The Power of the Queue

Ming Li, * Luc Longpré† and Paul Vitányi‡

1988, Published in final form in SIAM J. Computing, 21:4(1992), 697-712.

Abstract

Queues, stacks, and tapes are basic concepts which have direct applications in compiler design and the general design of algorithms. Whereas stacks (pushdown store or last-in-first-out storage) have been thoroughly investigated and are well understood, this is much less the case for queues (first-in-first-out storage). In this paper we present a comprehensive study comparing queues to stacks and tapes (off-line and with one-way input). The techniques we use rely on Kolmogorov complexity. In particular, 1 queue and 1 tape (or stack) are not comparable:

(1) Simulating 1 stack (and hence 1 tape) by 1 queue requires $\Omega(n^{4/3}/\log n)$ time in both the deterministic and the nondeterministic cases.

(2) Simulating 1 queue by 1 tape requires $\Omega(n^2)$ time in the deterministic case, and $\Omega(n^{4/3}/(\log n)^{2/3})$ in the nondeterministic case;

We further compare the relative power between different numbers of queues:

(3) Nondeterministically simulating 2 queues (or 2 tapes) by 1 queue requires $\Omega(n^2/(\log^2 n \log\log n))$ time and deterministically simulating 2

*Aiken Computation Laboratory Harvard University, Cambridge, MA 02138. Work supported by the NSF under Grant DCR-8606366 and by the ONR under Grant N00014-85-k-0445.

†College of Computer Science, Northeastern University, Boston, MA 02115. Research performed while a visiting faculty member in the Computer Science Department at the University of Washington.

‡Centrum voor Wiskunde en Informatica, Kruislaan 413, 1098 SJ Amsterdam, The Netherlands. Work performed in part at the Laboratory for Computer Science, Massachusetts Institute of Technology, and supported in part by the Office of Naval Research under Contract N00014-85-K-0168, by the U.S. Army Research Office under Contract DAAG29-84-K-0058, by the National Science Foundation under grant DCR-83-02391, and by the Defence Advanced Research Projects Agency under contract N00014-83-K-0125.

queues (or 2 tapes) by 1 queue requires $\Omega(n^2)$ time. The second bound is tight. The first is almost tight. We also obtain the upper bounds for queues. Below, we use "pushdown" and "stack" synonymously.

1 Introduction

It has been known for over twenty years that all multitape Turing machines can be simulated on-line by 2-tape Turing machines in time $O(n \log n)$ [HS2], and by 1-tape Turing machines in time $O(n^2)$ [HU]. Since then, many other models of computation have been introduced and compared. (See [Aa, DGPR, HS1, HS2, HU, LS, PSS, Pa, Vi2].) In addition to different storage mechanisms, real-time, on-line and off-line machines have been studied. An on-line simulation essentially simulates step-by-step each move of the simulated machine. In this paper, we consider off-line machines, where an answer is given only once the whole input has been read. There is no need to simulate the moves of the machine; it only matters that we give the right answer. We also use the one-way input convention, where the machine has a one-way input. As usual, the machines have a finite control and access to some storage.

The relative power of stacks and tapes is more or less well known. For example, for the nondeterministic case, we know that 1 stack $<$ 1 tape $<$ 2 stacks $<$ 3 stacks $=$ k stacks $=$ k tapes, where $A < B$ means that B can simulate A in linear time, while A cannot simulate B in linear time. In most of the cases, close lower and upper bounds are known for the simulation [Ma, Li1, Vi1, LV, Li2].

In this paper, we give a complete characterization of (off-line, one-way input) queue machines. The main theorems show that one queue machines are not comparable to one stack or one tape machines, both deterministically and nondeterministically. We also compare the relative power of machines having different number of queues. The current knowledge of upper and lower bounds for the simulation between queues and tapes is roughly summarized in Figures 1, 2 and 3. Figure 1 contains results that were previously known. The results of Figure 2 are covered in section 2. Notice that all the bounds in Figure 2 are valid also for simulating one stack or two stacks. The results of Figure 3 are covered in section 3.

	deterministic	nondeterministic
upper bound	$O(n^2)$ (straightforward)	$O(n^{3/2}\sqrt{\log n})$ (in [Li1])
lower bound	$\Omega(n^2)$ (in [LV])	$\Omega(n^{4/3}/\log^{2/3} n)$ (in [LV] or [Li3])

Figure 1: Simulating one queue by one tape

	deterministic	nondeterministic
upper bound	$O(n^2)$	$O(n^2)$
lower bound	$\Omega(n^{4/3}/\log n))$	$\Omega(n^{4/3}/\log n)$

Figure 2: Simulating one tape, or one stack, by one queue

	deterministic	nondeterministic
upper bound	$O(n^2)$	$O(n^2)$
lower bound	$\Omega(n^2)$	$\Omega(n^2/\log^2 n \log\log n))$

Figure 3: Simulating two queues by one queue

We use Kolmogorov complexity techniques [So, Ko, Ch] to prove the theorems, together with some new techniques to enable us to deal with queues. The Kolmogorov complexity of a string x, $K(x)$, is the length of the shortest program printing the string x. By a simple counting argument, we know that there are strings x of each length such that $K(x) \geq |x|$. These strings are called *incompressible* or *K-random*. For completeness we recall the notions of Kolmogorov complexity of binary strings and those of self-delimiting descriptions, see e.g. [PSS] or [LV]. Fix an effective coding C of all Turing machines as binary strings, such that no code is a prefix of any other code. Denote the code of Turing machine M by $C(M)$. The Kolmogorov complexity with respect to C, of a binary string x, denoted as $K_C(x)$ is the length of the smallest binary string $C(T)y$ such that T started on input y halts with output x. The crucial fact one uses is that in between effective enumerations C and D, for all x, $|K_C(x) - K_D(x)| < c$, with c a constant depending only on C and D (but not on x). So, up to an additive constant, the Kolmogorov complexity is independent of the particular effective enumeration choosen which allows us to drop the subscript. With some abuse of notation, in the sequel equalities and inequalities involving Kolmogorov complexity will always be assumed to hold up to additive constant only. To be able to differentiate between parts of y such that T is able to use different parts for different purposes (can compute an r-ary function), we need the notion of self-delimiting descriptions. If $a = a_1 a_2 ... a_n$ is a string of 0's and 1's, then $a_1 0 a_2 0 ... 0 a_n 1$ is a self-delimiting description of twice the original length. More efficiently, if $b = b_1 ... b_m$ is the length of a in binary, then the self-delimiting description of b concatenated with a is also a self-delimiting description of a, this time of length $n + 2\log n$ instead of $2n$. For example, 1000011101 is the self-delimiting version of 1101.

