Prof. Athina Markopoulou — Design and Analysis of Algorithms
Winter 2018 — EECS215

Homework 3 - Solution Sketch

Topic: Dynamic Programming (Ch.6)

Posted on March 1st, 2018.

1. *(70 Points)* **Variations of the Knapsack Problem.**

   (a) *(30 Points)* **Multidimensional Knapsack.** Consider each item $i = 1, ...n$ has a weight w_i, a volume z_i and a value v_i. You still want to choose a subset of the n items so as to maximize total value. However, you now need to meet two constraints: (1) the total weight of the selected items should not exceed a given W and (2) the total volume of the selected items should not exceed a given Z.

   i. *(10 Points)* Design a DP algorithm that finds the optimal solution to this problem.
   Answer:
   This is the general case of the 2-dimensional knapsack problem (2-dim KP). Let $z(j, W, Z)$ denote the optimal solution to the 2-dim KP when only item $i = 1, ..., j$ are considered. An extension of the dynamic programming solution for the traditional (1-dim) KP can be designed as follows:

$$z(j, W, Z) = \begin{cases} 0 & \text{if } j = 0 \\ z(j-1, W, Z) & \text{if } w_j > W \text{ or } z_j > Z \\ \max\{z(j-1, W, Z), v_j + z(j-1, W - w_j, Z - z_j)\} & \text{if } w_j \leq W \text{ and } z_j \leq Z \end{cases}$$

   ii. *(5 Points)* Prove that your algorithm finds indeed the optimal solution.
   Answer:
   Optimality of $z(j, W, Z)$ is guaranteed by the optimality of subproblems $z(j-1, W, Z), z(j-1, W - w_j, Z - z_j)$ and the Bellman-equation itself (a.k.a. "optimal substructure" property).

   iii. *(5 Points)* Analyze the running time.
   Answer:
   Computing the optimal solution, $z(n, W, Z)$, corresponds to filling a table of size $O(nWZ)$. The cost of computing each block of the table is $O(1)$. Thus, the overall complexity to solve the 2-dimensional KP is, $O(nWZ)$.

   iv. *(10 Points)* Consider the example problem from class: there are $n = 5$ items with values $\{v_1 = 1, v_2 = 6, v_3 = 18, v_4 = 22, v_5 = 28\}$ and weights $\{w_1 = 1, w_2 = 2, w_3 = 5, w_4 = 6, w_5 = 7\}$; the constraint on the total weight is $W = 11$. In addition, consider that the items have volumes $\{z_1 = 1, z_2 = 10, z_3 = 3, z_4 = 2, z_5 = 2\}$ and that the total allowed volume is $Z = 15$. Apply your algorithm to solve this example.
   Answer: Applying the algorithm to the above example, we get maximum $z(5, 11, 15) = 40$ achieved by selecting items $\{3, 4\}$. (You need to compute z for all subproblems, either by hand or using a program.)

(b) *(40 Points)* **Knapsack with Repetition.**

i. *(20 Points)* Given n types of items (with weights $w_1, ..., w_n$ and values $v_1, ..., v_n$, respectively), describe a DP algorithm that finds the most valuable set of items, subject to a total weight constraint W. You are allowed to use repetition, i.e., to use items of the same type multiple times. What is the asymptotic running time of your solution? Apply your algorithm to the example with the following $n = 4$ items and capacity $W = 10$:

Item i	1	2	3	4
Weight w_i	6	3	4	2
Value v_i	30	14	16	9

Answer:
Knapsack with repetition can be solved with an array with $W + 1$ length. the following Bellman-Ford equation produces the optimal solution:

$$OPT(w) = \begin{cases} 0 & \text{if } w = 0 \\ max_{i, w_i \leq w}(OPT(w - w_i) + v_i) & \text{if } w \leq W \end{cases} \quad (1)$$

The above equation applied in a bottom-up algorithm will produce the following result:

Weight	0	1	2	3	4	5	6	7	8	9	10
Value	0	0	9	14	18	23	30	32	39	44	48

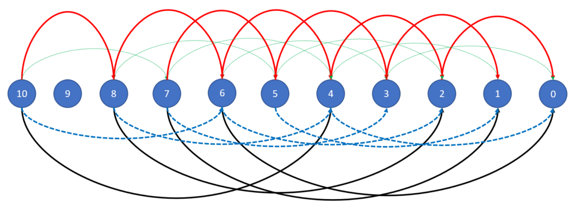
The optimal solution has value 48 and items $\{1, 4, 4\}$. For more details on the algorithm and examples, see the provided code!

