Δομές Δεδομένων

Μέρος 3ο: Λίστες

Εξάμηνο Σπουδών: 6ο

Κωδικός Μαθήματος: 681

Τμήμα Μαθηματικών
Πανεπιστήμιο Ιωαννίνων

Μιχάλης Α. Μπέκος
bekos@uoi.gr

ΑΤΔ: Λίστα

ADT: List

- Ο ΑΤΔ λίστα (list) μοντελοποιεί μια ακολουθία αντικειμένων.

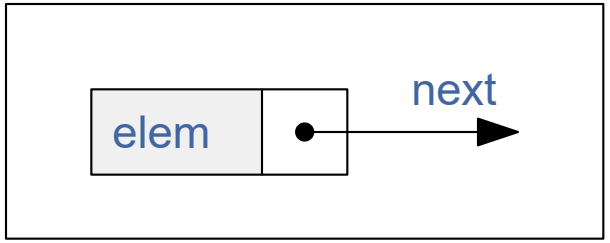
`empty()`	ελέγχει αν η λίστα είναι κενή	capacity
`size()`	επιστρέφει το πλήθος των στοιχείων της λίστας	capacity
`front()`	επιστρέφει το πρώτο στοιχείο της λίστας	access
`back()`	επιστρέφει το τελευταίο στοιχείο της λίστας	access
`push_front(e)`	ενθέτει το στοιχείο e ως πρώτο στη λίστα	modifiers
`push_back(e)`	ενθέτει το στοιχείο e ως τελευταίο στη λίστα	modifiers
`pop_front()`	διαγράφει το πρώτο στοιχείο της λίστα	modifiers
`pop_back()`	διαγράφει το τελευταίο στοιχείο της λίστα	modifiers
`insert(p,e)`	ενθέτει το στοιχείο e πριν τη θέση p της λίστας	modifiers
`erase(p)`	διαγράφει το στοιχείο στη θέση p της λίστας	modifiers

Μονά Συνδεδεμένες Λίστες

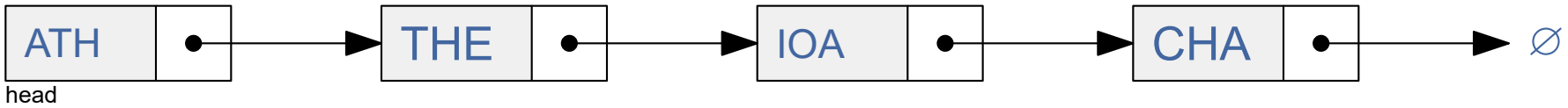
Singly linked lists

- Μια μονά συνδεδεμένη λίστα υλοποιεί τον ΑΤΔ λίστα. Αποτελεί μια συγκεκριμένη δομή δεδομένων, η οποία ορίζεται ως μια ακολουθία κόμβων.

- Κάθε κόμβος αποθηκεύει:
  - ένα στοιχείο
  - μια σύνδεση στον επόμενο κόμβο



- Παράδειγμα:



- Μια απλά συνδεδεμένη λίστα αποθηκεύει μια αναφορά (head) στον πρώτο κόμβο.

Υλοποίηση

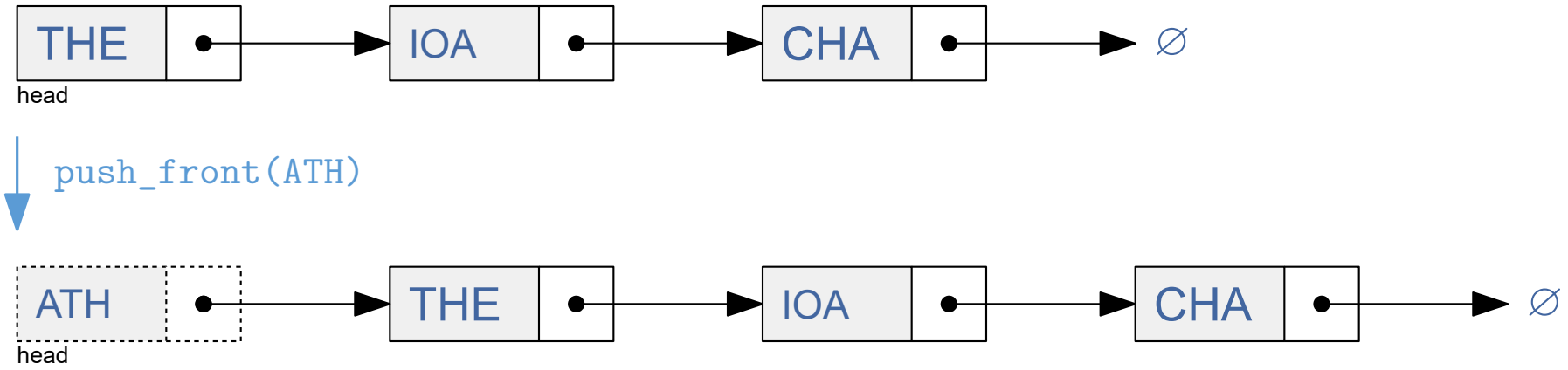
```
template <typename E>
class SLinkedList {
private:
  struct SNode { E elem;  SNode* next; };
  SNode* head;                      // head of list

public:
  SLinkedList();                    // constructor
  ~SLinkedList();                   // destructor
  bool empty() const;               // is empty?
  E& front() const;                 // return first element
  void push_front(const E& e);      // add as first element
  void pop_front();                 // remove first element
  ...
};
```

```
struct SNode {
  E elem;
  SNode* next;
};
```

Εισαγωγή Στοιχείου: `push_front(e)`

- Η ένθεση ενός νέου στοιχείου (ως πρώτο) στη λίστα περιλαμβάνει:
  - Τη δημιουργία νέου κόμβου
  - Την αποθήκευση του στοιχείου στον νέο κόμβο
  - Τη σύνδεση του νέου κόμβου με την κεφαλή της λίστας
  - Την ενημέρωση της αναφοράς head να δείχνει στον νέο κόμβο

- Παράδειγμα:



Εισαγωγή Στοιχείου: `push_front(e)`

- Η ένθεση ενός νέου στοιχείου (ως πρώτο) στη λίστα περιλαμβάνει:
  - Τη δημιουργία νέου κόμβου
  - Την αποθήκευση του στοιχείου στον νέο κόμβο
  - Τη σύνδεση του νέου κόμβου με την κεφαλή της λίστας
  - Την ενημέρωση της αναφοράς head να δείχνει στον νέο κόμβο

- Υλοποίηση:

```
void push_front(const E& e) {
  SNode* v = new SNode;
  v->elem = e;
  v->next = head;
  head = v;
}
```

