General Knapsack Problem

- We generalize the SKS to GKS by assigning a positive integer value v_i to each weight w_i.
- **Input:** $d, w_1, \ldots, w_d, v_1, \ldots, v_d, C \in \mathbb{N}$ (where C is the capacity)

  Output: Find $\max_S \{V(S) | K(S) \leq C\}$, where $S \subseteq \{1, \ldots, d\}$, $K(S) = \sum_{i \in S} w_i$, and $V(S) = \sum_{i \in S} v_i$.
- Thus $V(S)$ is the total value of the set S of weights. The goal is to maximize $V(S)$, subject to the constraint that $K(S)$ (the sum of the weights in S) is at most C.
- The Simple Knapsack Problem is the special case in which $v_i = w_i, 1 \leq i \leq d$.

- To solve GKS start by computing the same Boolean array $R(i,j)$ that was used to solve SKS. Thus $R(i,j)$ ignores the values v_i, and only depends on the weights w_i.
- Define another array $V(i,j)$ that depends on the values v_i as follows: For $0 \le i \le d$, $0 \le j \le C$

$$V(i,j) = \max\{V(S) | S \subseteq \{1,\ldots,i\} \text{ and } K(S) = j\}$$

- **Ex.** Give a recurrence for computing $V(i,j)$, using $R(i,j)$. (You may assume that the array $R(i,j)$ has already been computed.) Also, give a program for computing $V(i,j)$.
- **Ex.** If the above definition of $V(i,j)$ is changed so that we only require $K(S) \le j$ instead of $K(S) = j$, then the Boolean array $R(i,j)$ is not needed in the recurrence; why?

The Activity Selection Problem

- An **activity** i has a fixed start time s_i, finish time f_i, and profit p_i.
- Given a set of activities, we want to select a subset of non-overlapping activities with maximum total profit. (**Eg.** a set of lectures with fixed start and finish times that need to be scheduled in a single class room.)
- **Input:** A list of activities $(s_1, f_1, p_1), \ldots, (s_n, f_n, p_n)$. Assume $p_i > 0$, $s_i < f_i$, and $s_i, f_i, p_i \in \mathbb{R}^+$ where $1 \leq i \leq n$.

  Output: A set $S \subseteq \{1, \ldots, n\}$ of selected activities such that no two selected activities overlap, and the profit $P(S) = \sum_{i \in S} p_i$ is as large as possible.

1st step: Define an array of subproblems

- Sort the activities by their finish time: $f_1 \leq f_2 \leq \ldots \leq f_n$.
- Partition the activities according to their finish times. Denote these *distinct* finish times by $u_1 < u_2 < \ldots < u_k$.
- **Eg.** Times 1.24, 4, 3.77, 1.24, 5 and 3.77, would yield $u_1 = 1.24$, $u_2 = 3.77$, $u_3 = 4$, $u_4 = 5$.
- Let u_0 be $\min_{1 \leq i \leq n} s_i$, i.e., the earliest start time. Thus, $u_0 < u_1 < u_2 < \ldots < u_k$. Define an array $A(j)$ as follows:

$$A(j) = \max_{S \subseteq \{1,\ldots,n\}} \{P(S) | S \text{ is feasible, } f_i \leq u_j \ \forall i \in S\}$$

where S is **feasible** if no two activities in S overlap.

Note that $A(k)$ is the maximum possible profit for all feasible schedules S.

2nd step: give a recurrence

- In order to give a recurrence for computing $A(j)$ we define an auxiliary array $H(i)$ which is the index of the largest distinct finish time no greater than the start time of activity i.
- Formally, $H(i) = l$ if l is the largest number such that $u_l \leq s_i$.
- To compute $H(i)$, we need to search the list of distinct finish times. To do it efficiently, for each i, apply the binary search procedure that runs in logarithmic time in the length of the list of distinct finish times (try $l = \lfloor \frac{k}{2} \rfloor$ first).
- Since the length k of the list of distinct finish times is at most n, and we need to apply binary search for each element of the array $H(i)$, $1 \leq i \leq n$, the time required to compute all entries of the array is $O(n \log n)$.

- Initialization: $A(0) = 0$.
- Want $A(j)$ given $A(0), \ldots, A(j-1)$.
  - Consider $u_0 < u_1 < u_2 < \ldots < u_{j-1} < u_j$.
  - Can we beat profit $A(j-1)$ by scheduling some activity that finishes at time u_j?
  - Try all activities that finish at this time and compute maximum profit in each case. We obtain the following recurrence:

$$A(j) = \max\{A(j-1), \max_{1 \leq i \leq n} \{p_i + A(H(i)) \mid f_i = u_j\}\}$$

Eg.

Activity i:	1	2	3	4
Start s_i:	0	2	3	2
Finish f_i:	3	6	6	10
Profit p_i:	20	30	20	30
$H(i)$:	0	0	1	0

j:	0	1	2	3
u_j:	0	3	6	10
$A(j)$:	0	20	40	40

$$A(2) = \max\{20, 30 + A(0), 20 + A(1)\} = 40$$

$$A(3) = \max\{40, 30 + A(0)\} = 40$$

The answer is $\max P = A(k) = 40$.

Ex. Given that A has been computed, how do you find a set of activities S such that $P(S) = A(k)$?

Hint. Consider two cases:

1. If $A(k) = A(k-1)$, then we know that no selected activity finishes at time u_k, so we go on to consider $A(k-1)$.

2. If $A(k) > A(k-1)$, then some selected activity finishes at time u_k. How do we find this activity? We must find an i_0 such that $p_{i_0} + A(H(i_0)) = A(k)$ and $f_{i_0} = u_k$, output i_0, and go on to consider $A(H(i_0))$.

Ex. Write an algorithm (i.e., do the **3rd step**).