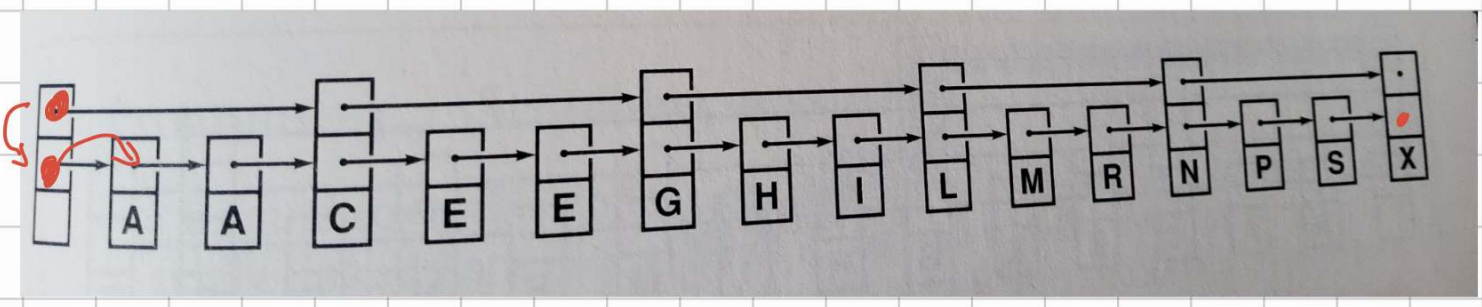
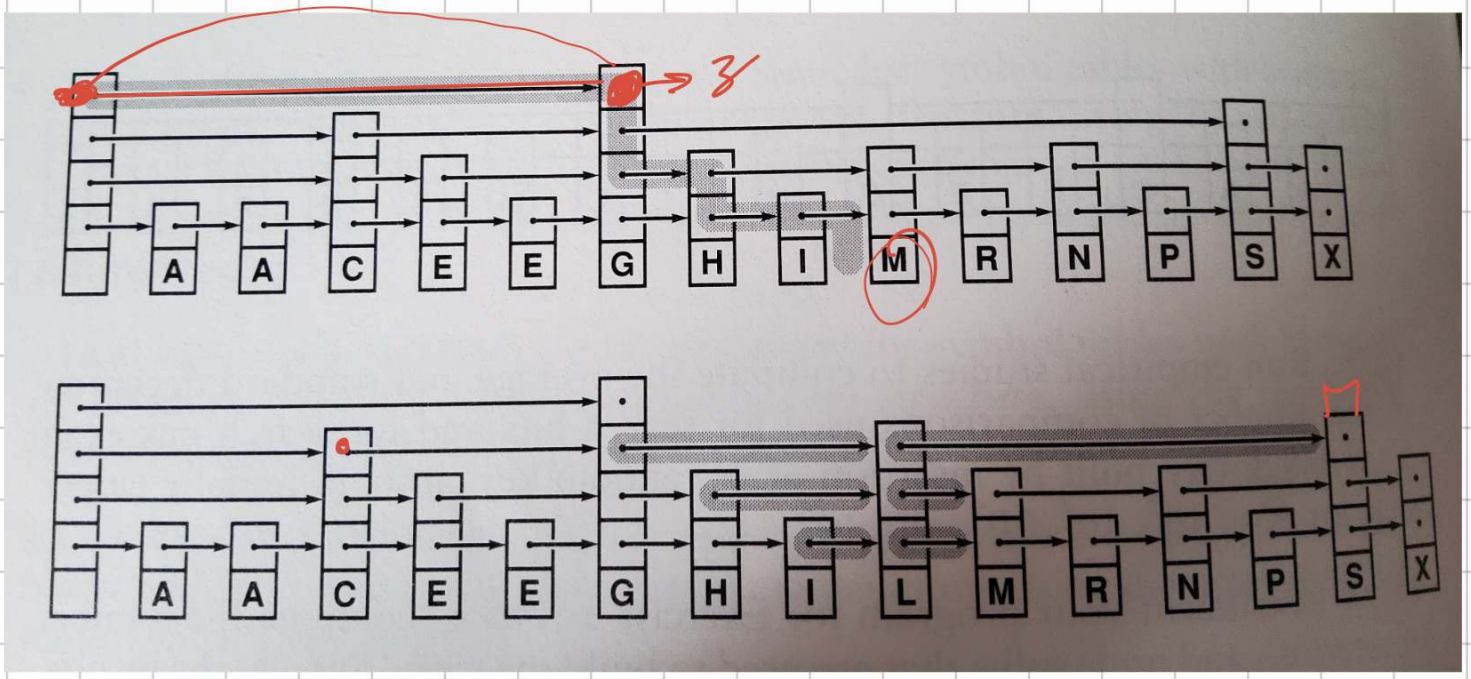
SKIP List



- Usa links extras nos nós da lista encadeada para pular várias entradas de uma vez.
- No exemplo acima: A cada 3 nós na lista encadeada ordenada possui um link adicional que permite pular 3 nós de lista.
- Usamos o link extra para acelerar a busca
  - Como fazer a busca?
  - Qual o fator de aceleração?
  - Examinamos até quantos nós?

