_1, x_2, \ldots, x_n)\)

Show that this ordering is optimal
Case 1: \(X = (1,1,\ldots,1)\) it’s optimal
Case 2: \(X = (1,1,\ldots,x_j,0,\ldots,0)\) s.t. \(\sum_{i=1}^{n} w_i x_i = M\)

where \(0 \leq x_j \leq 1\)
Assume $X$ is not optimal, and then there exists $Y = (y_1, y_2, \ldots, y_n)$

\[
s.t. \sum_{i=1}^{n} p_i y_i > \sum_{i=1}^{n} p_i x_i \quad \text{and } Y \text{ is optimal}
\]

examine $X$ and $Y$, let $y_k$ be the $1^{st}$ one in $Y$ that $y_k \neq x_k$.

\[
X = (x_1, x_2, \ldots, x_{k-1}, x_k, \ldots, x_n)
\]

\[
Y = (y_1, y_2, \ldots, y_{k-1}, y_k, \ldots, y_n)
\]

\[
y_k \neq x_k \implies y_k < x_k
\]

Now we increase $y_k$ to $x_k$ and decrease as many of $(y_{k+1}, \ldots, y_n)$ as necessary, so that the capacity is still $M$.

\[
\sum_{i=1}^{n} p_i z_i = \sum_{i=1}^{n} p_i y_i + (z_k - y_k) \cdot p_k - \sum_{i=k+1}^{n} (y_i - z_i) \cdot p_i
\]

\[
\therefore \quad \frac{p_k}{w_k} \geq \frac{p_i}{w_i} \quad \forall i \geq k + 1
\]

\[
\therefore \quad p_i \leq \left( \frac{p_k}{w_k} \right) \cdot w_i
\]

\[
\sum_{i=1}^{n} p_i z_i \geq \sum_{i=1}^{n} p_i y_i + \left[ (z_k - y_k) \cdot w_k - \sum_{i=k+1}^{n} (y_i - z_i) \cdot w_i \right] \cdot \frac{p_k}{w_k} = \sum_{i=1}^{n} p_i y_i
\]
So,

\[
\text{if } \text{profit}(z) > \text{profit}(y) \quad (\rightarrow \leftarrow) \n\]

\[
\text{else } \text{profit}(z) = \text{profit}(y) \n\]

(Repeat the same process.
At the end, \(Y\) can be transformed into \(X\).
\(\Rightarrow X\) is also optimal.
Contradiction! \(\rightarrow \leftarrow\) )