Comparison of Java and C for Graph Traversal Implementation

We will consider C and Java implementations for DFS and BFS graph traversal algorithms (4 programs) and compare their source codes and binaries to figure languages shortcomings and fortes in application to the given task.

BFS and DFS Algorithms in Psuedo Code

```
dfs(from vertex v)
    visit(v)
    for each neighbor w of v
        if w is not visited
            dfs(w)
            add edge vw to tree T

bfs(from vertex x)
    visit x
    list L = x
    while L nonempty
        choose some vertex v from the front of the list
        for each not visited neighbor w
            visit w
            add w it to end of L
```

Each traversal will be implemented on undirected graph of cities, where each node represents a city with information on its population, elevation and name. Algorithm stops when the target node is found or all vertices are visited.

Configurations

Java implementation is written for `Java 8` development platform, compiled without any flags and launched with the following flags: `java -XX:+AggressiveOpts -XX:+UseParNewGC -XX:+UseConcMarkSweepGC -server -Xmx8g -d64 BFS 6`. Clarification of options used may be found on oracle site. C implementation is written for GNU Compiler Collection and compiled with in the following way: `gcc BFS.c -O3 -o bfs.exe -include Expression.h`. Java implementation contains Recursive Descent Parser for parsing expressions that describe target graph node. C implementation instead takes search expressions in include file `Expression.h`. Measurements are conducted on `Windows 7x64` machine with 3gb of RAM and Pentium Dual Core processor (2 threads).

Measurements of execution times are conducted with separate measurements of time spent for input parsing, algorithm execution and output production. Measurements are realized with code instrumentation with calls to custom profiler and `time` command of `Cygwin` or other akin environment. The following source code excerpt demonstrates instrumentation:

```
Profiler.startTimeProfile("OUTPUT");
// ...writing output...
Profiler.endTimeProfile("OUTPUT");
```

Inputs are supplied in the following files:

- AdjacencyMatrix_of_Graph_G.txt
- Node_Information_of_Graph_G.txt
- SourceNode_StoppingCondition.txt

Outputs are written to files:

- Result*of_BFS_from_Source_to_Target_N*{number of visited nodes}.txt for BFS
- Profile*of_DFS_from_Source_to_Target_N*{number of visited nodes}.txt for DFS
- Profile_times_of_BFS.txt for BFS

Files of C implementation:

```
GraphTraversalC
+-- BFS.c
+-- DFS.c
+-- Graph.h // graph representation
+-- Profiler.h
L-- Tool.h  // error handling and input parsing procedures
```

Files of Java implementation:

