1 Introduction

A Choice Dictionary is a datatype that is initialized with a parameter $n \in \mathbb{N}$ that maintains an initially empty subset S of $\{1, ..., n\}$ under the following operations:

Name	Precondition	Description
insert(i):	$(i \in U)$	$S := S \cup \{i\}$.
delete(i):	$(i \in U)$	$S := S \setminus \{i\}$.
contains(i):	$(i \in U)$	Returns 1, if $l \in S$, 0 otherwise.
choice():		Returns $x \in S$, 0 if $S = \emptyset$.

For more information see 'An Optimal Choice Dictionary' by T. Hagerup.

This program contains several implementations of datastructures that realize the datatype. In addition you can realize your own datastructure and compare it with existing implementations.

2 Installation

Simply download the github archive and type *make*. Prerequisites are *make* and a compiler that supports *C++14*. You can include both header files *src/cd.h* and *src/cd_dev.h* in your code. When compiling you need to link the appropriate archive, i.e. *lib_cd.a* or *lib_cd_dev.a*.

3 Example

In the following example two Choice Dictionaries are profiled on one input.

```
auto c0 = create_choice_dictionary(1, cd_implementations::optimal);
auto c1 = create_choice_dictionary(1, cd_implementations::simple);
Profiling_file pf;
pf.set_name("random");
pf.set_max_argument_size(100000);
auto o = create_random_operations(100000, 100000);
pf.add_operations(o);
profile_cds(pf, "dst_path.txt", {c0,c1},{"optimal", "simple"});
```

The Profiling_file is used as a container for the input and information about the input. The last call will result in a file called *dst_path.txt* which will contain information about the result.

Let's look at another example of just using a Choice Dictionary:

```
cd c(100); // Create the Choice Dictionary with n=100.
c.insert(50);
cout << c.choice() << endl;
cout << c.contains(50) << endl;
c.del(50); // Choice Dictionary is empty.
```

4 For Users

All necessary declarations for users are in *cd.h.* A compilation wall exists in form of the *pimpl idiom.* The class *cd* contains the following operations (that are self explaining):

```
cd(_t::len_t n);
cd(_t::len_t n, std::vector<_t::len_t> init_list);
~cd();
void insert(_t::len_t i);
void del(_t::len_t i);
_t::len_t choice();
bool contains(_t::len_t i);
```

The second constructor will initialize the datastructure with the list (per insert).

5 For Developers

For a complete overview of the functions have a look at *cd_dev.cpp* and *Profiling_file.cpp.* Here we will only cover a couple basics.

All Profiling happens via a Profiling_file. Generally there are two cases:

1. Profiling several Choice Dictionaries with one Profiling File. Used to check which one is the best.
2. Profiling several Profiling Files with one Choice Dictionary. This is mainly helpful to achieve further insights into the datastructure.

The related functions are *profile_pfs* and *profile_cds.* For more information on profiling see the directory *Tests.*