Remarks on the pyramidal structure

Wojciech Rytter,
Institute of Informatics, Warsaw University,
PKiN VIII p, 00-901 Warszawa, Poland

1. Introduction

The pyramidal structure was invented by A.Schonhage [7] as a special structure of the memory of a storage modification machine simulating a Turing machine. In this paper we modify the original Schonhage's construction and we present the pyramidal structure in a more general setting as a data structure which implements efficiently some special sequences of the operations of storing and retrieving an information. The keys $\bar{x}$ are k-tuples of nonnegative integers. We introduce the operation find($\bar{x}$,v), which generalizes the dictionary operations member and insert. The value of find($\bar{x}$,v) is a node (or a record consisting of some consecutive registers of RAM) with the key $\bar{x}$, if there is no such a node then a new node with the key $\bar{x}$ is created and the value of find is set to the created node. The second argument is a node v (called a finger or a point of reference) and we can start searching $\bar{x}$ from v. The pyramidal structure implements in linear time and space sequences of operations find($\bar{x}$,v) such that the distance between $\bar{x}$ and the key of the finger is bounded. The number of fingers is unbounded (while in the Shonhage's construction is only one finger) The distance between keys is defined as follows. If $\bar{x}=(x_1,\ldots,x_k)$ and $\bar{y}=(y_1,\ldots,y_k)$ then

$$\text{dist}(x,y) = \sum_{i=1}^{k} |x_i-y_i| .$$

The distance is not defined in the terms of a linear order, hence we do not use sorted lists or any other method based on the linear order. We use a variation of digital search trees instead. Our model of the computation is a random access machine with the uniform cost criterion. The pyramidal structure can be applied to maintain efficiently sparse tables. We consider two-way deterministic pushdown automata with k input heads (2dpda(k)'s, for short). We simulate a given 2dpda(k) by a recursive program and then we apply the exact tabulation method (this gives a new simple $O(n^k)$ time simulation of 2dpda(k) disregarding the pyramidal structure).

Using the pyramidal structure we simulate 2dpda(k)'s in O(m) time and space, where m is the number of reachable surface configurations. If the computation of a 2dpda(k) is "sparse" then m is an improvement upon n^k. This generalizes the result of [4]. We obtain a similar result for nondeterministic two-way pushdown automata. The pyramidal structure can also serve to traverse efficiently a labyrinth whose nodes (which are integer points) are not given explicitely [6].

2. The pyramidal structure

The universe of possible keys is the set $U = [0..n]^k$ of k-tuples $\bar{x}$ of integers from the interval $[0..n]$. For $\bar{x}=(x_1,\dots,x_k)$ we define compress($\bar{x}$) = $(\lfloor x_1/2\rfloor,\dots,\lfloor x_k/2\rfloor)$. (If x_i are written in binary then compress($\bar{x}$) results by deleting the last digits of x_i's.) Let $r=\lceil \log_2(n+1)\rceil$. We say that the set S of keys is connected iff for every two elements $\bar{x},\bar{y} \in S$ there is a path $\bar{x}=\bar{x}_1,\bar{x}_2,\dots,\bar{x}_s=\bar{y}$ such that $\bar{x}_i \in S$ and $\mathrm{dist}(\bar{x}_i,\bar{x}_{i+1}) \leq 1$ for each $1 \leq i < s$. Let S be a connected subset of U.

The pyramidal structure P over the set S is the tree named pyramid (S) with the set V of nodes satisfying the conditions:

1) The keys are assigned to every node of V, key(v) is the key assigned to the node v;
2) The depth of all leaves is equal to r;
3) For every $\bar{x} \in S$ there is a leaf with the key x;
4) If the nodes v,w lying on the same level (having the same depth) have the same key then v=w (the nodes on a given level are identified by their keys);
5) If the node v is the father of w (we write v=father(w)) then key(v)=compress(key(w));
6) With every node v there are associated informations sons(v) and neighbours(v) such that
   sons(v) = $\{w \in V \mid \mathrm{father}(w)=v\}$ (v is not a leaf),
   neighbours(v) = $\{w \in V \mid$ w lies on the same level as v and $\mathrm{dist}(\mathrm{key}(v),\mathrm{key}(w)) \leq 1\}$.