DEFINIÇÃO: Uma "SkipList" é uma lista encadeada ordenada onde cada nó possui uma quantidade variada de links, sendo que, o i-ésimo link implementa uma lista simplesmente encadeada que pula os nós com menos que i links



- A representação é direta: trocamos o link por um vetor de links e um inteiro que possui a contagem de links.

OPERAÇÃO DE BUSCA

- Item searchR(link t, Key v, int k);
  Item ST searck(Key v);

```
Item searchR(link t, Key v, int k)
{
  Key tk = key(t->links[k]->item);

  if(eq(tk, v)) return t->links[k]->item;

  if(t->links[k] == z)
    if(k>0)
      return searchR(t, v, k-1);
    else
      return NULLitem;

  if(less(v, tk)) return searchR(t, v, k-1);

  return searchR(t->links[k], v, k);
}
```

É possível deixar mais elegente?

Inicialização da SKIP LIST

- Nós possuem um vetor de links
- NEW precisa alocar o vetor e apontar todos os links para "z"
- Temos uma constante lgNmax, representando o máximo de níveis que permitimos
- N mantém a contagem de itens na lista
- lgN é o número de níveis da lista
- Uma lista vazia é um nó cabeça com lgNmax links, todos apontando para "z", com N e lgN valendo 0.

```
typedef struct STnode *link;
struct STNode { Item item, link *links, int sz; };

static link head, z;
static int N, lgN;

link NEW(Item item, int k)
{
   link x = malloc(sizeof(*x));
   x->item = item;
   x->links = malloc(k * sizeof(x));
   x->sz = k;
   for(int i=0; i< k; i++)
      x->links[i] = z;
   return x;
}

void STinit(int maxN)
{
 N=0; lgN=0;
 z = NEW(NULLitem, 0);
 head = NEW(NULLitem, lgNmax+1);
}
```

Inserção em SKIP LISTS

- Criamos um nó com j-links com a probabilidade $1/2^j$
- Procede com a busca e faz o link quando descemos o nível para cada nível em j.

```
int randX()
{ int i, j, t = rand();
  for(i = 1, j = 2 ; i < lgNmax ; i++, j += j)
      if (t > RAND_MAX/j) break;
  if (i > lgN) lgN = i;
  return i;
}
```

```
Void insertR (link t, link x, int k)
{
  Key v = key(x -> item);
  if (less(v, key(t -> links[k] -> item)))
  {
    • if(k < x -> sz)
      {
        x -> links[k] = t -> links[k];
        t -> links[k] = x;
      }
    • if (k == 0) return;
    • insertR(t, x, k-1); return;
  }
  insertR(t -> next[k], x, k);
}

Void ST insert (Item item)
{
  insertR(head, NEW(item, randX()), lgN); N++
}
```

Ideia do algoritmo de inserção

- O primeiro passo é determinar quantos link o nó terá
  - Todos os nós tem pelo menos 1 link
  - Podemos pular "t" nós de uma vez no segundo nível para pular um em todos nós "t", devemos ter 2 links
    - Conclusão t^j nós devem ter ao menos j+1 links

- Para criar os nós com esta propriedade, usemos uma função que devolve j+1 com a probabilidade de $1/t^j$
  - Dado j, criemos um novo nó com j links e insiramos na SKIPLIST da mesma forma que fazemos a busca
    - Depois que alcançarmos o nível "j", fazemos o link para o novo nó sempre que descermos um nível

Propriedade: Busca e Inserção em uma SKIP LIST aleatorizada com parâmetro "t" requer aproximadamente $(t \log_t N)/2 = (t/2 \lg t) \lg N$ comparações, na média

- É esperado que a SKIP LIST tenha aproximadamente $\log_t N$ níveis, pois $\log_t N$ é maior que o menor "j" para $t^j = N$.

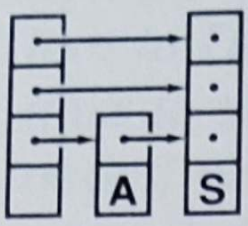
- Em cada nível, esperamos que tenha aproximadamente t nós que foram pulados do nível anterior, e que temos que visitar metade deles, na média, antes de descer para o próximo nível

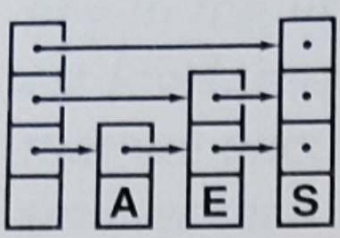
Propriedade SKIP LISTS possuem $(t/(t-1))N$ links na média.

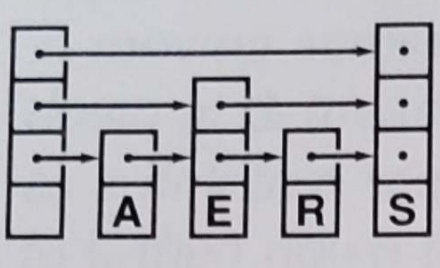
Existem N links no nível mais baixo, N/t links no primeiro nível, $\sim N/t^2$ no segundo, ...

