Pebbling Mountain Ranges and its Application to DCFL-Recognition *

by Kurt Mehlhorn **

Abstract: Recently, S.A. Cook showed that DCFL's can be recognized in $O((\log n)^2)$ space and polynomial time simultaneously. We study the problem of pebbling mountain ranges (= the height of the pushdown-store as a function of time) and describe a family of pebbling strategies. One such pebbling strategy achieves a simultaneous $O((\log n)^2/\log\log n)$ space and polynomial time bound for pebbling mountain ranges. We apply our results to DCFL recognition and show that the languages of input-driven DPDA's can be recognized in space $O((\log n)^2/\log\log n)$. For general DCFL's we obtain a parameterized family of recognition algorithms realizing various simultaneous space and time bounds. In particular, DCFL's can be recognized in space $O((\log n)^2)$ and time $O(n^{2.87})$ or space $O(\sqrt{n}\log n)$ and time $O(n^{1.5}\log\log n)$ or space $O(n/\log n)$ and time $O(n(\log n)^3)$. More generally, our methods exhibit a general space-time tradeoff for manipulating pushdownstores (e.g. run time stack in block structured programming languages).

I. Introduction

Recently, S.A. Cook showed how to recognize DCFL's in $(\log n)^2$ space and polynomial time simultaneously. The proof is an ingenious application of the pebble game.

Consider the height of the pushdown store as a function of time, a mountain range. In order to simulate the move of a DPDA at time t one needs to know the state (which comes from time t-1) and the top pushdown symbol (which either comes from t-1 or from t' where t' is the last node with height(t') = height(t)). (cf. Fig. 1). This is in complete analogy to the pebble game: a pebble may be put on a node if all predecessors hold pebbles.

Of course, the mountain range is not given as an input. Rather, Cook's simulation consists of two coroutines: Pebbling a mountain range and constructing a mountain range.

---

* Full version of paper is available from author

** FB 10, Universität des Saarlandes, 6600 Saarbrücken, West Germany

Nevertheless, in section I of this paper we concentrate on the first aspect only: pebbling mountain ranges. This will allow us to considerably simplify Cook's construction on the one hand and to extend his results on the other hand.

Definition: A mountain range of length n is a directed graph $G = (V,E)$ with

$V = \{0,1,\ldots,n-1\}$ and a function height: $V \to \mathbb{N} \cup \{0\}$ with

a) $height(x) > 0$ for all $x > 0$, $height(0) = 0$ and

$|height(x) - height(x-1)| \leq 1$.

b) $E = \{(x,x+1);\ 0 \leq x < n\} \ \cup$

$\{(x,y);\ x < y \text{ and } height(x) = height(y) < height(z)$ for all $x < z < y\}$

Definition: If $height(x-1) \leq height(x)$ then $x-1$ is the left neighbor of x, otherwise the unique y with $(y,x) \in E$ and $height(x) = height(y)$ is the left neighbor of x. Also x is the right neighbor of y in this case.

If x,y are nodes then x is visible from y if $x \leq y$ and $height(x) < height(z)$ for all z with $x < z \leq y$.

We assume that a mountain range is given by the sequence

$\{h(x) - h(x-1)\}_{x=1}^{n-1} \in \{-1,0,1\}^{n-1}$.

Our approach to pebbling mountain ranges is divide and conquer. (So is Cook's but in a disguised form). We feel that our approach is simpler. We describe a family (parameterized by function f) of pebbling strategies (f describes the division of a mountain range into subranges (e.g. a mountain range of length n could be divided in two pieces, or log n pieces or n pieces,...)),and analyse its space and time requirements. For one particular choice of f we obtain the following theorem :

Thm.: Mountain Ranges can be pebbled in space $O((\log n)^2/\log\log n)$ and polynomial time simultaneously.

It is easy to see that some mountain ranges require $\Omega(\log n)$ pebbles. Hence our strategy requires only $o(\log n)$ tape for recording positions of pebbles. Of course, this can only be achieved if pebbles are placed in a very regular fashion. This supports our intuitive feeling that our approach is simpler than Cook's.

In section II we apply our results to DCFL recognition. We first observe that for input-driven dpda's (= real time + input symbol determines the type of the move (i.e. push or pop)), Thm. 1 give the corresponding bounds for DCFL recognition.