Οι μέθοδοι `front()` και `empty()`

- `front()` επιστρέφει το πρώτο στοιχείο της λίστας

```
E& front() const {
  return head->elem;
}
```

- `empty()` ελέγχει αν η λίστα είναι κενή

```
bool empty() const {
  return head == NULL;    // requires cstdlib
}
```

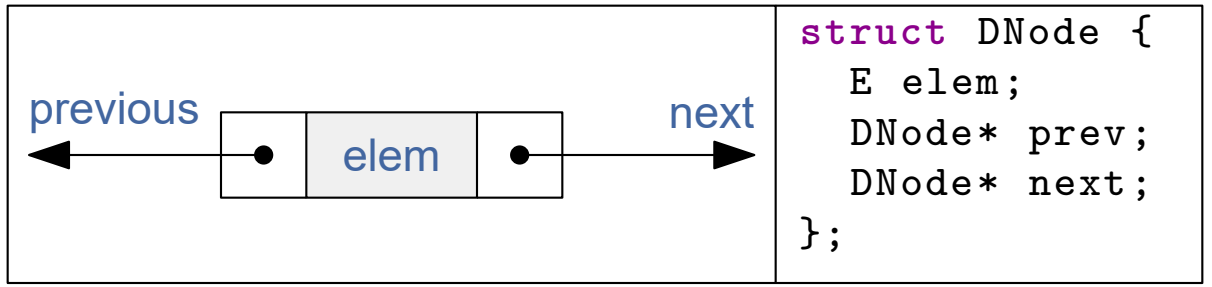
- Ερωτήσεις:
  - Πως θα υλοποιούσατε τη μέθοδο `size()` για τον υπολογισμό του μεγέθους της λίστας;
  - Πως θα υλοποιούσατε την ένθεση ενός νέου στοιχείου στο τέλος της λίστας;

Διπλά Συνδεδεμένες Λίστες

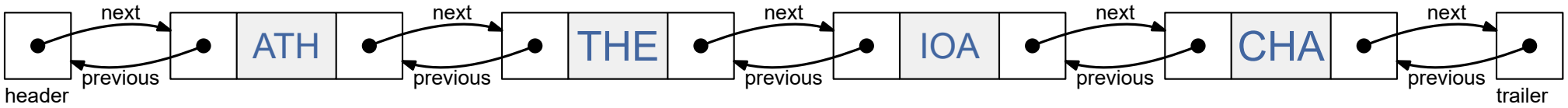
Doubly linked lists

- Μια διπλά συνδεδεμένη λίστα είναι μια επέκταση της μονά συνδεδεμένης λίστας, η οποία υλοποιεί τον ΑΤΔ λίστα καθιερώνοντας τη σχέση “προηγούμενος/επόμενος”’ μεταξύ των στοιχείων

- Κάθε κόμβος αποθηκεύει:
  - ένα στοιχείο
  - μια σύνδεση στον προηγούμενο κόμβο
  - μια σύνδεση στον επόμενο κόμβο



- Παράδειγμα:



- Μια διπλά συνδεδεμένη λίστα αποθηκεύει δύο αναφορές (header, trailer) στον πρώτο και τελευταίο κόμβο.

Διπλά Συνδεδεμένες Λίστες

Doubly linked lists

- Γενικές μέθοδοι:
  - `size()`, `empty()`
- Μέθοδοι ανανέωσης:
  - `push_front(e)`, `push_back(e)`
  - `pop_front()`, `pop_back()`
- Ανανεώσεις μέσω διαπροσπελαστών:
  - `insert(it,e)`, `erase(it)`
- Διαπροσπελαστές:

  Iterators
  - `begin()`, `end()`
- Διαπροσπελαστής: Ενθυλακώνει μια αναφορά σε ένα κόμβο (current) της λίστας

```
class Iterator {
private:
  DNode* current;
  Iterator(DNode* u);
public:
  E& operator*();      // ref. to the element
  Iterator& operator++();   // move forward
  Iterator& operator--();   // move backward
  bool operator==(const Iterator& it) const;
  bool operator!=(const Iterator& it) const;
  friend class DLinkedList;
};
```

Προσπέλαση μια λίστας

- Οι διαπροσπελαστές συνήθως χρησιμοποιούνται για την προσπέλαση των στοιχείων μια λίστας (ή γενικότερα μιας δομής δεδομένων)

- Παράδειγμα:

```
DLinkedList<std::string> list;
list.push_front("ATH");
list.push_front("THE");
list.push_front("IOA");
list.push_front("CHA");
```

```
DLinkedList<std::string>::Iterator it = list.begin();
while (it != list.end()) {
  std::cout << *it << " ";
  ++it;
}
```

Υλοποίηση ενός Διαπροσπελαστή

- Κατασκευαστής: δημιουργεί στιγμιότυπο το οποίο ενθυλακώνει τον δοθέν κόμβο

```
// private constructor
Iterator(DNode* u) {
  current = u;
}
```

Αντικείμενα τύπου `Iterator` μπορούν να κατασκευαστούν μόνο με χρήση των μεθόδων `begin()` και `end()` της κλάσης `DLinkedList`

- Τελεστής `*`: επιστρέφει το στοιχείο του τρέχοντος (current) κόμβου

```
E& operator*() {
  return current->elem;
}
```

Υλοποίηση ενός Διαπροσπελαστή

- Τελεστής ++: μετατοπίζει τον τρέχων (current) κόμβο στον επόμενο της λίστας

```
Iterator& operator++()
{
  current = current->next;
  return *this;
}
```

- Τελεστής ==: ελέγχει αν ο διαπροσπελαστής ενθυλακώνει το ίδιο στοιχείο με τον δοθέν

```
bool operator==(const Iterator& it) const {
  return current == it.current;
}
```

Οι τελεστές -- και != υλοποιούνται ομοίως

Διπλά Συνδεδεμένες Λίστες

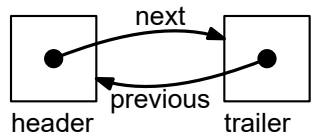
Doubly linked lists

```
class DLinkedList {
private:
  struct DNode { E elem; DNode* prev; DNode* next; };
  DNode *header,  *trailer;
  int numberOfElements;

public:
  DLinkedList(); virtual ~DLinkedList();                    //constructor - destructor

  class Iterator { ... }                                    // iterator declaration

  bool empty() const;  int size() const;                    // auxiliary methods

  E& front() const; E& back() const;                        // first/last elements
  void push_front(const E& e); void push_back(const E& e);
  void pop_front(); void pop_back();

  Iterator begin() const; Iterator end() const;             // Iterator- based methods
  void insert(const Iterator& it, const E& e);
  void erase(const Iterator& it);
}
```

