자가 조직 탐색 트리 조사 방법론
A Survey on Self-Organizing Search Trees

이혜자(의료정보시스템과), 송두헌(전자계산과)
Hiye-Ja Lee(Dept. of Medical Information System) and Doo-Heon Song(Dept. of Computer Science)

Key Words : 탐색 트리, Splay 트리, 동적 검색 트리.

ABSTRACT : Static data structures are easy to analyze and simple to implement. However, we use dynamic data structures more because of our computing environment. In the case of search trees, we know that OBST(Optimal Binary Search Tree) is the theoretically perfect optimal static one. Thus, there have been quite a lot of researches to find and prove a certain dynamic search tree being the optimal. This paper summarizes the result of various studies focused on a class of dynamic search trees that we call self-organizing search tree. Then we analyze the methods that the class of data structure uses and the splay tree which we find the best existing structure. Finally, we report a number of unproven conjectures which must be done if the structure is the optimal.

1. 서론

자료 검색에 자주 이용되는 데이터 구조에 검색 트리(search tree)가 있다. 검색 트리의 가지(branch) 수는 제한이 없지만 만약 가지 수 K가 고정되어 있다면 $O(\log_2 k)$에 이진 트리로 변환이 가능하기 때문에 보통 이진 검색 트리를 그 기본 데이터 구조로 설정한다. 이진 검색 트리에 대하여는 여러 연구가 진행되어 왔으나 정적(static) 이진 검색 트리의 경우 최적 이진 검색 트리(OBST: Optimal Binary Search Tree)가 알려져 있고 평균 접근 시간이 $O(\log_2 n)$이다. 다만 삽입, 삭제 명령의 경우 그것이 균형잡힌(balanced) 트리인 관계로 구조적 균형을 유지하기 위한 stack을 이용한 복잡한 알고리즘(Hu-Tucker 알고리즘[5]) 이 적용되고 그것의 수행시간은 $O(\log_2 n)$이다. 구조적 균형보다 확률적 균형을 중시한 확률 균형 검색 트리(probability balanced search tree)는 대략 1.44 × OBST의 검색 속도를 갖되 OBST같은 삽입, 삭제에서의 부가적 연산 부담은 없는 편이다.

그러나 대부분의 응용문제에서는 평균접근확률이 미리 알려져 있지 않으므로 이론적 최적치인 OBST를 사용할 수 없다. 때문에 동적 검색 트리(dynamic search tree)에 대한 연구의 필요성이 증대되어 왔다. OBST 혹은 B-tree 같은 정적 데이터 구조는 높이 또는 가중치(혹은 접근확률)가 완전히 혹은 거의 균형을 이룰 수 있어 효과적이었으나 오히려 그런 구조적 제한성(structural constraint) 때문에 접근확률이 알려져 있지 않은 동적 데이터 구조에서는 더 비효율적이다. 일반적으로 동적 데이터 구조 작성의 목표는 제반 연산(검색, 삽입, 삭제)의 컴플렉시티가 그에 상응하는 정적 데이터 구조 연산의 상수배로 제한되는 것인 바, 동적 검색트리의 제반 연산은 만약 그것이 효과적이려면 OBST 연산의 상수배로 수렴되어야 할 것이다.

부분적으로 성공적이었으나 효용성이 떨어지는 동적 검색 트리의 예로 Biased search tree가 있는데 평균 검색 속도는 빠르지만 그 구조를 유지하는 연산이 복잡하고 구조 균형 트리(structurally balanced tree)보다 오히려 구조적 제한성이 더 복잡하다[15]. Finger search tree는 그 검색은 빠르나 각 노드마다 부가적인 포인터를 두어야 하는 공간 부담(space burden)이 있어 덜 효율적이다.