2 Simulating one tape by one queue

2.1 Upper bound

Our upper bound is straightforward. However, it is for simulating any fixed number of stacks. Since two stacks can simulate one tape in real time, our upper bound applies for tapes as well.

Theorem 2.1 *For any fixed k, one queue can simulate k stacks in quadratic time, for both deterministic and nondeterministic machines.*

Proof. Simulate the k stacks by coding them sequentially onto the queue such that the top of each stack comes first. In front of each stack top, put a marker to indicate the separation between the stacks.

Each operation (push or pop on one stack) can be done on $O(n)$ time by just scanning the whole queue and performing the local transformation after the appropriate marker. The total time is then in $O(n^2)$. This simulation can be made for deterministic or nondeterministic machines. □

2.2 Lower bound

In this section, we show that it takes at least $\Omega(n^{4/3}/\log n)$ time for a nondeterministic one queue machine with a one-way input to recognize the language $\{w\#w^R : w \in \{0,1\}^*\}$.

Because this language can be recognized in linear time by a deterministic stack automaton, we can conclude that it takes at least $\Omega(n^{4/3}/\log n)$ time for a nondeterministic one queue machine to simulate a deterministic stack automaton.

The intuition behind the proof is that while the queue machine reads w, it has to store all the information in some sequential way on the queue. It turns out to be impossible to check the stored form of w for correspondence with w^R while the latter string is read from the input tape, so w^R must be stored in some sequential way as well. Using crossing sequence arguments, we show that whatever way the information is stored, the machine will be forced to scan the queue many times. This repeated scanning will then imply the lower bound on simulation time.

Let h_{in} be the read-only head on the one-way input tape. We identify the queue with a tape with two heads h_r and h_w. The head h_r is a read-only, one-way head on the queue. The head h_w is a write-only, one-way head on the queue. One step of the queue consists of all of the following. According to the old state and the contents of the cells scanned by the reading heads, the machine: (1) reads an empty or nonempty symbol from the input; (2) pops an empty or nonempty symbol from

the queue; (3) pushes an empty or nonempty symbol on the queue; and (4) changes state. This course of events is implemented as follows on the tape reperesentation: (1) if a nonempty symbol is read from the input tape, then h_{in} moves to the right adjacent cell; (2) if a nonempty symbol is written (pushed) on the queue, then h_w writes the symbol in the currently scanned cell and moves to the right adjacent cell; (3) if a nonempty symbol is read (popped) from the queue, then h_r moves to the right adjacent cell; and (4) the change of state is the same. Without loss of generality, we assume that the machine uses binary alphabets on the queue and accepts by empty queue. Let $h_k(t)$ denote the position of head $k \in \{in, r, w\}$ at time t on its respective tape. Let $c_1, c_2, \ldots, c_n$ be the individual cells on the input tape. Let $d_1, d_2, \ldots$ be the individual cells on the queue.

The contents of the tape from $h_r(t)$ through $h_w(t) - 1$ inclusive, is called the *actual queue* at time t, or $Queue(t)$. The length of $Queue(t)$, denoted as $|Queue(t)|$, is $h_w(t) - h_r(t)$. We say that two cells d_i and d_j are *contiguous* on $Queue(t)$ if $h_r(t) < j < h_w(t)$ and $j = i + 1$, or if $i + 1 = h_w(t)$ and $j = h_r(t)$ (the cells at both ends of the queue are also considered contiguous).

Theorem 2.2 *A nondeterministic one queue machine with a one-way input tape takes time* $\Omega(n^{4/3}/\log n)$ *to accept the language* $\{w\#w^R : w \in \{0,1\}^*\}$. [1]

Remark. This holds both for the worst-case time and the average time.

Proof. Toward a contradiction, assume there is a queue Q that accepts L in time $T(n)$ not in $\Omega(n^{4/3}/\log n)$.

Fix n large enough, such that the formulas below and the intended contradictions hold. Let $2\ell + 1 = n$ and let x be an incompressible string of length ℓ. We separate x into blocks: $x = x_0x_1x_2...x_m$. Let $|x_0| = \ell/2$. Each block x_i for $1 \leq i \leq m$ contains p symbols. For the proof of the theorem, we take $m = \ell^{1/3}/4$ and $p = 2\ell^{2/3}$. We look at any fixed accepting computation of the machine on input $x\#x^R$. Let t_j be the time step when h_{in} enters the block x_j. Let t'_j be the time step when h_{in} enters the block ${x_j}^R$. If z is a substring of x, then z' denotes the corresponding substring of x^R $(= x')$.

Claim 2.3 *If* $t_1 \leq t \leq t'_0$, *then the length of* $Queue(t)$ *is at least* $\ell/2 - O(\log \ell)$.

Proof. For any $t_1 \leq t \leq t'_0$, x can be reconstructed using the following information: a description of this discussion and of Q in $O(1)$ bits, the string $Queue(t)$ of

[1] Here we use the stronger version of Ω where $T(n) \in \Omega(f(n))$ if there are constants c and n_0 such that for all $n \geq n_0$, $T(n) \geq cf(n)$. Notice that there is no string of even length in the language. To be strict, we show that the time is in $\Omega(n^{4/3}/\log n \ : \ n \text{ is odd})$. With a slightly modified language: $\{x\#x^R\} \cup \{x\#\#x^R\}$, we could prove it for all n.

length $< \ell$, the string $x_1 \ldots x_m$ of length $\ell/2$, the state $q(t)$ of the machine in $O(1)$ bits, and $h_{in}(t)$ in $\leq \log \ell + 2$ bits. Apart from $x_1 \ldots x_m$, all items are encoded as self-delimiting strings, so that the total number of bits required is $< s + \ell/2 + O(\log \ell)$.

To reconstruct x from this information, run the machine with all possible candidate strings y substituted for x_0. Single out the strings y for which there is a time step for which $Queue(t)$, $h_{in}(t)$ and $q(t)$ correspond. Among those y, the machine should accept only if $y = x_0$, otherwise it would accept the string $x_0 x_1 \ldots x_m \# x_m^R \ldots x_1^R y$ which is not in the language.

