ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN

CHALLENGE 1: DANH SÁCH ĐA LIÊN KẾT

Môn: Cấu trúc dữ liệu và giải thuật

Thành phố Hồ Chí Minh - 2021

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN

CHALLENGE 1: DANH SÁCH ĐA LIÊN KẾT

GVHD: Thầy Văn Chí Nam – Thầy Bùi Huy Thông

Mã nhóm: 366_472_473

Danh sách sinh viên thực hiện:

1. Phạm Phú Hoàng Sơn _ 20120366
2. Thái Ngọc Vinh Hiển _ 20120472
3. Dương Minh Hiếu _ 20120473

Thành phố Hồ Chí Minh – 2021

MỤC LỤC

MỤC LỤC .....

DANH MỤC HÌNH .....

DANH MỤC BẢNG .....

Phần 1: Tìm Hiểu 1

1. Sorted Singly Linkedlist 1

1.1. Định nghĩa 1

1.2. Một số thao tác với Sorted Singly Linkedlist 1

2. Skip List 3

2.1. Định nghĩa 3

2.2. Một số thao tác với Skip List 4

2.3. Ứng dụng thực tế của Skip list 7

3. Indexable Skiplist 7

3.1. Định nghĩa 7

3.2. Một số thao tác với Indexable Skiplist 7

Phần 2: Trả lời câu hỏi 9

1. Câu a 9

2. Câu b 10

3. Câu c 10

Phần 3: Trả lời về độ phức tạp Big O 11

1. Câu a 11

2. Câu b 11

3. Câu c 12

Phần 4: Trình bày hướng giải quyết ở phần 1.2 13

TÀI LIỆU THAM KHẢO 15

DANH MỤC HÌNH

Hình 1: Minh họa Sorted Singly Linkedlist..... 1

Hình 2: Minh họa Skip list..... 3

Hình 3: Ví dụ tìm số 17 trong Skiplist..... 4

Hình 4: Ví dụ chèn số 17 trong Skip list 5

Hình 5: Ví dụ xóa số 6 khỏi Skip list..... 6

Hình 6: Minh họa Indexable Skiplist..... 7

Hình 7: Thêm 1 giá trị trong Indexable skiplist..... 8

Hình 8: Thêm 1 giá trị trong Indexable skiplist..... 8

Hình 9: Xóa phần tử trong indexable skiplist..... 9

Hình 10: Ý tưởng xây dựng thuật toán xác định level..... 10

DANH MỤC BẢNG

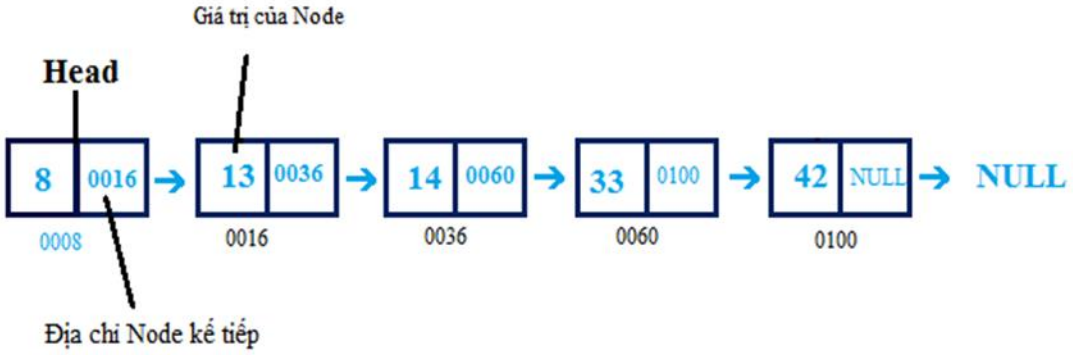
Bảng 1: Bảng BigO..... 11

Phần 1: Tìm Hiểu

1. Sorted Singly Linkedlist

1.1. Định nghĩa

Sorted Singly Linkedlist (SSL) là một cấu trúc lưu trữ dữ liệu có thứ tự. Sorted Linklist gồm các Node được cấp phát động, mỗi Node chứa dữ liệu và địa chỉ Node kế tiếp. Các thao tác trên SSL chỉ có thể thay đổi số lượng phần tử mà không làm mất tính thứ tự.



