Tabelas de símbolos de árvores binárias de busca

- Além do Sedgewick (**sempre** leiam o Sedgewick), veja
  - `http://www.ime.usp.br/~pf/mac0122-2002/aulas/symbol-table.html`
  - `http://www.ime.usp.br/~pf/algoritmos/aulas/bint.html`
  - `http://www.ime.usp.br/~pf/algoritmos/aulas/binst.html`

Tabelas de símbolos de árvores binárias de busca

- *Tabelas de símbolos*: estruturas de dados para se manter um conjunto de itens que admitem as operações básicas de *inserção* de itens e *busca de um item com uma dada chave*.
- Veremos implementações elementares e com *árvores binárias de busca* (ABBs)
- Outra implementação importante: *tabelas de espalhamento* (tabelas de *hashing*)

Tabelas de símbolos

```
/* prog12.1.c - ST.h */
void STinit(int);
 int STcount();
void STinsert(Item);
Item STsearch(Key);
void STdelete(Item);
Item STselect(int);
void STsort(void (*visit)(Item));
```

Um cliente

```
/* prog12.2.c */
#include <stdio.h>
#include <stdlib.h>
#include "Item.h"
#include "ST.h"

void main(int argc, char *argv[])
 { int N, maxN = atoi(argv[1]), sw = atoi(argv[2]);
    Key v; Item item;
    STinit(maxN);
```

Um cliente

```
/* prog12.2.c */
[...]
    for (N = 0; N < maxN; N++)
      { if (sw) v = ITEMrand();
          else if (ITEMscan(&v) == EOF) break;
        if (STsearch(v) != NULLitem) continue;
        key(item) = v;
        STinsert(item);
      }
    STsort(ITEMshow); printf("\n");
    printf("%d keys ", N);
    printf("%d distinct keys\n", STcount());
 }
```

Um cliente (levemente modificado); exemplo de execução

```
yoshi@erdos:~/IME/www/2003ii/mac122/exx/ST_basico$ a.out
We hold these truths to be self-evident, that all men are
created equal, that they are endowed by their Creator with
certain unalienable Rights, that among these are Life,
Liberty and the Pursuit of Happiness.

Creator
Happiness.
Liberty
Life,
Pursuit
Rights,
We
all
among
and
are
be
by
```

Um cliente (levemente modificado); exemplo de execução

```
certain
created
endowed
equal,
hold
men
of
self-evident,
that
the
their
these
they
to
truths
unalienable
with

35 keys, 30 distinct keys
yoshi@erdos:~/IME/www/2003ii/mac122/exx/ST_basico$
```

Implementações elementares

- Vetor indexado pelas chaves (que supomos serem inteiros pequenos)
- Vetor não-ordenado; vetor ordenado
- Lista ligada não-ordenada; lista ligada ordenada

Implementações elementares

Vetor indexado pelas chaves:

```
/* prog12.3.c */
static Item *st;
static int M = maxKey;

void STinit(int maxN)
  { int i;
    st = malloc((M+1)*sizeof(Item));
    for (i = 0; i <= M; i++) st[i] = NULLitem;
  }
```

Implementações elementares

```
[...]
int STcount()
  { int i, N = 0;
    for (i = 0; i < M; i++)
      if (st[i] != NULLitem) N++;
    return N;
  }
void STinsert(Item item)
  { st[key(item)] = item; }
Item STsearch(Key v)
  { return st[v]; }
void STdelete(Item item)
  { st[key(item)] = NULLitem; }
```

Implementações elementares

```
[...]
Item STselect(int k)
  { int i;
    for (i = 0; i < M; i++)
      if (st[i] != NULLitem)
        if (k-- == 0) return st[i];
  }
void STsort(void (*visit)(Item))
  { int i;
    for (i = 0; i < M; i++)
      if (st[i] != NULLitem) visit(st[i]);
  }
```

Implementações elementares

Vetor ordenado:

```
/* prog12.4.c */
static Item *st;
static int N;

void STinit(int maxN)
  { st = malloc((maxN)*sizeof(Item)); N = 0; }
int STcount()
  { return N; }
```