본 논문에서는 평균접근 확률은 알려져 있지 않으나 접근 시도간에는 독립성이 있는 동적 탐색 트리의 그간의 주요 연구 결과를 정리하고 그 장단점을 분석하고자 한다. 전술한 바와 같이 구조적 균형에 집착하면 검색 이외의 연산에 부담이 있으므로 구조적 제한성은 두지 않으나 각 연산마다 구조적 효율성을 위하여 간단한 재구성 법칙(restructuring rules)을 적용, 이후의 연산에 대비하는 정적 탐색 트리 계열이 본 논문의 연구 대상이다. 재구성 법칙에 대하여는 자가 조직 선형 리스트[4]에서 많이 연구되었는데 우리가 그 이론을 다수 차용하는 바 본 논문에서 분석되는 동적 탐색 트리 계열을 자가 조직 탐색 트리(SOST: Self-Organizing Search Tree)라 부르기로 한다.

우리의 목표는 각 연산의 코스트가 정적 최적 탐색 트리의 상수배로 제한되는 SOST의 구현인데 그렇게 되면 구조적 제한성을 가진 동적 탐색 트리보다 공간 사용이 효율적이며(부가적 데이터가 필요하지 않으므로) 프로그램으로의 구현도 더 쉬울 것이다. 본 논문의 2장에서는 여러 연구에서 사용된 재구성 법칙을 소개 분석하고, 3장에서는 이진 SOST의 가장 중요한 데이터 구조인 Splay 트리에 대해서, 4장에서는 Splay 트리의 일반형인 K-splay 검색 트리를 분석하고, 5장에서는 이러한 과정에서 발견된 미해결문제(conjectures and open problems)를 소개하여 후발 연구자가 나아갈 바를 제시하고자 한다.

2. 트리의 기본 재구성 법칙

2.1. 정의(Definitions)

본 논문에서 사용될 각종 수학적 용어에 대하여 아래와 같이 정의한다.

정의 1. 이진 검색 트리

1) T를 이진트리, x를 T의 한 노드(node)라고 하고 Key(x)가 있다고 하자. T_L과 T_R을 각각 T의 좌현부트리(left subtree), 우현부트리(right subtree)라 할 때 모든 노드 $y \in T_L$, $z \in T_R$에 대하여 $Key(y) \leq Key(x) \leq Key(z)$가 성립한다.
2) R_1, R_2, · · ·, Rn은 레코드로 키값 K_1, K_2, · · ·, Kn을 각각 갖고 접근빈도확률 P_1, P_2, · · ·, Pn이 주어진다.

정의 2. 정적 이진 검색 트리

T가 이진 검색 트리일 때 접근빈도 확률 Pi가 고정되고 미리 알려져 있으며 각 접근 시도간에 관계가 없으면 정적 이진 검색 트리라고 한다.

정의 3. 동적 이진 검색 트리

T가 이진 검색 트리일 때, 접근 빈도 확률 Pi가 접근 시도 전에 알려져 있지 않거나 각 시도간에 서로 영향이 있을 때 동적 이진 검색 트리라고 한다.

정의 4. 평균 접근 코스트

$C_A(Ri)$를 방법 A로 레코드 Ri를 접근하는 데 드는 코스트라 하고 $C_A(T)$를 트리 T의 모든 노드를 접근하는 데 필요한 평균접근 코스트, Pi를 Ri에 대한 접근 확률이라 할 때 $C_A(T) = \sum_{i=1}^{\infty} PiC_A(Ri)$가 성립한다.

정의 5. 최적 접근 코스트

C_{OPT}는 최적 접근 코스트로 정적 최적 이진 검색 트리의 접근 코스트를 말한다.

2.2 MTR 법칙(Move to Root)

만약 검색하고자 하는 노드가 트리 T에 있다면 포인터를 서로 바꾸어 그 노드를 루트로 만든다. MTR은 이 과정에서 오직 상수의 부가 공간(extra storage)과 시간을 사용하며 이론적으로 (1)에서 그것이 C_{OPT}의 상수배임이 증명되었다.