Thm. 2: Let M be an input-driven dpda. Then L(M), the language accepted by M, can be recognized in space $O((\log n)^2/\log\log n)$ and polynomial time.

Then we consider general DCFL's. We derive a parameterized class of recognition algorithms for DCFL's, realizing different simultaneous space, time bounds for DCFL-recognition. In particular, we show that DCFL's can be recognized in

space	and	time	
$O((\log n)^2)$		$O(n^{2.87})$	(Cook)
$O(\sqrt{n}\cdot\log n)$		$O(n^{1.5}\log\log n)$	
$O(n/\log n)$		$O(n\cdot(\log n)^3)$	

simultaneously. This establishes a general time/space trade-off for DCFL-recognition. More generally, our methods are applicable to any deterministic manipulation of pushdownstores, e.g. run time stack in block structured programming languages (B. Schmidt, Swamy/Savage, Gurari/Ibarra).

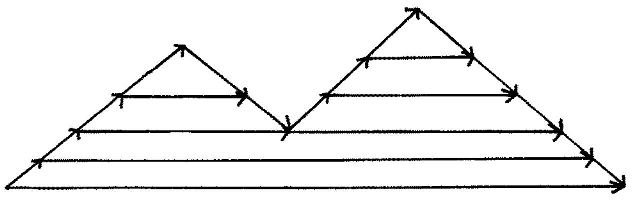
Figure 1: A mountain range.

Acknowledgement: Discussions with Bernd Schmidt are gratefully acknowledged.

I. The Algorithm

Our approach to pebbling mountain ranges is divide and conquer. The division is guided by function f.

Let $f : \mathbb{N} \cup \{0\} \to \mathbb{N}$ be any function with

1) $f(0) = 1$
2) $f(d+1) \geq 2f(d)$; in particular $f(d) \geq 2^d$
3) the binary representation of f(d) can be computed in space log f(d) (and hence in time p(f(d)) for some polynomial p) given the binary representation of d.

A d-order (d is the number of levels of the divide-and-conquer approach) strategy can be applied to a mountain range M of length m if $m \leq f(d+1)$. Let $v, v+1, \ldots, v+m-1$ be the nodes of M. The purpose of the strategy is to move a pebble on node $v+m-1$.

Any strategy uses two kinds of pebbles: recursion pebbles and ordinary pebbles. A d-order strategy uses d recursion pebbles, one for each d', $1 \leq d' \leq d$. Furthermore, for each d', $0 \leq d' \leq d$, it uses $\lceil f(d'+1)/f(d') \rceil$ ordinary pebbles of type d'. These pebbles are denoted by (d',0), (d',1),... . The second component of a pebble is called its index, the first component is its type.

We assume that a pebble of some type > d is on v initially. It will stay on v during the entire game.

A d-order strategy is played as follows. If $m \leq f(d)$ then it is identical to the (d-1)-order strategy on M. If d = 0, i.e. $m \leq f(1)$ then we put pebble (0,i) on node v+i one after the other. This will take m moves. Assume now that $d \geq 1$ and $m > f(d)$. Then range M is divided into subranges of length f(d). More precisely, the j-th subrange consists of points $v+j \cdot f(d), \ldots, v+\min(m-1, (j+1) \cdot f(d)-1)$. Then $0 \leq j \leq \lceil m/f(d) \rceil - 1 \leq \lceil f(d+1)/f(d) \rceil - 1$.

The game is started by putting pebble (d,0) on node v. (Note that index 0 pebbles are always put on nodes which have already a pebble).

Assume now that we just placed pebble (d,j) on the first node of the j-th subrange, $j \geq 0$. Let $i = v+j \cdot f(d)$ be that node. Then the following statement holds (later refered to as Invariant).

a) let $j' \leq j$. If the j'-th subrange contains a point visible from i then pebble (d,j') is on the leftmost visible point in that subrange.

b) if a pebble is on node u, $u \leq i$, then u is visible from i. Furthermore, let w be any point visible from i and let d', $0 \leq d' \leq d$, be arbitrary. Let $u \leq w$ be the rightmost node holding a pebble of type d', if any. Then $w < u+f(d')$.