Implementações elementares

```
[...]
void STinsert(Item item)
  { int j = N++; Key v = key(item);
    while (j>0 && less(v, key(st[j-1])))
      {  st[j] = st[j-1]; j--; }
    st[j] = item;
  }
Item STsearch(Key v)
  { int j;
    for (j = 0; j < N; j++)
      { if (eq(v, key(st[j]))) return st[j];
        if (less(v, key(st[j]))) break;
      }
    return NULLitem;
  }
```

Implementações elementares

```
[...]
Item STselect(int k)
  { return st[k]; }
void STsort(void (*visit)(Item))
  { int i;
    for (i = 0; i < N; i++) visit(st[i]);
  }
```

Implementações elementares

Lista ligada não-ordenada:

```
/* prog12.5.c */
typedef struct STnode* link;
struct STnode { Item item; link next; };
static link head, z;
static int N;

static link NEW(Item item, link next)
  { link x = malloc(sizeof *x);
    x->item = item; x->next = next;
    return x;
  }
```

Implementações elementares

```
[...]
void STinit(int max)
  { N = 0; head = (z = NEW(NULLitem, NULL)); }

int STcount()
  { return N; }
```

Implementações elementares

Lista ligada não-ordenada:

```
[...]
Item searchR(link t, Key v)
  {
    if (t == z) return NULLitem;
    if (eq(key(t->item), v)) return t->item;
    return searchR(t->next, v);
  }
Item STsearch(Key v)
  { return searchR(head, v); }

void STinsert(Item item)
  { head = NEW(item, head); N++; }
```

Busca binária

- Supõe itens em um vetor ordenado
- Muito mais rápido que uma busca seqüencial
- Número máximo de comparações entre itens $=$ número de bits na representação binária de N, que é $\lfloor \log_2 N \rfloor + 1$

Busca binária

```
/* prog12.6.c */
Item search(int l, int r, Key v)
  { int m = (l+r)/2;
    if (l > r) return NULLitem;
    if eq(v, key(st[m])) return st[m];
    if (l == r) return NULLitem;
    if less(v, key(st[m]))
         return search(l, m-1, v);
    else return search(m+1, r, v);
  }
Item STsearch(Key v)
  { return search(0, N-1, v); }
```

Tabelas de símbolos com ABBs

- Implementação mais sofisticada
- Existem diversas variantes: balanceadas e não-balanceadas
- Nesta disciplina veremos apenas ABBs não-balanceadas (ABBs balanceadas ficam para MAC323)

Árvores

- Árvores binárias T: (*i*) Λ ("árvore vazia") ou (r, T_L, T_R), (*ii*) $\square$ [nó externo] ou (r, T_L, T_R)
- Árvores binárias de busca: árvores binárias cujos nós estão associados a chaves de forma que valha a "propriedade de árvore binária de busca", a saber, as chaves associadas aos nós na subárvore esquerda (resp., direita) de um nó arbitrário x são menores ou iguais (resp., maiores ou iguais) à chave do nó x.

Uma implementação de uma TS com uma ABB

```
/* prog12.7.c */
#include <stdlib.h>
#include "Item.h"

typedef struct STnode* link;
struct STnode { Item item; link l, r; int N };

static link head, z;

link NEW(Item item, link l, link r, int N)
  { link x = malloc(sizeof *x);
    x->item = item; x->l = l; x->r = r; x->N = N;
    return x;
  }
```

Uma implementação de uma TS com uma ABB

```
[...]
void STinit()
  { head = (z = NEW(NULLitem, 0, 0, 0)); }

int STcount() { return head->N; }

Item searchR(link h, Key v)
  { Key t = key(h->item);
    if (h == z) return NULLitem;
    if eq(v, t) return h->item;
    if less(v, t) return searchR(h->l, v);
             else return searchR(h->r, v);
  }
Item STsearch(Key v)
  { return searchR(head, v); }
```

Uma implementação de uma TS com uma ABB

```
[...]
link insertR(link h, Item item)
  { Key v = key(item), t = key(h->item);
    if (h == z) return NEW(item, z, z, 1);
    if less(v, t)
         h->l = insertR(h->l, item);
    else h->r = insertR(h->r, item);
    (h->N)++; return h;
  }
void STinsert(Item item)
  { head = insertR(head, item); }
```