Algorithms and Data Structures

Author: Daria Shutina

Organization stuff

Professor Kinga Lipskoch

- Office: Research I, Room 94
- Telephone: +49 421 200-3148
- E-Mail: klipskoch@constructor.university
- Office hours: Mondays 10:00 - 11:00

Lecture slides: https://grader.eecs.jacobs-university.de/courses/ch_231_a/2023_1/

Homeworks:

Use Grader for homework submission. Do not forget to change semester from Fall 2022 to Spring 2023.

Submit ZIP file containing one PDF file and source code fileswith makefile.

Homeworks are not mandatory.

VPN for remote access to Jacobs Net:

Source: https://teamwork.jacobs-university.de/display/ircit/VPN+Access

Domain name: vpnasa.jacobs-university.de (Jacobs VPN)

Tutorials:

- 2 weekly tutorials given by one TA
- Tutorial before homework deadline
- Online via Teams, Saturdays, 19:00 – 21:00
- Online via Teams, Sundays, 19:00 – 21:00

23-02-02

Templates

Templates allow to write generic code, i.e., code which will work with different types. Later a specific type will be provided and the compiler will substitute it.

Motivation for using templates is to eliminate snippets of code that differ only in the type and do not influence on the logic.

While using templates, operators for some types need to be overloaded.

Why function definitions are written in `.h` files?

Both declaration an definition are written in `.h` files! Otherwise, the code will fail. The reason is that, when instantiating a template, the compiler creates a new class with a definite type. Consequently, it has to instantiate methods. But it cannot find methods with an appropriate type, since they are written in another `.cpp` file.

If you still want to have declaration and definition of a template in different files, there are some options (but think twice).

One possible solution is to instantiate a template in the same file where methods are written:

```cpp
// a.cpp
template <class T>
void MyTemplate<T>::do() {
    // do sth
}

template class MyTemplate<int>;
template class MyTemplate<std::string>;
```

Another solution is to include a `.cpp` file inside the header file (wtf??). Btw, you do not have to compile this file with others:

```cpp
// a.h
template <class T>
class MyTemplate {
    T prop;
public:
    void do() {}
}

#include a.cpp
```

Basic syntax

Type parameterization in classes:

```cpp
template <class T>
class MyClass {
    T i;
public:
    MyClass() = default;
    MyClass(T arg) : i(arg) {}
};

int main() {
    MyClass<int> aboba;
}
```

Type parameterization in functions:

```cpp
template <class T>
void my_function(T arr[], int size) {
    for (int i = 0; i < size; ++i) {
        std::cout << arr[i] << ' ';
    }
}
```

Note: `class` in `<class T>` can be replaced with `typename`.

Overloading an operator

```cpp
template <class T>
class MyClass {
    T array[size];
public:
    BoundedArray(){};
    T& operator[](int);
};

...

template<class T>
T& MyClass<T>::operator[](int pos) {
    if ((pos < 0) || (pos >= size))
        exit(1);
    return array[pos];
}
```

23-02-07

Fundamental Container Classes

- Sequence containers
    - vector
        - fast random access
        - fast inserting/removing at the end, slow inserting/removing in the middle
        - not efficient while resizing
    - deque
        - a dynamic array which can grow in both directions
        - random access with a constant complexity
        - fast inserting/removing at beginning/end, slow inserting/removing at the middle
    - list
        - no random access
        - fast inserting/removing at beginning/end, slow inserting/removing at the middle
- Associative containers
    - set
        - implemented as a BST
        - sorted
        - duplicates are not allowed
    - multiset
        - as a set, but duplicates are allowed
    - map
        - stores `key/value` pairs
        - the key is the basis for ordering
        - keys are not duplicated
        - called "associative array"
    - multimap
        - as a map, but keys can be duplicated
        - called "dictionary"
- Container adapters
    - stack
    - queue
    - priority queue

Set operations

For a set, you can use such operations as

- `set_union(...)` -- $A \cup B$
- `set_difference(...)` -- $A \backslash B$
- `set_intersection(...)` -- $A \cap B$
- `set_symmetric_difference(...)` -- $(A \backslash B) \cup (B \backslash A)$

23-02-14

decltype vs auto

Both tools are used in order to avoid writing long types.

The `decltype` keyword yields the type of a specified expression. Roughly, it is an operator that evaluates the type of passed expression. For example, it is useful with templates, when the type of the return result depends on the type of the template.

```cpp
#include <bits/stdc++.h>

int fun1() { return 10; }
char fun2() { return 'g'; }

int main() {
    decltype(fun1()) x;
    decltype(fun2()) y;

    std::cout << typeid(x).name() << std::endl;  // i
    std::cout << typeid(y).name() << std::endl;  // c
}
```

The `auto` keyword specifies that the type of the variable that is being declared will be automatically deducted from its initializer. The variable declared with auto keyword should be initialized at the time of its declaration, otherwise there will be a compile error.

```cpp
#include <bits/stdc++.h>

int main() {
    set<string> st = { "ab", "ob", "a" };
    auto it = st.begin();
    std::cout << *it << std::endl;  // a
}
```

Attention! `auto` emits reference (and `decltype` does not). If you want something like `int&`, you need to write `auto&`.