Υλοποίηση μιας Διπλά Συνδεδεμένης Λίστας

- Κατασκευαστής: δημιουργεί μια κενή λίστα

```
DLinkedList() {
  header = new DNode;
  trailer = new DNode;
  header->next = trailer;
  trailer->prev = header;
  numberOfElements = 0;
}
```



- Οι μέθοδοι `empty()` και `size()`:

```
bool empty() const {
  return (numberOfElements == 0);
}

int size() const {
  return numberOfElements;
}
```

Υλοποίηση μιας Διπλά Συνδεδεμένης Λίστας

- `begin()`, `end()`: επιστέφουν διαπροσπελαστές οι οποίοι ενθυλακώνουν το πρώτο και το τελευταίο στοιχείο της λίστας, αντίστοιχα

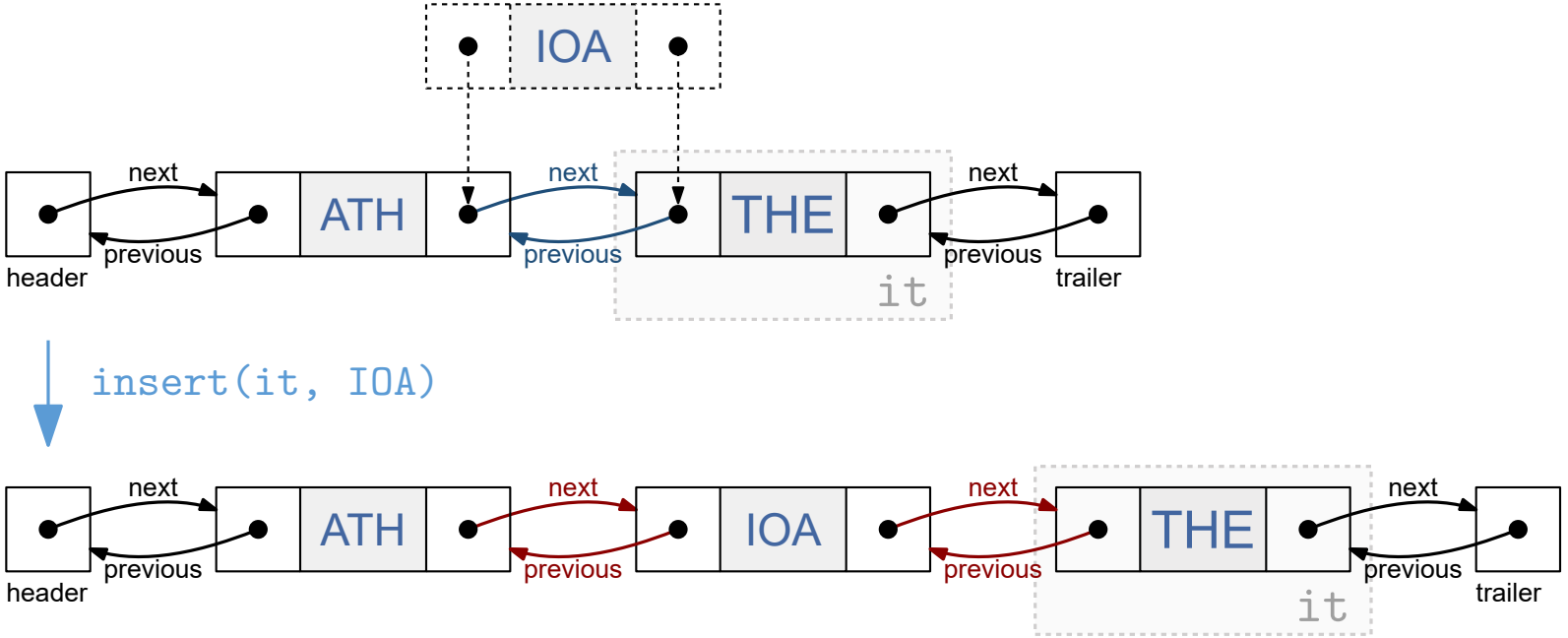
```
Iterator begin() const {
  return Iterator(header->next);
}

Iterator end() const {
  return Iterator(trailer);
}
```

- `front()`, `back()`: επιστέφουν το πρώτο και το τελευταίο στοιχείο της λίστας, αντίστοιχα

```
E& front() const {
  return header->next->elem;
}

E& back() const {
  return trailer->prev->elem;
}
```

Εισαγωγή Στοιχείου: `insert(it,e)`

- Η ένθεση ενός νέου στοιχείου (πριν τη θέση του διαπροσπελαστή) στη λίστα περιλαμβάνει:
  - Τη δημιουργία νέου κόμβου
  - Την αποθήκευση του στοιχείου στον νέο κόμβο
  - Την ενημέρωση των συνδέσεων των εμπλεκόμενων κόμβων
- Παράδειγμα:



Εισαγωγή Στοιχείου: `insert(it,e)`

- Η ένθεση ενός νέου στοιχείου (πριν τη θέση του διαπροσπελαστή) στη λίστα περιλαμβάνει:
  - Τη δημιουργία νέου κόμβου
  - Την αποθήκευση του στοιχείου στον νέο κόμβο
  - Την ενημέρωση των συνδέσεων των εμπλεκόμενων κόμβων

- Υλοποίηση:

```
void insert(const Iterator& it, const E& e) {
  DNode* w = it.current;     // it's node
  DNode* u = w->prev;        // it's predecessor
  DNode* v = new DNode;      // new node to insert
  v->elem = e;
  v->next = w; w->prev = v;    // link in v before w
  v->prev = u; u->next = v;    // link in v after u
  numberOfElements++;          // one more element
}
```

Εισαγωγή Στοιχείου: `insert(it,e)`

- Η ένθεση ενός νέου στοιχείου (πριν τη θέση του διαπροσπελαστή) στη λίστα περιλαμβάνει:
  - Τη δημιουργία νέου κόμβου
  - Την αποθήκευση του στοιχείου στον νέο κόμβο
  - Την ενημέρωση των συνδέσεων των εμπλεκόμενων κόμβων

