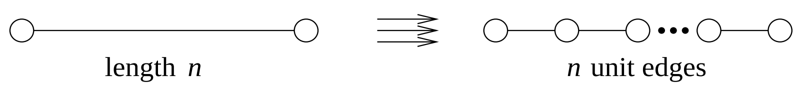
Analysis of Algorithms: Solutions 9

```
                                      X
                                      X
                                      X  X
  number of                        X  X  X
  homeworks                        X  X  X
                                X  X  X  X
                                X  X  X  X
                       X        X  X  X  X
                       X        X  X  X  X
             X  X      X        X  X  X  X
             --------------------------
             2  3  4   5  6  7  8  9  10
                         grades
```

Problem 1

Suppose that G is a weighted directed graph, where all weights are integers between 1 and 5, and let u and v be two vertices of G. Describe an efficient algorithm SHORTEST-PATH(G, u, v) that finds a minimal-weight path from u to v.

We construct a new graph, by replacing every edge of length n in the original graph with n unit edges, as shown in the picture. That is, we replace every edge of length 2 with two unit edges, every edge of length 3 with three unit edges, and so on. We then run the breadth-first search in the new graph, with the source vertex u, which finds a shortest path from u to v. If the original graph has V vertices and E edges, then the new graph has at most $V + 4 \cdot E$ vertices and $5 \cdot E$ edges, and the running time of the breadth-first search is $O(V + 4 \cdot E + 5 \cdot E) = O(V + E)$.



Problem 2

Write pseudocode of an algorithm GREEDY-KNAPSACK(W, v, w, n) for the 0-1 Knapsack Problem. The arguments are an weight limit W, item values $v[1..n]$, and item weights $w[1..n]$.

GREEDY-KNAPSACK(W, v, w, n)
sort items in the descending order of the $\frac{v[i]}{w[i]}$ ratios
items $\leftarrow \emptyset$ ▷ set of selected items
w-sum $\leftarrow 0$ ▷ sum of their weights
for $i \leftarrow 1$ **to** n ▷ in sorted order
    do if *w-sum* $+ w[i] \leq W$
        then *items* $\leftarrow$ *items* $\cup \{i\}$
            w-sum $\leftarrow$ *w-sum* $+ w[i]$
return *items*

The sorting takes $O(n \lg n)$ time, whereas the selection loop runs in linear time. Thus, the total time of GREEDY-KNAPSACK is $O(n \lg n)$.

Problem 3

Suppose that the weights of all items in the 0-1 Knapsack Problem are integers, and the weight limit W is also an integer. Design an algorithm that finds a *globally optimal* solution.

We use two arrays, $item[1..W]$ and $value[0..W]$, which are indexed on the size of a knapsack. For every size i between 0 and W, we compute the maximal value of items that can be loaded into a knapsack, and store this result in $value[i]$. If $value[i]$ is larger than $value[i-1]$, then $item[i]$ is the last added item; otherwise, $item[i]$ is 0.

We add items in their numerical order; that is, if items j_1 and j_2 must be in the knapsack, and $j_1 < j_2$, then we add j_1 before j_2.

The following algorithm computes the arrays $item[1..W]$ and $value[0..W]$, and returns the maximal value of items for size W; its time complexity is $\Theta(n \cdot W)$.

```
OPTIMAL-KNAPSACK(W, v, w, n)
value[0] ← 0
for i ← 1 to W        ▷ consider every size of a knapsack
    do item[i] ← 0
        value[i] ← value[i − 1]        ▷ initialize the maximal value for size i
        for j ← 1 to n        ▷ look through items, to find the best addition to a smaller load
            do if w[j] ≤ i        ▷ item j fits into the knapsack
                    and j > item[i − w[j]]        ▷ it does not violate the numerical order
                    and value[i] < value[i−w[j]]+v[j]        ▷ we get a good value by adding j
                then item[i] ← j        ▷ add j to the knapsack
                    value[i] ← value[i − w[j]] + v[j]
return value[W]
```

We also need an algorithm for printing out the list of selected items. The following output procedure uses the array $item[1..W]$, built by DYNAMIC-KNAPSACK, to print items in their numerical order; its running time is $O(n)$.

```
PRINT-KNAPSACK(item, W, w, i)
if i = 0
    then "do nothing"
elseif item[i] = 0
    then PRINT-KNAPSACK(item, W, w, i − 1)
else PRINT-KNAPSACK(item, W, w, i − w[item[i]])
      print item[i]
```