레마 2.2 $H-\log H-\log H \le C_{OPT} \le H+1$ 단, $H = \sum_{i=1}^{\infty} -Pi\log Pi$

정리 2.2 $C_{MTR} < 2\ln 2(C_{OPT} + \log C_{OPT}) + 3 \approx 1.3863(C_{OPT} + \log C_{OPT}) + 3$

만약 접근확률이 균일분포라면 (즉, $Pi = 1/n \ \forall i$), 정리 2.2는 $C_{MTR} \approx (2\ln 2)\log n$임을 보여 준다. 그러나 최악의 경우에 평균 검색시간이 O(n)에 달할 수도 있다는 것이 (1)에서 제시된 바 있다.

2.3 SE 법칙(Simple Exchange)

SE는 선형 검색 리스트의 변환(transposition)과 비슷하다. 레코드 Ri를 검색할 때 SE는 Ri와 Parent(Ri)의 포인터를 서로 바꾼다(단, Ri가 루트가 아닐 때). (13)에서 선형 검색 트리의 경우 SE가 MTR보다 다소 우수함이 밝혀졌지만 검색 트리의 경우는 그렇지 않다. 그 경우 SE는 유한 Markov 체인이 되고 최악의 경우 평균접근확률 $Pi = 1/n \forall i$일 때 $C_{SE} = O(\sqrt{\pi n})$이 된다.

2.4 SMR 법칙(Split Move to Root)

2.2에서 보인 MTR은 검색이 성공했을 때만 효율적이다. (1)은 그래서 T안에 있지 않은 Ri를 검색했을 경우도 포괄하는 SMR을 소개하고 있다. 만약 키값 $K \in T$이면 SMR은 MTR처럼 행동한다. 그러나 $K \notin T$이면 즉 $Ri < K < Ri+1$ $(1 \le i \le n-1)$이면 Ri 혹은 Ri+1이 루트와 교환된다. 이 때 $C_{SMR} < (4\ln 2)(C_{OPT} + \sum_{i=1}^{\infty} Pi\log C_{OPT}) + 6$이고, 이 때 Pi는 Ri가 T에서 발견될 확률이다.

2.5 MON 법칙(Monotonic Tree Rule)

MON은 heap처럼 접근빈도에 따라 최빈 노드가 루트가 되고 각 부트리 또한 재귀적

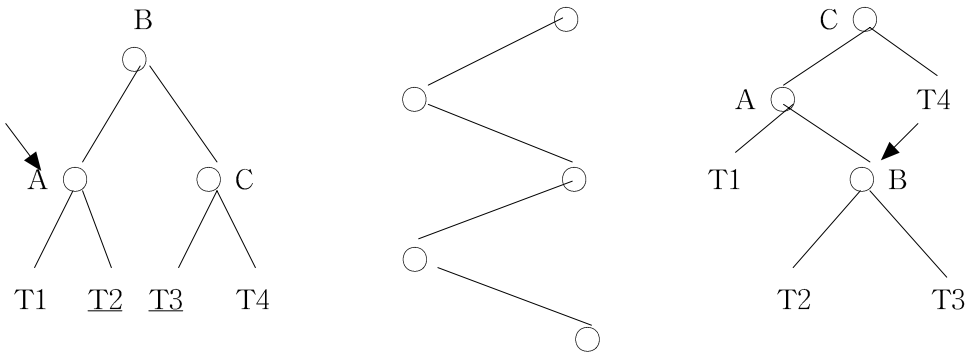
(recursively)으로 정의된다. 즉, 좌현부트리의 모든 노드 중 접근빈도가 가장 큰 노드가 좌현부트리의 루트가 된다. 이런 구조를 유지하기 위하여 MON은 각 노드마다 접근 횟수를 기록하고 매 접근 시마다 접근 횟수에 따라 구조를 조정하기 위하여 전환(rotation)알고리즘을 적용한다. 당연히 최악의 경우는 완전 선형인 트리(degenerated tree)이다. 즉, $Ki < Ki+1$ 이면서 $Pi \geq Pi+1 (1 \leq i \leq n-1)$인 경우이다. 만약 Pi가 서로 거의 같으면 최악의 경우이고 (12)는 $C_{MON} \leq (n/(4\log n))C_{OPT}$임을 증명하였다.