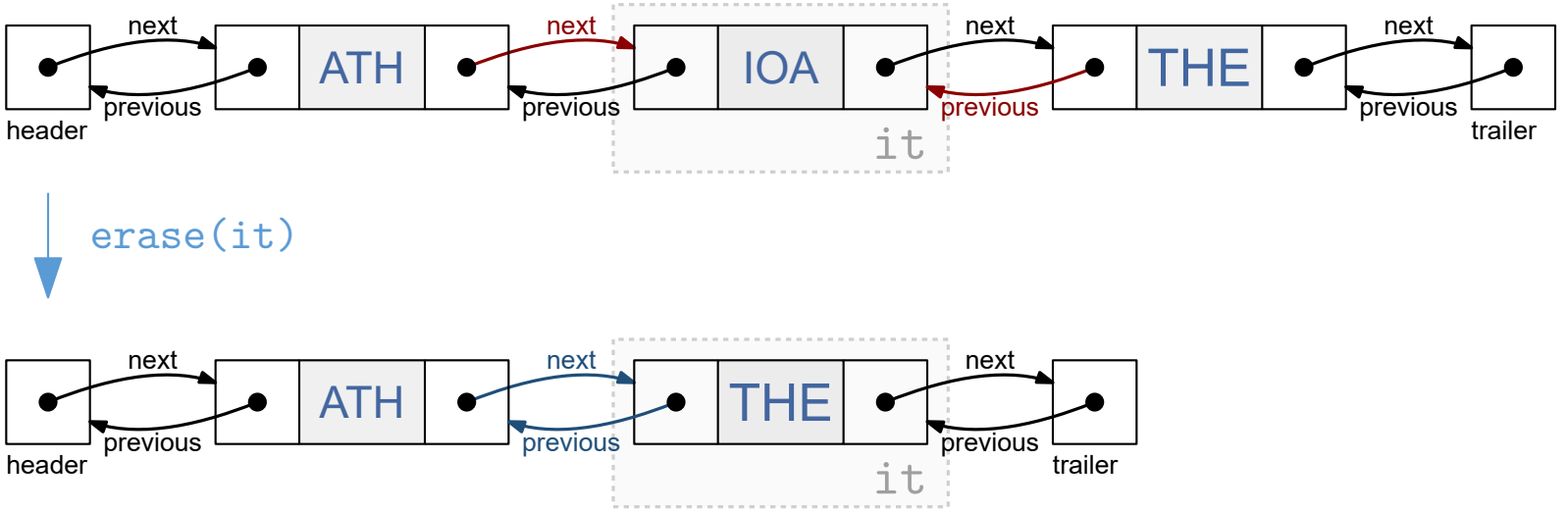
- Υλοποίηση:

```
void push_front(const E& e) {
    insert(begin(), e);
}

void push_back(const E& e) {
    insert(end(), e);
}
```

Διαγραφή Στοιχείου: `erase(it)`

- Η διαγραφή ενός στοιχείου (της θέση του διαπροσπελαστή) στη λίστα περιλαμβάνει:
  - Την ενημέρωση των συνδέσεων του προηγούμενο κόμβου
  - Την ενημέρωση των συνδέσεων του επόμενου κόμβου
  - Την απομάκρυνση του προς-διαγραφή κόμβου από τη λίστα

- Παράδειγμα:



Διαγραφή Στοιχείου: `erase(it)`

- Η διαγραφή ενός στοιχείου (της θέση του διαπροσπελαστή) στη λίστα περιλαμβάνει:
  - Την ενημέρωση των συνδέσεων του προηγούμενο κόμβου
  - Την ενημέρωση των συνδέσεων του επόμενου κόμβου
  - Την απομάκρυνση του προς-διαγραφή κόμβου από τη λίστα

- Υλοποίηση:

```
void erase(const Iterator& it) {
  DNode* v = it.current;          // node to remove
  DNode* w = v->next;             // successor
  DNode* u = v->prev;             // predecessor
  u->next = w; w->prev = u;       // unlink p
  delete v;                       // delete this node
  numberOfElements--;             // one fewer element
}
```

Διαγραφή Στοιχείου: `erase(it)`

- Η διαγραφή ενός στοιχείου (της θέση του διαπροσπελαστή) στη λίστα περιλαμβάνει:
  - Την ενημέρωση των συνδέσεων του προηγούμενο κόμβου
  - Την ενημέρωση των συνδέσεων του επόμενου κόμβου
  - Την απομάκρυνση του προς-διαγραφή κόμβου από τη λίστα

- Υλοποίηση:

```
void pop_front() {
  erase(begin());
}

void pop_back() {
   erase(--end());
}
```

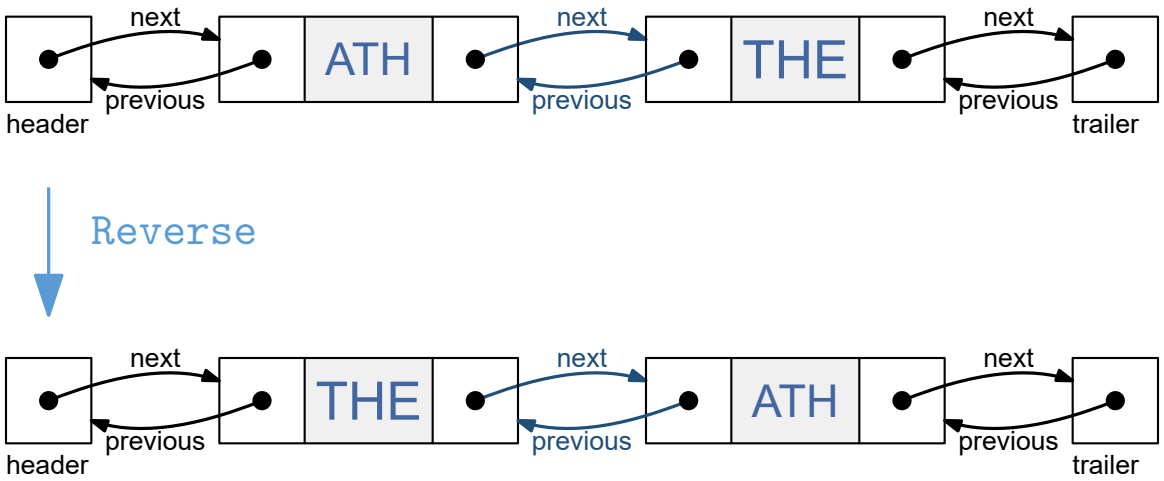
Σημειώσεις

- Στην υλοποίηση της διπλά συνδεδεμένης λίστας:
  - Ο χώρος που χρησιμοποιείται για την αποθήκευση n στοιχείων είναι $\mathcal{O}(n)$
  - Ο χώρος που χρησιμοποιείται για την αποθήκευση κάθε στοιχείου είναι $\mathcal{O}(1)$
  - Όλες οι μέθοδοι της λίστας εκτελούνται σε χρόνο $\mathcal{O}(1)$
  - Αποφύγαμε τη συζήτηση εκφυλισμένων περιπτώσεων

