1 Graphs, Circuits and Positional Number Systems

0pt

Problem 1.1 (Designing a Logic Circuit)
Given the following truth table:

x	y	z	OutputA	OutputB
0	0	0	1	1
0	0	1	0	0
0	1	0	0	1
0	1	1	1	0
1	0	0	1	1
1	0	1	0	0
1	1	0	0	1
1	1	1	1	1

1. Obtain and simplify (using any method) logic expressions for OutputA and OutputB.
2. Implement the corresponding logic circuit using only AND, OR, NOT gates, and using the least number of these.
3. Implement the same circuit (same logic expression) using the least number of NAND gates.

Note: For the last part, you do not have to prove that the number of gates you used is minimal. However the number of gates should be smaller than in the circuit obtained just by directly expressing each OR/AND/NOT gate with NAND gates.

Solution:

2 Machine Languages

0pt

Problem 2.1 (ASM reverse)
Write an ASM program which reverses a substring of a string of integer. D(0) will contain the index b of the element at the begining of the substring and D(1) will contain the index e of the last element of the substring. D(3) through D(∞) will be the elements of the string, which are encoded as integers. The indexing starts with D(3) having index 0. It is guarantied that b and e are non-negative. Note that D(2) was left intentionaly for you to use as temporary storage in your algorithm, should you choose to do so. After the program finishes, the values in D(0), D(1) and D(2) can be anything you want. If b $\geq$ 1. the values from D(3) to D(b+2) should be the same as before the program was run. Also, the values of D(e+4) through D(∞) should also be the same as before. If s < e, the values of D(b+3) through D(e+3) should appear in reverse order.

Note that you are only allowed to use labels in jump statements. You are very much encouraged to write comments and even a description of your approach, which cand help the grader award partial points. Here is a list of ASM commands (only for reference).

load i	jump(=) i
store i	jump(>) i
loadi i	jump(<) i
addi i	jump($\geq$) i
subi i	jump($\leq$) i
add i	sub i
move S T	jump($\neq$) i
loadin i j	storein i j
jump i	stop 0

Hint: You might want to make use of the "storein" and "loadin" instructions which deal with relative addressing. After all, at the begining D(s+2) == D(D(0) + 2).

There might just be a "hidded" induction somewhere which could make life easy for you.

Hint:

Solution:

The recursion is simple: base case: D(0) ¿= D(1) =¿ nothing to do. step case: swap D(D(0)+3) and D(D(1)+3), increment D(0) and decrement D(1). This is an induction by the difference n = D(1)-D(0), where negative cases are treated as the base case.

The swapping uses the memory cell D(3) as temporary storage: D(2) = D(D(0)+3); D(D(0)+3) = D(D(1)+3); D(D(1)+3) = D(2);

The relative referencing is done using registers IN1 and IN2: IN1 = D(0) IN2 = D(1) D(2) = D(IN1+3); D(IN1+3) = D(IN2+3); D(IN2+3) = D(2);

Here is the code with comments

```
load 1
sub 0
# check if we are at the base case
```

```
jump(<=) 18
# prepare IN1 and IN2 for relative referencing
load 0
move ACC IN1
load 1
move ACC IN2
# D(2) = D(IN1+3)
loadin 1 3
store 2
# D(IN1+3) = D(IN2+3)
loadin 2 3
storein 1 3
# D(IN2+3) = D(2)
load 2
storein 2 3
# increment D(0)
load 0
addi 1
store 0
# decrement D(1)
load 1
subi 1
store 1
# continue the recursion
jump -19
stop 0
```

Problem 2.2 (Factorial and $\mathcal{L}$(VMP)) 0pt

Write a VM procedure which takes an argument n and computes $n!$. Also, draw the VM stack at each computation step done by the virtual machine.