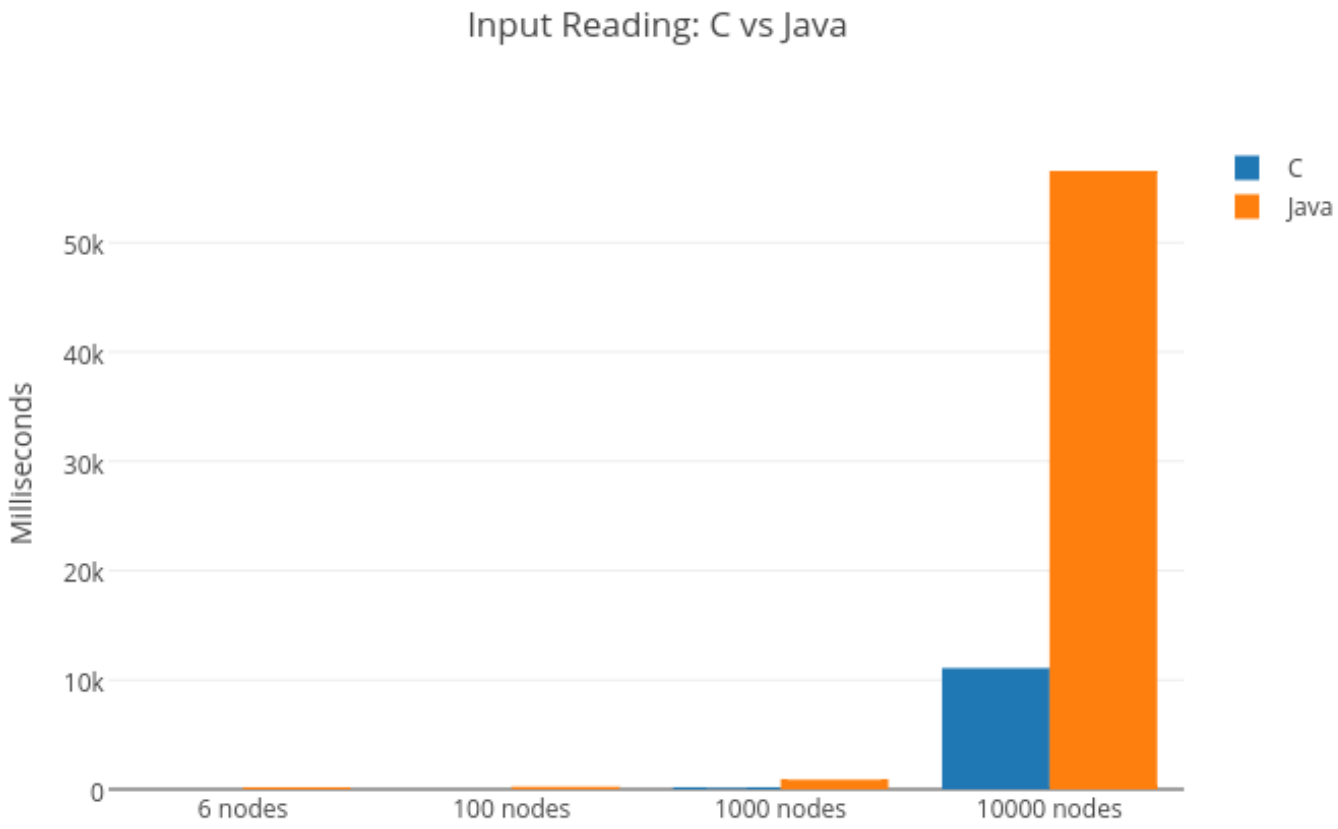
```
GraphTraversalJava
+-- BFS.java
+-- DFS.java
+-- Graph.java
+-- Profiler.java
+-- RecursiveDescentParser
¦   +-- AST.java // Abstract Syntax Tree
¦   +-- ExpLexer.java
¦   +-- ExpParser.java
¦   L-- Token.java
+-- Tool.java
L-- TraverseAlgorithm.java // Base class for algorithms
```

Languages Comparison

Implementations will be compared on the basis of number of characteristics and features present to figure out shortcomings and strong parts of the languages.