Observe that $v \in$ neighbours(v) and observe that the condition (5) implies that key(root)=(0,..,0). Denote by ground(P) the set of leaves of P (this set is also called the ground level). The ground level represents the set S. Fig.1 presents an example of the pyramidal structure.

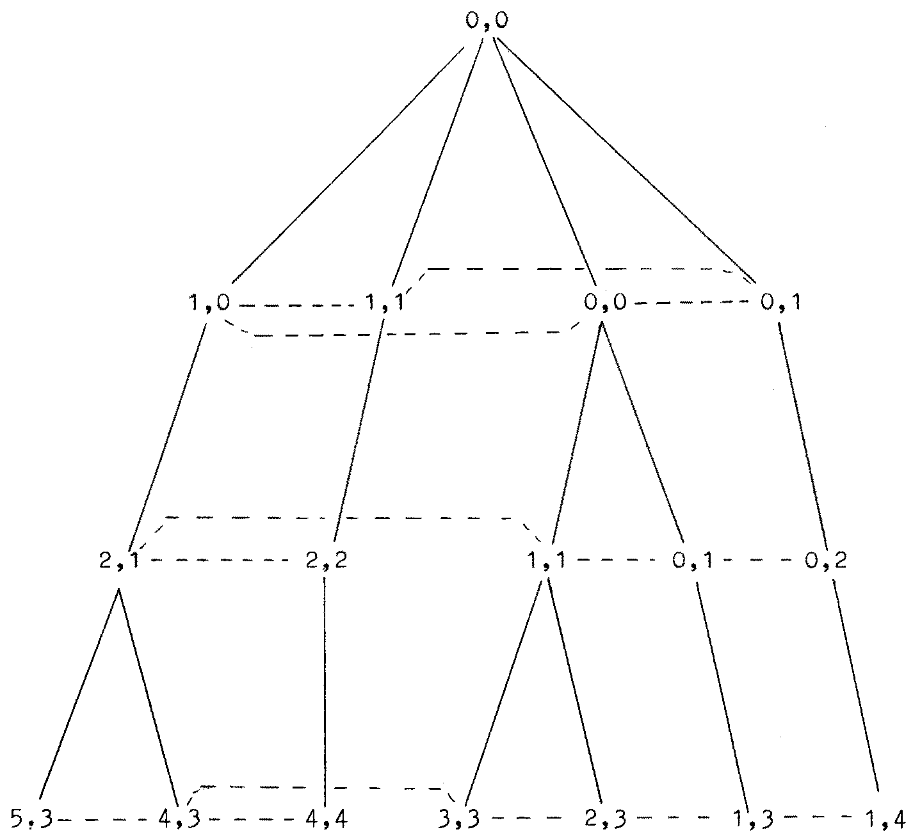

Fig.1. pyramid({(5,3),(4,3),(4,4),(3,3),(2,3),(1,3),(1,4)}).
The keys are placed in the nodes. The horizontal links correspond to the sets neighbours(v). The computation of find((3,4),v), where v is a leaf and key(v)=(3,3), leads to the computation of FIND((3,4),v), FIND((1,2),v_1), FIND((0,1),v_2). Here v_1=father(v) and v_2=father(v_1). Two new nodes are created with the keys (3,4),(1,2). The computation of FIND((0,1),v_2) does not produce new nodes because there is already a neighbour of v_2 with the key (0,1).

Let $|S|$ denote the cardinality of the set S and compress(S) = $\{compress(\bar{x}) \mid \bar{x} \in S\}$. The number k is treated as a constant.

Lemma 1. (key lemma)

Let S be a connected subset of U. Then

a) $|compress(S)| \leq c_k |S| + 1$, where c_k is a constant, $0 \leq c_k < 1$;

b) the size of pyramid(S) is $O(|S|)$ if $|S| \geq \log n$.

