CSCI3390-Assignment 1 Solutions

1 Encoding problem instances as strings.

The scheme used for encoding graphs is illustrated in Figure 1. I have deliberately chosen a flawed encoding scheme so that you can correct it in the last part of this problem, but for all other parts of the problem, you should stick with this method.

A *Hamiltonian path* in a graph is a path that includes every vertex of the graph exactly once. A *Hamiltonian circuit* is a path that starts and ends at the same vertex and visits every other vertex exactly once.

(a) The pictured graph has a Hamiltonian circuit. Find it.

Solution. 1-2-5-6-4-3-1. This is not the only solution! Since it's a circuit, you can choose any vertex as the start and ending vertex, or traverse counter-clockwise instead of clockwise, etc.

(b) Eliminate one edge of the pictured graph so that the resulting graph has a Hamiltonian path but not a Hamiltonian circuit. (Tell which edge you are elminating, and what the path is.)

Solution. Any edge (except the edge between 4 and 5, whose deletion preserves the Hamiltonian circuit property) will work. For example, we can eliminate the edge from 5 to 6 and get the Hamiltonian path 6-4-3-1-2-5. There is no Hamiltonian circuit in the resulting graph, however, because no path can both enter and leave vertex 6 without hitting vertex 4 twice.

(c) Let $HAMPATH$ be the language representing the decision problem 'Does this graph have a Hamiltonian path?' relative to this encoding. Let $HAMCIRC$ be the language representing the problem 'Does this graph have a Hamiltonian cricut?' Let w be the following string.

$$1\#2\#1\#3\#1\#4\#2\#4.$$

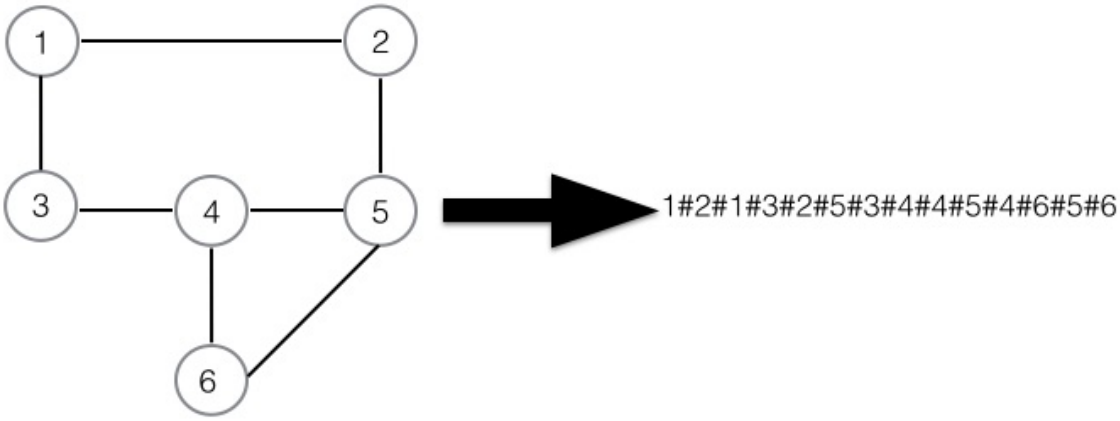

Figure 1: The graph encoding scheme for Problem 1.

Is $w \in$ HAMPATH? Is $w \in$ HAMCIRCUIT? (Don't just tell me yes or no; explain your answer for all parts.) It helps to draw a picture.

Solution. The graph is pictured below. The translation of the problem is 'does this graph have a Hamiltonian path?' a Hamiltonian circuit?' The answer is 'yes' and 'no'. 3-1-2-4 is a Hamiltonian path, but there is no Hamiltonian circuit because you cannot both enter and leave vertex 3 without hitting vertex 1 twice.

(d) We can encode anything over the two-letter alphabet $\{0, 1\}$. Describe how to adapt the encoding scheme for graphs so that it uses only these two symbols. (A simple answer is: 'encode the vertex numbers in binary, rather than decimal' but that doesn't tell us how to handle the separator symbol # ! There are, however, several different correct answers.)

Solution. Here are three solutions. (1) We can encode each vertex number in binary, but let's use *two* bits (00) for a binary 0, and two bits (01) for a binary 1. Then we can encode the separator $\#$ as 11. We then encode the graph $1\#2\#2\#3$ as

$$01 \quad 11 \quad 0100 \quad 11 \quad 0100 \quad 11 \quad 0101.$$

I put spaces in just to show you where the separate fields are, but the point is, you don't need the spaces. You parse the string by reading successive pairs of