Remark: Our invariant captures the following idea : a pebble of type d' supports the exploration of a subrange of length at most $f(d')$. If w is visible and u is the rightmost point $\leq w$ holding a pebble then the pebble on u supports w.

□

Consider the j-th subrange in more detail. Let the sequence $v_o, v_1, \ldots, v_k$ of points in the j-th subrange be defined by

a) v_o is the first point in the j-th subrange

b) $v_{\ell+1} = \min\{w;\ v_\ell < w \text{ and height } (v_\ell) \geq \text{height}(w)\}$

Furthermore, v_{k+1} is the first point in the (j+1)-th subrange. Then either height $(v_\ell) \leq$ height $(v_{\ell+1})$ (= for $\ell < k$) or height $(v_{\ell+1}) =$ height$(v_\ell)-1$ and $v_{\ell+1} = v_\ell + 1$. (cf. Fig. 2).

The j-th subrange is played as follows.

```
for $\ell = 0$ to k do
begin co pebble (d,j) is on node $v_\ell$ and the invariant holds with $i = v_\ell$;
      if height $(v_\ell) \leq$ height $(v_{\ell+1})$
      then let d' be minimal with $v_\ell + f(d') \geq v_{\ell+1}$ ;
           co then $d' \leq d$;
           remove all pebbles of type $\leq d' - 1$ from the graph;
(A)        play (d'-1)-order strategy on mountain $v_\ell, v_\ell+1, \ldots, v_{\ell+1}-1$;
           co at this point pebble (d,j) is still on $v_\ell$ and there is
           a pebble on $v_{\ell+1}-1$;
      fi;
      replace the pebble on node $v_{\ell+1}-1$ by the recursion pebble of type d
      let t' be the left neighbor of $v_{\ell+1}$;
      let $u \leq t'$ be the rightmost node which holds a pebble, let d' be
      the type of the pebble on u;
```

```
    co by part b) of the Invariant $u+f(d') \geq t'$;
    let $d''$ be minimal such that $u+f(d'') \geq t'$;
    remove all pebbles of type $\leq d''-1$ from the graph;
(B)     play $(d''-1)$-order strategy on $u,\dots,t'$;
(C)     move pebble $(d,j)$ ($(d,j+1)$) on node $v_{\ell+1}$ if $\ell < k$ ($\ell=k$), and
        remove pebbles from points which are not visible from $v_{\ell+1}$;
        co the invariant holds with $i = v_{\ell+1}$
end
```

This finishes the description of a d-order strategy given by function f. On an arbitrary mountain M we will always play a d-order strategy with the smallest possible d, i.e. $f(d) <$ length $M \leq f(d+1)$.

Definition: Let f be any function as described above, let $m \in \mathbb{N}$. Define

$$d_o(m,f) := \min\{d;\ m \leq f(d+1)\}$$

$$N(m,f) := \sum_{d=0}^{d_o(m,f)} (1 + \lceil f(d+1)/f(d) \rceil)$$

□

Remark: $N(m,f) = \Omega(\log m)$ for all functions f.

Lemma 1: Let f be any function with $f(0) = 1$ and $f(d+1) \geq 2 \cdot f(d)$ for $d \geq 0$.

a) Let M be any mountain with m nodes. Then our strategy uses at most $N(m,f)$ pebbles on M.

b) An $O(N(m,f) \cdot \log m)$ space bounded Turing machine can play our strategy.

c) An $O(N(m,f) \cdot \log N(m,f) + d_o(m,f) \cdot \log m)$ space bounded TM can play our strategy.

Proof: part a) is an immediate consequence of the description of our strategy. Part b) follows from a) and the observation that space $O(\log m)$ suffices to record the position of a pebble. A proof of c) can be found in the full paper. It uses the fact that ordinary pebbles are placed in a very regular fashion and that it suffices to know the order in which the pebbles appear on the mountain from left to right in order to be able to compute their positions.

□

Next we turn to the timing analysis. We will first derive a bound on the number of moves.