Proof.

a) Take $c_k = (2^k - 1)/2^k$. We prove (a) by induction on $|S|$. If $|S| \leq 2^k$ then the thesis follows from the fact that in this case $c_k|S| + 1 \geq |S|$. Assume now that $|S| > 2^k$ and the thesis holds for all connected subsets with the cardinality less than $|S|$. It can be proved that if S is connected and $|S| > 2^k$ then there are two distinct elements $\bar{x}, \bar{y} \in S$ such that compress($\bar{x}$)=compress($\bar{y}$). We partition S into two disjoint connected subsets S_1, S_2 , where S_1 is the maximal connected subset of S such that $\bar{x} \in S_1$ and $\bar{y} \notin S_1$, and $S_2 = S - S_1$. The connectivity of S_2 follows from the connectivity of S. The cardinalities of S_1, S_2 are less than $|S|$, hence (a) holds for S_1, S_2. $|compress(S_1) \cap compress(S_2)| \geq 1$ because compress($\bar{x}$)=compress($\bar{y}$). We have $|compress(S)| = |compress(S_1)| + |compress(S_2)| - |compress(S_1) \cap compress(S_2)| \leq c_k|S_1| + 1 + c_k|S_2| + 1 - 1 = c_k|S| + 1$. This completes the proof of (a). The point (b) follows easily from (a). This completes the proof.

Let P=pyramid(S), where S is a connected subset of U, and let $\bar{x} \in U$, $v \in$ ground(P). We consider the operation find($\bar{x}$,v) which returns the found or created node $w \in$ ground(P) with the key $\bar{x}$. After computing find($\bar{x}$,v) P=pyramid($S \cup \{\bar{x}\}$). It is easier to design the function FIND($\bar{x}$,v), which has almost the same effect as find, however v is not necessarily contained in the ground level. FIND treates the level of v as a ground level. It is assumed that S is connected and dist($\bar{x}$, key(v)) ≤ 1 whenever FIND($\bar{x}$,v) is called.

The operation FIND is easy if there is already a node with the key $\bar{x}$ on the same level as v. If such a node is to be created then we can use the following property of the pyramidal structure.

Assume that w$\neq$root. Then

$$\text{neighbours}(w) \subseteq \bigcup_{w1 \in \text{neighbours}(\text{father}(w))} \text{sons}(w1) \qquad (*)$$

This property allowes to design the function FIND recursively.

```
function FIND(x̄,v);   {x̄ ∈ U, dist(x̄,key(v)) ≤ 1}
var w,w1,w2: node
begin
if there is a node w ∈ neighbours(v) such that key(w)=x̄
  then FIND:=w
else
  begin  {insertion}
  create a new node w with the key x̄;
  FIND:= w;
  father(w):= FIND(compress(x̄),father(v));
  add w to sons(father(w));
  neighbours(w):= empty_set;
  for each w1 ∈ neighbours(father(w)) do
   for each w2 ∈ sons(w1) do
   if dist(key(w2),x̄) ≤ 1  then
     add w2 to neighbours(w) and add w to neighbours(w2)
  end
end function;
```

Lemma 2.

Let S be a connected subset of U and P=pyramid(S). If $v \in$ ground(P) and $dist(\bar{x},key(v)) \leq 1$ then after executing w:=FIND($\bar{x}$,v) we have P=pyramid($S \cup \{\bar{x}\}$), $w \in$ ground(P) and key(w)=$\bar{x}$ (in this case the effect of FIND is the same as the effect of find).

Proof.

If initially $\bar{x} \in U$ and $v \in$ ground(P) then later in the moment when FIND($\bar{x}$,v) is called and v is the root we have $\bar{x}$=(0,...,0) (in this case the result is set to the root and the call terminates). Hence FIND terminates. When the resulting node w is to be created then the information associated with w is to be computed. The key of father(w) is equal to compress($\bar{x}$), hence father(w) can be computed using FIND which treats now the level of father(v) as a ground level and searches compress(x) starting from the finger father(v). After computing father(w) the information associated with father(w) can be used. The crucial point is the correctness of the computed sets neighbours (w). This follows from the property (*). This completes the proof.