Because x is incompressible, we know that $K(x) \geq \ell$, so it must be that our program reconstructing x has size $\geq \ell$. So, we have $s + \ell/2 + O(\log \ell) \geq \ell$, from which the claim follows. □

The machine needs to remember what it reads on the input and code it in some way on the queue. What is written on the queue is determined by the input and the rear of the queue. We say that a cell d_j is *directly influenced* by a cell c_i if $h_{in}(t) = i$ at the time t when h_w writes on d_j. (That is, $h_w(t) = j$ and $h_w(t+1) = j+1$.) Similarly, a cell d_j is *directly influenced* by a cell d_i if $h_r(t) = i$ at the time t when h_w writes on d_j.

The *influence relation* is the transitive closure of the direct influence relation. We say that c_i (or d_i) influences d_j if there is a chain of direct influences from c_i to d_j. A block of cells influences a cell if and only if at least one of the cells in the block influences it. A block of cells is influenced by a block of cells, if at least one cell of the first block is influenced by the second block. The influence relation will allow us to talk about where the information can be stored on the queue.

In the following, we need the notion of cycles. A *cycle* $c(t) = [t, \hat{t})$ is a half open interval (of time) such that $h_r(\hat{t}\,) = h_w(t)$. Given a time t_1, we will be interested in non overlapping contiguous cycles $c_1(t_1), c_2(t_2), \ldots$ starting at time t_1, such that $c_1(t_1) = [t_1, t_2)$, $c_2(t_2) = [t_2, t_3)$, and so on. In what follows, whenever we count cycles, the start time t_1 will be either specified or clear from context and we count the successive non overlapping contiguous cycles, as induced by the computation of Q.

Let $\tilde{x}$ be the string $x_1 \ldots x_m$.

Claim 2.4 *If t is fewer than s cycles away from t_1, then each cell in $Queue(t)$ is influenced by at most s input cells in $\tilde{x}\#\tilde{x}^R$.*

Proof. Let the chain of cycles starting from t_1 be $c_1(t_1)$, $c_2(t_2)$,.... The proof is by induction on the indices s. At time t_1, no cell in $Queue(t)$ is influenced by any input cell in $\tilde{x}\#\tilde{x}^R$. During c_1, each cell written can be influenced by at most one of those input cells. Suppose the claim is true for cycles c_1 through c_{s-1}. During the cycle $c_s(t_s)$, each cell written is influenced by one new input cell (possibly) and

by each input cell that influences the cell scanned by h_r. This adds up to at most s input cells. □

For each i, we say that x_i is a *valid* block if $Queue(t_0')$ contains a cell which is not influenced by x_i nor by ${x_i}'$. ($t_i - 1'$ is the time when $h_i n$ leaves ${x_i}'$.) The following claims show that valid blocks exist. (We only need the existence of one valid block, but in fact, the majority of blocks are valid.)

Claim 2.5 *For each cell d on $Queue(t_0')$, there is a block x_i such that d is influenced by neither x_i nor by ${x_i}^R$.*

Proof. Suppose there is a cell d on $Queue(t_0')$ such that for all i, d is influenced by either x_i or ${x_i}^R$. It means that d is influenced by at least m different cells. By claim 2.3, we know that then the machine makes at least $m-1$ cycles from t_1 to t_0'. By claim 2.2, the queue has length at least $\ell/2 - O(\log \ell)$ for each cycle, so the algorithm will take at least $(m-1)(\ell/2 - O(\log \ell)) \geq \Omega(n^{4/3})$, which is a contradiction. □

Claim 2.6 *If at some time t there is a cell on $Queue(t)$ not influenced by a block y, then for all $\hat{t} \leq t$, there is a cell on $Queue(\hat{t})$ not influenced by y.*

Proof. If at some time all the cells on the actual queue are influenced by y, then no new cell can be written without being influenced by y. □

From the last two claims it follows that there is always a valid block in the computation. The following two facts explain why a valid block is a part of the input that has been coded sequentially on the queue. The facts can be derived almost directly from the definition of the influence relation and the observation that a valid block is read within one cycle.

Fact 2.7 *For each valid block x_j, each cell in x_j influences a disjoint set of cells on the queue. Moreover, cells in x_j' also influence a disjoint set of cells on the queue. However, some cells on the queue can be influenced by both a cell of x_j and a cell of x_j'.*

Fact 2.8 *For any time t, ${t_0}' > t > t_{j+1}$, the* regions *influenced by the sequence of cells of a valid block x_j form a contiguous ordered sequence on $Queue(t)$. (The same statement holds for x_j'.)*

□

A *partial configuration* of the machine at some time t is the state of the machine and the position of all the heads on their respective tape.

The *crossing sequence* associated with a cell d_i is the partial configuration when h_r goes from cell d_i to cell d_{i+1} ($h_r(t-1) = d_i$ and $h_r(t) = d_{i+1}$). Because for us the queue is in fact a one-way tape, the crossing sequence associated with a cell will have exactly one entry. Each entry contains the position of 3 heads and a machine state. Since writing using more than n^2 tape cells would take too much time, we may assume that each head position can be described in $O(\log n)$ bits.

The *crossing sequence* around a region $d_i \ldots d_j$ is the crossing sequence associated with d_{i-1} concatenated with the one associated with d_j.

The *crossing sequence* around a list of regions is the concatenation of the crossing sequences around each of the regions.

Any substring y of the input influences a series of regions on the tape, one for each cycle of the queue. The rest of this paragraph addresses degenerate cases and the reader who is not interested in verifying the details may proceed to the next paragraph. The situation can degenerate in two different cases. It could happen that at some time t, y influences (1) all of $Queue(t)$ or (2) no cell on $Queue(t)$. Each of those two situations is irreversible, i.e. the situation will remain for all $\hat{t} \geq t$. We want to have one different region for each cycle of the queue even in the degenerate cases. In the case when no cell on $Queue(t)$ is influenced by y, we can still talk about the region influenced by y on $Queue(t)$ as an empty region that has a specific position. In particular, if, while h_r is reading the cells influenced by y, h_w does not write anything, then the region influenced by y is an empty region between $h_w(t)-1$ and $h_w(t)$ (between the cell written into before accessing y and the cell written into after accessing y). Also, if, at time t, h_r goes from cell d_i to cell d_{i+1} while there is an empty region between d_i and d_{i+1}, then there is also an empty region between $h_w(t)$ and the previous cell. For the second degenerate case, the regions influenced by y also form a series of regions, one for each cycle. In this case, the regions are side by side, with empty gaps between them having a precise position defined an a similar fashion as in degenerate case (1).