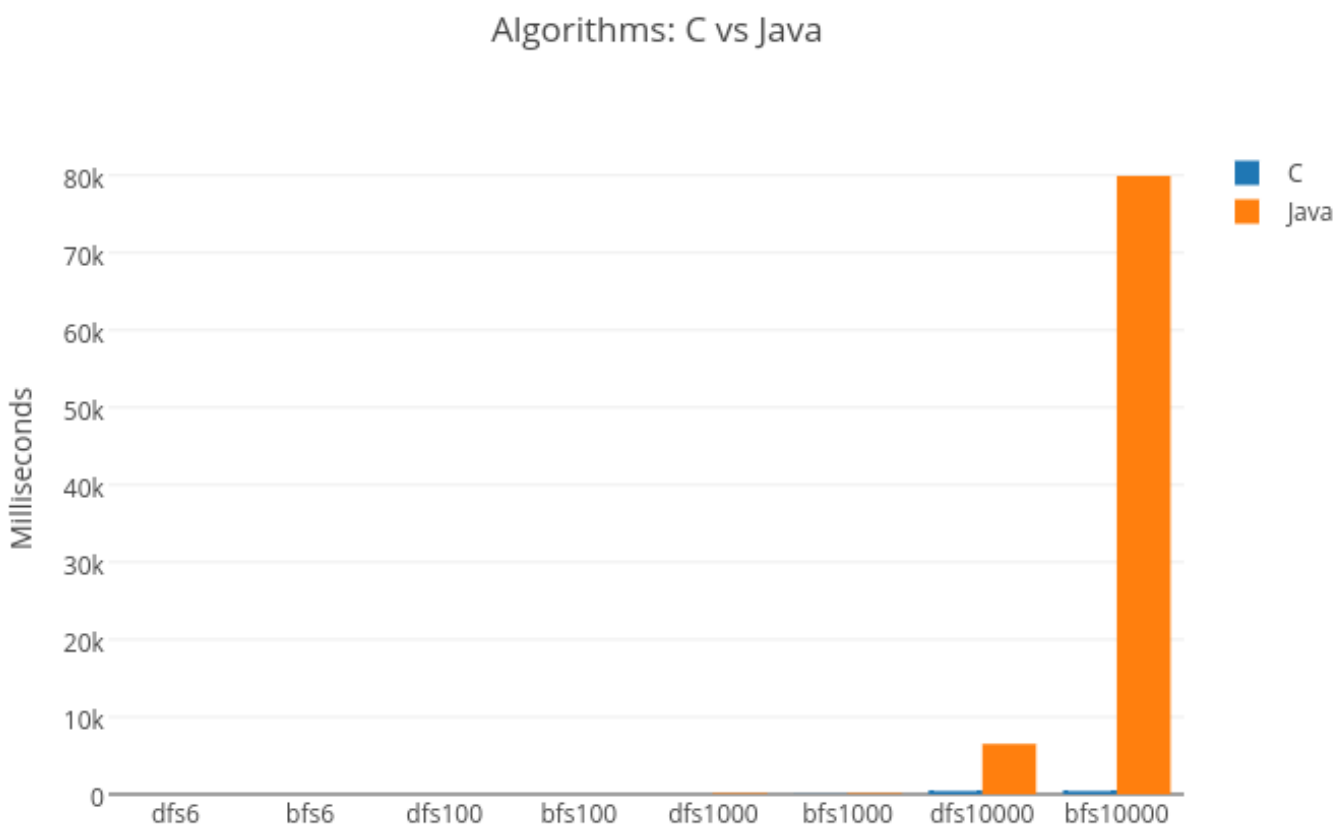
Time Performance

The following chart demonstrates time consumed for reading input files by C and Java implementations for different number of nodes. We can see that C outperforms Java in read performance for sufficiently large input.



Though C is faster, its code is not always portable. E.g. large read performance is reached by using `gnu getline` function in C which is fast but copes only with Unix-style line endings `\n` and doesn't treat `\r` of old Macs as line breaks. That't why input files for C implementation first have to be processed with `mac2unix` and `dos2unix` utilities called by `prepareInputs.sh` bash script in the source codes. My attempts of implementing `crossplatformGetline` led to dramatically worse performance and may be found in `Tool.h`

The following chart demonstrated time consumed for algorithm execution by C and Java implementations. C outperforms Java again.



Memory Consumption

Memory footprint of C implementation is significantly fewer. Java implementation runs into OutOfMemory exception if launched with default flags (default `MaxHeapSize` is 768mb on the experimental machine) on 10000 nodes of input. Higher level abstractions of Java and virtual machine require more space. Compare C and Java code structures used for storing edges. C, thin low level structures:

```
typedef struct ListNode ListNode;
struct ListNode {
    int number;
    ListNode *next;
};
typedef struct List List;
struct List {
    ListNode *first;  // To start traversal.
    ListNode *last;   // To add new nodes.
};
```

Java, high-level classes :

```
// Modified for report, exerts ordered set.
this.edges = new HashMap<Integer, LinkedHashSet<Integer>>(n);
```

Readability and Orthogonality

Readability is defined by how easy your code to read and understand by other

experienced programmers. It is relevant to orthogonality which is defined by how many programming concepts and their combinations you have to use to reach your goal. As a rule of thumb the more freedom the language gives you the less readable it is. Both C and Java are quite readable in comparison to, e.g., `C++` or `Perl`. But C gives the programmer more freedom than Java, especially, on the low level. More than that as Java offers rich Standard Class Library out of the box and C doesn't offer something as portable, in C you have to write low level code or be dependent on the third party libraries. In this aspect Java is more readable as it works with high level abstractions. It may be stated that in Java you have to be familiar with more higher-level concepts than in C but in C instead you have to be familiar with large number of combinations of low level concepts that's why Java doesn't loose in orthogonality. Both languages don't impose any naming convention but Java prompts the convention used throughout its Standard Library. More than that Java strives to be clear and verbose but C is terse and that's why less readable. C offers wider aliasing techniques with preprocessor macros which give inexperienced programmers the freedom to obfuscate code.

To sum it up, Java outperforms C in readability:

- C gives low-level freedom and forces low-level programming.
- C doesn't offer Standard Class Library alternative out of the box which means you have to invite your own bicycles.
- Java is verbose, clear and prompts naming convention and C is terse and intricate.
- Java outperforms in orthogonality.
- C gives the freedom of aliasing with preprocessor macros which may be abused.