    π.χ. τη διαγραφή ενός στοιχείου από μια κενή λίστα

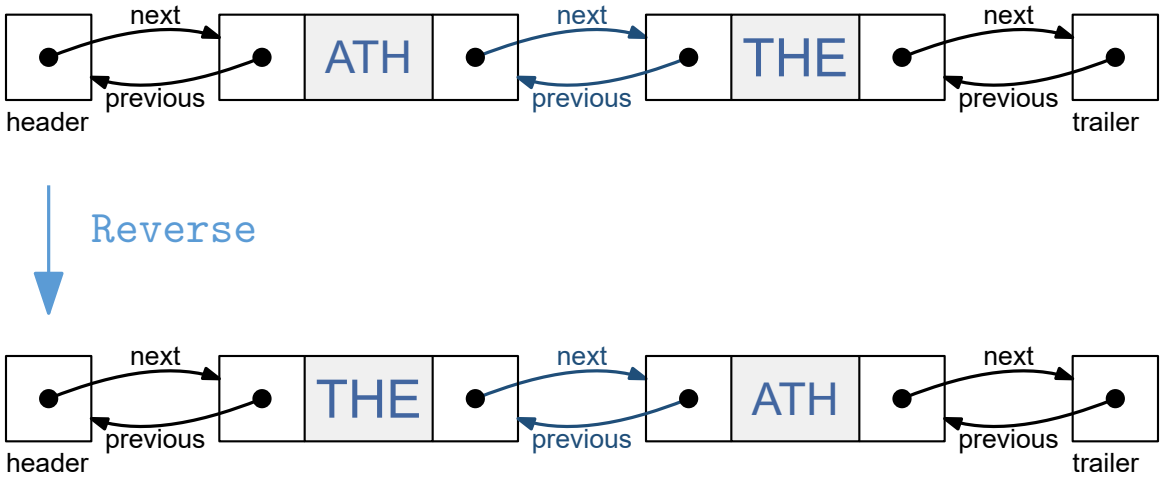
Ερωτήσεις

- Πώς θα αντιστρέφατε μια λίστα?



Ερωτήσεις

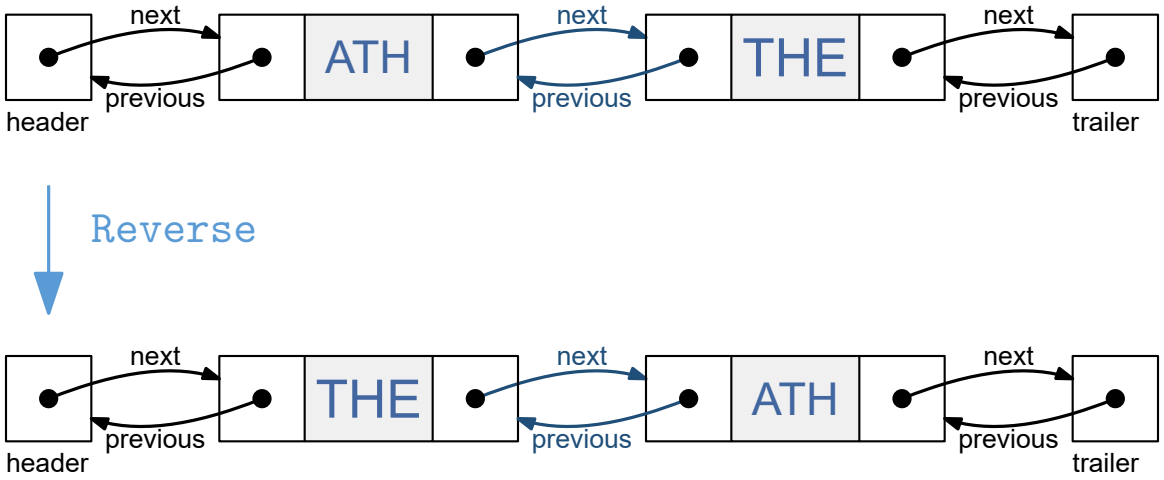
- Πώς θα αντιστρέφατε μια λίστα?




```
template <typename E>
void listReverse(DLinkedList<E>& L) {
  DLinkedList<E> T;
  while (!L.empty()) {
    E e = L.front();
    L.pop_front();
    T.push_front(e);
  }
  while (!T.empty()) {
    E e = T.front();
    T.pop_front();
    L.push_back(e);
  }
}
```

Ερωτήσεις

- Πώς θα αντιστρέφατε μια λίστα?




```
template <typename E>
void listReverse(DLinkedList<E>& L) {
  DLinkedList<E> T;
  while (!L.empty()) {
    E e = L.front();
    L.pop_front();
    T.push_front(e);
  }
  while (!T.empty()) {
    E e = T.front();
    T.pop_front();
    L.push_back(e);
  }
}
```

- Πώς θα παραθέτατε δύο λίστες?

`L1.append(it, L2)`

ΑΤΔ: Λίστα

	Singly Linked List	Doubly Linked List
`empty()`	$\mathcal{O}(1)$	$\mathcal{O}(1)$
`size()`	$\mathcal{O}(1)$	$\mathcal{O}(1)$
`front()`	$\mathcal{O}(1)$	$\mathcal{O}(1)$
`back()`	$\mathcal{O}(n)$	$\mathcal{O}(1)$
`push_front(e)`	$\mathcal{O}(1)$	$\mathcal{O}(1)$
`push_back(e)`	$\mathcal{O}(n)$	$\mathcal{O}(1)$
`pop_front()`	$\mathcal{O}(1)$	$\mathcal{O}(1)$
`pop_back()`	$\mathcal{O}(n)$	$\mathcal{O}(1)$
`insert(p,e)`	$\mathcal{O}(n)^*$	$\mathcal{O}(1)^*$
`erase(p)`	$\mathcal{O}(n)^*$	$\mathcal{O}(1)^*$

*Υποθέτουμε υλοποιήσεις μέσω διαπροσπελαστών

STD Lists

```
#include <iostream>
#include <list>

int main ()
{
  std::list<int> list;
  std::list<int>::iterator it;

  // set some initial values:
  for (int i=1; i<=5; ++i) list.push_back(i); // 1 2 3 4 5

  it = list.begin();
  list.insert(it,0);                          // 0 1 2 3 4 5

  for (it=list.begin(); it!=list.end(); ++it)
    std::cout << ' ' << *it;
  std::cout << std::endl;
}
```

---

https://www.cplusplus.com/reference/list/list/insert/

Εισαγωγή στις Λίστες Παράλειψης

- Υπενθύμιση: Αν ένας πίνακας n στοιχείων είναι ταξινομημένος, τότε μπορούμε να αποφασίσουμε αν ένα δοθέν στοιχείο x ανήκει στον πίνακα σε χρόνο $\mathcal{O}(\log n)$