Theorem 1.

Assume that initially P=pyramid($\{\bar{x}_0\}$), where $\bar{x}_0 \in U$ and ground(P)=$\{v_0\}$ Assume also that a given sequence σ of m operations σ = find($\bar{x}_1$,v_1),find($\bar{x}_2$,v_2),...,find($\bar{x}_m$,v_m) satisfies for $1 \leq i \leq m$:

a) $dist(key(v_i),\bar{x}_i) \leqslant c$, where $\bar{x}_i \in U$ and c is a constant;
b) $v_i \in ground(P)$ after executing the first $i-1$ operations find.

Then the sequence σ can be executed on-line in $O(m)$ time and space.

Proof.
Each operation $find(\bar{x},v)$ such that $dist(\bar{x},key(v)) \leqslant c$ can be implemented by a sequence of at most c operations of the form $FIND(\bar{y},w)$, where $dist(\bar{y},key(w)) \leqslant 1$. Hence we can assume that c=1. Observe that the cost of computing FIND is constant or it is proportional to the number of created nodes. Henc e time and space complexity of computing σ is linear or it is proportional to the size of $pyramid(\{\bar{x}_0,\bar{x}_1,\ldots,\bar{x}_m\})$. However if c=1 then it follows from (a) that $\{\bar{x}_0,\ldots,\bar{x}_m\}$ is connected. Hence if $m \geqslant \log n$ then the thesis follows from Lemma 1(b).

If $m < \log n$ then we can use the cutted pyramidal structure. If a level contains less than 2^k+1 nodes then we can link all these nodes directely to the root. If the root is linked to more than 2^k nodes then we create the next upper level. We leave the details to the reader. This completes the proof.

Remark.
Observe that the number of fingers is unbounded. The assumption that the bound n is known in advance can be dropped at the cost of more complicated algorithms. In any case it is not a major improvement. There are possible many variations of the pyramidal structure. If we label all edges of the tree representing the pyramidal structure with k-tuples of binary digits then we can delete keys from internal nodes and we obtain the digital search tree.

3. Recursive programs and the tabulation method.

We consider recursive programs given in the following form:

$$f(\bar{x}) = \text{if } p(\bar{x}) \text{ then } \bar{x} \text{ else } f(\, a(\bar{x},\, f(b(\bar{x})))),$$

where $\bar{x} = (x_1,\ldots,x_k)$ is the vector of integer variables and p,a,b are given functions computable in constant time and space. Let $V(\bar{x}_0)$ be the set of all values of $\bar{x}$ for which $f(\bar{x})$ is called during the computation of $f(\bar{x}_0)$. Denote $m = |V(\bar{x}_0)|$ (m is the number of nodes in the dependancy graph of the computation of $f(\bar{x}_0)$ and it is the natural size of the input when we consider the exact tabulation [8]).

We are given an initial value $\bar{x}_0$ of $\bar{x}$ and a natural number n. Assume that:

(a) $f(\bar{x}_0)$ is defined; (b) $V(\bar{x}_0) \subseteq [0..n]^k = U$; (c) for each $\bar{x},\bar{y} \in V(x_0)$ $dist(b(\bar{x}),\bar{x}) \leq c$ and $dist(a(\bar{x},\bar{y}),\bar{y}) \leq c$, where c is a constant.

