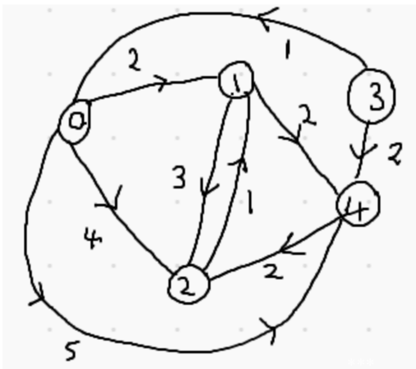
Informatics 2 – Introduction to Algorithms and Data Structures

Solutions for Tutorial 6

Mary Cryan

week 3: 27th-31st January, 2025

1. *Execute Dijkstra's Algorithm from node 0 on the following graph, showing the steps/updates to the d and π arrays.*



answer: I was vague about how much detail I want for the solution, but had been thinking simpler rather than detailed ... hence skipping the Heap details. However, given that I asked for both the d, π arrays, some people may have worked wrt the final implementation.

Therefore I will take the convention that I *do* update d, π with values whenever a better fringe edge *option* becomes available, even if it is not the committed vertex. I will indicate "addition to S" by using bold font in the arrays.

We start with the d array (of length 5) initialised to ∞ everywhere, and the π-array initialised to NULL.

0 is the initial vertex being added to the set S (with distance 0), so after that we have the following update to d (no update to π yet):

0	∞	∞	∞	∞
0	1	2	3	4

We next examine the outgoing edges from 0 to $\{1, 2, 3, 4\}$ - there are edges of weight 2 (to 1), weight 4 (to vertex 2) and weight 5 (to vetex 4).

Hence our three *fringe* edges have the costs $d[0] + 2 = 2$ (for $(0 \to 1)$), $d[0] + 4 = 4$ (for $(0 \to 2)$) and $d[0] + 5 = 5$ (for $(0 \to 4)$); hence *we add vertex 1 to S*, setting $d[1] \leftarrow 2$ and $\pi[1] \leftarrow 0$.

0	**2**	4	∞	5
0	1	2	3	4

null	**0**	0	NULL	0
0	1	2	3	4

After this step, the fringe edges $(0 \to 2)$ (with cost 4) and $(0 \to 4)$ (with cost 5) are still fringe edges; and we have two new fringe edges $(1 \to 2)$ and $(1 \to 4)$; Overall the current costs of our fringe edges are:

$(0 \to 2)$: (with cost 4, already shown in the $d[2], \pi[2]$ cells)

$(0 \to 4)$: (with cost 5, already shown in the $d[4], \pi[4]$ cells)

$(1 \to 2)$: Cost is $d[1] + 3 = 5$

$(1 \to 4)$: Cost is $d[1] + 2 = 4$. This will give a new/better option for 4.

We can take either of the cost-4 options, let's choose $(0 \to 2)$; hence 2 is committed to S, which becomes $\{0, 1, 2\}$, and we fix $d[2], \pi[2]$ to these values. We also use the details of the $(1 \to 4)$ edge to update $d[4], \pi[4]$:

0	**2**	**4**	∞	4
0	1	2	3	4

null	**0**	**0**	NULL	1
0	1	2	3	4

Now the newly-added vertex 2 only has one outgoing edge, and it is $(2 \to 1)$, so within S; hence we have no new fringe edges. We have lost two prior fringe edges (the two leading to 2), hence we just commit 4 to S via the better route, which is already encoded in the arrays.

0	**2**	**4**	∞	**4**
0	1	2	3	4

null	**0**	**0**	NULL	**1**
0	1	2	3	4

So we have $S = \{0, 1, 2, 4\}$, but no fringe edges any more.

Hence the algorithm terminates with this p, π.

2. For this question, the first thing to do will be to talk a bit about the way these Greedy strategies will operate in the context of fractional knapsack, before you actually present the specific answers to (a) and (b).

   I think the most important thing with new questions like this is to help students understand the notation. So to give an example with about 4 items maybe, and write out the values as numbers (with v_i annotations), same for the weights, and then pick a capacity C that will make things (slightly) interesting.

   It may be wise to actually *run* the two Greedy strategies on an example, step by step. May be nice to use the example in (i) which gives the counter-proof for strategy (a).