Solution:

Problem 2.3 (Reverse and $\mathcal{L}$(VMP)) 0pt

Suppose you enhance the VM language with two operations similar to `peek` and `poke` by allowing indirect addressing, i.e. one can now write `peeki 3`, which would push on the stack the content of $S(S(3))$ and `pokei 3`, which would move the top of the stack into $S(3)$.

Now, you are given a positive integer n in $S(0)$ and n digits in the cells $S(2) \dots S(n+1)$, which represent a number in decimal representation. Your task is to find out if this number is a palindrome or not, and write the answer in $S(1)$. You may use the cells below $S(0)$ to do your computations.

A slightly more tedious (but not more difficult) excercise is to reverse the content of the VM stack, since you also have to compute $n/2$. Try it out if you want.

Solution:

```
        con 1 poke 1                    // assume it is palindrome
        peek 0 poke -1                  // S(-1) will be the iterator i from n --> 0
<loop>  peek -1 cjp <halt>              // we reached i = 0
        peek -1 con 1 add poke -2       // S(-2) = i+1
        peek -1 con 2 peek 0 add sub  poke -3 // S(-3) = n+2-i
        con 1 peek -1 sub poke -1       // i--
        peeki -2 peeki -3 sub cjp <loop>       // s[i+1] =? s[n+2-i]
        con 0 poke 1 halt               // stop if not
<halt>  halt
```

Problem 2.4 (Largest Binary Number) 0pt

Design a Turing Machine that finds the largest binary number in a list of unsigned positive binary numbers/ strings, and appends it with a # to the right of the input.
All numbers/ strings are of the same length and are separated by #'s.

Sample input:	#1001#1100#0110#0011
Sample output:	#Whatever − Here#1100

Assume the head is positioned at the first #. You may use additional symbols in your alphabet.
Describe the idea behind your algorithm and provide a transition (action) table.

Solution: Basically you compare every two consecutive numbers bit-by-bit, and if the left one is bigger, it overwrites the right one.
Here is a decision table on what to do at the bit-by-bit comparison:

l bit	r bit	action
0	1	skip left string, move to right
1	0	overwrite right string with left string
0	0	check next bit
1	1	check next bit

---

3 Internet

0pt

Problem 3.1 (Sending emails) 10min

The GenCS TAs were very satisfied by the interest students had for their tutorials and wanted to write a thank you email to everyone. Since the *Information and Software Architecture of the Internet and WWW* topic was not that detailed when they had the class, they are very interested to learn more from you.

1. (briefly) describe the TCP/IP layers that their email passes as it is sent and what transformations happen at each stage.
2. what is HTML and CSS and briefly explain how they can be used to write a *fancy* email (It is sufficient to just write an example email using them both).
3. describe how an asymmetric-key encription method would enable sending confidential messages.

Solution:

1. The layers of the TCP/IP suite, with descriptions, are:

Layer	Description
Application Layer	high level data
Transport Layer	UDP,TCP header added to the data as it gets encapsulated in a packet
Internet Layer	IP header and data
Link Layer	packets are divided to frames