Note

There are two contradicting approaches in language design:

1. "There should be one -- and preferably only one -- obvious way to do it." (Python principle)
2. "There's More Than One Way To Do It" (Perl principle)

Both Java and C are idiomatic, i.e., they both has idioms for achieving practical goals. This makes them more readable. But I find Java to be more inclined to the first principle and thus more readable.

Writability and Expressivity

It is more difficult to write code in C as it uses contractions which are not always easy to remember. C is less expressive, e.g., not all standards allow type declarations like `for(int i=0;....)`, there are no foreachs like Java's `for(int i : someList)`. With Java abstractions under your fingertips it is easier to write concise code and not to reinvent bicycles in C.

Reliability

Reliability is defined by how robust your program to failures in input, runtime exceptions, programming errors, etc. Java offers exception control concepts such as `try-catch-finally` blocks and `throws` declarations in method signatures. It makes Java code more robust to failures. Moreover Java offers Garbage Collection so you

are less prone to memory programming errors in Java than in C. The penalty of this is additional cost of execution. As C lacks exception control while porting Java graph traversal implementation to C exceptions handling mechanisms were substituted with custom `failWtih`function. C intricate programming interfaces are harder to remember which leads to more programming errors. Both languages are strict-typed which makes them more robust.

Learnability

Learnability is defined by how easy it is for a newcomer to pick up the language. With Java you have to learn OOP concepts (Class, Inheritance, Encapsulation, Abstraction, Polymorphism, etc.), exception control and idiomaitc use cases of Standard Class Library. With C you have to learn low level manipulation (pointer, address, casting), memory management (`malloc`, `calloc`, `free`), preprocessor and types, use cases of intricate standard libraries functions. From my point of view it is more easier to learn Java than C but both are worth learning and both will pay off in the future.

Portability

Portability is defined by how portable your code to other platforms. Java is devised with portability in mind: it may boast virtual machine and portable Standard Class Library. Though C may be compiled for virtual machine too, its standard libraries can't offer the same multitude of crossplatform solutions as Java, e.g., reading file line by line is not present in C standard libraries and third library solutions like `gnu getline` may be not portable. In the task of graph traversal you have to write your own portable `getline` (which may be slow), or rely on third parties. In other words, it takes pains to write portable code in C.

Simplicity

Code in Java is more simple than in C. Let's consider one excerpt from the source codes.

```
// concat_malloc("One", "Two", NULL);
char* concat_malloc(char *first, ...) {
    size_t messageLen = strlen(first);
    char *message = (char*) malloc(messageLen + 1);
    strncpy(message, first, messageLen + 1);

    va_list argp;
    va_start(argp, first);
    char *tail = concatValist_malloc(argp);
        int tailLen = strlen(tail);
        messageLen += tailLen;
        message = (char*) realloc(message, messageLen + 1); // Length + /0
        strncat(message, tail, tailLen + 1);
    free(tail);
    va_end(argp);
    return message;
}
```

This is a function in C to concatenate variable number of arguments into one string. In C you have to know number of arguments beforehand or you have to use some sentinel like `NULL` in this case. This function calls to another variable

number of arguments function `concatValist_malloc` which operates on `va_list` type instead. We have two functions: one for calling as ordinary variable number arguments function, another for calling from within another variable number of arguments function (passing `va_list`), e.g., from this function:

```
void failWith(char *first, ...) {
    va_list argp;
    va_start(argp, first);
    char *tail = concatValist_malloc(argp);
    va_end(argp);
    char *message = concat_malloc(first, tail, NULL);
    free(tail);
    printf("Error: %s", message);
    free(message);
    exit(EXIT_FAILURE);
}
```

Firstly, variable number of arguments mechanism is more simpler for Java, where you don't have to know number of arguments, use NULL sentinel or write multiple variants of multiargument functions:

```
void foo(String... args) {
    for (String arg : args) {
        System.out.println(arg);
    }
}
```

Secondly, you don't have to implement such bicycles in Java as they are in Standard Class Library already. Thirdly, you don't have to bother with memory arrangement in Java. The trade off of this is performance impact of Java Garbage Collector which may be difficult to predict. Obviously, Java code is simpler than C.

Generality and Reusability

Generality of code is defined by whether it may be applied to vast majority of use cases, e.g., different types of inputs. General functions are those that are not dependent on the context or that take any number of arguments. It is related to reusability which is defined whether the code may be reused in different scenarios. It takes more efforts to write crossplatform reusable code in C as it is less portable. Further, OOP paradigm promotes reusability of code to further extent than structural programming of C. Moreover Java offers generics and method overloading and C respond is preprocessor macros. It's worth noting that Java parameter passing is better than in C.