The problem consists in the computation of $f(\bar{x}_0)$. The numbers n,m characterize the size of the input. Applying the tabulation method we can compute $f(\bar{x}_0)$ in $O(n^k)$ time (observe that the straightforward computation can require exponential time). However we are interested in O(m) time and space which can be much less than n^k. We begin with $O(n^k)$ time computation. We store the computed values of $f(\bar{x})$ in the table T of the size $O(n^k)$. Initially each entry of T contains the special value "undefined".

```
function f1(x̄);
 begin
 if T(x̄)="undefined" then
   T(x̄):= if p(x̄) then x̄ else f1(a(x̄,f1(b(x̄))));
 f1:= T(x̄)
 end;
```

The cost of computing $f1(\bar{x}_0)$ is proportional to the size of the table T, which is $O(n^k)$. We say that the computation of $f(x_0)$ is sparse iff the table T is sparse (during the computation the most part of T contains the value "undefined"). The initialization of T can be omitted using the trick from [8]. However this trick does not reduce space complexity. We show that $f(\bar{x}_0)$ can be computed in O(m) time and space if the conditions (a-c) are satisfied.

Theorem 2.
$f(\bar{x}_0)$ can be computed in O(m) time and space if f and $\bar{x}_0$ satisfy the conditions (a-c).

Proof.
We modify the function f1. The role of the table T will play now the ground level of the pyramidal structure P. With every node $v \in$ ground (P) there is associated an additional information T(v) , where T(v) $=(\bar{x},w)$ for some $w \in$ ground(P), key(w)=$\bar{x}$, or T(v)="undefined". When a new node v is created then initially T(v)="undefined". Initially P=pyramid($\{\bar{x}_0\}$) and ground(P)=$\{v_0\}$, where $T(v_0)$="undefined". At the end $T(v_0)=(\bar{y},w)$, where $\bar{y}=f(\bar{x}_0)$,key(w)=$\bar{y}$. Instead of storing the computed results in the table we store them in the ground level of P. The actual parameters $\bar{x}$,v always satisfy key(v)=$\bar{x}$, $v \in$ ground(P).

The designed function $f2(\bar{x},v)$ returns as a result a pair $(\bar{y},w)$ such that $\bar{y}=f(\bar{x})$, $key(w)=\bar{y}$, $w \in ground(P)$.

```
function f2(x̄,v);      {key(v)=x̄, v ∈ ground(P)}
var ȳ:key; w:node
begin
if T(v)="undefined" then
  begin
  if p(x̄) then T(v):= (x̄,v)
  else
    begin
    ȳ:=b(x̄); w:=find(ȳ,v);          {dist(ȳ,key(v))≤ c}
    (ȳ,w):= f2(ȳ,w);                {y=f(b(x̄))}
    ȳ:=a(x̄,ȳ);                      {dist(ȳ,key(w))≤ c}
    w:= find(ȳ,w);
    T(v):= f2(ȳ,w)
    end;
  end;
 f2:= T(v)
end;
```

The correctness of f2 follows from the correctness of f1. Similarly the number of executed assignments statements is $O(m)$. Now the thesis follows from Theorem 1. This ends the proof.

Remark

Theorem 2 is also valid for recursive programs of another form (for example the one defined in [8]) if it is satisfied a condition analogous to (c). It will be seen in the next section why we have chosen such a form of the recursive program.

4. Two-way pushdown automata

Let A be a 2dpda(k) and let $w=a_1 \ldots a_n$ be the input string of the length n. By a surface configuration (configuration, for short) we mean a vector $\bar{x}=(d,s,h_1,h_2,\ldots,h_k)$, where d is a top element of the stack, s is a state and $h_1,\ldots,h_k$ are positions of the input heads. In the computation of A we look only at the sequence of consecutive (surface) configurations.

We say that a configuration $\bar{x}$ is reachable (for a given input w) iff $\bar{x}$ occures in the computation of A on w. Let m denote the number of different reachable configurations. Assume that $m \geq n$ (A scans all the iput string). The number m is our size of the input.

Theorem 3.
Each 2dpda(k) can be simulated in O(m) time and space.