At this point, we assumed that there is a valid block x_i. For this block, both x_i and $x_i{}'$ have been coded sequentially on the queue. Now we have to show that it takes a lot of time to check $x_i{}' = x_i{}^R$. Intuitively, we can check only a constant number of bits of $x_i{}'$ at each cycle. Each cycle takes as much time as the size of the queue at that time.

Claim 2.9 *The crossing sequence around the list of regions influenced by a substring of the input from time t_1 up to time t_0' has at most $\ell^{1/3}$ entries.*

Proof. Each addition to the crossing sequence corresponds to one cycle. By Claim 2.2, each cycle takes at least $\ell/2 - O(\log \ell)$ steps. So, the machine will take $\Omega(n^{4/3})$ time, contradicting the assumption. □

Claim 2.10 *If $t > t_{i-1}$ is fewer than r cycles away from t_{i-1}, then $Queue(t)$ has length at least $\ell^{2/3} - O(n^{1/3}\log n) - O(r \log n)$.*

Proof. Let x_i be a valid block, $i > 0$. Then, t'_{i-1} is the time step when h_{in} leaves ${x_i}^R$. Let $x_i = uv$, where u and v are strings of equal size. Select a cell d on $Queue(t'_{i-1})$ which is not influenced by x_i nor by ${x_i}^R$. By claim 2.4 and 2.5, such a cell exists. The cells of $Queue(t'_{i-1})$ influenced by the cells of x_i form an ordered sequence of regions. This is also true for the cells of ${x_i}^R = v^R u^R$.

If the region influenced by u on $Queue(t'_{i-1})$ is disjoint from the region influenced by u^R, then let $y = u$ and $y' = u^R$. If the region influenced by u^R is empty, we need to treat it as a special case. We then consider the region influenced by u^R as a border between two cells, and if this border is inside a region influenced by u, it is considered not to be disjoint. If the regions are not disjoint, it must be the case that the regions influenced by v and v^R are disjoint, because v comes after the region influenced by both u and u^R, and v^R comes before. In this case, choose $y = v$ and $y' = v^R$.

Let S be the set of cells influenced by y. Let $\bar{x}$ be the string x where y is deleted, and let t be as in the statement of the claim. We show below that x can be computed from $\bar{x}$, t, the position of y in $\bar{x}$ and the crossing sequence around S up to time t. Encoding each item self-delimiting, except $\bar{x}$ which we give literally, this description takes $\ell - \ell^{2/3} + O(n^{1/3}\log(n)) + O(r\log(n)) + |Queue(t)|$ bits. Because $K(x) \geq n$, it then follows that $|Queue(t)| \geq \ell^{2/3} - O(n^{1/3}\log(n)) - O(r\log(n))$.

We compute y with the information provided in the following way. For all binary strings z of equal length as y, let x_z be the string x where z has been substituted for y. Run Q on all strings $x_z \# {x_z}^R$ until you find one that matches the description. By construction, $z = y$ matches the description. Suppose $z \neq y$ matches the description as well. Then, by cutting and pasting the two computations on $x\#x^R$ and $x_z\#{x_z}^R$, we can construct a legal computation of Q on $x_z\#x^R$. Let S be the set of cells influenced by y in the computation on $x\#x^R$, and let S_z be the set of cells influenced by z in the computation of $x_z\#{x_z}^R$. Because the crossing sequence includes the position of all heads, the regions in S and in S_z occupy the same positions on the queue. Therefore, we can compose an accepting computation that starts out as the computation on $x\#x^R$, continues with the computation on $x_z\#{x_z}^R$ when a head enters S, switches back to the computation on $x\#x^R$ when the last head leaves S again, and so on. Since the crossing sequences of both computations are the same up to t, the resulting computation is a legal computation of Q (up to t) on $x_z\#x^R$. Now we can derive a contradiction. The contents of the intersection of $Queue(t)$ and S is the same for the computation on $x_z\#{x_z}^R$ and for the computation on $x_z\#x^R$ by construction, and for the computation on $x\#x^R$ and $x_z\#{x_z}^R$ by assumption. The contents of $Queue(t) - S$ is the same for the computation on $x\#x^R$ and $x_z\#x^R$

by construction. Hence $Queue(t)$ is the same for Q's computation on $x\#x^R$ and $x_z\#x^R$. By assumption and construction the head positions and state of Q at time t are identical as well. Therefore, we can continue the computation on $x_z\#x^R$ by Q and accept this string since $x\#x^R$ is accepted. So Q accepts a string that is not in the language, which is a contradiction. □

Claim 2.11 *The machine makes $\Omega(n^{2/3}/\log n)$ cycles after t'_{i-1}.*

Proof. Let T be the time Q accepts. Then, $Queue(T)$ is of length 0, and by the previous claim, r has to be $\Omega(n^{2/3}/\log n)$. □

Now, for at least 1/2 of the cycles in claim 2.10, $Queue(t)$ will have size in $\Omega(n^{2/3})$ (by claim 2.9). Therefore, the machine makes $\Omega(n^{4/3}\log n)$ steps, contradiction our assumption. The proof of theorem 2.1 is finished. □

3 More queues versus fewer queues

In this section we study the power of queue machines with different number of queues. We first provide some easy upper bounds: Two queues are as good as k queues in the nondeterministic case. This motivates our research focussing on small numbers of queues; One queue can simulate k queues in quadratic time, deterministically or nondeterministically. We then provide tight, or almost tight, lower bounds for our simulations mentioned above.

