Algorithms and Data Structures

Seminars Guide, 2023–24

José Luis Balcázar

Departament de Ciències de la Computació, UPC

Session 1, 11/01: Review of various notions, including recursion

Today:

- Short briefing
- Higher order
- Graphs and trees
- Recursion
- Evaluable exercise (slight corrections made the day after)
- Exercises for the week
- Additional related material

Short, partial briefing including Jutge courses.

- Most info, including evaluation, deferred to the theory class on Monday.
- Mandatory lab exercises to do *in class* most Thursdays and to be submitted through the Aula Virtual are part of the evaluation, including *today*'s.
- Attendance to the Lab Sessions is *mandatory.* Only Aula submissions made from the classroom will be considered. *I want to see you submitting from our room.*
- Jutge courses:
    - Specific, *mandatory*, non-public (invitations sent round):
        * Algorithms and Data Structures Winter 2023-24 ESCI.
        * This course will be expanded along the quarter with the addition of some new lists or new problems;
        * further, exams will refer to it.
    - Optional, open, *recommended*: Ad computationem vIA reGIA.
    - Optional on demand: ask me for invitations to the first-year courses of 2023-24 if you believe they can be helpful to you.

Higher order

1. Functions as arguments *or results* of other functions. Examples.
2. Higher order for today:
    - Reading items with `pytokr` and
    - looping on them all.
    - The problems in the first three lists of the Jutge course will give you practice with `pytokr` and with general higher order programming.
    - They are also part of the current version of Programming and Algorithms 1.

Graphs and trees

Review and terminology:

- graphs, directed and undirected,
- free trees,
- rooted general trees,
- binary trees.

How are we representing binary trees in our programs?

And in our input data?

Recursion

The fourth list in the Jutge course contains problems to be solved *recursively.* You are likely to have iterative solutions of many of them. They are no longer what is required now! Work *only* recursively, *no loops.*

Goal for our first couple of weeks: proficiency with recursive programs with *more than one* recursive call per call.

Examples:

- reading binary trees,
- reading and traversals of binary trees (P98436).

Specific programming exercises:

1. Write first a program that reads binary trees from `stdin` as per P98436, but assuming just one integer per line, so that the standard function `input()` will keep providing you with the next element to be read.

2. Test it out, e. g. by asking for specific components of the tree just read within the Python interpreter.

3. Move on into a program that reads the tree without assuming anymore that the values come in different lines, using `pytokr()`, and test it out in the same way.

4. Then, complete the program with traversals and try to get green light (or at least yellow due to presentation error) in the Jutge problem P98436.

Evaluable exercise

Upload to the Aula a screen capture of your best attempt to solving Jutge problem P98436 before leaving the classroom.

Exercises for the week

All can be found in our course on `jutge.org`, within the list Recursion Revisited.

1. Solve P61120.
2. Generating (of course, *recursively*, no loops) all subsets of a given set of words: solve P18957.
3. Evaluating prefix expressions: solve P20006.
4. Size-based encoding of binary trees: solve X30150 (midterm exam problem last year).
5. Reading and traversal of general trees: solve P66413. Feel free to choose to ignore the `C++` code suggestions and work instead fully in Python, if you prefer that option.
6. Once you have working Python solutions, consider trying `C++` (P57669).

We will get back to all this later on with alternative programs for tree traversals and with the *graph variant* of tree preorder traversal, namely, *depth-first search* (DFS).

Additional related material

To be added here two weeks after the session.

Session 2, 18/01: Backtracking, I: Knapsack variants

Today:

- Individual review: Decisional Knapsack by backtracking, first solution
- Decisional Knapsack by backtracking, all solutions
- Subset Sum
- Evaluable exercise
- Exercises for the week
- Additional related material

Individual Review: Decisional Knapsack by backtracking, first solution

Review the notion and first examples of **set-based backtracking** from the theory session, namely, single-solution Knapsack by exhaustive search and by set-based backtracking. The codes discussed in the theory session are available through the Aula Virtual.

You will find also there a text file with examples to cut and paste from. How to read in the data for one of these cases? Observe that, first, we find the desired value and the weight bound and, after them, the number of objects is given. Then, we need to read a known number of integers (two for each object). For all these readings, the first function returned by `pytokr` suffices: no iterator is necessary.

Decisional Knapsack by backtracking, all solutions

1. *Optionally*: Start by modifying the exhaustive search algorithm based on the tree of subsets (file name `...simple_first_no_back.py`) so as to return the list of all the valid solutions for a given upper bound on weight and a given lower bound on value. Try it on the examples in the text file mentioned above. *Alternatively*, you can choose to skip this part and move on directly to the next point.
2. Modify the single-solution backtracking (file name `...simple_first.py`) so as to return the list of all the valid solutions for a given upper bound on weight and a given lower bound on value. Try it on the examples in the text file mentioned above.
3. Once your backtracking program works correctly on all these examples, adapt it a bit so as to try it out in the Jutge: X94664. Your Jutge submission can print out the solutions in any order; also, within each solution, the object numbers may be printed out in any order. Follow the format of the public tests.

Subset Sum

Consider the following problem, similar to Knapsack but slightly simpler. Input consists of a given positive integer as a *goal* (you may think of it as change to be given for a payment) and a sequence of positive integers, the *addends*, possibly repeated (you may think of it as the values of the currency items available to us like coins and bills). Is there a way of selecting a set of addends from the sequence that adds up exactly to the goal?

This problem is widely known as the *Subset Sum* problem.

Your program must read these data and answer the question by applying the **set-based backtracking** schema.

1. First, write a program that returns one solution, provided that one exists.
2. Then, change your program so that it returns all solutions.

Evaluable exercise

Solve P40685 in list Combinatorial Search Schemes I; upload to the Aula a screen capture of your best attempt before leaving the classroom.

This exercise is to be solved individually.

You will recognize in P40685 yet another variant of Subset Sum.

However, **be careful:** the condition that all input integers are positive is not guaranteed anymore. Hence, your programs for Subset Sum that assume that goal and addends are positive might not work. Given this change, how do you propose to organize your algorithm so as to detect subtrees where we can safely cut out the exploration?

Exercises for the week

1. Solve some pending exercises you may have still left from the list Recursion Revisited.
2. Like P40685, except that you only look for one solution, and stop the search as soon as found. (Of course, it's no use submitting this solution to the Jutge.)
3. Read the statement of Interval Subset Sum, X56351 in list Combinatorial Search Schemes I. The text refers to an optimization criterion ("shortest solution") that will be covered in the next sessions. For the time being, solve instead the following two simplified forms: finding just one arbitrary solution and finding all the solutions. (Of course, again it would be no use submitting these intermediate programs to the Jutge yet.)

Additional related material

To be added here two weeks after the session.