2.6 WR 법칙(Weighted Rotations)

(2)는 누적 접근 빈도에 따른 두 개의 알고리즘을 제시하였다. 기본 아이디어는 SE가 개별적인 노드의 위치를 바꾸지만 많은 경우 부트리 전체의 위치가 바뀌는 것이 더 효과적이라는 것이다.

LSR(Limited Single Rotation)은 주어진 키값 K의 탐색 경로상에서 위로 올려질 부분의 접근 빈도가 아래로 내려갈 부분의 누적 접근 빈도보다 많을 때 전환 알고리즘을 적용한다. 그림 1의 (a)는 이것의 예로서, A가 전환점(point of rotation)일 때 $W(A)+W(T1) > W(B)+W(C)+W(T3)+W(T4)$이면 LSR을 적용한다($W(X)$는 $\sum_{y \in x}$(y의 접근 빈도)).

그러나 LSR은 그림 1 (a)의 T2, T3같은 "내부"(inside) 부트리는 고려하지 않는 관계로 만약 트리 T가 그림 1의 (b)같이 균형이 거의 잡히지 않은 경우에는 T가 거의 변하지 않는다 (blind problem).



T2, T3 in (a) are "blind" subtrees when SLR is applied

그림 1. Weighted Rotations

DR(Double Rotation)은 바로 이 문제에 대한 해결책이다. 그림 1의 (c)는 DR을 노드 B에 적용하는 예로서, 그 결과는 그림 1의 (a)가 되고 DR이 적용되려면 $2W(B)+W(T2)+W(T3) > W(C)+W(T4)$가 성립해야 한다.

불행히도 LSR과 DR에 대한 이론적 분석은 정확히 증명되지 않았다. 프로그램을 통한 실험 결과는 $C_{LSR} \approx 1.06 \times C_{OPT}$이고 $C_{DR} \approx 1.0384 \times C_{OPT}$이다.

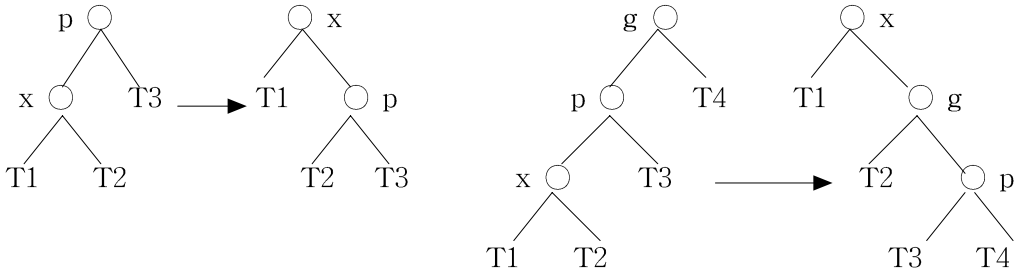
3. Splay 트리

2장에서 본 트리 재구성 법칙들은 그러나 전환이 검색 대상 노드 하나에 국한된다는 단점이 있다. 따라서 이 경우 종종 트리 T의 높이(height)가 늘어나 MTR의 경우처럼 허용할 수 없는 최악의 경우가 발생할 수 있다. Splaying 법칙[15]은 접근 경로의 구조에 따라 쌍(pair)으로 전환을 시도하며 이것이 대략적으로 트리의 높이를 절반으로 줄일 수 있다. 개별적인 연산으로 볼 때 Splaying은 코스트가 높다. 그러나 우리의 주안점은 개별 연산이 아니라 여러 번의 연산이 수행된 후의 평균 접근 코스트이므로 또 이런 높은 코스트의 연산 후에는 트리 T의 높이가 줄어 향후 연산의 코스트가 낮아지므로 개략적 평균 분석법(amortized analysis)에 따르면 Splay 트리의 코스트는 OBST의 상수배임이 증명되어 있다[17].