Hình 1: Minh họa Sorted Singly Linkedlist

1.2. Một số thao tác với Sorted Singly Linkedlist

Với kiểu số nguyên ta cài đặt cấu trúc SSL như sau:

```
struct Node {
      int data;
      Node* next;
};

struct Slist {
      Node* head;

}
```

Tìm kiếm một giá trị cho trước

- Mã giả

```
Node* findElement(Slist s, int key) {
    Node* temp = s.head;
    // Duyệt phần tử đến khi nào tìm thấy hoặc đã lớn hơn key
    while (temp != NULL && temp->data <= key) {
        if (temp->data == key)
            return temp;
        temp = temp->next;
    }
    return NULL;
}
```

Thêm một giá trị cho trước

- Mã giả

```
void addElement(Slist& s, int key) {
    Node* new_element = createNode(key);
    // Sử dụng Node fake để không phải xét rỗng hoặc thêm vào đầu
    Node* fake = new Node;
    fake->next = s.head;
    Node* temp = fake;
    // Tìm Node có data > key để chèn vào trước Node đó
    while (temp->next != NULL) {
        if (key <= temp->next->data)
        break;
        temp = temp->next;
    }
    new_element->next = temp->next;
    temp->next = new_element;
    s.head = fake->next; // Cập nhật lại head
}
```

Xóa một giá trị cho trước

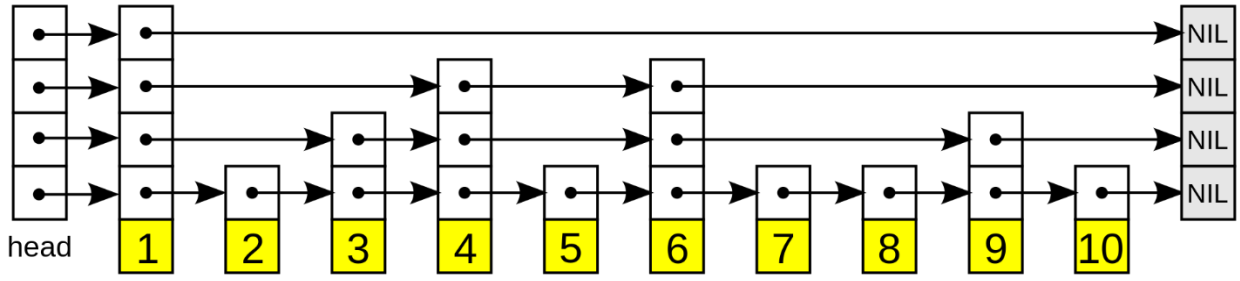
```
bool deleteElement(Slist& s, int key) {
    Node* temp = s.head;
    if (key == temp->data) {
            s.head = s.head->next;
            delete temp;
            return true;
    }

    while (temp->next != NULL && temp->next->data <= key) {
            if (temp->next->data == key) {
                    Node* delete_item = temp->next;
                    temp->next = temp->next->next;
                    delete delete_item;
                    return true;
            }
            temp = temp->next;
    }
    return false;
}
```

2. Skip List

2.1. Định nghĩa

Skip List là một cấu trúc dữ liệu được xây dựng bằng nhiều tầng Sorted Linked List một cách ngẫu nhiên, trong đó tầng cao chứa những bước nhảy dài hơn và tầng thấp chứa những bước nhảy ngắn hơn. Skip List cho phép ta thực hiện thao tác tìm kiếm với độ phức tạp xấp xỉ **O(log(N)).**