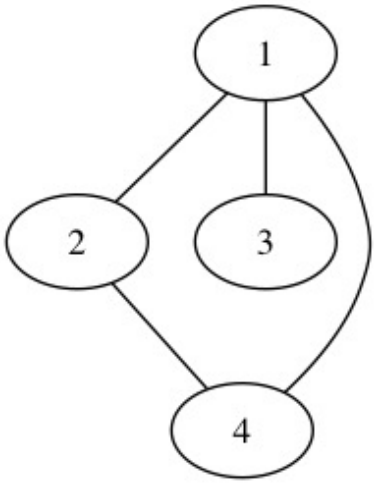

Figure 2: The graph of Problem 1(c).

bits. The pair 11 can only mean a separator symbol. Observe that this doesn't work if we try to encode # as 11 but represent the integers in binary in the ordinary way. If we tried to do things this way, what does

$$11111111111110$$

represent? Is it

$$1 \quad 11 \quad 11 \quad 11 \quad 11 \quad 11 \quad 110,$$

that is, $1\#3\#3\#6$, or is it

$$111111 \quad 11 \quad 111110,$$

which is $63\#62$?

(2) That was the first solution I thought of, but several students in office hours came up with a better one: Let's encode the separator # by 0 and each vertex number n by n 1's. Thus $1\#2\#2\#3$ would be encoded as

$$10110110111.$$

(3) There is still another method commonly used in Computer Science, and that is to represent the graph by an $n \times n$ matrix, where n is the number of

vertices. The (i, j)-entry of the matrix is 1 if there is an edge between i and j and 0 otherwise. Thus $1\#2\#2\#3$ is encoded by the matrix

$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}.$$

To get a string out of this, we have to 'flatten' it and just write

$$010101010.$$

Doing so removes the information about where the rows end, but we can still recover this (with difficulty): Since the matrix must be square, there is only one possible way to arrange the string into rows and have the same number of rows and columns.

(e) Let V and E represent, respectively, the sets of vertices and edges of a graph, and $|V|$ and $|E|$ the number of elements in these sets. Write an expression using these numbers that gives the length of the encoding of the graph, or at least a reasonable upper bound for the length of the encoding. (This problem is a little subtle: It is tempting to write $4 \cdot |E|$ because we use about 4 symbols to encode each edge, or $4|V|^2$, because there are $|V|^2$ pairs of vertices. But these are not correct, because we will need more than one symbol to encode a vertex number if we have ten or more vertices.)

Solution. Let's do this first in terms of the number of edges. If every vertex number was encoded by a single symbol, then we would get $4|E| - 1$ as the *exact* length of the encoding. The point is that in there are $|V|$ vertices, then it will require as many as $\log_{10} |V|$ decimal digits to encode a vertex number. If we multiply by this 'fudge factor', we get an *upper bound* of $4|E| \cdot \log_{10} |V|$. If we bound the number of edges by $|V|^2$, then we get an upper bound of $4|V|^2 \log_{10} |V|$.

That is really all you need to write to solve the problem, but let's pursue this a bit, because there is an important point to make here. This is an upper bound, which means we've over-counted. For instance, not every vertex number requires all $\log_{10} |V|$ digits, and the separator symbols require only 1 symbol. Furthermore, the graph cannot have $|V|^2$ edges: In the worst case, where every edge is connected to every other edge (the so-called *complete* graph), we have

$$\binom{|V|}{2} = \frac{1}{2}(|V|^2 - |V|)$$

edges. But we have not overcounted that badly. If $V = 10^k$, then ninety per cent of the vertices will require all $k = \log_{10} |V|$ digits to encode them, and thus ninety percent of the edges will contain a vertex that uses at least this many symbols. Thus for the complete graph there are *at least*

$$0.45(|V|^2 - |V|) \log_{10} |V|$$

symbols in the encoding. If there are 2 or more vertices, then $|V|^2 - |V| \geq \frac{1}{2}|V|^2$, so we get a *lower bound* of

$$0.2|V|^2 \log_{10} |V|.$$

Thus we have lower and upper bounds that both have the form

$$c \cdot |V|^2 \log_{10} |V|.$$

In the CS lingo we would say something like, 'the length of the encoding in the worst case is proportional to $n^2 \log n$ where n is the number of vertices'.

(f) The problem with this encoding scheme is that it only includes vertices that are endpoints of edges. But a general graph can have *isolated* vertices that are not connected by an edge to any other vertex. Describe a scheme for encoding graphs by strings that behaves correctly for all undirected graphs. (There are many good answers here.) Give an example illustrating how your encoding works.

Solution. Almost anything you do here to include the isolated vertices will work. For instance, we could include a list of the vertices at the beginning of the encoding, and separate this from the list of edges by $\#\#$. Thus the graph from part (c) with four vertices would be encoded by

$$1\#2\#3\#4\#\#1\#2\#1\#3\#1\#4\#2\#4,$$

but if we added a fifth vertex that was not part of any edge we would get

$$1\#2\#3\#4\#5\#\#1\#2\#1\#3\#1\#4\#2\#4.$$

2 A Turing Machine

This problem refers to the Turing machine in Figure 3.