   (i) Here is a specific input which will demonstrate the non-optimality of the "largest v_i first" strategy: values $v_1 = 3, v_2 = 3, v_3 = 4, v_4 = 5$, weights $w_1 = 3, w_2 = 4, w_3 = 4, w_4 = 9$, capacity $C = 12$.

   In this case we will consider the items in order of value, so will consider items in order of index $i = 4, i = 3, i = 1, 2$.

   Taking $i = 4$ first, we take that entire item (as $w_4 < 12$), and set $x_4 \leftarrow 1$ and $C' \leftarrow 12 - 9$, $C' \leftarrow 3$.

   Next we consider item $i = 3$, we have $w_3 = 4$, so we can't fit all of item $i = 3$: we must set $x_3 \leftarrow (C'/w_3)$-fraction, which is $x_3 \leftarrow 0.75$, with the new $C' \leftarrow 3 - w_3 \cdot 0.75 = 0$.

At this point the leftover capacity is now 0, so we don't consider the other items. We have $x_1 = 0, x_2 = 0$.

The total value we get with this version of Greedy is $5 + 0.75 \cdot 4 = 8$.

However, it is easy to see by inspection that we could have got a value of 10.555555... by taking $x_1 = 1, x_2 = 1, x_3 = 1, x_4 = 1/9$.

(ii) For greedy strategy (b), we need a proof, as we are claiming that when we rank by v_i/w_i and consider items i in that order, we are guaranteed to construct an optimal $x_1, \ldots, x_n$ for the input.

PROOF: It is natural to think of designing a proof by induction, but we need to choose our Induction Hypothesis carefully. We will choose the following:

Induction Hypothesis (I.H.): For the set I of top-ranked k items (according to the v_i/w_i ranking), there is some *optimal* solution $x'_1, \ldots, x'_n$ such that $x'_i = x_i$ for $i \in I$.

Base case: If we consider the case of $k = 0$, it is certainly the case that there is some *optimal* solution $x'_1, \ldots, x'_n$ such that the top-0 items match the values assigned by greedy (b).

Induction step: We assume the (I.H.) for the top-ranked item set I. Our goal is to argue that we can then construct an optimal solution which satisfies the (I.H) for $I \cup \{i^*\}$, where i^* is the "next most highly ranked item" after the items in I.

We will let val_{opt} be the value $\sum_{i \in [n]} x'_i \cdot v_i$.

For the current working optimum x', compare x'_{i^*} to x_{i^*}.

If it is the case that these two values are identical, then we already have shown the induction step, and we do not need to change x' (note this includes the case where both these are 0).

The more interesting argument is when $x'_{i^*} \neq x_{i^*}$.

We know by the rules of greedy (b) that greedy always sets x_i to the maximum possible, which is $x_i \leftarrow \min\{1, \frac{C'}{w_i}\}$ (for the current remaining capacity C'). Our (I.H.) ensures that $x_i = x'_i$ for all the items considered before i^* (items in I). Hence the leftover capacity for the $[n] \setminus I$ items is identical for x and x', and greedy (b) has set x_{i^*} to the maximum possible. Therefore, the only way x'_{i^*} and x_{i^*} can differ is if $x'_{i^*} < x_{i^*}$.

We will now show how to transform x' to a new assignment with $x'_{i^*} = \min\{1, \frac{C'}{w_{i^*}}\})$ where we also maintain overall value val_{opt}.

Consider some $j \in [n] \setminus I \cup \{i^*\}$ with $x'_j > 0$ such that $x'_j > x_j$.

(*) We assume there must be such a $j \in [n] \setminus I \cup \{i^*\}$ ... if this was not the case, then we would have spare capacity to increase the value of x'_{i^*} in x' to achieve an assignment of value *strictly greater* than val_{opt} (which itself is a contradiction).

For such a $j \in [n] \setminus I \cup \{i^*\}$ with $x'_j > x_j$, we will re-distribute the extra item weight $(x'_j - x_j)w_j$ (for x') towards the i^* item. We do not know whether x'_{i^*} is small enough to absorb all possible extra weight from j, hence we will consider scaling by any $\alpha > 0$:

- We reduce x'_j to now be $x'_j - (x'_j - x_j)\alpha$
- We increase x'_{i^*} to now be $x'_{i^*} + \alpha(x'_j - x_j)\frac{w_j}{w_{i^*}}$.

