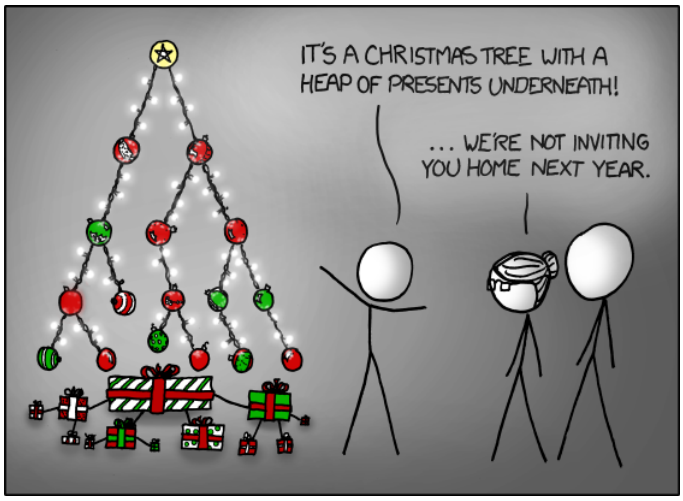
Assignment 9: Data structures

ETH Zurich

Hand-out: 19 November 2012
Due: 28 November 2012



Tree © Randall Munroe (http://xkcd.com/835/)

Goals

- Figure out when to use which data structure.
- Practice using data structures.
- Practice implementing your own data structures.

1 Choosing data structures

In the lecture you have learnt that different data structures can represent different kinds of information and support efficient implementation of different operations.

To do

For each of the use cases below pick a data structure (from the ones you have seen in the lecture) that is best suited for that use case, and explain your choice. If you think there is more than one suitable data structure, briefly discuss the trade-offs.

Example. You want to store the names of weekdays and access them by the number of a day within the week.

Answer. An array, because the number of weekdays is fixed, and access by index has to be efficient.

1. You want to store the stations of a public transportation line. New stations can be added to both ends of the line, but not between existing stations. You should be able to traverse the line in both directions.

2. You want to store a phone book, which supports looking up a phone number by name, as well as adding and removing entries.

3. You are looking for a way out of a maze, and you are not allowed to use recursion. You have to store the path you are currently exploring, and be able to go back one step whenever you find yourself in a dead-end and explore a new possibility from there.

4. You want to store a sorted list of strings and support the operation of merging two sorted lists into one, in place (without creating a copy of the lists).

5. You are writing software for a call center. When a client calls, his call should be stored until there is a free operator to pick it up. Calls should be processed in the same order they are received.

To hand in

Hand in your answers for questions 1–5.

2 Short trips: take two

In the previous assignment you had to implement a feature that highlights all the stations that are reachable from a given station in two minutes or less. If your implementation is similar to the master solution, then it doesn't keep track of the stations it already visited and keeps visiting the same stations again and again. As a result, highlighting the short-trip range for stations with a lot of connections (for example, Bürkliplatz) takes a long time.

Note that it is incorrect to simply stop the search whenever the next station you are about to visit is already highlighted: the route you had found the first time around could be longer than the one you are currently building, and thus you could miss some reachable stations.

In this task you have to find a way to make *highlight_short_distance* more efficient by keeping track of the state of traversal using an appropriate data structure.

To do

1. You can start either from your own solution to "Short trips" from the previous assignment or from the master solution: `http://se.inf.ethz.ch/courses/2012b_fall/eprog/assignments/09/traffic.zip`.

2. What kind of information do you have to keep track of, so that the traversal does not miss reachable stations, but also does not visit the same station unnecessarily many times? Choose an appropriate data structure class from the EiffelBase2 library (see Table 1) to store this kind of information, and introduce an attribute of the corresponding type into *SHORT_TRIPS*. Modify your implementation of *highlight_reachable* to make use of the introduced data structure.

Data structure	Concrete implementations	Description
V_ARRAY	*V_ARRAY*	Arrays: elements are indexed with integers from a continuous interval
V_LIST	*V_ARRAYED_LIST*, *V_LINKED_LIST*	Lists: elements can be inserted and removed at any position
V_TABLE	*V_HASH_TABLE*, *V_SORTED_TABLE*	Tables: values are indexed by keys; key-value pairs can be updated, added and removed
V_SET	*V_HASH_SET*, *V_SORTED_SET*	Sets: elements are unique and lookup is efficient
V_STACK	*V_LINKED_STACK*	Stacks: last in first out policy
V_QUEUE	*V_LINKED_QUEUE*	Queues: first in first out policy

Table 1: Data structure classes in EiffelBase2

3. Compare the performance of *highlight_short_distance* before and after the modification (the difference is especially clear if you change the initial time interval from two to three minutes).

To hand in

Hand in the code of *SHORT_TRIPS*.

3 Bags

A *bag* (also called *multiset*) is a generalization of a set, where elements are allowed to appear more than once. For example, the bag $\{a, a, b\}$ consists of two copies of a and one copy of b. However, a bag is still unordered, so the bags $\{a, b, a\}$ and $\{a, a, b\}$ are equivalent.

In this task you have to implement some features for a linked representation of bags. This representation is very similar to a regular singly-linked list, except for the following:

- In addition to the value and the reference to the next cell, each bag cell stores the number of copies of its value (see Figure 1), which is always positive.
- For a given value, at most one cell storing that value should appear in the data structure.

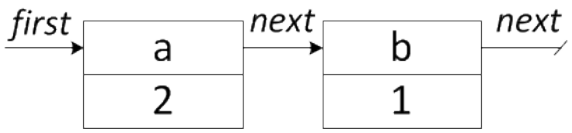


Figure 1: A possible linked representation of the bag $\{a, a, b\}$.

To do

1. Download `http://se.inf.ethz.ch/courses/2012b_fall/eprog/assignments/09/bag.zip` unzip it and open `linked_bag.ecf`. Open class *LINKED_BAG*.
2. Fill in the implementations of the following features:

- *add* (*v*: *G*; *n*: *INTEGER*), which adds n copies of v to the bag.
- *remove* (*v*: *G*; *n*: *INTEGER*), which removes as many copies of v as possible, up to n. For example, removing one copy of a from the bag $\{a, a, b\}$ will result in a bag $\{a, b\}$, while removing two copies of c from the same bag will not change it.
- *subtract* (*other*: *LINKED_BAG* [*G*]), which removes all elements of *other* from the current bag. For example, taking the bag $\{a, a, b\}$ and subtracting $\{a, b, c\}$ from it will yield the bag $\{a\}$.

Your implementation should satisfy the provided contracts.

3. Run the program: this will test you implementation on a simple example. Make sure the test does not fail.

To hand in

Hand in the code of *LINKED_BAG*.