COMPUTER SCIENCE TRIPOS Part IB

Monday 2 June 2014 1.30 to 4.30 pm

COMPUTER SCIENCE Paper 3

*Answer **five** questions.*

*Submit the answers in five **separate** bundles, each with its own cover sheet. On each cover sheet, write the numbers of **all** attempted questions, and circle the number of the question attached.*

You may not start to read the questions printed on the subsequent pages of this question paper until instructed that you may do so by the Invigilator

STATIONERY REQUIREMENTS
Script paper
Blue cover sheets
Tags
Rough work pad

SPECIAL REQUIREMENTS
Approved calculator permitted

1 Algorithms II

(*a*) In the context of multithreaded algorithms, define *work* and *span*, and state the work law and the span law. [3 marks]

(*b*) Prove that the performance of a greedy scheduler is optimal to within a factor of 2. (Proving all intermediate theorems is not required if you state them correctly.) [4 marks]

(*c*) Version A of a multithreaded algorithm takes 500 seconds on a uniprocessor machine and 50 seconds on a 32-processor machine. Version B takes the same time as A on a single processor but only 24 seconds on the 32-processor machine.

(*i*) Define the *parallelism* of a computation and compute the parallelism of algorithms A and B. Which of the two has higher parallelism, and by how much? (*Hint:* use one of the greedy scheduler theorems to derive an approximation for one of the unknowns.) [6 marks]

(*ii*) Estimate the running times of algorithms A and B on a 4-processor and on a 1024-processor machine, explaining how you obtain them. [3 marks]

(*iii*) Sketch possible computation DAGs for algorithms A and B and use them to discuss the results obtained. As the number of processors in the host machine varies, is A or B faster? [4 marks]

2 Algorithms II

(*a*) Consider van Emde Boas (vEB) trees.

(*i*) On its own page for legibility, draw the smallest vEB tree storing keys 0, 3, 6, 7. The correctness of the structure and the accuracy of all fields of all nodes are important. Once done, write each of the keys under the cluster in which it is logically stored. [8 marks]

(*ii*) vEB trees store the minimum and maximum key of a subtree in the root node, but do not store the minimum key in any of the descendent clusters. Explain all the reasons why this provides a performance advantage compared with proto-vEB trees. [4 marks]

(*b*) Consider proto-vEB trees.

The following pseudocode attempts to implement a method to delete a key from a proto-vEB node. Give a clear explanation of the strategy that it uses. Fix any bugs that it may contain. Give more meaningful identifiers for the variables `c` and `s`. Give appropriate comments for the four positions marked "`COMMENT HERE`". Explain why the method returns a value and what the value means. [8 marks]

```
 0   boolean delete(self, key)
 1       # HEADER COMMENT HERE (1)
 2       if self.u == 2:
 3           if self.A[key] == 0
 4               # COMMENT HERE (2)
 5               return False
 6       else:
 7           c = self.cluster[high(key)].delete(low(key))
 8           if c:
 9               # COMMENT HERE (3)
10               s = self.summary.delete(high(key))
11               # COMMENT HERE (4)
12               return s
13           else:
14               return False
```

3 Programming in C and C++

(*a*) Write a C function `revbits()` which takes a single 8-bit `char` parameter and returns a `char` result by reversing the order of the bits in the `char`. [4 marks]

(*b*) Write a C function `revbytes()` taking two parameters and returning no result. The first parameter is a pointer to memory containing n contiguous bytes (each of type `char`), and the second is the number of bytes. The function should have the side effect of reversing the order of the bits in the n contiguous bytes, seen as a bitstring of length $8n$. For example, the first bit of the first `char` should be swapped with last bit of the last `char`. [6 marks]

(*c*) You have been assigned the following seemingly working C code, which processes files controlling the behaviour of a system. You observe that, after obtaining several `ERR_MALFORMED` errors, subsequent calls to `fopen` fail due to too many files being open:

```
int process_file(char *name)
{   FILE *p = fopen(name, "r");
    if (p == NULL) return ERR_NOTFOUND;
    while (...)
    {   ...
        if (...) return ERR_MALFORMED;
        process_one_option();
        ...
    }
    fclose(p);
    return SUCCESS;
}
```

(*i*) Explain how to fix the program using facilities in C. [2 marks]

(*ii*) Now suppose the function above was part of a system written in C++ (but still using the C file-processing commands such as `fopen` and `fclose`), and that `process_one_option()` might raise one or more exceptions. Using a class with a destructor, show how to fix the "too many files open" bug above. [8 marks]

4 Compiler Construction

This question concerns the run-time call stack.