3.1 Upper bounds

Theorem 3.1 *Two stacks can simulate one queue in linear time, for both deterministic and nondeterministic machines.*

Proof. We design a machine P with 2 stacks $pd1$, $pd2$. To simulate a queue, every time a symbol is pushed into the queue, P pushes the same symbol into $pd1$. If a symbol is taken from the queue, then P pops a (the same) symbol from $pd2$ if $pd2$ is not empty. If $pd2$ is empty then P first unloads the entire contents of $pd1$ into $pd2$ and then pops the top symbol from $pd2$. At the end of the input, P accepts iff the 1 queue machine accepts. □

Theorem 3.2 *Two queues can nondeterministically simulate k queues for any fixed k in linear time.*

Proof. This theorem follows from the method used by Book and Greibach [BG] in order to nondeterministically simulate k tapes by 2 tapes in linear time. For the sake of completeness, we will describe the idea. The 2 queue machine guesses the computation of the k-queue machine and puts this guess on 1 queue in the form: $ID_1, ID_2, ...$, where ID_i contains the state of the k queue machine and the $k+1$ queue symbols scanned by the k queue heads and the input head, at step i. First check that the state in each ID is consistent with the previous ID and check the correctness of the guessed input symbol in each ID_i by scanning the ID's and moving the input head when necessary. Then, scan the ID's again k times, each time simulating one of the k queues of the simulated machine on the other queue. This simulation takes $O((k+1)n) = O(n)$ time. □

Theorem 3.3 *3 stacks can nondeterministically simulate k queues in linear time.*

Proof. Combine the ideas from the above 2 theorems. *I.e.*, guess the computation of the k queue machine as before, and put the guess into one stack. Save this guess also to another stack (but put a marker on the top). Then simulate a queue and check the correctness of the guess. (The simulation needs 2 stacks, one of the stacks has the guessed computation saved in the bottom.) After simulating one queue, retrieve the guessed contents and again put it into 2 stacks. Repeat this process for each queue. □

Remark. It is a folklore fact, and easily verified, that 1 queue machines accept precisely the *r.e.* languages. In contrast, 1 stack machines accept only CFL's. Hence, 1 queue is better than 1 stack. However, when we have more stacks, more stacks seem to be better than queues because they are more efficient. It was proved in [HM] that 4 stacks can simulate a queue in real time.

Theorem 3.4 *One queue can simulate k queues in quadratic time, both deterministically and nondeterministically.*

Proof. This is similar to the simulation of k tapes by 1 tape by Hartmanis and Stearns [HS1]. See [HU, page 292]. □

This also relates to a interesting problem of whether "2 heads (on one tape) are better than 2 tapes (each with one single head)". Vitányi [Vi3] showed that 2 tapes cannot simulate a queue in real time if at least one of the tape-heads is within $o(n)$ cells from the start cell at all times. We saw that 2 stacks can simulate a queue in linear time and 4 stacks can do this in real time. It would be interesting to know whether 2 or 3 stacks can do this in real time. The question of how to deterministically simulate k queues by 2 queues in $o(n^2)$ time, like the Hennie-Stearns simulation in the tape case [HS2], remains open.

3.2 Lower bounds

We now prove optimal lower bounds for the above simulations. Let L be the following language.

$$
\begin{aligned}
L \;=\; \{\;\; & a \,\&\, b_0^1 b_1^1 \ldots b_k^1 \# \\
& b_0^2 b_0^3 b_1^2 b_2^2 b_1^3 b_3^2 \ldots b_{2i}^2 b_i^3 b_{2i+1}^2 \ldots b_{k-1}^2 b_{(k-1)/2}^3 b_k^2 \\
& b_0^4 b_{(k+1)/2}^3 b_1^4 b_2^4 b_{(k+3)/2}^3 b_3^4 \ldots b_{2i \bmod (k+1)}^4 b_i^3 b_{(2i+1) \bmod (k+1)}^4 \ldots b_{k-1}^4 b_k^3 b_k^4 \\
& \,\&\, a : \\
& \quad b_i^1 = b_i^2 = b_i^3 = b_i^4 \text{ for } i = 0, \ldots, k \\
& \qquad \text{and all } b_i^j \text{ have format } \$x\$ \text{ where } x \in \{0,1\}^* \\
& \quad k \text{ is odd, and } a \in \{0,1\}^*, \;\}
\end{aligned}
$$

When we prove the lower bound, all the b_i^j will have the same length. The string between the first & and second & can be obtained by copying $b_0 b_1 \ldots b_k$ three times: $b_0 b_1 \ldots b_k \; \# b_0 b_1 \ldots b_k \; b_0 b_1 \ldots b_k$, and then adding one more copy of $b_0 b_1 \ldots b_k$ by inserting block b_i after $2i$ blocks, starting from $\# b_0$ in above. The superscripts on the b_i's are used only to facilitate later discussions. L can be considered as a modified version of a language used in [Ma]. We have added a string a on both ends. The purpose of a is to prevent the queue from shrinking, since if we choose a to be a long K-random string, then before the second a is read the size of the queue has to be at least about $|a|$. We have to prevent the queue from shrinking because otherwise the crossing sequence argument would not work. In addition to the techniques in [Ma, LV], we will need the techniques introduced in this paper to treat queues.

An alternate way to describe the language L is as follows. Let y and z be sequences of blocks, each block is of form $\$u\$$ where $u \in \{0,1\}^*$. Define $intermingle(y) = z$ if (1) the blocks of z in positions $i \equiv 2 \pmod 3$ form the string y ($z_2 z_5 z_8 ... = y_1 y_2 y_3 ...$), and (2) the remaining blocks of z form the string yy.

Then, $L = \{a \& y \# intermingle(y) \& a : y \text{ contains an even number of blocks}\}$.

Theorem 3.5 *Simulating a deterministic two queue machine with a 1-way input tape by a nondeterministic one queue machine with a 1-way input tape requires* $\Omega(n^2/\log^2 n \log\log n)$ *time.*

Proof. We will show below that L defined above requires $\Omega(n^2/\log^2 n \log\log n)$ time on a nondeterministic one-queue machine. Since L can be trivially accepted by a deterministic two-queue machine in linear time, the theorem will follow.

Now, aiming at a contradiction, assume that a one-queue machine M accepts L in time $T(n)$ which is not in $\Omega(n^2/\log^2 n \log\log n)$. Without loss of generality, we assume that M has a binary queue alphabet and that M accepts with a final state and an empty queue. We use the same notation and definitions as in the previous section such as $Queue$, $|Queue(t)|$, h_{in}, h_r, h_w, cycles and crossing sequence.

Choose a large n and a large enough C, such that $C >> |M| + c$ and all the subsequent formulas make sense, where $|M|$ is the number of bits needed to describe M and c is a constant given in Claim 3.9 below. Choose an incompressible string $X \in \{0,1\}^{2n}$, $K(X) \geq |X|$. Let $X = X'X''$ where $|X'| = |X''| = n$. Equally divide X'' into $k+1 = n/(C \log\log n)$ parts, $X'' = x_0 x_1 ... x_k$, where each x_i is $C \log\log n$ long. Consider a word $w \in L$ where $a = X'$ and $b_i^j = x_i$ for $1 \leq j \leq 4$ and $0 \leq i \leq k$. Fix a shortest accepting path, P, of M on w. We will show that M takes $\Omega(n^2/\log^2 n \log\log n)$ time on P. Since n is linearly related to the size of the input, this will provide the lower bound in the theorem[2].