Lemma 2: Let f be any function as described above. Let T(m,d) be the maximal number of moves in a d-order strategy on a mountain M of length m ($m \le f(d+1)$). Then

$$T(1,d) = 0 \qquad\qquad T(m,0) = m$$

$$\begin{aligned} T(m,d) \le\ & m + T_A + T_B + T_C + \\ & (\lceil m/f(d)\rceil - 1)\ T(f(d-1),d-2) + \\ & (\lceil m/f(d)\rceil - 2)\ T(f(d),d-1) \end{aligned}$$

where

$$T_A = \max\left\{ \sum_{i=0}^{k} T(y_i,e_i);\quad \begin{array}{l} 0 \le e_i \le d-1, \\ f(e_i) < y_i \le f(e_i+1) \\ y_0+\ldots+y_k \le m \end{array} \right\}$$

$$T_B = \max\left\{ \sum_{i=0}^{k'} T(f(h_i+1),h_i);\ 0 \le h_i \le d-1, \atop f(h_0)+\ldots+f(h_{k'}) \le m \right\}$$

$$T_C = \sum_{g=1}^{d-1} \left(\frac{\lceil m\rceil}{f(d)} - 1\right) \frac{\lceil f(d)\rceil}{f(d-1)} \cdots \frac{\lceil f(g+1)\rceil}{f(g)} \cdot T(f(g),g-1)$$

Proof: Let M be a mountain range of m points $v, v+1, \ldots, v+m-1$, $m \le f(d+1)$, such that a d-order strategy on M uses a maximal number of moves. If m = 1 then no move is required and hence T(1,d) = 0. If d = 0 then the number of moves is bounded by m, hence $T(m,0) \le m$.

Suppose now that $d \ge 1$. Let $x_0, x_1, \ldots, x_k$ be the set of points which receive pebbles of type d, let $x_{k+1} = v+m$. Then $x_0 = v$.

Consider the description of our d-order strategy. We will count the moves in lines A, B and C of the algorithm separately.

line C: the number of moves in line C is certainly bounded by the number m of nodes of mountain M.

line A: in line A the games on subranges $x_i,\dots,x_{i+1}-1$ are played, $0 \le i \le k$; say an e_i order strategy is used. Then $f(e_i) < x_{i+1} - x_i \le f(e_i+1)$. The cost of an e_i-strategy on $x_i,\dots,x_{i+1}-1$ is at most $T(y_i,e_i)$ where $y_i := x_{i+1}-x_i$. Hence the total cost arising in line A is at most T_A where T_A is defined as above.

line B: For i, $0 \le i \le k$, let index (i) be the index of the type d pebble which was used on x_i. Then index (0) = 0 and index (1) = 1 since pebble (d,0) is only used on v. Furthermore, index(i) $\le$ index(i+1) and index(i) = index(i+1) implies height(x_i) $\ge$ height(x_{i+1}).

Let t_i be the left neighbor of x_i. Of course, a t_i is either identical to one of the x's (and then repebbling t_i in line B is free) or t_i lies properly between two x's. So let

$$Q = \{i; 1 \le i \le k,\ t_i \text{ is not one of the x's}\}$$

and for $i \in Q$ let int(i) be such that $x_{int(i)-1} < t_i < x_{int(i)}$. Then int(i) $\le$ i, height($x_{int(i)-1}$) < height($x_{int(i)}$) and hence index(int(i)-1) < index(int(i)). Furthermore, for $i \in Q$ let

$$\text{left}(i) = \begin{cases} \max\{\ell; \ell < i \text{ and } \text{int}(\ell) = \text{int}(i)\} & \text{if such an } \ell \text{ exists} \\ \text{undefined} & \text{otherwise} \end{cases}$$

(cf. Figure 3). Note that left(i) undefined is equivalent to int(i) = i and that left(i) defined implies height($x_{left(i)}$)-1 = height (x_i). Note further that left is injective on the points on which it is defined.

The total cost of line B is

$$\sum_{i \in Q} \text{cost of repebbling } t_i \text{ in line B.}$$

We will split this cost in four parts.

part 1: Let Q_1 = {i; $i \in Q$ and left(i) undefined}. Then int(i) = i by the remark above. From index (int(i)-1) < index(int(i)) we conclude $Q_1 \le \lceil m/f(d) \rceil - 1$. Furthermore, t_i needs to be repebbled just after we finished pebbling the interval $x_{int(i)-1},\dots,x_{int(i)}-1$.