(a) Trace the execution of the machine (that is, give the complete sequence of configurations) on the input 000, 11 and 1010. You can do this by hand (which is a little tedious for the last example, but not too terrible), or you can use the

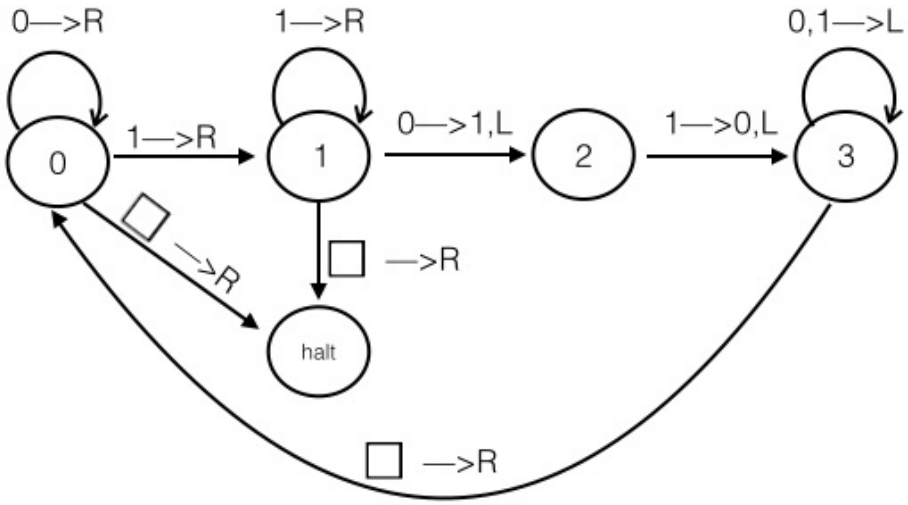

Figure 3: The Turing machine for Problem 2.

Turing machine simulator program. In the latter case, you should include your specification file with your submitted assignment, as well as the output printed by the program.

Solution.

Below is the specification file of this machine for the Turing machine simulator.

You can use this specification to generate the required runs (omitted here). The one for `1010` takes 24 steps and ends with `0011`. The runs for `000` and `11` halt with the inputs unchanged, after many fewer steps.

(b) Describe *in general* the function f computed by this Turing machine.When you figure out the answer, you should be able to find $f(011011011011011011011011)$ quickly, without running the machine.

Solution. If you study the runs, or just run on a lot of examples, you can figure out what is going on: The machine is *sorting* its input string, moving all the 0's to the left and the 1's to the right. So for the example in this problem the result is

$$000000000111111111111111,$$

```
0   0   0   0   R
0   1   1   1   R
0   B   -3  B   R
1   0   2   1   L
1   1   1   1   R
1   B   -3  B   R
2   1   3   0   L
3   0   3   0   L
3   1   3   1   L
3   B   0   B   R
```

Figure 4: The Turing machine of Problem 2, as a specification for the simulator.

which you can also write as

$$0^9 1^{16}.$$

The way the sorting takes place is this: In each phase, the machine scans the tape left to right, looking for a `0` immediately preceded by a `1`. It changes the 0 to 1, then moves left and changes the 1 to 0—that is, it interchanges these two symbols, and then returns to the left end of the tape. If it discovers that there is no `0` preceded by a `1`, then the string is sorted, and the machine halts.

(c) Here is a harder, but important question. How many steps does it take (*i.e,* how long is the complete sequence of configurations) on an input of length n? The answer depends on the input string, of course: If it is 0^n (all zeros) then the computation terminates very quickly. Here you should give the *worst-case* answer—what input causes the computation to take as long as possible, and how many steps does it require?

An *inversion* in the input is a pair of symbols that are out of order—any 1 that is somewhere to the left of any 0 is an inversion. Thus

$$111000$$

has 9 inversions, because each of the three ones precedes each of the three zeros, while

$$101010$$

has 6 inversions, and

$$010101$$

has 3. During the sorting process, each 1 has to be interchanged with each 0 , and the machine requires a pass back and forth over its input to do the interchange. The pair to be interchanged might be found close to the beginning of the string, or close to the end, but each pass will require at most $2n$ steps on an input of length n. The number of such passes is the number of inversions. How many inversions are there in the worst case? The worst case is

$$1^{n/2}0^{n/2},$$

which gives $n^2/4$ inversions. Thus the number of steps is never larger than

$$2n \cdot n^2/4 = n^3/2.$$

We have neglected the last pass, which requires an additional n steps, so we get an upper bound of $n^3/2 + n$. In this worst case, at least half of the interchanges take place in second half of the string, and thus there are at least $n^2/8$ passes that require at least n steps, and consequently the number of steps is at least $n^3/8$. So we have both lower and upper bounds of the form cn^3 on the number of steps in the worst case

(That's even more than I intended you to do, but it is possible to carefully count the number of interchanges performed at each location in the string and find an exact formula for the worst case number of steps. The answer is a cubic polynomial with leading term $n^3/4$.)