– These two changes to x' ensure that the new x' has identical total item weight to before, hence the total capacity is unchanged. CHECK THIS!
– The reduction of value x'_j will *reduce* val_{opt} by $v_j(x'_j - x_j)\alpha$,
– The increase to value x'_{i^*} will *increase* val_{opt} by $\alpha(x'_j - x_j)\frac{w_j}{w_{i^*}} \cdot v_{i^*}$

We consider the difference

$$\begin{aligned} & \alpha(x'_j - x_j)\frac{w_j}{w_{i^*}} \cdot v_{i^*} - v_j(x'_j - x_j)\alpha \\ = \quad & \alpha(x'_j - x_j)\left(\frac{w_j}{w_{i^*}} \cdot v_{i^*} - v_j\right) \end{aligned}$$

We know that $\frac{v_i}{w_i} \leq \frac{v_{i^*}}{w_{i^*}}$ for every $i \in [n] \setminus I \cup \{i^*\}$ (including j), which implies $v_j \leq \frac{v_{i^*}}{w_{i^*}} \cdot w_j$. We also know that $\alpha > 0$ and $(x'_j - x_j) > 0$, hence the overall change to val_{opt} is non-negative.

We have shown how we can use any extra weight from any x'_j for $j \in [n] \setminus I \cup \{i^*\}$ to *strictly* increase the value of x'_{i^*} without reducing our value from val_{opt}. We can iterate this until x'_{i^*} achieves the value $\min\{1, \frac{C'}{w_{i^*}}\}$. We know from (*) above that until x_{i^*} achieves this value, that there must be j indices with $x'_j > x_j$. Hence we are guaranteed to build an assignment x' with value val_{opt} where x'_{i^*} has the value assigned by greedy Criterion (b).

After this step, we have constructed an optimal x' where the top $|I| + 1$ ranked items match the values assigned by greedy (b), completing the Induction Step.

By induction, we can therefore infer that there is an optimal solution x' such that x'_i matches the value assigned by greedy (b) for every $i \in [n]$. Hence we have proven our claim. □

Alternative approach: Some students on the class (Hubert Wach, Jonathan Ambler, and others who spoke to me at lectures) suggested a way we could potentially simplify the proof, if we were to *normalise* the input to an "weights all 1" version. We have discussed, and it is possible to get a simpler proof this way, given the original assumption that all weights were from $\mathbb{N}$ (as was the case in our problem statement). It's not massively shorter, but is more intuitive.

UNIT-WEIGHT FRACTIONAL KNAPSACK: We are given a set of items $i = 1, \ldots, n$ of *unit weight* each, and with values $v_1, \ldots, v_n \in \mathbb{Q}^+$ respectively. We are also given a capacity $C \in \mathbb{N}$. Our aim is to find binary values $x_i \in [0, 1], i \in [n]$ such that $\sum_{i=1}^n x_i \leq C$ and such that $\sum_{i=1}^n x_i \cdot v_i$ is maximised subject to the capacity constraint.

(note: we need to allow the values to be rational numbers, as in our proof we will build unit-weight instances which have values v_i/w_i)

ALTERNATIVE PROOF FOR THE WEIGHTED KNAPSACK:

I. Every input instance of WEIGHTED FRACTIONAL KNAPSACK $w_i \in \mathbb{N}, v_i \in \mathbb{N}, i \in [n]$ and capacity $C \in \mathbb{N}$ can be transformed to an *equivalent* instance of UNIT-WEIGHT FRACTIONAL KNAPSACK where we have the same capacity C and where a single item i of value v_i and weight w_i gets mapped to w_i new unit-weight items with value v_i/w_i. There will be a total of $\widehat{n} = \sum_{i=1}^n w_i$ items in this unit-weight instance.

item i: weight w_i, value v_i	$\Leftrightarrow$	w_i different items, each value v_i/w_i
capacity C	$\Leftrightarrow$	capacity C
n original items	$\Leftrightarrow$	$\sum_{i=1}^{n} w_i$ items altogether (call this $\widehat{n}$)

We can define a "break-ties" ordering on the items of the unit-weight construction which groups the unit-weight items from the same item i together. When we do this, the operation of Greedy (b) on the constructed unit-weight instance is equivalent to the operation of Greedy (b) on the original instance (we assign x_i on the original instance to be the sum of the x-values for its corresponding unit-weight items).