Proof.
We can assume without the loss of generality that the next move of A is always defined if the stack is nonempty, each move is a pop or a push move, and when A accepts then the stack is one element and the next move is a pop move. Assume also that top symbols and states are numbered, hence configurations $\bar{x}$ are elements of the set $U=[0..n]^{k+2}$ for n sufficiently big. We define the functions popf, pushf, terminal and terminator as follows:

1) if $\bar{x}$ is a push configuration and $\bar{y}$ results from $\bar{x}$ in one step then $\bar{y}$=pushf($\bar{x}$);
2) if $\bar{y}$ is a pop configuration, $\bar{z}$ results from $\bar{y}$ by a pop move and top element of $\bar{z}$ is the same as the top element of $\bar{x}$ then $\bar{z}$=popf($\bar{x}$,$\bar{y}$);
3) pop($\bar{x}$)=true iff $\bar{x}$ is a pop configuration;
4) terminator($\bar{x}$)=$\bar{y}$ iff pop($\bar{y}$) and there is a computation of A starting with the configuration$\bar{x}$ and one element stack and ending with $\bar{y}$ and one element stack.

The simulation of A (on w) can be reduced to the computation of terminator($\bar{x}_0$), where $\bar{x}_0$ is the initial configuration[2]. The function terminator can be computed recursively:

terminator($\bar{x}$)= <u>if</u> pop($\bar{x}$) <u>then</u> $\bar{x}$ <u>else</u> terminator(popf($\bar{x}$,terminator (pushf($\bar{x}$))))

If we know that A does not loop (terminator($\bar{x}_0$) is defined) then the thesis follows directely from Theorem 2 taking the functions f=terminator, p=pop, b=pushf, a=popf. The condition c follows from the fact that in one step the input heads move to neighbouring positions. The functions pop, popf, pushf can be computed in constant time from the description of the automaton.

When we do not know if A loops then we have to detect looping of A during the computation. This can be done by assigning to the nodes $v \in$ ground(P) boolean values onstack(v) which inform whether before a given moment of the computation f2($\bar{x}$,v) was called and this call is still not terminated.

If we call f2($\tilde{x}$,v) and in this moment onstack(v)=true then we know that the function loops and a rejecting procedure is to be called. We leave the details to the reader. The bound on the complexity follows from Theorem 2. This completes the proof.

It could be proved that Theorem 3 holds if the model of the computation ia a storage modification machime [7].
In [4] (p,q)-dpda was defined as a deterministic pushdown automaton with p two-way input heads and q one-way heads. The total number of moves of one-way heads is O(n), hence $m=O(n^{p+1})$, if A is a (p,q)-dpda. Now the result of [4] follows directely from Theorem 3.
Corollary.
Each (p,q)-dpda can be simulated in $O(n^{p+1})$ time and space.

Let A be a two-way nondeterministic pushdown automaton. We say that A is loop-free iff there is no possible an infinite computation of A on any input string. The looping of nondeterministic pushdown automata is much more complicated than the looping of deterministic ones. The following theorem can be proved.

Theorem 4.

Each loop-free nondeterministic multihead two-way pushdown automaton can be simulated in $O(m^3)$ time and space.
Proof.
The (lengthy) proof is similar to that of Theorem 3.

References.

[1] Aho A.V,Hopcroft J.E,Ullman J.D. Time and tape complexity of pushdown automaton languages. Inf.and Control 13:3 (1968)
[2] Aho A.V,Hopcroft J.E,Ullman J.D. The design and analysis of computer algorithms. Section 9.4. Addison-Wesley (1976)
[3] Cook S.A. Linear time simulation of deterministic two-way pushdown automata. Proc.IFIP Congress 1971.
[4] Rytter W. An efficient simulation of deterministic pushdown automata with many two-way and one-way heads. Inf.Proc.Letters 12:5 (1981)
[5] Rytter W. The dynamic simulation of recursive and stack manipulating programs. Inf.Proc.Letters 13:2 (1981)
[6] Rytter W. A note on the complexity of traversing a labyrinth. Graph Theory Conf. Lagov,Poland, february 1981
[7] Schonhage A. Storage modification machines. SIAM J.Comp. august1981
[8] Bird R. Tabulation techniques for recursive programs. ACM Comp. Surveys 12:4 (1980)