Hence the cost of repebbling t_i is bounded by T(f(d-1),d-2). This shows

$$\sum_{i \in Q_1} \text{cost of repebbling } t_i \leq (\lceil \frac{m}{f(d)} \rceil - 1) \cdot T(f(d-1),d-2)$$

<u>part 2:</u> Let $Q_2 \in \{i; i \in Q$ and left(i) defined and index(left(i)) $\neq$ index(i)$\}$. Since the cost of repebbling t_i is certainly bounded by the cost of pebbling the entire interval $x_{int(i)-1}, \ldots, x_{int(i)}-1$, and this in turn is bounded by T(f(d),d-1), we have

$$\sum_{i \in Q_2} \text{cost of repebbling } t_2 \leq |Q_2| \cdot T(f(d),d-1)$$

<u>claim 1:</u> $|Q_2| \leq \lceil m/f(d) \rceil - 2$

<u>proof:</u> It suffices to show $i_1, i_2 \in Q_2$, $i_1 \neq i_2$ implies index(left(i_1)) $\neq$ index(left(i_2)). (Note that C < index(left(i)) $\leq$ $\lceil m/f(d) \rceil - 2$ for $i \in Q_2$).So assume i_1, $i_2 \in Q_2$, $i_1 \neq i_2$. Since left is injective on the points on which it is defined we may assume w.l.o.g. that left(i_1) < left(i_2). If $i_1 \leq$ left(i_2) then we are done. So suppose left(i_2) < i_1. Since height($x_{left(i_1)}$) = height(x_{i_1}) + 1 $\leq$ height(j) for all j with $x_{left(i_1)} \leq j \leq x_{i_1} - 1$ (this follows from int(left(i_1)) = int(i_1)) and height($x_{left(i_2)}$) = height(x_{i_2}) + 1 we conclude $i_2 < i_1$. Hence height(left(i_1)) < height(left(i_2)) and thus index(left(i_1)) < index(left(i_2)). This proves the claim. □

Using claim 1 we get

$$\sum_{i \in Q_2} \text{cost of repebbling } Q_2 \leq (\lceil m/f(d) \rceil - 2) \cdot T(f(d),d-1)$$

<u>parts 3 and 4:</u> Let $Q_3 = \{i; i \in Q$ and left(i) defined and index(left(i)) = index(i)$\}$. Let h_i be the maximal type ($\neq$ d) of pebble used in going from $x_{left(i)}$ to $x_i - 1$, i.e. $h_i = \max(e_{left(i)}, \ldots, e_{i-1})$. Let g_i be the type of pebble which was used on $t_{left(i)}$, i.e. pebble (g_i, r), r > 0, was used on $t_{left(i)}$. Then $g_i \leq d-1$.

<u>claim 2:</u> A $\max(g_i-1,h_i)$-order strategy suffices to repebble t_i.

<u>proof:</u> When $x_{left(i)}$ was pebbled there was pebble (g_i,r) on $t_{left(i)}$. Let T_o be moment of time when pebble (g_i,r) was put on $t_{left(i)}$. At time T_o pebbles $(g_i,0)$, $(g_i+1,0),\dots,(e_{int(i)-1},0)$ where also on the interval $x_{int(i)-1},\dots,t_{left(i)}$. Since t_i is the rightmost point which is visible from $t_{left(i)}$, all these pebbles are to the left or at t_i. Hence at time T_o a (g_i-1)-order strategy would suffice to pebble node t_i, This is still true when $x_{left(i)}$ gets its pebble since (g_i,r) is still on $t_{left(i)}$ at that moment of time. When we proceed from $x_{left(i)}$ to x_i-1 all pebbles of type $> h_i$ stay where they are. Hence a $\max(g_i-1,h_i)$-order strategy suffices to play t_i. This proves claim 2.

□

Let $Q_{31} = \{i;i \in Q_3 \text{ and } g_i-1 \leq h_i\}$ and let $Q_{32} = \{i;i \in Q_3 \text{ and } g_i-1 > h_i\}$

<u>claim 3:</u> $\sum_{i\in Q_{31}}$ cost of repebbling $t_i \leq T_B$

where T_B is defined as above.

<u>proof:</u> Since a pebble of type h_i is used in going from $x_{left(i)}$ to x_i-1 we have $x_i-x_{left(i)} > f(h_i)$ Hence claim 3 follows immediately from claim 4.

