Engineering 9867 Advanced Computing Concepts

Assignment #2 – Sample solutions

Due: Tuesday, April 2 at 0900

1. [10 points] Consider the following implementation of the palindrome checking problem (question 4 on assignment 1):

```
bool
isPalindrome(const string& s)
{
  bool result = true;
  int size = s.size();
  int i = 0;

  while (result && i < size/2) {
    result = (s[i] == s[size-1-i]);
    i++;
  }
  return result;
}
```

a) [5 points] Give an expression for the worst case time (note **not** complexity) of this algorithm.

> The worst case time occurs when `s` is a palindrome, in which case the loop repeats `size`/2 times, so
> $WT_{\texttt{isPalindrome}}(\texttt{size}) = k_0 + k_1 \times \lfloor \texttt{size}/2 \rfloor$

b) [5 points] Assuming that the strings contain only of the letters "a" and/or "b", and that all strings are equally likely, what is the average case time for this algorithm?

> Let N be the length of the string. For strings containing only "a" or "b", the probability any particular comparison will find the letters unequal, and thus cause the algorithm to stop is $P(\texttt{s}[\texttt{i}] \neq \texttt{s}[N-1-\texttt{i}]) = \frac{1}{2}$. The probability that it will stop at exactly the $\texttt{i}^{\text{th}}$ itteration $P(\text{stop at } \texttt{i}) = \begin{cases} \frac{1}{2^i} & \text{, for } \texttt{i} < \lfloor \frac{N}{2} \rfloor \\ 1 - \sum_{j=1}^{i-1} P(\text{stop at } j) = \frac{1}{2^{i-1}} & \text{, for } \texttt{i} = \lfloor \frac{N}{2} \rfloor \end{cases}$ and thus the expected number of itterations is
>
> $$\begin{aligned} E(N) &= \sum_{j=1}^{\lfloor \frac{N}{2} \rfloor} j \times P(\text{stop at } j) \\ &= \frac{\lfloor \frac{N}{2} \rfloor}{2^{\lfloor \frac{N}{2} \rfloor - 1}} + \sum_{j=1}^{\lfloor \frac{N}{2} \rfloor - 1} \frac{j}{2^j} \\ &\approx 2 \end{aligned}$$
>
> so
>
> $AT_{\texttt{isPalindrome}}(N) = k_2$

2. [10 points] Consider the following algorithm that satisfies the specification as follows:

Pre: $1 \leq \mathtt{m} \leq \mathtt{n}$

Post: $\mathtt{result} \leq \mathtt{n} - \mathtt{m} \rightarrow (\forall i, 0 \leq i < \mathtt{m} \rightarrow \mathtt{P}[i] = \mathtt{S}[\mathtt{result} + i]) \;\wedge$
$\mathtt{result} > \mathtt{n} - \mathtt{m} \rightarrow (\neg\exists j, 0 \leq j \leq \mathtt{n} - \mathtt{m} \wedge (\forall i, 0 \leq i < \mathtt{m} \rightarrow \mathtt{P}[i] = \mathtt{S}[j + i]))$

```
result = −1
matched = false
while (result < n − m ∧ ¬matched) do
  result = result + 1
  i = 0
  matched = true
  while (i < m ∧ matched) do
    matched = matched ∧ (P[i] == S[result + i])
    i = i + 1
  end while
end while
if (¬matched) then
  result = result + 1
end if
```

a) [9 points] Give an expression for the exact worst case (i.e., Θ) complexity for this algorithm. Show your workings.

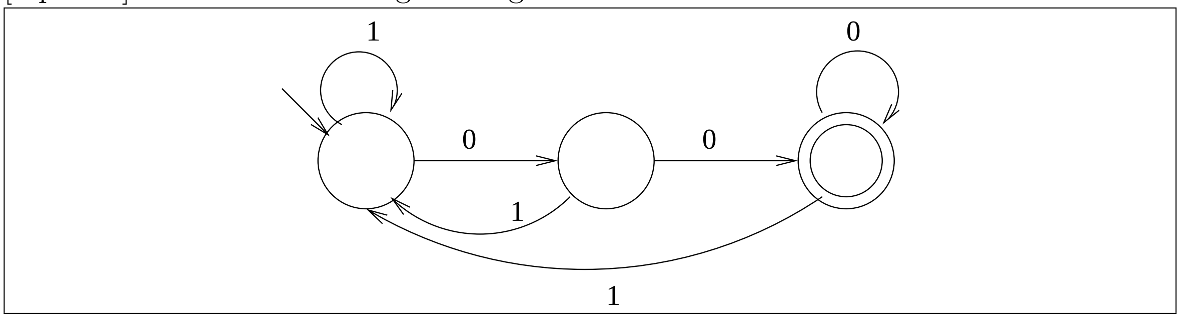
In the worst case the outer loop will execute $\mathtt{n} - \mathtt{m}$ times, and each time the inner loop will execute $\mathtt{m}$ times (this would not actually happen, but it doesn't make any difference for the complexity), so $WT_{\mathtt{match}}(\mathtt{n}, \mathtt{m}) \in \Theta(\mathtt{n} \times \mathtt{m})$.

b) [1 point] What does this tell us about the complexity of the problem solved by this algorithm?

