Job Scheduling with Profits

- We have n **jobs**, each of which takes unit time, and a processor on which we would like to schedule them in as profitable a manner as possible.
- Each job has a **profit** associated with it, as well as a **deadline**; if the job is not scheduled by its deadline, then we don't get its profit.
- Because each job takes the same amount of time, we will think of a schedule S as consisting of a sequence of job "slots" $1, 2, 3, \ldots$, where $S(t)$ is the job scheduled in slot t.
- Formally, the input to the problem is a sequence of pairs $(d_1, g_1), (d_2, g_2), \ldots, (d_n, g_n)$ where $g_i \in \mathbb{R}^+$ represents the profit obtainable from job i, and $d_i \in \mathbb{N}$ is the deadline for job i.

- A **schedule** is an array $S(1), S(2), \ldots, S(d)$ where $d = \max d_i$ (i.e. the latest deadline, beyond which no jobs can be scheduled), such that if $S(t) = i$, then job i is scheduled at time t, $1 \le t \le d$. If $S(t) = 0$, then no job is scheduled at time t.
- A schedule S is **feasible** if:
  1. If $S(t) = i > 0$, then $t \le d_i$ (i.e. every scheduled job meets its deadline)
  2. If $t_1 \ne t_2$ and $S(t_1) \ne 0$, then $S(t_1) \ne S(t_2)$ (i.e., each job is scheduled at most once).
- Let the total profit of schedule S be $P(S) = g_{S(1)} + g_{S(2)} + \ldots + g_{S(d)}$, where $g_0 = 0$.

- We want to find a feasible schedule S whose profit $P(S)$ is as large as possible.

- Consider the following algorithm:

  Sort the jobs in non-increasing order of profits:

  $$g_1 \geq g_2 \geq \ldots \geq g_n$$

  $d \longleftarrow \max_i d_i$

  for $(t = 1; t \leq d; t++)$

  $\quad S(t) \longleftarrow 0$

  for $(i = 1; i \leq n; i++)$

  $\quad$ Find largest t s.t. $(S(t) = 0$ and $t \leq d_i)$, and let $S(t) \longleftarrow i$

  $\quad$ [*find the latest possible free slot meeting the deadline*]

- A context-free grammar is in **Chomsky Normal Form (CNF)** if all the rules are of the form:
  - $A \longrightarrow BC$
  - $A \longrightarrow a$

  A, B, C are variables (not necessarily distinct), a a terminal.
- Every context-free grammar can be transformed to an equivalent grammar (i.e., generating the same set of strings of terminals) in CNF.
- Given a grammar G in CNF, and a string $w = a_1 a_2 \dots a_n$, we can test whether $w \in L(G)$ using the CYK algorithm.
- The CYK algorithm, on input $\langle G, w = a_1 a_2 \dots a_n \rangle$, builds an $n \times n$ table T, where each entry contains a subset of V.
- At the end, $w \in L(G)$ if the start variable is in position $(1, n)$.

- Let $V = \{X_1, X_2, \ldots, X_m\}$. Initialize T as follows:

  for $(i = 1; i \leq n; i++)$
      for $(j = 1; j \leq m; j++)$
          Place variable X_j in (i, i) iff $X_j \longrightarrow a_i$ is a rule of G

- Then, for $i < j$, the contents of (i, j) are computed as follows:

  for $(k = i; k < j; k++)$
      if $(\exists X_p \in (i, k) \wedge \exists X_q \in (k+1, j)$ & $\exists$ rule $X_r \longrightarrow X_p X_q)$
        Put X_r in (i, j)

Eg.

x	(2,2)	(2,3)	(2,4)	**(2,5)**
x	x			(3,5)
x	x	x		(4,5)
x	x	x	x	(5,5)

Eg. Consider

$G = (\{S\}, \{a - z, (,), \wedge, \vee, \neg\}, S, \{S \longrightarrow \texttt{a-z} | (S \wedge S) | (S \vee S) | \neg S\}).$

We convert it to CNF:

$X_0 \longrightarrow \texttt{a-z}, X_1 \longrightarrow (, X_2 \longrightarrow), X_3 \longrightarrow \wedge, X_4 \longrightarrow \vee, X_5 \longrightarrow \neg$

$X_0 \longrightarrow X_1 X_6, X_6 \longrightarrow X_0 X_7, X_7 \longrightarrow X_3 X_8, X_8 \longrightarrow X_0 X_2$

$X_0 \longrightarrow X_1 X_9, X_9 \longrightarrow X_0 X_{10}, X_{10} \longrightarrow X_4 X_{11}, X_{11} \longrightarrow X_0 X_2$

$X_0 \longrightarrow X_5 X_0$