<u>claim 4:</u> Let $i_1, i_2 \in Q_3$, $i_1 \neq i_2$. Then the intervals $x_{left(i)},\dots,x_{i_1}-1$ and $x_{left(i_2)},\dots,x_{i_2}-1$ are disjoint.

<u>proof:</u> Assume $i_1,i_2 \in Q_3$, $i_1 \neq i_2$. Since $left(i_1)$ and $left(i_2)$ are defined we may assume w.l.o.g. that $left(i_1) < left(i_2)$. If $i_1 < left(i_2)$ then we are done. So suppose $left(i_2) < i_1$. As in the proof of claim 1 we conclude $index(left(i_1)) < index(left(i_2))$. But $left(i_2) < i_1$ implies $index(left(i_2)) \leq index(i_1)$. Hence $index(left(i_1)) < index(i_1)$ which contradicts $i_1 \in Q_3$. This proves claim 4 and 3.

□ □

<u>claim 5:</u> $\sum_{i\in Q_{32}}$ cost of repebbling $t_i \leq T_C$

where T_C is as defined above.

proof: If $i \in Q_{32}$ then the cost of repebbling t_i is bounded by $T(f(g_i),g_i-1)$ where (g_i,r) is the pebble used on $t_{left(i)}$ $(r > 0)$. For g, $1 \leq g \leq d$, let $u(g) = |\{i \in Q_3 \text{ and } g_i = g\}|$. Then $\sum_{i \in Q_{32}}$ cost of repebbling $t_i \leq \sum_{g=1}^{d-1} u(g)T(f(g),g-1)$ since $g_i \leq d-1$ always. An induction argument can be used to show $u(g) \leq (\lceil m/f(d) \rceil -1) \cdot \lceil f(d)/f(d-1) \rceil \ldots \lceil f(g+1)/f(g) \rceil$.

□

Putting everything together we obtain

$$T(m,d) \leq m + T_A + T_B + T_C + \left(\left\lceil \frac{m}{f(d)} \right\rceil - 1\right) T(f(d-1),d-2)$$

$$+ \left(\left\lceil \frac{m}{f(d)} \right\rceil - 2\right) T(f(d),d-1)$$

□

Lemma 3: $T(m,d) \leq m \cdot \prod_{g=1}^{d} [3 + f(g)/f(g-1)]$

proof: by induction on d; we refer the reader to the full paper.

□

Better bounds can be obtained for specific functions f; e.g. for $f = 2^d$, one obtains $T(m,d) \leq m^{2.86}$.

Theorem 1: Mountain Ranges can be pebbled in space $O((\log n)^2/\log \log n)$ and polynomial time simultaneously.

proof: Use $f(d) = d!$. The space bound follows from Lemma 1, part c, the time bound follows from lemma 3 and the observation that a TM can simulate one move of the pebble game in polynomial time.

□

Applications: The methods of this paper show a general space, time tradeoff in manipulating pushdownstores; in particular they are applicable to space and time efficient realizations of run time stacks in block structured programming languages (cf. B. Schmidt, Swamy/Savage, Gurari/Ibarra) and to the simulation of deterministic pushdown automata. In this section we will sketch very briefly the application to DCFL-recognition.

Definition: A deterministic pushdown automaton is input-driven if the input symbol determines the type (push, pop, change of top pushdown symbol) of the move.

Theorem 2: Let N be an input driven dpda. Then L(N), the language accepted by N, can be recognized in space $O((\log n)^2/\log n)$ and polynomial time simultaneously on a multitape TM.

proof: Consider the height of the pushdown store as a function of time; this defines a mountain range. Store in each pebble state and top pushdown symbol. Then apply Theorem 1. □

For input driven dpda's the input string encodes a mountain range in a natural way. In the general case, one has to store in each pebble its position (the time of the move), the position of the largest visible node, a pointer to the current input symbol, state and top pushdown symbol. This will require space $O(\log n)$ per pebble.