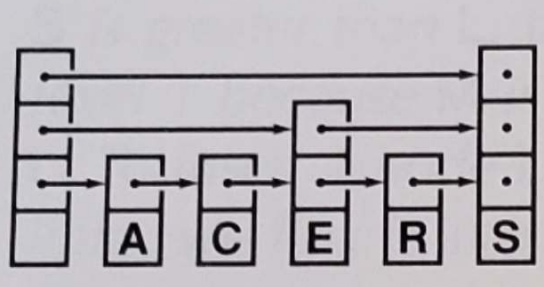
$$N(1 + 1/t + 1/t^2 + 1/t^3 \ldots) = N/((1-1)/t)$$

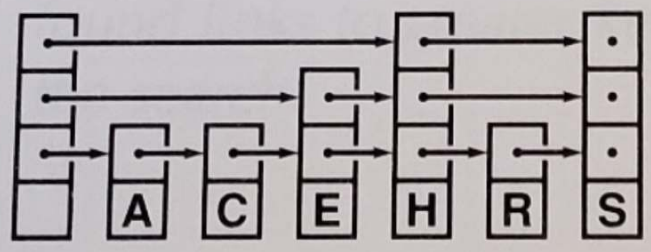

A

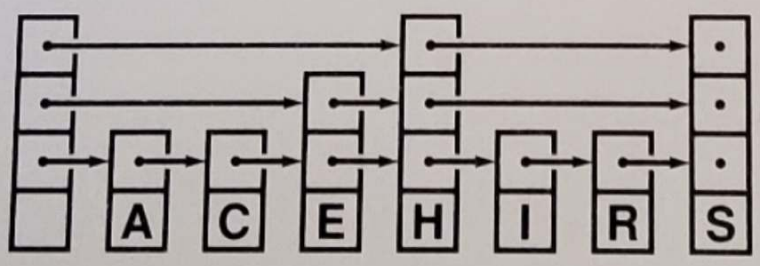

A
S

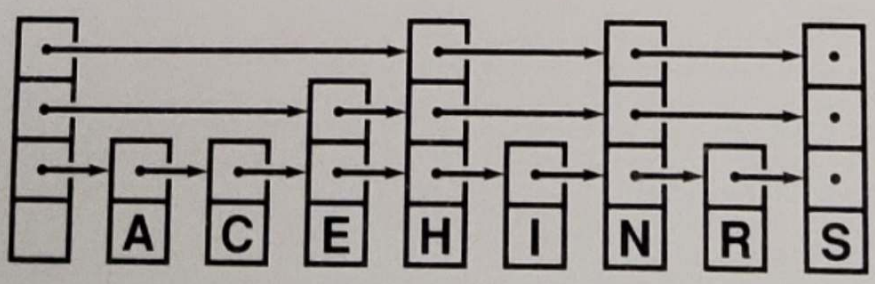

A
E
S

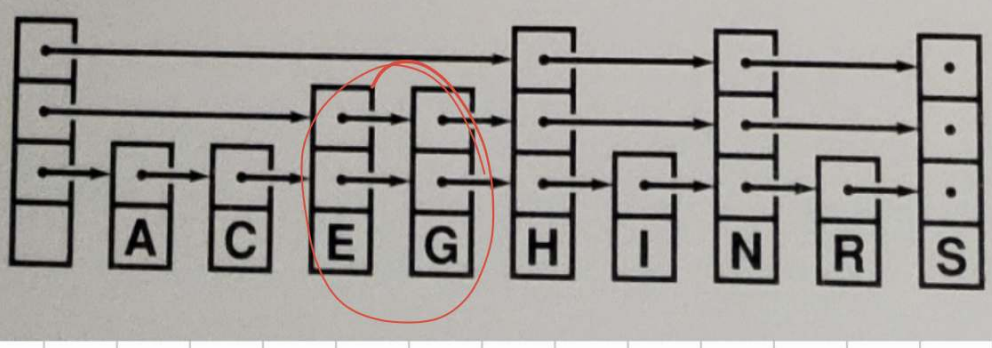

A
E
R
S


A
C
E
R
S


A
C
E
H
R
S


A
C
E
H
I
R
S


A
C
E
H
I
N
R
S


A
C
E
G
H
I
N
R
S

	construction						search misses					
N	B	T	R	S	C	L	B	T	R	S	C	L
1250	0	1	3	2	1	2	1	1	0	0	0	2
2500	2	4	6	3	1	4	1	1	1	2	1	3
5000	4	7	14	8	5	10	3	3	3	3	2	7
12500	11	23	43	24	16	28	10	9	9	9	7	18
25000	27	51	101	50	32	57	19	19	26	21	16	43
50000	63	114	220	117	74	133	48	49	60	46	36	98
100000	159	277	447	282	177	310	118	106	132	112	84	229
200000	347	621	996	636	411	670	235	234	294	247	193	523

Key:

B Standard BST(Program 12.7)
T BST built by root insertion (Program 12.12)
R Randomized BST (Program 13.2)
S Splay BST(Exercise 13.33 and Program 13.5)
C Red–black BST (Program 13.6)
L Skip list (Programs 13.7 and 13.9)