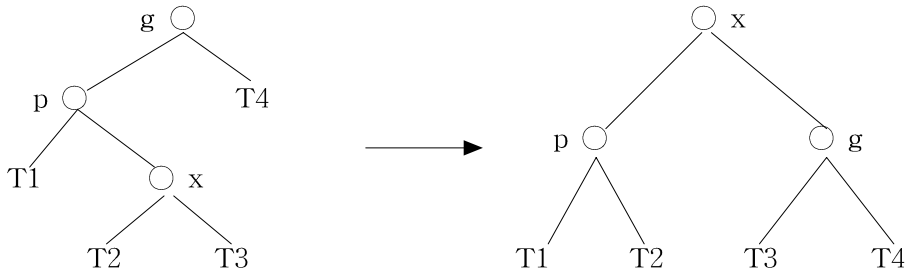
정의 6. Splay step

트리 T의 노드 x에 대하여 P=parent(x), g=grandparent(x)라 하자.

1. (Zig) 만약 p=root면 r(x, p)에 의하여 x를 루트로 한다. 단 r(a, b)는 노드 a와 b의 전환을 말한다.
2. (Zig-zig) 만약 p와 x가 공히 이진 트리의 좌현 혹은 우현 자손이면 먼저 r(p, g)를 행한 후 r(x, p)를 행한다.
3. (Zig-zag) 만약 x, p, g가 한 줄로 있지 않다면 r(x, p)를 한 후 g를 새로운 parent(x)로 하고 r(x, g)를 행한다.



(a) Zig (b) Zig-zig



(c) Zig-zag

그림 2. Splaying steps

Splaying 연산의 개략적 평균 분석은 아래와 같다. potential function법[17]을 이용하면 아래의 공식이 성립한다.

$$\sum_{j=1}^{\infty} t_j = \sum_{j=1}^{\infty} a_j + (\theta m - \theta o)$$ - 공식 3.1

여기서 m은 연산 횟수, t_j와 a_j는 각각 j번째 연산의 실제의 그리고 개략적 코스트를 말하며 $\theta m \geq \theta o$ 이고 각각 최종 potential과 초기 potential을 나타낸다. 코스트는 전환의 횟수로 측정되며 Splay트리 T의 potential은 아래와 같이 정의된다.

정의 7. potential

트리 T의 각 노드 i가 양의 비중 W(i)를 갖고 Tx 는 x가 루트인 부트리를 말한다. x의 크기 $S(x) = \sum_{i \in Tx} W(i)$이고 랭크 r(x)=log S(x)이면 트리 T의 potential은 $\sum_{i \in T} r(i)$로 주어진다.

Splay 트리에서 알려진 유용한 정리가 아래에 정리되어 있다.

레마 3.1 (접근 레마[15])

루트가 t이고 노드 x에서 Splay를 행하는 개략적 코스트는 $C_T^x \leq 3(r(t)-r(x))+1 = O(\log(s(t)/s(x))$이다. 충분히 긴 m연산이 n노드의 트리 T에 행해진 후 다음의 두 정리가 증명되었다.

정리 3.1 (균형 정리) 누적 접근시간은 O((m+n)logn)이다.

정리 3.2 (정적 최적치 정리)

만약 모든 노드가 적어도 한 번 접근되었다면 qi가 노드 i의 접근빈도라 할 때 누적 접근 시간은 $O(m + \sum_{i=1}^{\infty} qi \log(m/qi))$이다.

정리 3.1과 3.2의 결과는 Splay 트리가 모든 정적 탐색 트리와 그 효율면에서 차이가 없음을 말하며 이는 아래에 정리된 각종 갱신(update) 연산에서도 마찬가지이다.

정리 3.3 (갱신) 아래의 연산들이 Splay 트리 T의 노드 i에 적용되었을 때 그 컴플렉시티는 O(3log(W/wi))이다. wi는 노드 i의 비중이고 W는 전체의 비중(여기서는 접근빈도)이다.

- 검색(i, T) : T에서 노드 i를 찾되 i∉T이면 null을 기록한다.
- 삽입(i, T) : 노드 i∉T이면 i를 삽입하고 노드 i로부터 Splaying을 실시한다.
- Join(T1, T2) : ∀x∈T1, y∈T2, x<y, T1과 T2를 묶어 한 트리로 만든다.
- 삭제(i, T) : 탐색(i, T)를 실시한 후 T1과 T2가 i를 루트로 하는 부트리일 때 Join(T1, T2)를 실시한다.
- Split(i, T) : i를 중심으로 T1, T2를 ∀x∈T1, y∈T2, x≤y, y≥i와 같이 형성한다.