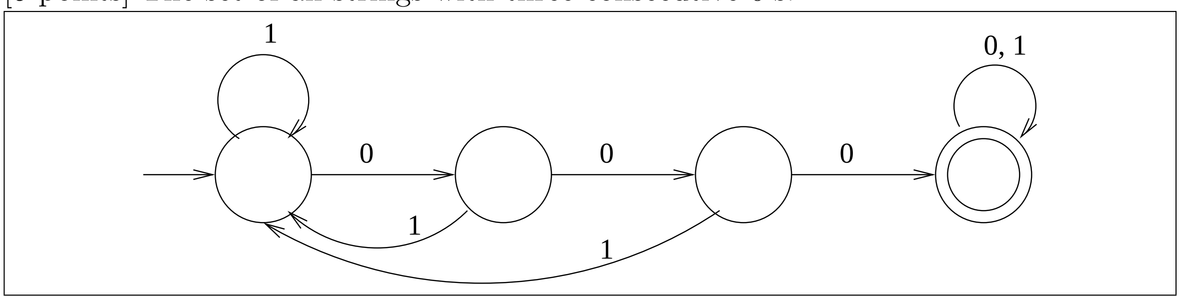
This tells us that the problem of pattern matching is $O(\mathtt{n} \times \mathtt{m})$ (i.e., the problem complexity is no worse than linear in the length of the string to search). Note that this does **not** tell us that the complexity is no better than this, in fact there are sub-linear algorithms for this problem.

3. [15 points] Give a (deterministic) finite state automata on the input language $\Sigma = \{0, 1\}$ accepting each of the following languages:

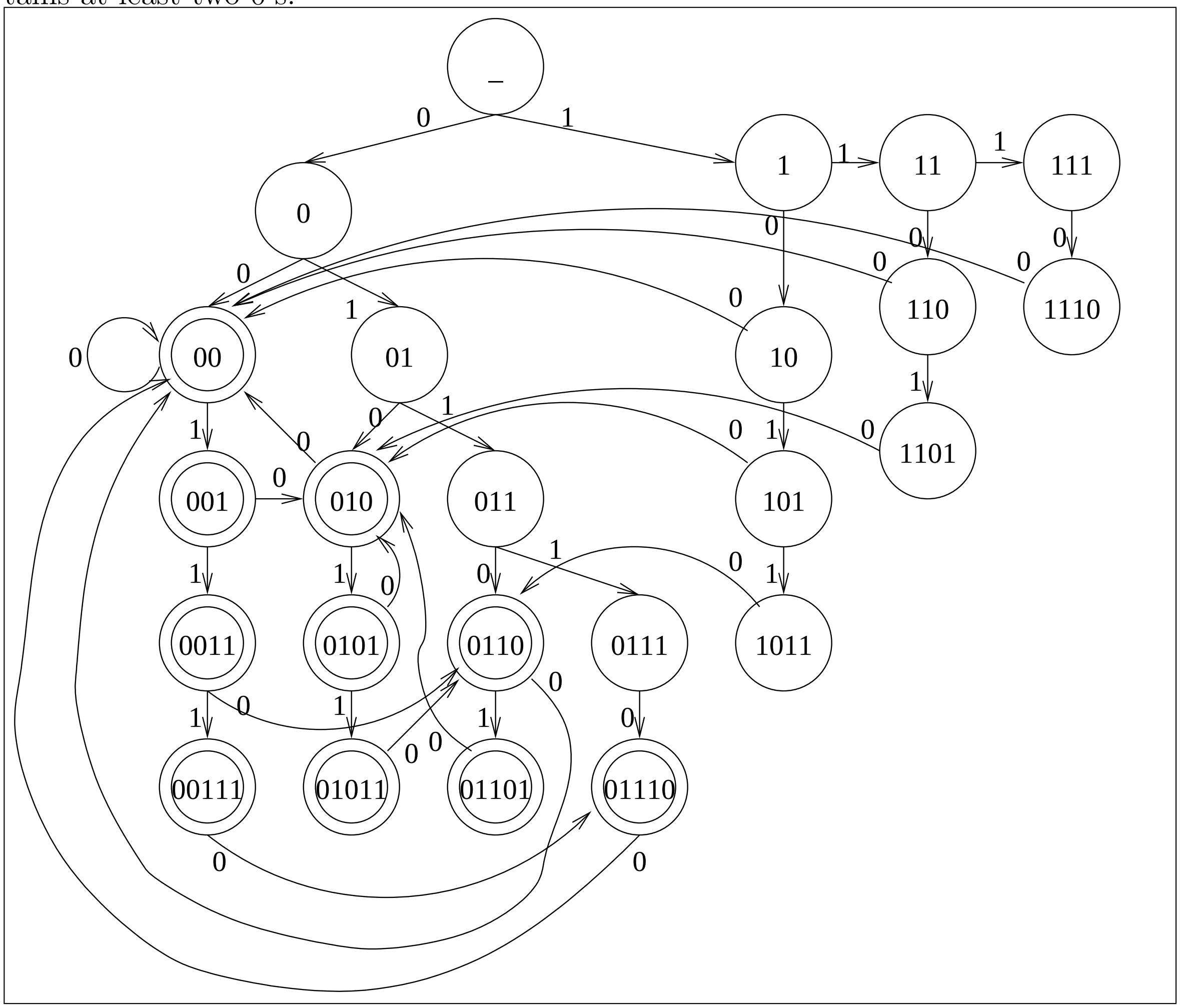
a) [5 points] The set of all strings ending in 00.