Consider only the path P. Let $t_\&$ be the time when h_{in} reaches the first &, $t'_\&$ be the time h_{in} reaches the second &, and $t_\#$ be the time when h_{in} reaches #.

Claim 3.6 *$|Queue(t)| \geq n - O(\log n)$, for every $t_\& \leq t \leq t'_\&$.*

Proof. The proof of this claim is same to that of Claim 2.3. We will only sketch the idea. Assume the claim is false, i.e., $|Queue(t)| \geq n - g(n)$, with $g(n)$ not $O(\log n)$. Then we can conclude that $K(X) < |X|$ as follows. For every Y such that $|Y| = |X'|$, replace the second a ($a = X'$) after the second & sign in w by Y to form w'. Using the description of the queue, start to simulate M on w' from time $t_\&$. By a standard argument, $Y = X'$ ($= a$) iff M accepts. The information used in this simulation is self-delimiting descriptions of $|M|$, $t_\&$, $h_{in}(t_\&)$, $Queue(t_\&)$, and a literal description of X'' of size n. Therefore $K(X) < |X|$, a contradiction. □

Claim 3.7 *The number of cycles from time $t_\&$ to $t'_\&$ is less than $n/C^5 \log^2 n \log\log n$.*

Proof. This follows directly from the previous claim. Each cycle is of length $\Omega(n)$ and hence takes $\Omega(n)$ time. If M requires at least $n/C^5 \log^2 n \log\log n$ cycles from $t_\&$ to $t'_\&$, then M used $\Omega(n^2/\log^2 n \log\log n)$ time, contradiction. □

For each time t, we say that a substring s of the input w is *mapped into (onto)* a set Q of cells on $Queue(t)$ if all the cells influenced by s on $Queue(t)$ are (exactly those) in Q. Notice that Q is a contiguous region on $Queue(t)$.

[2]Here, like in the previous section, the language does not have a string of each length. The proof provides an input which causes the machine to take a long time for each length that has at least one string in the language. If we want a hard string for each length, just add a finite padding in the definition of the language, for example allowing markers to repeat up to 4 or 5 times

Claim 3.8 *Let $k' = k/2 - n/C^5 \log^2 n \log\log n$. At time $t_\#$, $Queue(t_\#)$ can be partitioned into two segments, $S_1(t_\#)$ and $S_2(t_\#)$, such that k' b_i^1's, say $b_{i_1}^1, ..., b_{i_{k'}}^1$, are mapped into $S_1(t_\#)$, and k' other b_i^1's, say $b_{j_1}^1, ..., b_{j_{k'}}^1$, are mapped into $S_2(t_\#)$.*

Proof. Consider any cell c_0 on the $Queue(t_\#)$. By the nature of the queue and Claim 3.7, at most $m = \frac{n}{C^5 \log^2 n \log\log n}$ b_i^1's can influence c_0 at $t_\#$ because M made no more than m cycles on the queue from $t_\&$ to $t_\#$. Hence for any partition of $Queue(t_\#)$ into two parts, $S_1(t_\#)$ and $S_2(t_\#)$, there can be at most m b_i^1 blocks each influencing both $S_1(t_\#)$ and $S_2(t_\#)$. Each of the rest of $k+1-m$ b_i^1 blocks either influences only $S_1(t_\#)$ or influences only $S_2(t_\#)$. It is now trivial to make $|S_1(t_\#)| - |S_2(t_\#)| \leq m$ to satisfy the claim. □

Look at the regions on $Queue(t_\&)$ that influence $S_1(t_\#)$ and $S_2(t_\#)$. These regions form two disjoint regions but these region may not form a partition of $Queue(t_\&)$. Partition $Queue(t_\&)$ into two regions $S_1(t_\&)$ and $S_2(t_\&)$, each exactly influencing $S_1(t_\#)$ and $S_2(t_\#)$ respectively. From now on, for $t_\& \leq t \leq t_\#$, we will always consider $Queue(t)$ to be partitioned as $S_1(t)$ and $S_2(t)$, where $S_1(t)$ is the region on $Queue(t)$ influenced by $S_1(t_\&)$ and $S_2(t)$ the one influenced by $S_2(t_\&)$.
When there is no ambiguity, we simply write S_1 and S_2 for $S_1(t)$ and $S_2(t)$. Notice that the sizes of $S_1(t)$ and $S_2(t)$ may change with t, but $S_1(t)$ and $S_2(t)$ will remain a partition of $Queue(t)$.

The next claim is a simple generalization of a theorem proved by Maass in [Ma] (Theorem 3.1). The proof of the claim is a simple reworking of the Maass' proof and hence omitted.

Claim 3.9 *Let S be a sequence of numbers from $0, ..., k$, where $k = 2^l$ for some l. Assume that every number $b \in \{0, ..., k\}$ is somewhere in S adjacent to the numbers $2b \pmod{k+1}$ and $2b \pmod{k+1}+1$. Then for every partition of $\{0, ..., k\}$ into two sets G and R such that $|G|, |R| > k/4$, there are at least $k/(c \log k)$ (for some fixed c) elements of G that occur somewhere in S adjacent to a number from R.* □

A $k/\sqrt{\log k}$ upper bound corresponding to the lower bound in this lemma is contained in [Li3]. A more general, but weaker, upper bound can be found in [Kl].

Remark 3.1 For each word $w \in L$, the sequence of the subscripts of the substrings (in the order they appear) in w between the # sign and the second & satisfies the requirements in Claim 3.9. For example, given k, such a sequence is formed by inserting i after $2i$th number, $i = 0, 1, ..., k$, in the following sequence,

$0, 1, 2, ...k, 0, 1, 2, ..., k$.

So each number i is adjacent to $2i \pmod{k+1}$, and $2i+1 \pmod{k+1}$. In what follows we will also say that a pair of b_i blocks are adjacent if their subscripts are adjacent in above sequence.

