0-1 Knapsack Problem

Given items \( x_1, x_2, \ldots, x_n \) with weights \( w_1, w_2, \ldots, w_n \) and profits \( p_1, p_2, \ldots, p_n \), and a knapsack of capacity \( B \), we want to maximize the profit \( \sum x_i \cdot p_i \) subject to \( \sum w_i \cdot x_i \leq B \) (\( B, w_i \)'s are integral).

Let \( F(i, j) \) denote the maximum profit that we can obtain from \( x_1, x_2, \ldots, x_i \), and a knapsack of capacity \( j \leq B \).

In this notation, \( F(n, B) \) is the desired solution.
\[ F(1, 1) = \begin{cases} \text{if } w_1 \leq 1 \text{ then} & \text{profit } = p_1, \\ 0 & \text{otherwise} \end{cases} \]

\[ F(1, j) = \begin{cases} \text{if } w_1 \leq j \text{ then } p_1, \\ 0 & \text{else} \end{cases} \]

\[ F(2, j) = \begin{cases} \text{if } w_1 + w_2 \leq j \text{ then } p_1 + p_2, \\ 0 & \text{else if } \ldots \ldots \ldots \end{cases} \]

\[ F(3, j) = \begin{cases} \text{either soln includes } x_2 & \phi_2 + F(1, j - w_2) \\ \text{soln doesn't include } x_2 & F(1, j) \end{cases} \]

\[ F(i, j) = \max \left\{ \phi_i + F(i-1, j - w_i), F(i-1, j) \right\} \]

\[ 1 \leq i \leq n, \quad 0 \leq j \leq B \quad (x) \]
We can iteratively compute
\[ F(1) F(2) \ldots F(n,-) \]
(in this ordering)
\[ \vdots \]
for all values \( 0 \leq j \leq B \)

If we store the previously computed term, we need only \( O(1) \) to compute
\[ F(i, j) \]
for \( F(i-1, -) \)
\[ = \text{total of } O(nB) \text{ steps} \]
\[ \left( \begin{array}{c} \text{not} \\ \frac{2^n}{2^n} \end{array} \right) \]

If \( B \) is bounded by \( n^{o(1)} \)
\[ = \text{total of polynomial steps} \]

What is the problem size?
\[ (c_0, w_1, \ldots, w_n, p_1, k_2, \ldots, p_n, B) \]

2n+1 parameters

Size of output is \( l_{w_1} + l_{w_2} + \ldots + l_{B} \)

So \( |B| \) can be \( \frac{1}{2} \) the size of the input

Running time \( 2^{|B|} \times n \)
Problem: We are given a sequence of \( n \) nos
\[
\begin{array}{c}
1 & 1 & 2 & 2 & 3 & 3 & 3 & 4 & 3 \\
5 & 3 & 8 & 6 & 20 & 11 & 51 & 9 \\
x_1 & x_2 & x_3 & \ldots & x_n
\end{array}
\]
An increasing subsequence is
\[
x_{i_1} & x_{i_2} & x_{i_3} & \ldots & x_{i_k}
\]
i_1 < i_2 < i_3 \ldots < i_k

and \( x_{i_1} < x_{i_2} < x_{i_3} \ldots x_{i_k} \)

We want to find the longest increasing subsequence.

In any sequence of \( n \) nos, there is either an increasing or a decreasing subsequence of length \( \lfloor \sqrt{n} \rfloor \).

Proof: Consider the longest increasing subsequence ending at \( x_{i_k} \) and denote it by \( l_i \), \( 1 \leq l_i \leq n \).
If no subsequence is longer than $\sqrt{n}$, some $x_i$ must be repeated at least $\sqrt{n}$

\[
\begin{array}{cccccccc}
1 & 2 & 3 & \cdots & \sqrt{n} \\
\end{array}
\]

By pigeonhole

\[
\begin{array}{cccccccc}
x_1 & x_2 & x_3 & x_4 & \cdots & x_i & x_n \\
\downarrow & & & & & \uparrow & \uparrow \\
k & k & k & k & \text{at least } \sqrt{n} \text{ times}
\end{array}
\]

What can we say about $x_2$ and $x_4$?

\[
x_2 > x_4
\]

\[
\therefore \text{ There must be a decreasing subsequence of length } \geq \sqrt{n}
\]

Erdős - Szekeres - them
If \( l_i \) is the longest subsequence ending at \( x_i \)

\[
l_i = \max \{ l_j \} + 1 \quad j < i, \quad x_j < x_i
\]

\( l_i \) can be computed \( l_i = 1 \)

To compute each \( l_i \), we need at most \( i-1 \) lookups.

Total time = \( \sum_{i=1}^{n} i-1 \) \( \sim O(n^2) \)

How much space - we need to store all the previously computed \( l_j \)

\[
\begin{array}{cccccccc}
1 & 2 & 3 & 4 & \ldots & n \\
\hline
l_1 & & & & & \\
\end{array}
\]

Space required

Knapsack: \( i \rightarrow 1 \ldots \rightarrow B \)

\( n \times B \)
Faster than $O(n^2)$ sohn?

$l_{ij}$: longest sequence up to $i$ of length $j < i$

Moreover, for each $l_{ij}$ we want to retain the sequence that has the smallest last element.