Splay 트리는 priority queue, deque 등에 응용되어 좋은 결과를 보였으며[10], 데이터 압축에도 실제적으로 응용된 결과가 보고된 바[7] 이것이 단순한 이론적 데이터 구조는 아니라고 할 것이다.

4. K-splay 탐색 트리

3장에서 살펴본 Splay 트리는 이진 트리이다. OBST의 경우는 K진 트리를 $O(\log_2 K)$에 이진 트리로 만들 수 있어 K-ary 트리에 대한 분석이 불필요하나 Splay 트리는 다르다. 특히 이진 Splay 트리는 현실적으로 virtual memory 체제 하에서는 지역 접근성(locality)에 문제가 있다. (14)는 그에 따라 Splay 트리를 K-ary로 확장하였다. 이것을 K-splay라고 하는데 각 K-splay step은 최종 step을 제외하고 k+1 노드와 k edge가 그 탐색 경로에 연결되어 있다.

우선 탐색이 종료된 노드를 star 노드라고 부르기로 하자. K-splay가 진행되면 그 노드의 값이 탐색 경로를 따라 변화한다고 생각해도 좋다. 이렇게 그 값이 서로 바뀌는 일련의 노드 집합을 'framework'라고 한다. K-splay는 아래의 두 경우가 있다.

보통 step

이것은 star 노드가 그 깊이(depth)가 적어도 k였다가 k 미만이 되는 경우이다. 이 경우 (k+1)(k-1)개의 구 framework 값이 깊이 1의 k부트리에 재정리된다. 이 구 framework은 star 노드와 그보다 직속 상위인 k노드로 (탐색 경로 상의) 구성되어 있는데 새 framework의 루트가 새 star 노드가 되고 여기서부터 다음 K-splaying이 이루어진다.

종료 step

이것은 star 노드의 깊이가 k 미만일 경우이다. 그 깊이를 l이라 할 때 루트는 l개의 부트리를 갖고 각 부트리는 k개의 값을 갖도록 레코드의 키값이 재정리된다.

K-splay트리의 코스트는 접근되는 노드 수로 측정되는데 보조기억장치가 필요하지 않은 경우라면 Splay step마다 $\Omega(n^2)$개의 값이 다른 노드로 전해지는 부담이 있기 때문이다.

만약 S가 임의의 접근, 삭제, 삽입연산이라하고 n이 트리 T의 노드 수라 할 때 처음에 비어 있던 K-splay 트리가 이 S연산을 수행하는 데 걸리는 시간은 $O(m\log_2 n)$이다[14].

5. 미해결 과제들

위에서 본 것 같이 Splay 트리는 우리가 찾던 최적 SOST처럼 보인다. 그러나 Splay 트리가 개략적 분석법에 의해서만이 그 수행결과를 짐작할 수 있는 관계로 아직 명확히 증명되지 않은 정리들이 있다. 만약 이 모두가 증명되다면 Spaly 트리가 최적 SOST 구조임을 말할 수 있을 것이다. 여러 연구[3, 8, 16, 19]가 여기에 관하여 보다 진전된 결과를 보고하였으나 아직 완전하지는 않다. SOST의 후발 연구자들을 위하여 우리는 여기에 미해결 과제들을 정리하는 것으로 본 논문을 마치고자 한다.

과제 1. (동적 최적치[15])

n노드의 이진 트리 T에서 임의의 m개의 검색을 생각한다. 이 일련의 검색을 S라 하고 T(S)는 이것의 이론적 최저 코스트라 한다. 루트에서 노드 x까지의 탐색 코스트가 1+depth(x)이고 전환 코스트를 1이라 할 때 Splaying으로 이것을 검색하는 데 O(T(s)+n)이 든다.