Claim 3.10 *At time $t'_{\&}$, the b_i's between # and the second & are mapped into $Queue(t'_{\&})$ in the following way: either*

1. *a set, $\bar{S}_1$, of $k/(3c\log k)$ b_j's, which belong to $\{b^1_{j_1},...,b^1_{j_{k'}}\}$, are mapped into $S_1(t'_{\&})$; or*
2. *a set, $\bar{S}_2$, of $k/(3c\log k)$ b_i's, which belong to $\{b^1_{i_1},...,b^1_{i_{k'}}\}$, are mapped into $S_2(t'_{\&})$,*

where $c << C$ is the small constant in Claim 3.9.

Proof. By Claim 3.7, from time $t_{\#}$ to time $t'_{\&}$, M makes fewer than $\frac{n}{C^5\log^2 n\log\log n}$ cycles. Hence, h_w can alternate between S_1 and S_2 fewer than $\frac{2n}{C^5\log^2 n\log\log n}$ times. Each time h_w alternates between S_1 and S_2, h_w can map at most one adjacent pair of b^j_i blocks into both $S_1(t'_{\&})$ and $S_2(t'_{\&})$. All other pairs are each mapped totally into $S_1(t'_{\&})$ or totally into $S_2(t'_{\&})$. There are $\theta(k)$ such pairs in L.

Combining Claims 3.8 and 3.9 and Remark 3.1 we know that there are at least $k/c\log k - \frac{n}{C^5\log^2 n\log\log n}$ pairs of b^j_i's such that each of these pairs contains a component belongs to $G = \{b^1_{i_1},...,b^1_{i_{k'}}\}$ and another component belongs to $R = \{b^1_{j_1},...,b^1_{j_{k'}}\}$. Most of these pairs, except $\frac{n}{C^5\log^2 n\log\log n}$ of them by the previous paragraph, are mapped either totally into $S_1(t'_{\&})$ or totally into $S_2(t'_{\&})$. Hence one of (1) or (2) must be true. □

Without loss of generality, assume that (1) of claim 3.10 is true.

Claim 3.11 *Let t_{end} be the time M accepts. $|Queue(t_{end})| = 0$. Then there exists a time $t'_{\&} \le t_0 \le t_{end}$ such that $|Queue(t_0)| \le \frac{n}{C^5\log n}$ and from $t'_{\&}$ to t_0 M made fewer than $\frac{n}{C^5\log n\log\log n}$ cycles.*

Proof. Otherwise M spends $\Omega(\frac{n^2}{\log^2 n\log\log n})$ time, a contradiction. □

By Claim 3.7 the number of cycles M made from $t_{\&}$ to $t'_{\&}$ is less than $\frac{n}{C^5\log^2 n\log\log n}$. And by above claim, M made at most $\frac{n}{C^5\log n\log\log n}$ cycles from time $t'_{\&}$ to t_0. Hence the length of the crossing sequence at the boundary of S_1 and S_2 from $t_{\&}$ to t_0 is shorter than $n/C^4\log n\log\log n$. For every j, if a $b^k_j \in \bar{S}_1$ for some k, then b^1_j is mapped into S_2 by Claim 3.10.

Now we describe a program that reconstructs X with less than $|X|$ information. Consider every Y such that $|Y| = |X|$ and $Y = a\ y_0...y_k$ for some $y_0...y_k$.

1. Check if Y is the same as X at positions other than those places occupied by $b^k_j \in \bar{S}_1$.

2. If (1) is true, then construct the input, w_Y, the same way w was constructed except with x_i replaced by y_i for $i = 0, 1, ..., k$.

3. Record the crossing sequences between S_1 and S_2 from $t_\&$ to t_0 of length less than $n/C^4 \log n \log\log n$. Also record the contents of S_2 at time t_0, which is shorter than $\frac{n}{C^5 \log n}$ by Claim 3.11. Simulate M from the beginning to $t_\&$ normally. Then simulate M from $t_\&$ to t_0 such that h_r never goes into S_2. Whenever h_r reaches the border of S_2 it compares the current ID with the corresponding one in the crossing sequence. If they match, then M *jumps* over S_2 and, using the next ID on the other side of S_2 to start from, M continues until time t_0. At time t_0, if above simulation is consistent, *i.e.* M's status matches the crossing sequence every time M reaches the boundary of S_1, we put the (short) contents of S_2 recorded beforehand back to the position of S_2 and simulate M from t_0 to the end in the normal way.

4. By the end M has an accepting path iff $Y = X$.

The information we used in this program is only the following.

1. $X - \bar{S}_1$, plus the information to describe the relative locations of $b_j^k \in \bar{S}_1$ in X. This would require at most

   $|X| - |\bar{S}_1||b_j^k| + O(|\bar{S}_1| \log(k/|\bar{S}_1|))$

   $\leq 2n - |\bar{S}_1| C \log\log n + O(|\bar{S}_1|(\log\log n + 2\log C))$

   $\leq 2n - (|\bar{S}_1| C \log\log n)/2$

   $\leq 2n - n/C^2 \log n,$

   where in the first line the second term is for the b_j's in $\bar{S}_1$, the third term is for the information to describe the relative positions of $b_j \in \bar{S}_1$: To represent $|\bar{S}_1|$ elements of $\{0, 1, ..., k\}$, sort the elements, determine the sequence of their differences, and use a self-delimiting encoding of the natural numbers to write each difference. The final encoding has approximately $O(|\bar{S}_1| \log(k/|\bar{S}_1|))$ bits. (See for example [LV,Lo,El]).

2. Description of the crossing sequence, of length less than $\frac{n}{C^4 \log n \log\log n}$, around S_2. Again by the above efficient encoding method, this requires at most $\frac{n}{C^3 \log n}$ bits. The detail of this encoding can be found in [LV]. The idea is as follows: Each item in the c.s. is (state of M, h_{in}'s position). Trivial encoding of $\frac{n}{C^4 \log n \log\log n}$ long c.s. needs $\frac{n}{C^4 \log\log n}$ bits. But we can use the above method and encode only the differences of h_{in}'s positions, thus use less than $\frac{n}{C^3 \log n}$ bits.

3. Description of the contents of S_2 at time t_0. But $|Queue(t_0)| \leq \frac{n}{C^5 \log n}$.

4. Extra $O(\log n)$ bits to describe the program discussed above.

The total is less than $2n - \frac{n}{C \log n}$. Therefore $K(X) < |X|$, a con tradiction. □

Corollary. Simulating two deterministic tapes by one nondeterministic queue requires $\Omega(n^2 / \log^2 n \log\log n)$.