(*a*) What is a *run-time stack* and why is it important to a compiler writer? [3 marks]

(*b*) The implementation of a run-time call stack typically uses a *stack pointer* and a *frame pointer*. What are their roles and why do we need two pointers? [3 marks]

(*c*) For some compilers the activation records (stack frames) contain *static links*. What problem are static links used to solve and how do they solve this problem? [3 marks]

(*d*) (*i*) Consider a programming language that does not allow functions to be returned as results, but does allow the nesting of function declarations. Using ML-like syntax, we have the following code in this language.

```
let fun f(x) =
    let
        fun h(k) = k * x

        fun g(z) = h(x + z + 1)
    in
       g(x + 1)
    end
in
    f(17)
end
```

Draw a diagram illustrating the call stack from the call of `f` up to and including the call of function `h`. Make sure all function arguments are included in the diagram and clearly indicate static links. [5 marks]

(*ii*) Using your diagram, explain how the code generated from the body of function `h` can access the values associated with the variables `k` and `x`. In each case make it clear what information is known at compile-time and what information is computed at run-time. [6 marks]

5 Compiler Construction

Functional programmers will often rewrite a recursive function such as

```
fun fact1 n =
    if n <= 1
    then 1
    else n * (fact1 (n -1))
```

to one such as

```
fun fact2 n =
let fun aux (m, a) =
        if m <= 1
        then a
        else aux(m-1, m * a)
in aux (n, 1) end
```

using an accumulator (the parameter `a` of `aux`) and *tail recursion.*

(*a*) Clearly explain the optimisation such programmers are expecting from the compiler and how that optimisation might improve performance. [4 marks]

(*b*) The desired optimisation can be performed by a compiler either directly on the source program or on lower-level intermediate representations. Treating it as a source-to-source transformation, rewrite `fact2` to ML code that has been transformed by this optimisation. You will probably use references and assignments as well as the construct `while EXP do EXP`. [8 marks]

(*c*) Suppose that the programmer used instead a function as an accumulator.

```
fun fact3 n =
let fun aux (m, h) =
        if m <= 1
        then h(1)
        else aux(m-1, fn r => m * (h r))
in aux (n, fn x => x) end
```

Will your optimisation still work in this case? Explain your answer in detail. [8 marks]

6 Concepts in Programming Languages

(*a*) Write a LISP program for detecting whether a LISP interpreter treats the language as being dynamically scoped (as was the case in historical LISP) or as being statically scoped (as is the case in modern LISP). You may use pseudo-code and should explain your answer in detail. [4 marks]

(*b*) You manage two junior programmers and overhear the following conversation:

> A: "I don't know why anyone needs a language other than Java, it provides clean thread-based parallel programming."
> B: "Maybe, but I write my parallel programs in a functional programming language because they are then embarrassingly parallel."

Discuss the correctness of these statements and the extent to which they cover the range of languages for parallel programming. [6 marks]

(*c*) Explain why the SML interpreter accepts the declarations

```
datatype 'a FBtree = node of 'a * 'a FBtree list;

fun dfs P (t: 'a FBtree)
= let  exception Ok of 'a;
       fun auxdfs( node(n,F) ) = if P n then raise Ok n
           else foldl (fn(t,_) => auxdfs t) NONE F;
  in  auxdfs t handle Ok n => SOME n  end;
```

while it does not accept the declaration

```
exception Ok of 'a;
```

[4 marks]

(*d*) Consider the declarations

```
structure Z = struct type t = int; val z = 0 end;

structure A = Z  :   sig type t       ; val z: t end;
structure B = Z  :>  sig type t = int ; val z: t end;
structure C = Z  :>  sig type t       ; val z: t end;
```

in the SML Modules language.

