MANCHESTER 1824
The University of Manchester

COMP26120 Revision

Summer Exam

Joshua Knowles

School of Computer Science

The University of Manchester

COMP26120 Semester 2, Week 11 LT 1.1, 28 April 2014

Form of the exam

- No change from recent years
- 2 hours
- 4 Questions
- Answer any 3
- Two of the questions are on the labs: 0/1 Knapsack and Optimisation; and Trees and Graph Algorithms (small world hypothesis lab)

Advice document

http://studentnet.cs.manchester.ac.uk/ugt/COMP26120/revision2014.pdf

Revising for Knapsack

- Be able to describe the 0/1 Knapsack problem in detail
- Understand how ***enumeration*** works, and limitations of enumeration (complexity)
- Understand how ***greedy*** works and limitations (not exact for 0/1 problems) and its complexity
- Understand how ***branch-and-bound*** works in detail for the 0/1 problem
  - Understand how bounds are calculated, and how they are used to make the search efficient
  - Know the worst-case complexity
  - Understand about what instances would be easy for branch-and-bound and why
- Understand how and why ***dynamic programming*** works in detail
  - Complexity of the dynamic programming approach (how this relates to table size)
  - Which instances are easy for DP and which are hard
  - How the actual list of items packed are stored and reported by DP.
- Applications of knapsack problems
- The fractional knapsack problem — solvable (exactly) by greedy. Why?

The 2013 Exam

http://studentnet.cs.manchester.ac.uk/assessment/exam_papers/UG_sem2_2012/COMP26120sem2.pdf

Model Answer to 2(a)

The bound is an estimate of the value of a partial solution (or equivalently a set of solutions descended from that solution), that is a solution with only some of the items to be taken or left out specified.

It is an upper bound which means it must be greater than or equal to the true value.

A partial solution may be represented like this 010010****....

The bound is calculated by adding in those items specified as 1 and leaving out those specified by 0. Note: items must be in decreasing value-to-weight order.

The greedy algorithm is applied to the remaining unspecified part of the solution. It "packs" the items in the order given until the capacity of the knapsack is reached or exceeded, at which point pack that fraction of the previous item needed to precisely fill the knapsack. The value of the part-item is its value times its fraction. The bound is the total value from this calculation of items packed.

The bound is used in branch-and-bound to determine whether a node is worth explanding or could be pruned. A node with upper bound that does not exceed the current best solution found should not be expanded.

Marking scheme: Up to 2 marks for definition, up to 2 marks for description/naming of greedy method (with items ordered appropriately), up to 2 marks for pruning the search tree based on whether greater or less than best/incumbent solution.

Model Answer to 2(b)

(i) The optimal solution has value 3003+274+47 = 3324. The solution found by greedy has value 3003+174+83 = 3260.

Marking scheme: 1 mark for the greedy solution. 2 marks for the optimal or for a precise argument why the greedy is not optimal in this instance.

(ii) Process the items in the order given. But when considering an item, check if there are subsequent items with the same value:weight ratio. If there are, take the largest of these items that will fit next. This would give the following sequence of considering the items for addition to the knapsack

argmaxvalue(item1,item2) = item1 (add to knapsack)
item2 does not fit
argmaxvalue(item3,item4) = item4 (add)
item3 does not fit
argmaxvalue(item5,item6) = item5, but item5 does not fit
item 6 does fit so add.

This is optimal for this instance as required.

Marking scheme: A solution such as this, or any alternative scheme that works gets 4 marks. Lose marks for lack of detail or accuracy.

(iii) Dynamic programming is not efficient in this case because the instance has few items but large item weights that are also not easily factorizable (relatively large primes). DP has complexity O(NC). Here C is three orders of magnitude larger than N so DP would be inefficient compared to greedy which is O($N \log N$) including sorting, or even Branch and Bound which is O(2^N).

Marking scheme: 1 mark for mentioning item sizes relative to number of items. 1 mark for not factorizable. 1 mark for comparison to either branch and bound complexity or greedy (does not need to be in Big-Oh notation).

Model answer to 2(c)

The table entries store a trace of optimal solution values (profits) for subproblems (optimal subsolutions).

The decisions are whether to store an item or not.

The subproblems involve a smaller capacity knapsack and a subset of the items only.

The optimal decision for the current item and knapsack capacity does not need to consider every item before it again, but is based on looking up (in the table) and comparing (i) the relevant optimal subsolution for the smaller knapsack problem created by taking away the current item's size from the knapsack capacity, and considering only the items "before" the current one (in case we pack the item) and (ii) the optimal subsolution of the same knapsack capacity without the current item.

It is easy to solve the 0/1 knapsack problem for 1 item (for all capacities), and the above method makes it easy for $j + 1$ items, given a solution for j items, and so on (by induction). This is how the DP exploits the Bellman Principle.

Marking scheme: 1 mark for identifying the subsolution as a profit value. 1 mark for subproblem as being a certain capacity and certain subset of the items. 1 mark for identifying the decisions. 1 mark for explaining how the method of building up from an easy problem works.

Answers to some student questions

If you were a student, how would you prepare for the exam?

Follow the advice given. Go through the lecture slides. Use the course textbook. Look back over relevant labs. Do the past examination papers. Ask by email if there is anything I still don't understand after all that.

What are you expecting of the students to learn from this unit?

This is detailed on the webpage for the course. We are interested that students understand some core algorithms, are able to design their own algorithms, and able to reason about the complexity and correctness of algorithms.

How are is this reflected in the exam? What are you looking for in the exam?

Questions are asked about the algorithms we have studied, and about correctness and complexity.

We probe understanding by asking to explain particular cases and to explain how fixes or variations of algorithms compare in terms of complexity.

What would be an ideal exam?

I think the current exam is ideal. We try to make the questions clear, fair and able to distinguish understanding. Our marking scheme is detailed.

The only way to do it better would be to interview every student individually (as is done in Italy) ;-)