Proof. Since L can also be accepted by a two tape Turing machine in linear time. □

Theorem 3.12 *To simulate two deterministic queues by one deterministic queue requires $\Omega(n^2)$ time.*

Proof Idea. Define a language L_1 as follows. (Below, $a, x_i, y_i \in \{0,1\}^*$.)

$$L_1 = \{a \,\&\, x_1\$x_2\$...\$x_k\#y_1\$...\$y_l\#(1^{i_1}, 1^{j_1})(1^{i_2}, 1^{j_2})...(1^{i_s}, 1^{j_s}) \,\&\, a \mid$$
$$x_p = y_q \,\&\, (p = i_1 + ... + i_t,\ q = j_1 + ... + j_t) \,\&\, 1 \leq t \leq s\}.$$

L_1 can be accepted by a deterministic two queue machine in linear time. Using the techniques in the above theorem and in the proof of one deterministic Turing machine tape requiring square time for this language (See [LV]), it can be shown that L_1 requires $\Omega(n^2)$ for a one queue deterministic machine. We omit the proof. □

Acknowledgement. We are grateful to the referee for has careful analysis and extensive comments on the first version of this paper.

References

[Aa] Aanderaa, S.O., "On k-tape versus (k-1)-tape real-time computation, in *Complexity of Computation,* ed. R.M. Karp, SIAM-AMS Proceedings, vol. 7, pp. 75-96, American Math. Society, Providence, R.I., 1974.

[BG] Book, R. and S. Greibach, "Quasi real-time languages," *Math. System Theory*, vol. 4, pp. 97-111, 1970.

[BGW] Book, R., S. Greibach, and B. Wegbreit, "Time- and tape-bound Turing acceptors and AFL's," *J. Computer and System Sciences*, vol. 4, pp. 606-621, 1970.

[Ch] Chaitin, G.J., "Algorithmic Information Theory," *IBM J. Res. Dev.*, vol. 21, pp. 350-359, 1977.

[DGPR] Duris, P., Z. Galil, W. Paul, and R. Reischuk, "Two nonlinear lower bounds for on-line computations," *Information and Control*, vol. 60, pp. 1-11, 1984.

[El] Elias, P., "Universal codeword sets and representation of integers," *IEEE Trans. Information Theory*, vol. IT-21, pp. 194-203, 1975.

[GKS] Galil, Z., R. Kannan, E. Szemeredi, "On nontrivial separators for k-page graphs and simulations by nondeterministic one-tape Turing machines," in Proceedings 18th Annual ACM Symposium on Theory of Computing, pp. 39-49, 1986.

[HM] Hood, R and R. Melville, "Real-time queue operations in pure LISP," *Information Processing Letters*, vol. 13, pp. 50-54, 1981.

[HS1] Hartmanis, J. and R.E. Stearns, "On the computational complexity of algorithms," *Trans. Amer. Math. Soc.*, vol. 117, pp. 285-306, 1969.

[HS2] Hennie, F.C. and R.E. Stearns, "Two tape simulation of multitape Turing machines," *J. Ass. Comp. Mach.*, vol. 4, pp. 533-546, 1966.

[HU] Hopcroft, J.E. and J.D. Ullman, *Formal Languages and their Relations to Automata*, Addison-Wesley, 1969.

[Kl] Klawe, M., "Limitations on explicit construction of expanding graphs," *SIAM J. Comp.*, vol. 13, no. 4, pp. 156-166, 1984.

[Ko] Kolmogorov, A.N., "Three approaches to the quantitative definition of information, *Problems in Information Transmission*, vol. 1, no. 1, pp. 1-7, 1965.

[Kos] Kosaraju S., "Real time simulation of concatenable double-ended queues by double-ended queues," 11th Annual ACM Symposium on Theory of Computing, pp. 346-351, 1979.

[Li1] Li, M., "Simulating two pushdowns by one tape in O(n**1.5 (log n)**0.5) time," 26th Annual IEEE Symposium on the Foundations of Computer Science, pp. 56-64, 1985.

[Li2] Li, M., "Lower Bounds in Computational Complexity," Ph.D. Thesis, Report TR-85-663, Computer Science Department, Cornell University, march 1985.

[Li3] Li, M., "Lower bounds by Kolmogorov-complexity", 12th ICALP, Lecture Notes in Computer Science, 194, pp. 383-393, 1985.

[LV] Li, M. and P.M.B. Vitányi, "Tape versus queue and stacks: The lower bounds," *Information and Computation* Vol. 77, 1988.

[L0] Loui, M.C., "The complexity of sorting on distributed systems," *Information and Control* Vol. 60, pp. 70-85, 1984.

[LS] Leong, B.L. and J.I. Seiferas, "New real-time simulations of multihead tape units," *J. Ass. Comp. Mach.*, vol. 28, pp. 166-180, 1981.

[Ma] Maass, W., "Combinatorial lower bound arguments for deterministic and nondeterministic Turing machines," *Trans. Amer. Math. Soc.*, 292,2, pp. 675-693, 1985.

[MSZ] Maass, W., G. Schnitger, and E.Szemeredi, "Two tapes are better than one for off-line Turing machines," Proceedings 19th ACM Symposium on Theory of Computing, pp. 94-100, 1987.)

[PSS] Paul, W.J., J.I. Seiferas, and J. Simon, "An information theoretic approach to time bounds for on-line computation, *J. Computer and System Sciences*, vol. 23, pp. 108-126, 1981.

[Pa] Paul, W.J., "On-line simulation of k+1 tapes by k tapes requires nonlinear time," *Information and Control*, pp. 1-8, 1982.

[So] Solomonoff, R., "A formal theory of inductive inference, Part 1 and Part 2," *Information and Control*, vol. 7, pp. 1-22,224-254, 1964.

[Vi1] Vitányi, P.M.B., "One queue or two pushdown stores take square time on a one-head tape unit," Computer Science Technical Report CS-R8406, CWI, Amsterdam, March 1984.

[Vi2] Vitányi, P.M.B., "An N**1.618 lower bound on the time to simulate one queue or two pushdown stores by one tape," *Information Processing Letters,* vol.

21, pp. 147-152, 1985.

[Vi3] Vitányi, P.M.B., "On two-tape real-time computation and queues," *J. Computer and System Sciences*, vol. 29, pp. 303-311, 1984.