Explain the behaviour of the SML interpreter on inputting each of the expressions

```
Z.z = A.z;   Z.z = B.z;   Z.z = C.z;
```

[6 marks]

7 Further Java

Five housemates run a "status" server on their home network. The server stores the current status of each housemate as a string of text. For example, housemate Eva might set her status to "Gone to the exam hall."

Messages are passed between clients and the server as text strings sent over TCP. The new line character is used exclusively as the last character in every message. On connection with the server, a client can either (i) query the status of a user by sending the user's name to the server as a string (and the server responds with the current status message), or (ii) set the status of a user by sending the user's name followed by a colon and the new status message. For example, "Eva:Gone to the exam hall." sets the status message for Eva.

(*a*) Implement a status server in Java. The server should run indefinitely, responding to client requests. Once a client request has been fulfilled, the server should close the connection. You may assume current status messages are lost if the server is restarted and you do not need to handle exceptions. [8 marks]

(*b*) One housemate suggests the server and client should communicate by serialising Java objects rather than sending messages as text.

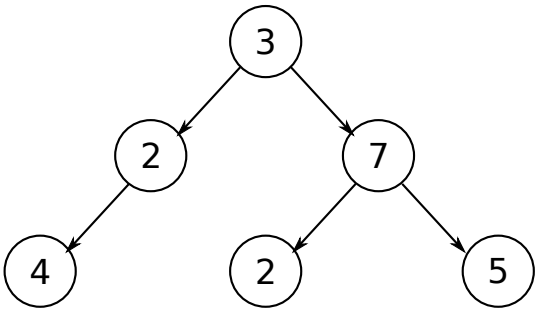
(*i*) Describe in words the changes you would make to your server implementation to send messages as serialised Java objects. [3 marks]

(*ii*) List two advantages and two disadvantages of an implementation based on serialised Java objects versus sending messages as text. [4 marks]

(*c*) Another housemate suggests that the server should not close the client's connection after answering the request. Instead the connection should remain open until the client sends another request or closes the connection. Describe in words what changes you would need to make to your implementation in part (*a*) to achieve this and comment on the advantages and disadvantages of this idea. [5 marks]

8 Prolog

You are asked to write a Prolog program to work with binary trees. Your code should not rely on *any* library predicates and you should assume that the interpreter is running without occurs checking.



(*a*) Describe a data representation scheme for such trees in Prolog and demonstrate it by encoding the tree shown above. [3 marks]

(*b*) Implement a Prolog predicate `bfs/2` which effects a *breadth-first* traversal of a tree passed as the first argument and unifies the resulting list with its second argument. For example, when given the tree shown above as the first argument the predicate should unify the second argument with the list `[3,2,7,4,2,5]`. [4 marks]

(*c*) Explain why the `bfs/2` predicate might benefit from being converted to use difference lists. [2 marks]

(*d*) Implement a new predicate `diffbfs/2` which makes use of a difference list to exploit the benefit you identified in part (c). Your predicate should take the same arguments as `bfs/2`. [6 marks]

(*e*) A friend observes that a clause in `diffbfs/2` will need to contain an empty difference list and proposes two possible ways of representing it, either `[]-[]` or `A-A`.

Consider your implementation of `diffbfs/2`. For each use of an empty difference list, justify your choice and explain what can go wrong using the alternative form. [2 marks]

(*f*) Is your implementation amenable to *last call optimisation* (LCO)? If so, explain why. If not, give details of the minimal changes you would make to make LCO possible. [3 marks]

9 Software Engineering

(*a*) Describe the main lessons learned from the report into the collapse of the London Ambulance Service. [12 marks]

(*b*) To what extent have the developments in software tools and management practices of the last twenty years improved the situation, and which of the lessons do we still have to be careful of today? [8 marks]

END OF PAPER