Hình 2: Minh họa Skip list

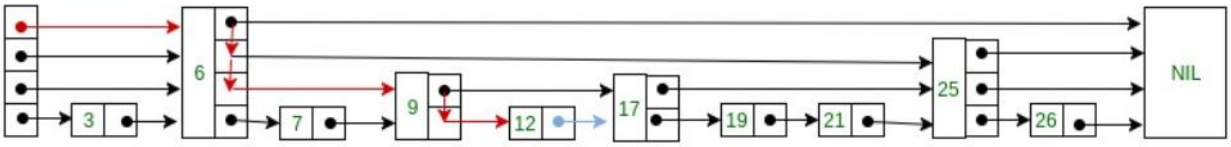
2.2. Một số thao tác với Skip List

Tìm kiếm một giá trị cho trước

- Ý tưởng
  - Nếu giá trị tại node tiếp theo mà nhỏ hơn giá trị node đang tìm kiếm thì chúng ta sẽ tiếp tục tiến đến node tiếp theo trên cùng một tầng.
  - Nếu giá trị tại node tiếp theo mà lớn hơn giá trị node đang tìm kiếm thì ta dừng lại tại vị trí **i** đang đứng rồi di chuyển xuống một tầng và tiếp tục tìm kiếm.
- Mã giả

```
Search(list, searchKey)
x := list -> header
-- loop invariant: x -> key  level downto 0 do
    while x -> forward[i] -> key  forward[i]
x := x -> forward[0]
if x -> key = searchKey then return x -> value
else return failure
```

- Ví dụ tìm số 17 trong **Skip list**



Hình 3: Ví dụ tìm số 17 trong Skiplist

Thêm một giá trị cho trước

- Ý tưởng

Việc thêm một giá trị vào **Skip list** sẽ làm thay đổi cấu trúc của nó. Vì vậy, các liên kết ở mỗi tầng không nhất thiết phải chuẩn. Để thuật toán vẫn chạy tốt và không mất quá nhiều lần nhảy ở mỗi tầng thì ta sẽ sử dụng thuật toán random như sau:

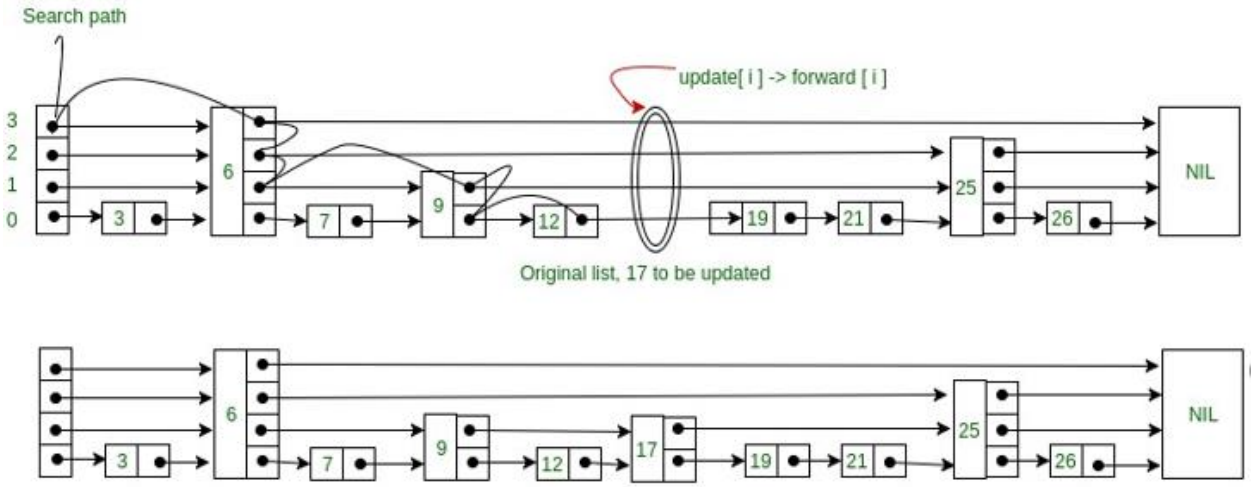
```
randomLevel()
lvl := 1
//random() that returns a random value in [0...1)
while random() < p and lvl < MaxLevel do
lvl := lvl + 1
return lvl
```

Chúng ta sẽ bắt đầu với tầng lớn nhất trong **Skip List** và so sánh với các giá trị ở node tiếp theo như *"Tìm kiếm một giá trị cho trước"* nêu ở trên.