과제 2. (동적 Finger 정리[15])

n노드 Splay 트리 T에서 m개의 성공적 검색을 하는 데에는 $O(m+n+\sum_{j=0}^{m-1}\log(|i_{j+1}-i_j|+1))$이 든다. 단, $1\le i\le m$, i_j는 노드 i의 j번째 접근이다.

과제 3. (deque 정리[18])

m deque 연산을 n노드 Splay 트리에 적용하는 시간은 $O(m+n)$이다.

과제 4. (Turn 정리[18])

우선 Turn은 Splaying step에서 Zig-zig case의 쌍 전환을 말한다고 하자. n노드 트리 T에서 우현 Turn과 우현 전환을 섞은 임의의 연산을 실시하면 Turn의 개수가 $O(n)$이고 전환은 nC_2개가 된다.

과제 5. (Split 정리[8])

만약 n노드 Splay 트리를 계속 나누면 결국 n개의 단수 노드 트리가 될 것인데 이렇게 하는 데에는 $O(n)$시간이 필요하다.

참고문헌

(1) Allen, B. and Munro, I., "Self-organizing Binary Search Tree", *Journal of ACM*, Vol. 25, No. 4, 1978, pp. 526-535

(2) Bitner, J.R., "Heuristics that dynamically organize data structures", *SIAM Journal of Computing*, Vol. 8, No. 1, 1979, pp. 82-110

(3) Cole, R., "On the Dynamic Finger Conjecture for Splay Trees Extended Abstract", *In Proceedings of 31th IEEE Annual Symposium on Foundations of Computer Science*, 1990, pp. 8-17

(4) Hester, J.H., and Hirchberg, D,S., "Self-Organizing Linear Search", *Computing Surveys*, Vol. 17, 3., 1985, pp. 296-311

(5) Hu, T.C., *Combinatorial Algorithms*, Addison-Wesley, 1982

(6) Jones, D.W., "An empirical comparison of priority queue and event-set simulation", *Communications of ACM*, Vol. 29, No. 4, 1986, pp. 300-311

(7) Jones, D.W., "Application of Splay Trees to Data Compression", *Communications of ACM*, Vol. 31, No. 8, 1988, pp. 996-1007

(8) Lucas, J.M., Arbitrary splitting in Splay Trees, Tech. Report No. DCS-TR234, Computer Science Dept. Rutgers University, 1988

(9) Mäkinen, E., On Top-down Splaying, BIT, Vol. 27, 1987, pp.330-339

(10) Mäkinen, E., "Splay Trees as Priority Queues", *International Journal of Computer Mathematics*, Vol. 31, 1989, pp. 55-62

(11) Martel, C. "Self-adjusting multi-way search trees", *Intormation Processing Letters*, Vol. 38, No. 3, 1991, pp. 135-141

(12) Melhorn, K., "Nearly optinal binary search trees", *Acta Informatica*, Vol. 5, 1975, pp. 287-295

(13) Revest, R., "Self-organizing sequential search heuristics", *Communication of ACM*, Vol. 19, No. 2, 1976, pp. 63-67

(14) Sherk, M., "Self-adjusting k-ary search trees, In Algorithms and Data Srructures", *Lecture Notes in Computer Science 382*, Springer-Berlag, Berlin, 1989, pp. 381-392
(15) Sleator, D.D., and Tarjan, R.E., "Self-adjusting Binary Search Tree", *Journal of ACM*, Vol.32, No. 3, 1985, pp. 652-686
(16) Sundar, R., "Twists, Turns, Cascades, Deque conjecture, and Scanning theorem", *In Proceeding of 30th IEEE Annual Symposium on Foundations of Computer Science*, 1989, pp. 555-559
(17) Tarjan, R.E., "Amortiaed computational complexity", *SIAM Journal of Computing*, Vol. 6, No. 2, 1985, pp 367-378
(18) Tarjan, R.E., "Sequential access in splay trees takes linear time", *Combinatorica*, Vol. 5, No. 4, 1985, pp. 367-378
(19) Wilber, R., "Lower bounds for accessing binary search trees with rotations", *SIAM Journal of Computing*, Vol. 18, No. 1, 1989, pp. 56-67