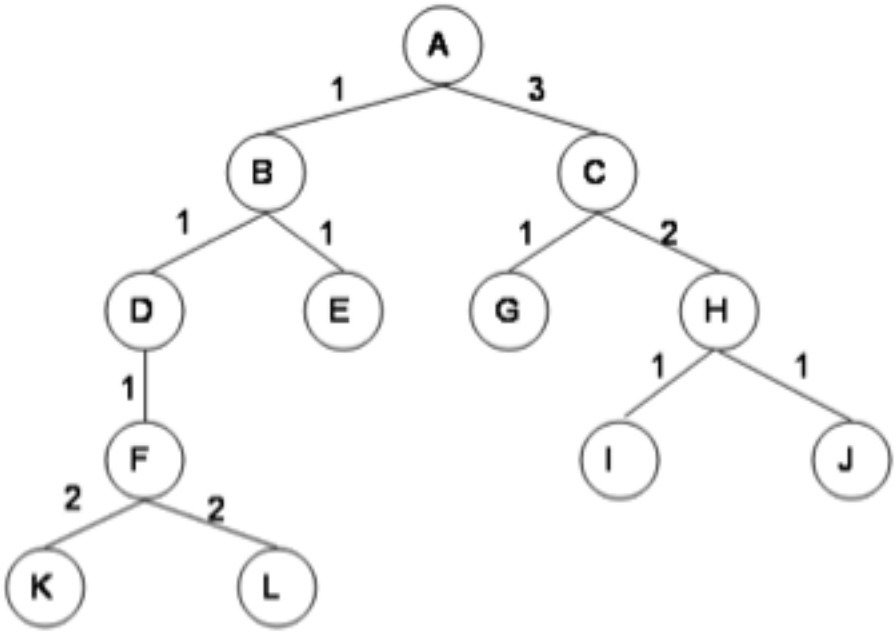
2. HTML is a representation format for web pages and CSS is a style sheet language that allows authors and users to attach style (e.g., fonts and spacing) to structured documents. A basic example with HTML tags and CSS style is sufficient to show the use of them.
3. In an asymmetric-key encryption method, the key needed to encrypt a message is different from the key for decryption. Such a method is called a public-key encryption if the encryption key (called the public key) is very diffcult to reconstruct from the decryption key (the private key). To send a confidential message the sender encrypts it using the intended recipient's public key; to decrypt the message, the recipient uses the private key.

---

4 Problem Solving and Search

0pt

Problem 4.1 (Search algorithm)
Apply BFS, DFS, IDS, UCS, Greedy, and Astar to the following graph:



For the last two search strategies, experiment with some heuristic functions of choice. Also, review the properties of each search and the properties of heuristics.

Solution:

BFS A, B, C, D, E, G, H, F, I, J, K, L

DFS A, B, D, F, K, L, E, C, G, H, I, J

IDS A, A, B, C, A, B, D, E, C, G, H, A, B, D, F, E, C, G, H, I, J, A, B, D, F, K, L, E, C, G, H, I, J

UCS A, B, D, E, C, F, G, K, L, H, I, J

Greedy depends on the heuristic

Astar also depends on the heuristic (to be explained in tutorial)

Properties of searches:

Name	Complete	Time	Space	Optimal
BFS	Yes	$O(b^{d+1})$	$O(b^{d+1})$	Yes/no
UCS	Yes	nr explored nodes	nr explored nodes	Yes
DFS	Yes/no	$O(b^m)$	$O(b \cdot m)$	No
IDS	Yes	$O(b^{d+1})$	$O(b \cdot d)$	Yes
Greedy	No (loops)	$O(b^m)$	$O(b^m)$	No
Astar	Yes	exponential in error of h $\times$ length of sol	same as BFS	yes

Properties of heuristics:

1. admissibility: $0 \leq h(n) \leq h^*(n)$ (optimal Astar)
2. dominance: h_2 domainates h_1 if $h_2(n) \geq h_1(n)$ for any node (better Astar)

---

5 Programming in Prolog

0pt

Problem 5.1 (The Zip Function)

Remember the SML Grand Tutorial from last semester? You will now remember the good old times by implementing one of those functions in Prolog: the zip function.

The zip function takes two lists with lengths that differ at most by 1, and outputs a list of lists containg one element from the first list and the element with the same index from the other list. Create a Prolog predicate with 3 arguments: the first two would be the two lists you want to zip, and the third one would be the result.

For instance:

```
?- zip([1,2,3],[4,5,6],L).
L = [[1, 4], [2, 5], [3, 6]].

?- zip([1,2],[3,4,5],L).
L = [[1, 3], [2, 4], [5]].
```

Feel free to implement any helper functions.

Solution:

```
zip([L],[],[[L]]).
zip([],[L],[[L]]).
zip([A],[B],[[A,B]]).
zip([H1|T1],[H2|T2],L) :- zip(T1,T2,T), append([[H1,H2]],T,L).
```