Lambda functions

```cpp
auto my_lambda = [ /*captureList*/ ] ( /*params*/ ) -> /*returnType*/
{
    ...
}
```

Options for the capture list:

- capture by value

```cpp
int num = 100;

auto my_lambda = [num] (int val) {
  return num + val;
};

std::cout << my_lambda(100) << '\n'; // 200
std::cout << num << '\n';            // 100
```

- capture by reference

```cpp
int num = 100;

auto my_lambda = [&num] (int val) {
  num += val;
  return num;
};

std::cout << my_lambda(100) << '\n'; // 200
std::cout << num << '\n';            // 200
```

Variadic templates

Variadic templates are templates that can take zero or positive number of arguments.

```cpp
#include <iostream>

void print() {
    std::cout << "I am empty function\n";
}

// Variadic function Template that takes variable
// number of arguments and prints all of them.
template <typename T, typename... Types>
void print(T var1, Types... var2) {
    std::cout << var1 << std::endl;
    print(var2...);
}

int main() {
    print(1,
          2,
          3.14,
          "Pass me any number of arguments",
          "I will print\n");
}
```

Variadic functions

In comparison with variadic templates, variadic functions have a fixed amount and a fixed type of given arguments.

You can use `cstdarg` library for variadic functions. There are library functions such as `va_list`, `va_start`, `va_copy`, `va_end`, etc.

```cpp
#include <iostream>
#include <cstdarg>

// `count` is the number of parameters
// `...` reflect the fact that the function is variadic
double average(int count, ...) {
    va_list ap; // the list of given arguments
    int j;
```

```
    double sum = 0;

    va_start(ap, count);
    for (j = 0; j < count; j++) {
        sum += va_arg(ap, double); // gets one argument and
                                   // increments `ap` to the next argument.
    }
    va_end(ap);

    return sum / count;
}

int main() {
    std::cout << average(4, 1.0, 2.0, 3.0, 4.0) << '\n';
    std::cout << average(2, 3.0, 5.0) << '\n';
    std::cout << average(5, 3.2, 3.2, 3.2, 3.2, 3.2) << '\n';
    return 0;
}
```

23-02-16

Analysis of time complexity

Types of analisys:

- worst case
- average case

Asymptotic Analysis

$$f = \Theta(g) \Leftrightarrow \exists n_0, c_1 > 0, c_2 > 0 \ : \ c_1 \cdot g(n) \leqslant f(n) \leqslant c_2 \cdot g(n), \forall n > n_0.$$

Upper bound:

$$f = O(g) \Leftrightarrow \exists n_0, c > 0 \ : \ 0 \leqslant f(n) \leqslant c \cdot g(n), \forall n > n_0.$$

Lower bound:

$$f = \Omega(g) \Leftrightarrow \exists n_0, c > 0 \ : \ 0 \leqslant c \cdot g(n) \leqslant f(n), \forall n > n_0.$$

Non-tight upper bound:

$$f = o(g) \Leftrightarrow \forall c > o\ \exists n_o\ :\ 0 \leqslant f(n) < c \cdot g(n), \forall n > n_0$$
$$\Leftrightarrow \lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$$

Non-tight lower bound:

$$f = \omega(g) \Leftrightarrow g = o(f)$$
$$\Leftrightarrow \lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$$

Computation with limits

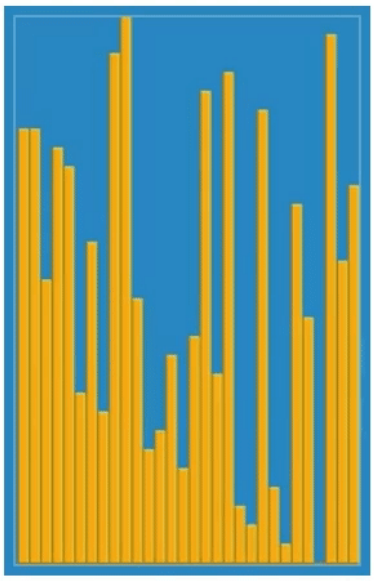
$f \in o(g)$	$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$
$f \in O(g)$	$\lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty$
$f \in \Omega(g)$	$\lim_{n\to\infty} \frac{f(n)}{g(n)} > 0$
$f \in \Theta(g)$	$0 < \lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty$
$f \in \omega(g)$	$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$

23-02-21

Sorting algorithms

Insertion sort

Go from 0 to $n-1$. Consider a current element $a[k]$ and insert it in $a[0..k-1]$.



```
INSERTION-SORT(A, n)
 for j = 2 to n
     key = A[j]
     // Insert A[j] into the sorted sequence A[1 . . j − 1].
     i = j − 1
     while i > 0 and A[i] > key
         A[i + 1] = A[i]
         i = i − 1
     A[i + 1] = key
```

Time complexity:

$O(n^2)$. We make $n-1$ iterations in total. Imagine that, on every iteration, we have to compare an element with all elements in front of it. Thus, we have $1+2+\ldots+(n-1)=\frac{n(n-1)}{2}=O(n^2)$ comparisons in total.