Paradigms Support

Java supports OOP, imperative, structured, functional, procedural (static methods), generic, reflective, concurrent paradigms. C supports imperative, structured, procedural, generic (preprocessor macros), concurrent (`pthread.h`) paradigms. Java support of functional programming became better recently. Both languages support closures. The following Java code exerts lambdas:

```
this._ifTargetFunction =
        nodei -> {
            ExpResult result = ast.evaluate(
                varName -> lookupVariable(nodei, varName)
```

```
            );
            return result.ifTrue();
        };
return;
```

In C you may pass pointer to a function and thus emulate some paradigms:

```
bool defaultIfTargetFunction(Graph *graph, int noden) {
    return graph->target == noden;
}
graph->_ifTargetFunction = &defaultIfTargetFunction;
return graph;
```

Some OOP techniques may be emulated in C, e.g., methods may be defined as function which takes `this` object as the first argument:

```
void ListAddListNode(List *list, int noden) {
    ListNode *last = list->last;
    last->next = list->last = newListNode_malloc(noden);
}
```

To sum it up, Java boasts better paradigm support than C, yet C may emulate some paradigms.

Ease of Code Maintenance

As Java is simpler and more readable its code is easier to maintain. It is easier to write unmaintainable code in C. Advanced code editors like Eclipse offer code refactor techniques for both languages, though C is more reluctant in this way.

Cost

Both languages are free, C tool set is more liberal. There are plenty of free of cost profilers for GCC C under Unix, and fewer for Java. Fewer maintainability of C requires additional costs. If you are interested in language specifications -- official C specs are paid for and drafts are free, Java specs are free.

Abstraction

Abstraction is the main principle of OPP and Java adheres to it vividly. Java works at a higher level of abstraction by the penalty of performance. C lacks Garbage collection and forces programmer to work on the lower level.

Well Definedness

> Well-definedness – The completeness and precision of the language‘s official definition.

Both languages are well defined (e.g., both has specifications, both defined by formal grammar). Both offer verbose documentation though I personally find Java more well documented than C. In C documentation I sometimes stumble upon phrases like "if argument is NULL the behavior is not defined." which are absent in Java documentation (Java simply throws exceptions in such cases). This makes Java more predictable and well-defined.

Dependability

C better meets goals of independent projects, e.g., military projects. With C you depend on a compiler which can be bootstrapped and on C standards committee. With Java it is legal to devise your own virtual machine (e.g., `Dalvik`) though you have to keep with some restrictions (see Oracle vs. Google#Licensing_and_patents)).

Other Characteristics

Types & Type Checking

Both Java and C are strong-typed languages. Java offers generics out of the box.

Parameter Passing

Both languages lack default function arguments, but Java offers method overloading. C lacks function overloading and has worse support for variable number of arguments.

Concurrency & Parallel Programming

Java has support for concurrent programming, in C you have to rely on third party libraries.

Memory Management

Java has Garbage Collection at the penalty of execution time checks.

Consistency

C gives you more programming freedom and doesn't prompt naming convention as Java.

Standardization

Java platform standardization is driven by Oracle, C -- by ISO. Both are standardized and documented.

Web Applications & Web Services Development

Though it is possible to program for web in C, it is not a common case. Java fits better for this objective.

Compilation/Interpretation

C is compiled and executed. Java is compiled to bytecode, bytecode is then compiled just-in-time.

Using in Startups and Prototyping

Java is easier to prototype in and thus to write startups.

Conclusion

Java outperforms C in overwhelming majority of aspects. But if you have to write high-performance code, e.g., system driver, or independent project then Java is no choice in favor of C. The followig table summarizes comparison.

+ means "wins" - means "looses" | Characterisitc | Java | C
| ---------- |:-----:|:-----: | Time performance | - | + | Memory Consumption| - | + | Readability | + | - | Writability | + | - | Reliability | + | - | Learnability | + | - | Portability | + | - | Simplicity | + | - | Reusability | + | - | Paradigms Support | + | - | Maintainability | + | - | Cost | + | + | Abstraction | + | - | Well Definedness | + | + | Dependability | - | +

Some materials used.

Principles of Programming Language, G. NARAYANAMMA INSTITUTE OF TECHNOLOGY & SCIENCE.