Theorem 3: Let N be a dpda. Then L(N) can be recognized in

	space	time	
a)	$O((\log n)^2)$	$O(n^{2.87})$	[Cook]
b)	$O(\sqrt{n} \log n)$	$O(n^{1.5} \log \log n)$	
c)	$O(n/\log n)$	$O(n(\log n)^3)$	

simultaneously on a unit cost RAM.

proof: we sketch a proof of b). Let c be such that N uses at most cn moves on any input of length n. Let $h(m) = \lfloor\sqrt{m}\rfloor$. Let d_o be minimal such that $h^{(d_o+1)}(cn+1) = 1$. Then $d_o = O(\log \log n)$. Let $f(i) = h^{(d_o-i+1)}(cn+1)$. Then $N(n,f) = O(\sqrt{n})$. This gives the space bound.

For the time bound use Lemma 3 and observe that a unit cost RAM can simulate one move of the pebbling strategy in time $O(d_0)$. For part c) use $h(m) = (\log m)^2$.

□

References:

S.A. Cook : Deterministic CFL's are accepted simultaneously in polynomial time and log squared tape, 11th ACM Symposium on Theory of Computing, 1971, 338-345

E.M. Gurari, O.H. Ibarra : On the space complexity of recursive algorithm, Inf. Proc. Letters, Vol. 8, No. 5, 1979, 267-272

B. Schmidt : Ph.D. Thesis, Universität des Saarlandes, Fachbereich 10, 6600 Saarbrücken, in preparation

S. Swami, J. Savage : Space-Time Tradeoffs for linear Recursion, 6th ACM Symposium on Principles of Programming Languages, 1979, 135-142

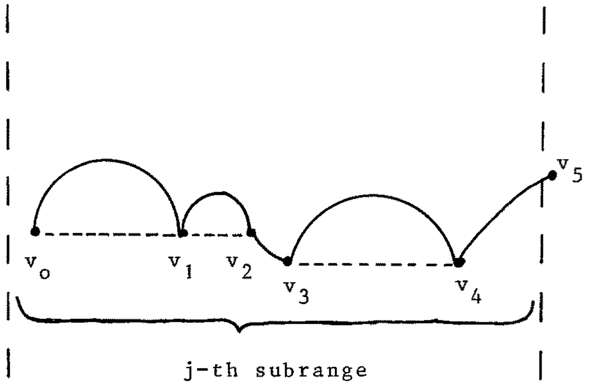

Figure 2: a subrange and its v-nodes

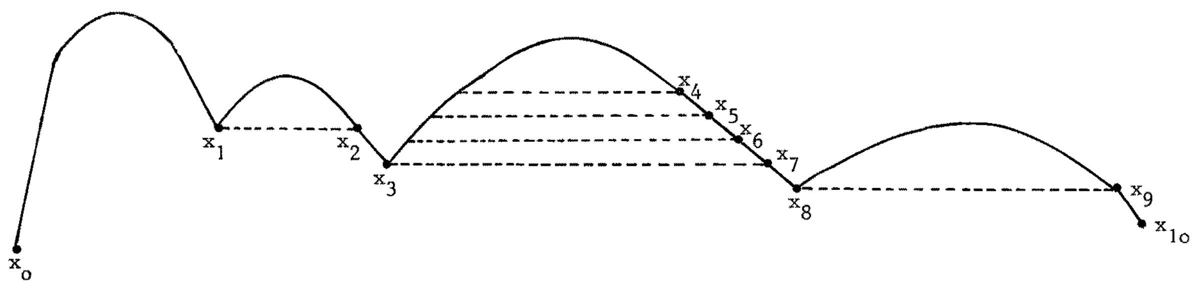

Figure 3: Assumption: index(0) = 0, index(1) = ... = index(3) = 1, index(4) = ... = index(10) = 2

Then Q = {1,3,4,5,6,7,8,10} int(1) = int(3) = int(8) = int(10) = 1, int(4) = int(5) = int(6) = 4, left(1) = left(4) = undef., left(5) = 4, left(6) = 5, left(3) = 1, left(8) = 3, left(10) = 8; Q_1 = {1,4}, Q_2 = {8}, Q_3 = {3,5,6,10}

Further assumptions: $c_1 = 5$, $e_8 = 8$; $g_3 = 6$, $g_{10} = 4$, $g_5 = g_6 = 1$.

Then Q_{31} = {3,10} , Q_{32} = {5,6}