  $\rightarrow$ δυαδική αναζήτηση

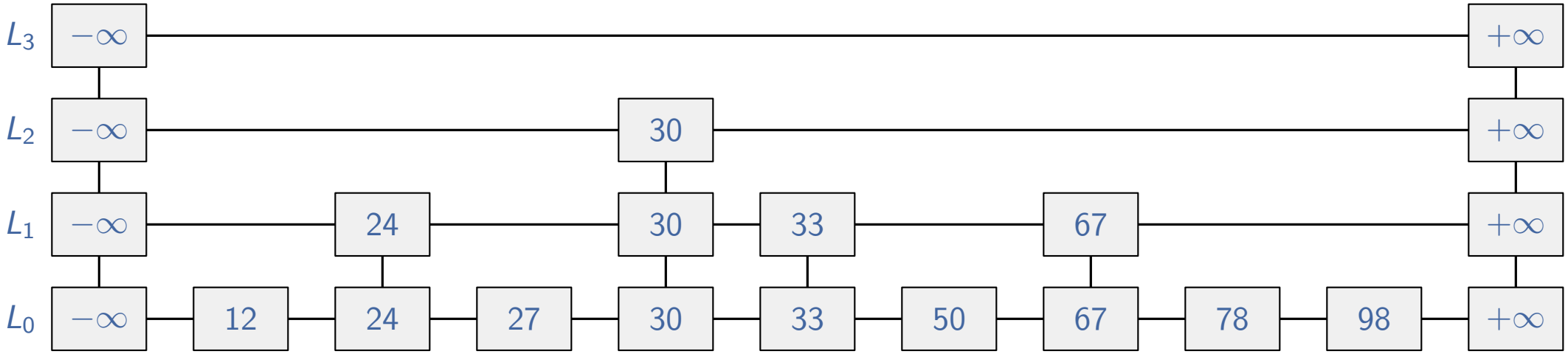
- Ερώτημα: Μπορούμε να αναπτύξουμε αντίστοιχη στρατηγική αν τα n στοιχεία βρίσκονται ταξινομημένα σε μία λίστα;

  $\rightarrow$ λίστες παράλειψης

Λίστες Παράλειψης

Skip lists

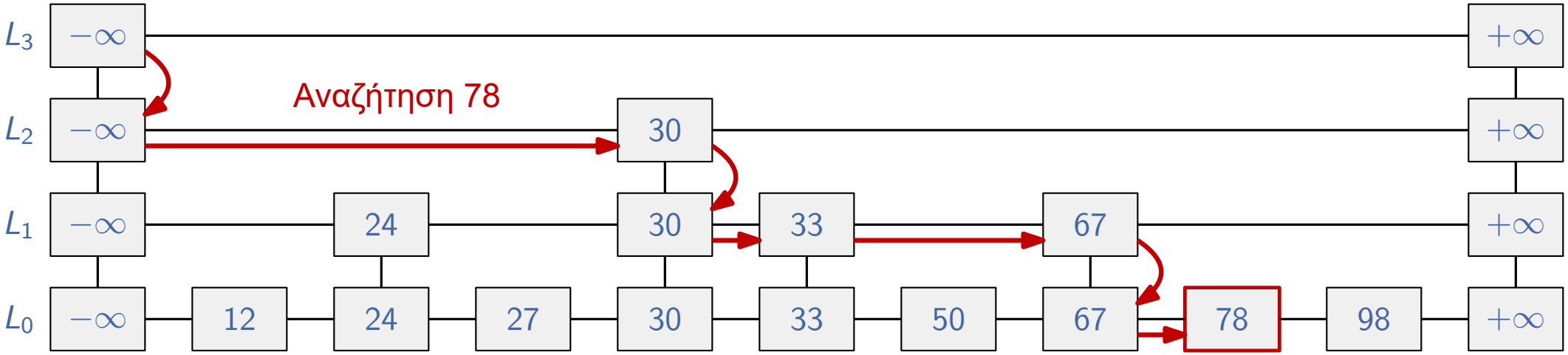
- Μια λίστα παράλειψης ενός πεπερασμένου συνόλου S διακριτών στοιχείων είναι μια ακολουθία λιστών $L_0, L_1, \ldots, L_h$, έτσι ώστε:
  - κάθε λίστα L_i περιέχει δύο ιδιαίτερα στοιχεία: $-\infty$ και $+\infty$
  - η λίστα L_0 περιέχει όλα τα στοιχεία του συνόλου S σε αύξουσα διάταξη
  - η λίστα L_i είναι υπολίστα της $L_{i-1} \Rightarrow L_0 \supseteq L_1 \supseteq \ldots \supseteq L_h$
  - η λίστα L_h περιέχει μόνο τα δύο ιδιαίτερα στοιχεία: $-\infty$ και $+\infty$



Αναζήτηση σε μια Λίστα Παράλειψης

Search in a skip list

- Η αναζήτηση ενός στοιχείου x σε μια λίστα παράλειψης γίνεται ως εξής:
  - Ξεκινάμε από το πρώτο στοιχείο της "ανώτατης" λίστας L_h
  - Στην τρέχουσα θέση `p`, συγκρίνουμε το στοιχείο `x` με το στοιχείο `y` της θέσης `p.next()`
    - → `x = y`: Επιστρέφουμε τη θέση `p.next()`
    - → `x > y`: Συνεχίζουμε την αναζήτηση "προς τα εμπρός"
    - → `x < y`: Συνεχίζουμε την αναζήτηση "προς τα κάτω"
  - Αν η αναζήτηση πρέπει να συνεχιστεί κάτω από την "κατώτατη" λίστα L_0, επιστρέφουμε `NULL`



Τυχαιοκρατικοί Αλγόριθμοι

Randomized algorithms

- Η εισαγωγή στοιχείων σε μια λίστα παράλειψης γίνεται μέσω ενός τυχαιοκρατικού αλγορίθμου
- Ορισμός: Ένας τυχαιοκρατικός αλγόριθμος εκτελεί ρίψεις νομισμάτων κατά την εκτέλεση του (σ.σ., χρησιμοποιεί τυχαία bits)
- Παράδειγμα:

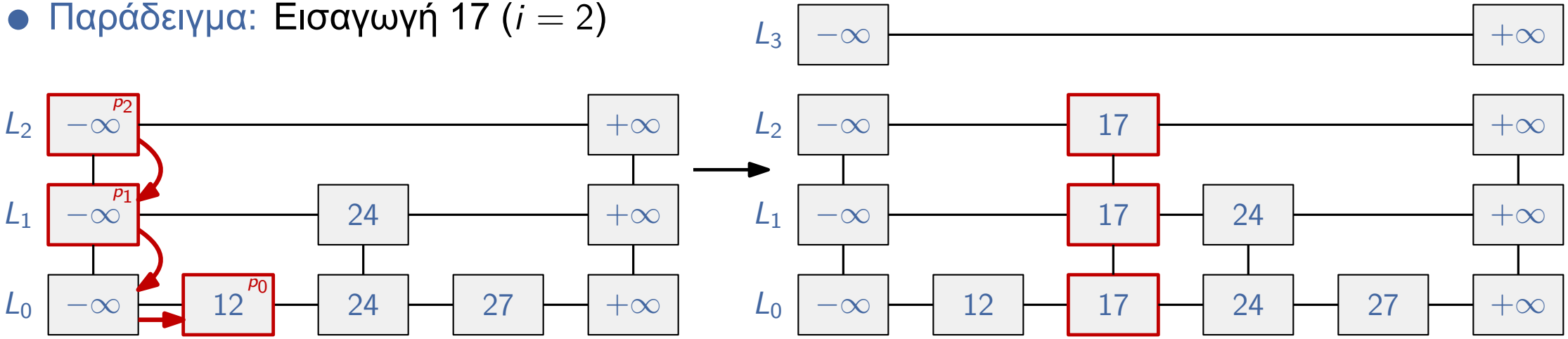
```
1 x ← random();
2 if x ≥ 0 then
3 |   methodA();
4 else
5 |   methodB();
```

- Τόσο ο χρόνος εκτέλεσης όσο και η έξοδος ενός τυχαιοκρατικού αλγορίθμου εξαρτάται από τις ρίψεις των νομισμάτων
- Αναλύουμε τον αναμενόμενο χρόνο εκτέλεσης ενός τυχαιοκρατικού αλγορίθμου υποθέτοντας:
  - ότι τα νομίσματα είναι αμερόληπτα
  - οι ρίψεις των νομισμάτων είναι ανεξάρτητες

Εισαγωγή σε μια Λίστα Παράλειψης

Insertion in a skip list

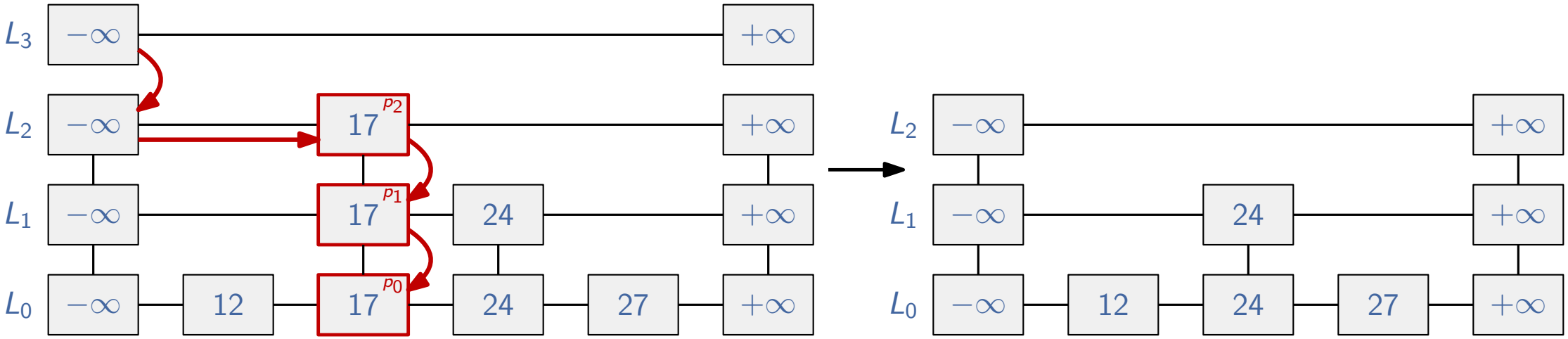
- Η εισαγωγή ενός στοιχείου x σε μια λίστα παράλειψης γίνεται
  - Ρίχνουμε επανειλημμένα ένα νόμισμα μέχρι να φέρουμε “γράμματα”
  - $i \leftarrow$ το πλήθος των ρίψεων που κατέληξαν σε “κορώνα”
  - Αν $i \geq h$, προσθέτουμε στην λίστα παράλειψης νέες λίστες $L_{h+1}, \ldots, L_{i+1}$ (με τα στοιχεία $\pm\infty$)
  - Αναζητούμε το στοιχείο x στην λίστα παράλειψης και εντοπίζουμε τις θέσεις $p_0, p_1, \ldots, p_i$ των μεγιστικών στοιχείων τα οποία είναι μικρότερα του x στις λίστες $L_0, L_1, \ldots, L_i$.
  - Για κάθε $j = 0, 1, \ldots, i$, εισάγουμε το στοιχείο x στην λίστα L_j μετά τη θέση p_j.
- Παράδειγμα: Εισαγωγή 17 ($i = 2$)



Διαγραφή από μια Λίστα Παράλειψης

Deletion from a skip list

- Η διαγραφή ενός στοιχείου x από μια λίστα παράλειψης γίνεται ως εξής:
  - Αναζητούμε το στοιχείο x στη λίστα παράλειψης
  - Έστω $p_0, p_1, \ldots, p_i$ οι θέσεις που εμφανίζεται το στοιχείο x στις λίστες $L_0, L_1, \ldots, L_i$
  - Διαγράφουμε τις θέσεις $p_0, p_1, \ldots, p_i$ από τις λίστες $L_0, L_1, \ldots, L_i$
  - Διαγράφουμε όλες τις λίστες που περιέχουν μόνο τα στοιχεία $-\infty$ και $+\infty$, εκτός από μια
- Παράδειγμα: Διαγραφή 17



Παρατηρήσεις

- Η εισαγωγή, η αναζήτηση και η διαγραφή ενός στοιχείου x από μια λίστα παράλειψης εξαρτάται γραμμικά από:
  - το ύψος της λίστας παράλειψης, σ.σ., το πλήθος των λιστών που την συναποτελούν
  - το πλήθος των "προς τα εμπρός" μετακινήσεων που εκτελούνται σε καθεμία από τις λίστες που συναποτελούν τη λίστα παράλειψης

- Στην συνέχεια, δείχνουμε ότι:
  - με μεγάλη πιθανότητα, μια λίστα παράλειψης με n στοιχεία έχει ύψος το πολύ $\mathcal{O}(\log n)$
  - το αναμενόμενο πλήθος των "προς τα εμπρός" μετακινήσεων ανά λίστα είναι $\mathcal{O}(1)$

