

Exercise 1: Sparse vectors

A *sparse vector* is a vector in which most of the elements are zero. To save space, we can store only the non-zero elements.

Implement a sparse vector using a symbol table of index-value pairs so that the memory is proportional to the number of nonzeros. The **set** and **get** operations should take $\log n$ time in the worst case; taking the dot product of two vectors should takes time proportional to the number of nonzero entries in both.

Implement **Svector** class with the following methods:

- **void set(int i, double x):** set the *i*th entry to *x*
- **double get(int i) const:** return the *i*th entry
- **double dot(const Svector& that) const:** return the dot product of **this** vector with **that** vector
- **double norm() const :** return the Euclidean norm $\|x\|$ of the vector *x*,
where $\|x\| = \sqrt{x_1^2 + x_2^2 + \dots + x_n^2}$
- **Svector add(const Svector& that):** return the sum of **this** vector with **that** vector
- **void scale(double alpha):** multiply **this** vector with a scalar

You may use **std::map** for the symbol table.

Exercise 2: LRU cache

Create a data structure that supports the following operations: **access** and **remove**. The **access** operation inserts the item onto the data structure if it's not already present. The **remove** operation deletes and returns the item that was least recently accessed.

Hint: maintain the items in order of access in a doubly linked list, along with pointers to the first and last nodes. Use a symbol table with keys = items, values = location in linked list. When you access an element, delete it from the linked list and re-insert it at the beginning. When you remove an element, delete it from the end and remove it from the symbol table.

Implement the **LRU** class with the following methods:

- **void access(int item):** insert the item if not already present
- **int remove():** remove and return the least recently accessed item
- **void print():** print the items in the order of access
- **bool contains(int item):** return **true** if the item is present
- **int size():** return the number of items in the cache
- **bool empty():** return **true** if the cache is empty

Useful classes from STL:

- **std::map:** Red-black trees based symbol table with $\log n$ time for **get** and **set** operations. <https://en.cppreference.com/w/cpp/container/map>
- **std::list** Double linked list <https://en.cppreference.com/w/cpp/container/list>