II. We will prove that the Greedy Algorithm ranking according to (b) is guaranteed to return an optimal solution for every instance of UNIT-WEIGHT FRACTIONAL KNAPSACK.

PROOF OF II. In the UNIT-WEIGHT case of fractional knapsack we assume all weights are 1, and we allow the values $\widehat{v_i}$ (for $i \in [n]$) to be any positive *rational numbers*. We can assume $n > C$ (if not, then the capacity allows us to set $x_i \leftarrow 1$ for all items, and optimal and Greedy (b) are exactly the same). N

We observe that (since all v_i are > 0) if $n > C$, we expect an *optimal* x' solution to satisfy $\sum_{i=1}^{n} x'_i = C$ (to use all capacity).

We observe that Greedy (b) always constructs a solution which uses all capacity (assuming $n > C$).

Now let us consider the first step of Greedy (b), assuming $C > 0$.

- Let i^* be the item of maximum value $\widehat{v_{i^*}}$ in the collection of items.
- Greedy (b) chooses this item to add to the knapsack, setting $x_{i^*} \leftarrow \min\{1, C\}$, which (as C is a whole number, and $C > 0$) is 1.
  We will show there must be some optimal solution x' with $x'_{i^*} = 1$.
  The reason is as follows: consider the optimal solution x' which has the maximum assignment to x'_{i^*} among all optima. Suppose that this x'_{i^*} is *not* equal to 1. Then we can find $j \in [n] \setminus \{i^*\}$ such that $x'_j > x_j$ (due to our observations that both optimal x' and Greedy (b) x will use all of C).
  By choice of i^*, we know that $\widehat{v_{i^*}} \geq \widehat{v_j}$.
  Hence we can transform x' by increasing x'_{i^*} by $\min\{1 - x'_{i^*}, x'_j - x_j\}$ and decreasing x'_j by the same amount ... and the *value* of x' does not decrease.
  This contradicts the choice of x' being the one with maximum value x'_{i^*} (among all optimum assignments x') Hence there is an optimum assignment x' with $x'_{i^*} = 1 = \min\{1, C\}$, just as Greedy (b) has assigned.

This above justifies the initial step of Greedy (b), taking the first item.

If we had $C = 1$, then this is also the final step of Greedy (b), and we are finished.

Otherwise, we note that continuing Greedy (b) is equivalent to the new instance $(\widehat{v_i}, i \in [n] \setminus \{i^*\}$ with $C - 1)$ of UNIT-WEIGHT FRACTIONAL KNAPSACK. We can apply induction to infer that there is an optimal assignment that matches the one constructed by Greedy (b).

note: something to think about is what Greedy (a) will achieve for the special case of unit weights.

3. (a) The algorithm is driven by two nested for-statements, the outer iterating n times, the inner one iterating C times. The statements within the inner loop just carry out $\Theta(1)$ operations (comparison, addition, subtraction) on each iteration, so overall $\Theta(nC)$ time.

   (b) The following is the main dynamic programming table, where the cell value for (i, j) is the value of the "max-knapsack which uses items 1 to i to achieve weight at most j".

	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	2	2	2	2	2
2	0	0	3	3	3	5	5	5
3	0	0	3	4	4	7	7	7

   (c) This proposed Greedy algorithm will *not* deliver an optimal solution for all instances of the 0/1 knapsack problem.

   One counterexample is $v_1 = 3, v_2 = 5, v_3 = 2$ and $w_1 = 3, w_2 = 4, w_3 = 2$. $C = 5$ In this case Greedy (b) will first add item 2 ($v_2/w_2 = 1.25$). We then have residual capacity $C' = 5 - 4 = 1$, and in the 0/1 setting, this means that we cannot add any extra items (as weights are 2 and 3), hence we return value 4.

   However, if we had taken items 1 and 3, we would have used capacity $3 + 2 = 5 = C$, and would have achieved value $3 + 2 = 5$.