- Οι πράξεις εισαγωγής, αναζήτησης και διαγραφής έχουν αναμενόμενη πολυπλοκότητα $\mathcal{O}(\log n)$

Αναμενόμενος Αριθμός Κόμβων μιας Λίστας Παράλειψης

Expected size usage of a skip list

- Θεώρημα: Ο αναμενόμενος αριθμός κόμβων μιας λίστας παράλειψης n στοιχείων είναι $\mathcal{O}(n)$

  Απόδειξη:

  - Γεγονός 1: η πιθανότητα του ενδεχομένου να έρθει i φορές συνεχόμενα "κορώνα" είναι $\frac{1}{2^i}$
  - Γεγονός 2: δοθέντος n στοιχείων, καθένα από τα οποία ανήκει σε ένα σύνολο με πιθανότητα p, το αναμενόμενο μέγεθος του συνόλου είναι np
  - Το Γεγονός 1 συνεπάγεται ότι ένα στοιχείο βρίσκεται στη λίστα L_i με πιθανότητα $\frac{1}{2^i}$
  - Το Γεγονός 2 συνεπάγεται ότι το αναμενόμενο μέγεθος της λίστας L_i είναι $\frac{n}{2^i}$
  - Συνολικά: Ο αναμενόμενος αριθμός κόμβων της λίστας παράλειψης είναι:

$$\sum_{i=0}^{h} \frac{n}{2^i} \leq n \cdot \sum_{i=0}^{\infty} \frac{1}{2^i} = 2n$$

Το ύψος μιας Λίστας Παράλειψης

The height of a skip list

- Θεώρημα: Μια λίστα παράλειψης n στοιχείων έχει ύψος το πολύ $3 \log n$ με πιθανότητα τουλάχιστον $1 - \frac{1}{n^2}$

Απόδειξη:

- Γεγονός 1: η πιθανότητα του ενδεχομένου να έρθει i φορές συνεχόμενα "κορώνα" είναι $\frac{1}{2^i}$
- Γεγονός 3: δοθέντος n ενδεχομένων, καθένα από τα οποία έχει πιθανότητα p, η πιθανότητα να συμβεί τουλάχιστον ένα είναι το πολύ np

- Το Γεγονός 1 συνεπάγεται ότι ένα στοιχείο βρίσκεται στη λίστα L_i με πιθανότητα $\frac{1}{2^i}$
- Το Γεγονός 3 συνεπάγεται ότι η πιθανότητα η λίστα L_i να έχει τουλάχιστον ένα στοιχείο είναι $\frac{n}{2^i}$
- Θέτοντας $i = 3 \log n$, η πιθανότητα η λίστα $L_{3 \log n}$ να έχει τουλάχιστον ένα στοιχείο είναι:

$$\frac{n}{2^{3 \log n}} = \frac{n}{n^3} = \frac{1}{n^2}$$

“Προς τα εμπρός” Μετακινήσεις

- Θεώρημα: Το αναμενόμενο πλήθος των “προς τα εμπρός“ μετακινήσεων ανά λίστα είναι $\mathcal{O}(1)$

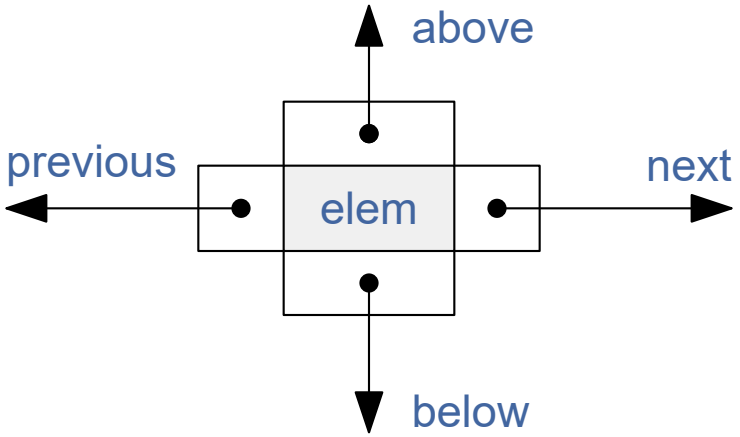
  Απόδειξη:

  - Ιδιότητα 1: σε μια “προς τα εμπρός” μετακίνηση, το στοιχείο-προορισμός δεν ανήκει σε υψηλότερη λίστα
  - Γεγονός 4: ο αναμενόμενος αριθμός ρίψεων ενός νομίσματος μέχρι την εμφάνισης της ένδειξης “γράμματα” είναι 2
  - Η Ιδιότητα 1 συνεπάγεται ότι κάθε “προς τα εμπρός” μετακίνηση μπορεί να συσχετιστεί με μια προηγούμενη ρίψη που κατέληξε σε “γράμματα”
  - Το Γεγονός 4 συνεπάγεται ότι το αναμενόμενο πλήθος των “προς τα εμπρός“ μετακινήσεων ανά λίστα είναι 2

Υλοποίηση

- Για την υλοποίηση μιας λίστας παράλειψης χρησιμοποιούμε ένα τετραμερή κόμβο

- Ένας τετραμερής κόμβος στη λίστα L_i αποθηκεύει:
  - ένα στοιχείο
  - μια σύνδεση στον επόμενο κόμβο της λίστας L_i
  - μια σύνδεση στον προηγούμενο κόμβο της λίστας L_i
  - μια σύνδεση στον “από κάτω” κόμβο της λίστας L_{i-1}
  - μια σύνδεση στον “από πάνω” κόμβο της λίστας L_{i+1}



Σύνοψη

- Σε μια λίστα παράλειψης n στοιχείων:
  - ο αναμενόμενος αριθμός κόμβων είναι $\mathcal{O}(n)$
  - οι πράξεις εισαγωγής, αναζήτησης και διαγραφής έχουν αναμενόμενη πολυπλοκότητα $\mathcal{O}(\log n)$
- Χρησιμοποιώντας μια πιο προχωρημένη πιθανοθεωρητική ανάλυση, μπορεί κάποιος να δείξει ότι τα παραπάνω όρια απόδοσης ισχύουν επίσης με μεγάλη πιθανότητα
- Οι λίστες παράλειψης είναι ιδιαίτερα αποδοτικές και απλές στην υλοποίηση

Επιπλέον υλικό

- Ενότητες 2.2, 19.6:
  Michael T. Goodrich, Roberto Tamassia, Αλγόριθμοι σχεδίαση και εφαρμογές
  ISBN: 9789605126971, εκδόσεις Γκιούρδα, 2016.