Alternative solutions
Knapsack with repetition can be also solved by a small change in the standard Knapsack problem in the following way:

$$OPT(i, w) = \begin{cases} 0 & \text{if } w = 0 \text{ or } i = 0 \\ max(OPT(i, w - w_i) + v_i, OPT(i - 1, w)) & \text{if } w \leq W \end{cases} \quad (2)$$

Both solutions run in $O(nW)$ time to find optimal solution.

ii. *(10 Points)* For the above example, draw the graph with nodes the DP subproblems and edges the dependencies between them. Is it a DAG? Can you re-state this particular variant of the KP problem as finding a path on this graph?
Answer:

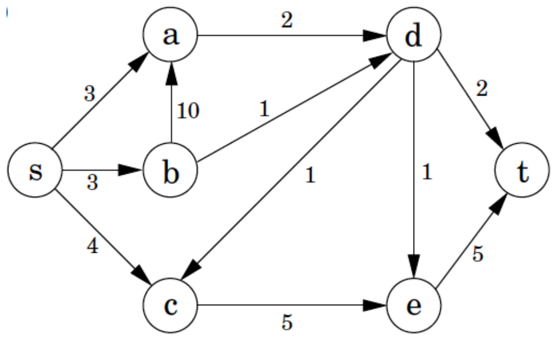


Yes, in general, in all DP formulations, this is a DAG. Nodes in the DAG represent the subproblems, and larger subproblems point (forward) to smaller subproblems. The costs of the edges represent values we obtain as per the Bellman-Ford equation.

Finding the optimal DP solution is equivalent to finding the longest path from the problem to any node. Finding longest paths can be solved in linear time in DAGs. In the particular example above, we draw the solution obtained with the top-down approach. Finding $OPT(W)$is equivalent to finding the longest path from subproblem W to 0 to any node

iii. *(10 Points)* Implement one of your DP algorithms, in `Python`, and run it on the example above. You should submit your code - the grader should be able to run your code with the input text file (format and example provided on Canvas - see `KP all.ZIP` and `README` therein) and test that it gives the correct output!
Answer:
Sample code for both solutions is uploaded on canvas: `KP_code.zip`

2. *(30)* **Shortest Paths on Graphs using Bellman-Ford Algorithm.** Consider the directed graph shown. The numbers on the edges indicate costs of these edges.



(a) *(15 Points)* Find the shortest paths from all nodes to destination t, using the Bellman-Ford algorithm. Show (some of) your intermediate steps and the final result in the following form: [next hop][distance to t] for every node.
Answer:
A summary of the synchronous execution of the Bellman-Ford algorithm is shown in the following table:

	0	1	2	3	4
t	t/0	t/0	t/0	t/0	t/0
s	∞	∞	∞	b/6	b/6
a	∞	∞	d/4	d/4	d/4
b	∞	∞	d/3	d/3	d/3
c	∞	∞	e/10	e/10	e/10
d	∞	t/2	t/2	t/2	t/2
e	∞	t/5	t/5	t/5	t/5

Note: the algorithm can stop at the 4th iteration since the last two columns are the same. We can read the "[next hop] [distance to t]" for each node from the last column, the result is represented in the following table:

Node	Next hop	Distance to t
s	b	6
a	d	4
b	d	3
c	e	10
d	t	2
e	t	5

(b) *(15 Points)* After the algorithm reaches steady state, somebody cuts off edges d-t and b-d at the same time. Use Bellman-Ford to recalculate the paths to t after the change. Does the algorithm converge? If yes, show your calculations and the final "[next hop][destination] for every node. If not, explain why.

Answer:

The changes affect nodes d and b simultaneously. Using asynchronous Bellman-Ford algorithm it is easier to recover, there are different intermediate steps depending on the order that nodes start their updates. One of the possible solutions is:

- d detects the down link and switches from next hop t to next hop e with a cost of 6. Notifies a and b about its change.
- a receives notification from d and updates the cost of its path through d from cost 4 to cost 8. Notifies s and b about its change.
- b initiates its update and switches from next hop d to next hop a with cost 18. Notifies s about its change.
- s initiates its update and switches from next hop b to next hop a with cost 11.

For each node, the final "[next hop] [distance to t]" is represented as a row in the following table:

Node	Next hop	Distance to t
s	a	11
a	d	8
b	a	18
c	e	10
d	e	6
e	t	5