Correctness:

base $n=1$. An array of size 1 is sorted.

Transition: $a[0..k-1]$ is sorted. We consider an element $a[k]$ and insert it between $a[i]$ and $a[i+1]$ so that $a[i] \leqslant a[k] \leqslant a[i+1]$. Thus, we get a sorted array $a[0..k]$.

Merge Sort

Recursively sort $a[0..\frac{n}{2})$ and $a[\frac{n}{2}..n)$. Then merge them.

```
1  void merge(std::vector<int> &a, int left, int mid, int right)
2  {
3      int leftSubArraySize = mid - left;
4      std::vector<int> leftSubArray(leftSubArraySize);
```

```
    for (int i = 0; i < leftSubArraySize; ++i) {
        leftSubArray[i] = a[i + left];
    }

    int rightSubArraySize = right - mid;
    std::vector<int> rightSubArray(rightSubArraySize);
    for (int i = 0; i < rightSubArraySize; ++i) {
        rightSubArray[i] = a[i + mid];
    }

    int indexOfRightSubArray = 0;
    int indexOfGivenArray = left;

    for (int indexOfLeftSubArray = 0; indexOfLeftSubArray < leftSubArraySize;
indexOfLeftSubArray++) {
        while (indexOfRightSubArray < rightSubArraySize
               && rightSubArray[indexOfRightSubArray] <=
leftSubArray[indexOfLeftSubArray]) {
            a[indexOfGivenArray++] = rightSubArray[indexOfRightSubArray++];
        }
        a[indexOfGivenArray++] = leftSubArray[indexOfLeftSubArray];
    }

    while (indexOfRightSubArray < rightSubArraySize) {
        a[indexOfGivenArray++] = rightSubArray[indexOfRightSubArray++];
    }
}

void mergeSort(std::vector<int> &a, int left, int right) {
    if (right - left <= 1) return;
    int mid = left + (right - left) / 2;
    mergeSort(a, left, mid);
    mergeSort(a, mid, right);
    merge(a, left, mid, right);
}
```

Time complexity:

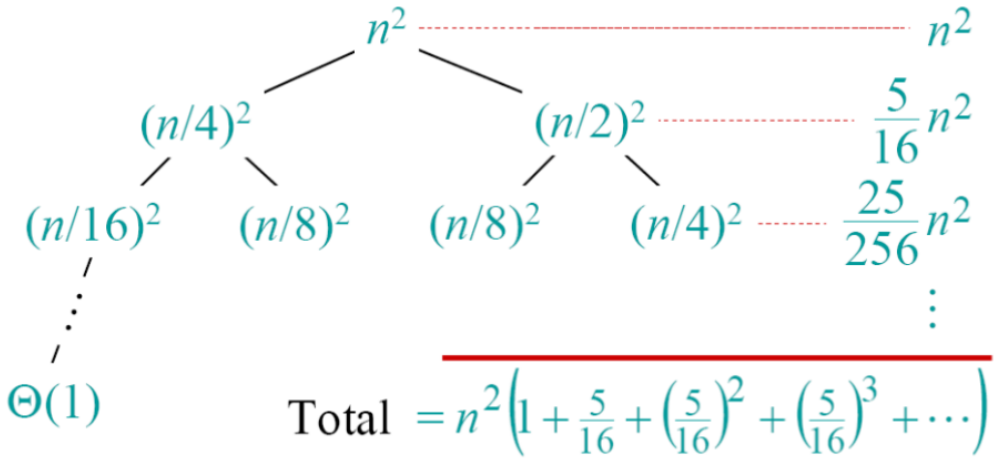
$$T(n) = \begin{cases} O(1),\ if\ n = 1 \\ T(\frac{n}{2}) + \Theta(n),\ if\ n > 1 \end{cases}$$

$T(n)$ is divided by 2, so we can build a recursion tree, which height is $O(\log n)$ and the ampunt of leaves is $O(n)$ $\Rightarrow$ total time complexity is $O(n \log n)$.

Recurrence Solving Methods

Recursion tree

$$T(n) = \begin{cases} T(\frac{n}{4}) + T(\frac{n}{2}) + n^2, \ n > 1 \\ \Theta(1), \ if \ n = 1 \end{cases}$$



$n^2(1 + \frac{5}{16} + (\frac{5}{16})^2 + \ldots) = O(n^2)$.