**Weighted Interval Scheduling**

- No greedy algorithm known.
- Dynamic programming works.

**Problem** - Given \( n \) jobs with start and finish time

\[ J_1 = [s_1, f_1], \ J_2 = [s_2, f_2], \ldots, J_n = [s_n, f_n]. \]

- Job \( J_i \) has value \( V_i \geq 0 \).

**Goal** To find a set of pairwise non-overlapping jobs with maximum total value.

**Recall** The interval scheduling problem studied earlier corresponds to the case where all \( V_i = 1 \). The algorithm (greedy) sorts the jobs according to finish time, say,

\[ f_1 \leq f_2 \leq f_3 \ldots \leq f_n \]

and for \( i=1,2,\ldots,n \), picks \( i \)th job if it does not overlap with previously chosen jobs.

This does not work for general values.
Example

\[ v_1 = 5 \]

\[ v_2 = 12 \]

\[ v_3 = 5 \]

The greedy algorithm will pick \( \{1, 3\} \) but the optimal solution is \( \{2\} \).

Dynamic Programming Algorithm

- Order jobs according to finish time, say

\[ f_1 \leq f_2 \leq f_3 \ldots \leq f_n \]

Idea

- If \( J_n \) is not selected then the problem reduces to \( \{J_1, J_2, \ldots, J_{n-1}\} \).

- If \( J_n \) is selected, then problem reduces to \( \{J_1, \ldots, J_k\} \) s.t. \( k \) is largest index

\[ \text{s.t. } f < S_n \quad (k=3 \text{ in picture}) \]
Subproblems

\[ \text{OPT}(J_1, J_2, ..., J_t) = \max \text{ value of non-overlapping set of jobs from} \{J_1, J_2, ..., J_t\}. \]

These are all "prefixes".

# subproblems = n.

original problem = \{J_1, ..., J_t\}.

Recursive formula

\[ \text{OPT}(J_1, ..., J_t) = \]

\[ \max \left\{ \begin{array}{l}
\text{OPT}(J_1, J_2, ..., J_{t-1}) \\
V_t + \text{OPT}(J_1, ..., J_k) \quad \text{where } k \text{ is largest index s.t. } f_k \leq s_t
\end{array} \right\} \]

Order According to length of prefix.

Base case \[ \text{OPT}(J_1) = V_1. \]
Shortest Path in Directed Graphs

Def A directed graph $G(V,E)$ consists of a set of vertices $V$ and set of directed edges $E$.

$$E = \{(u,v) \mid u, v \in V, u \neq v\}$$

ordered pair

Example

$$V = \{s, a, b, c, d, s, t\}$$

$$E = \{(s, a), (a, c), (a, b), (b, t), (d, c), (t, a), (t, b)\}$$

Problem

Given a directed graph $G(V,E)$
- each edge $(u,v) \in E$ has cost $c_{uv} \geq 0$
- gives start node $s$ and target node $t$

Goal - to find the path from $s$ to $t$ that has least cost.
Bellman-Ford Algorithm

Observation: For any two nodes \( u, v \), the \( u \rightarrow v \) path with minimum cost has no cycles, so has \( \leq n-1 \) edges. (\( |V| = n \)).

Subproblems

\( \text{OPT}(u, i) = \text{minimum cost of a path from } s \text{ to } u \text{ that has } \leq i \text{ edges} \) (\( \infty \) if no such path exists).

- Original problem: \( \text{OPT}(t, n-1) \).
- Number of subproblems: \( O(n^2) \).

Base case: \( \text{OPT}(u, 0) = \begin{cases} 0 & \text{if } u = s \\ \infty & \text{otherwise} \end{cases} \)

Recursive formula:

\( \text{OPT}(u, k) = \min \left\{ \text{OPT}(u, k-1), \min_{w : (w, u) \in E} \left[ c_{wu} + \text{OPT}(w, k-1) \right] \right\} \)
Observe that the recursive formula is based on the following fact: the shortest $s \rightarrow u$ path with at most $k$ edges has either at most $k-1$ edges or has some $w$ as the "last but u" vertex and has shortest $s \rightarrow w$ path as prefix with at most $k-1$ edges.
Maximum Independent Set in Trees

**Def** In an undirected graph $G(V,E)$, an independent set $S$ is a subset of the vertex set $V$ that contains no edge inside it, i.e., $\forall u,v \in S$, $\{u,v\} \notin E$.

![Graph with nodes labeled 1 to 7 and edges connecting them]

Independent sets:
- $\{1, 3, 6\}$
- $\{2, 7, 3\}$
- $\{1, 5\}$

**Note** Finding independent sets in general graphs is NP-complete.

**Problem** Given a tree $T(V,E)$, find an independent set of the maximum size!
W.l.o.g. $T$ is a rooted tree.

**Idea** Let $r$ be the root. We consider two cases depending on whether $r$ is included in an independent set or not.

Define:

- $\text{IS}(T) = \text{size of maximum independent set in } T$
- $\text{IS}_{\text{with root}}(T) = \text{maximum size of an independent set in } T \text{ that includes the root}$
- $\text{IS}_{\text{without root}}(T) = \text{max size of an indep. set in } T \text{ that excludes the root}$
Note.

\[ IS(T) = \max \{ IS_{\text{with-root}}(T), \]
\[ IS_{\text{without-root}}(T) \} . \]

\[ T = \]

\[ T_1 \quad T_2 \quad \ldots \quad T_k \]

**Recursive Formula**

\[ IS_{\text{without-root}}(T) = \sum_{i=1}^{k} IS(T_i) . \]

\[ IS_{\text{with-root}}(T) = 1 + \sum_{i=1}^{k} IS_{\text{without-root}}(T_i) . \]

\[ IS(T) = \max \{ IS_{\text{with-root}}(T), \]
\[ IS_{\text{without-root}}(T) \} . \]