b) [5 points] The set of all strings with three consecutive 0's.



c) [5 points] The set of all strings such that every block of five consecutive symbols contains at least two 0's.



4. [15 points] Let G be an undirected graph consisting of a set of nodes, N, and a set of edges $E \subseteq N \times N$. A set of nodes $N' \subseteq N$ is called a *vertex cover* for G, if for every edge in E, at least one of its end-points is in N'. The *vertex cover problem* is, given a graph, G, and a positive integer, K, determine whether there is a vertex cover for G with at most K nodes.

   Show that the vertex cover problem is NP-complete.

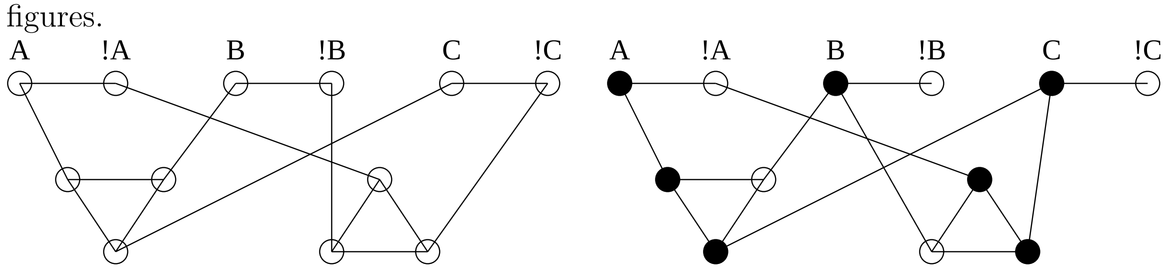
First we must show that VCP is in NP. To do this we give the non-deterministic polynomial time algorithm for VCP, as follows:
Assume that $N = \{n_0, n_1, \ldots n_M\}$, for some $M > K$, and that $E = \{e_0, e_1, \ldots e_Q\}$.

for $i = 1$ to M **do**
  if ($\texttt{magicCoin}() \wedge |N'| < K$) **then**
    Add n_i to N'
  end if
end for
for $i = 1$ to Q **do**
  if Neither of the endpoints of $e_i \in N'$ **then**
    output "No"
    stop
  end if
end for
output "Yes"

Clearly this algorithm is polynomial time, so VCP is in NP.
Now we must show that another problem that is known to be NP-complete can be reduced to VCP. I'll use 3SAT.
For an instance of 3SAT, assume we have the variables $v_1, v_2, \ldots v_n$, and the 3CNF formula "$(a_{1,1} \vee a_{1,2} \vee a_{1,3}) \wedge (a_{2,1} \vee a_{2,2} \vee a_{2,3}) \ldots (a_{m,1} \vee a_{m,2} \vee a_{m,3})$" where each $a_{i,j}$ is v_k or $\neg v_k$.
We construct an instance of VCP as follows: The set of nodes in G is
$N = \left\{ \begin{array}{l} v_1, v_2, \ldots v_n, \neg v_1, \neg v_2, \ldots \neg v_n, \\ a_{1,1}, a_{1,2}, a_{1,3}, a_{2,1}, a_{2,2}, a_{2,3}, \ldots a_{m,1}, a_{m,2}, a_{m,3} \end{array} \right\}$ and the set of edges is

$$E = \left\{ \begin{array}{l} (v_1, \neg v_1), (v_2, \neg v_2), \ldots (v_n, \neg v_n), \\ (a_{1,1}, a_{1,2}), (a_{1,2}, a_{1,3}), (a_{1,1}, a_{1,3}), \\ (a_{2,1}, a_{2,2}), (a_{2,2}, a_{2,3}), (a_{2,1}, a_{2,3}), \ldots \\ (a_{m,1}, a_{m,2}), (a_{m,2}, a_{m,3}), (a_{m,1}, a_{m,3}) \end{array} \right\} \cup \{(a_{i,j}, v_k) | a_{i,j} \text{ is } v_k\} \cup \{(a_{i,j}, \neg v_k) | a_{i,j} \text{ is } \neg v_k\}$$

That is, G consists of an edge for each variable and its negation, a triangle for each clause and an edge connecting each term in the clause to its value. Choose $K = n + 2m$ — the number of variables plus twice the number of clauses. The formula is satisfiable iff there is a vertex cover of G containing at most K nodes. This is illustrated in the following figures.



(A+B+C)&(!A+!B+!C)   (A+B+C)&(!A+B+C)