- Mã giả

```
Insert(list, searchKey)
local update[0...MaxLevel+1]
x := list -> header
for i := list -> level downto 0 do
    while x -> forward[i] -> key  forward[i]
update[i] := x
x := x -> forward[0]
lvl := randomLevel()
if lvl > list -> level then
for i := list -> level + 1 to lvl do
    update[i] := list -> header
    list -> level := lvl
x := makeNode(lvl, searchKey, value)
for i := 0 to level do
    x -> forward[i] := update[i] -> forward[i]
    update[i] -> forward[i] := x
```

- Ví dụ chèn số 17 trong **Skip List**



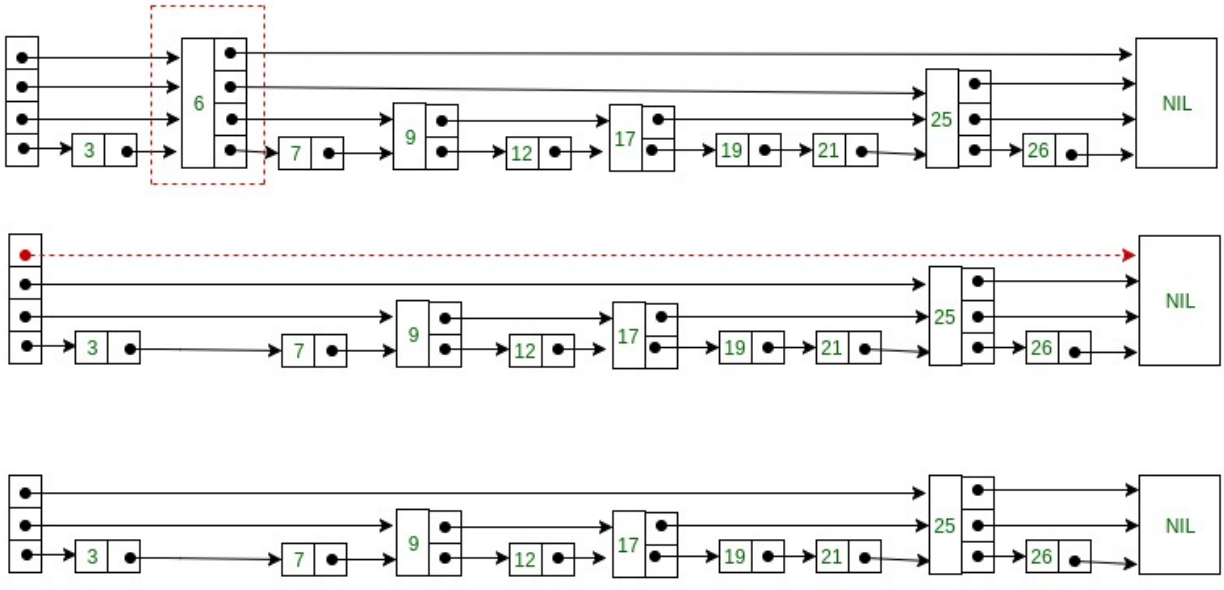
Hình 4: Ví dụ chèn số 17 trong Skip list

Xóa một giá trị cho trước

- Ý tưởng
  - Chúng ta dựa trên thuật toán tìm phần tử ở trên để xác định vị trí của phần tử cần xóa
  - Sau đó xóa phần tử đã tìm thấy giống như danh sách liên kết đơn.
- Mã giả

```
Delete(list, searchKey)
local update[0..MaxLevel+1]
x := list -> header
for i := list -> level downto 0 do
    while x -> forward[i] -> key  forward[i]
    update[i] := x
x := x -> forward[0]
if x -> key = searchKey then
    for i := 0 to list -> level do
        if update[i] -> forward[i] ≠ x then break
        update[i] -> forward[i] := x -> forward[i]
    free(x)
    while list -> level > 0 and list -> header -> forward[list -> level] = NIL do
        list -> level := list -> level – 1
```

- Ví dụ xóa số 6 khỏi **Skip list**



Hình 5: Ví dụ xóa số 6 khỏi Skip list

2.3. Ứng dụng thực tế của Skip list

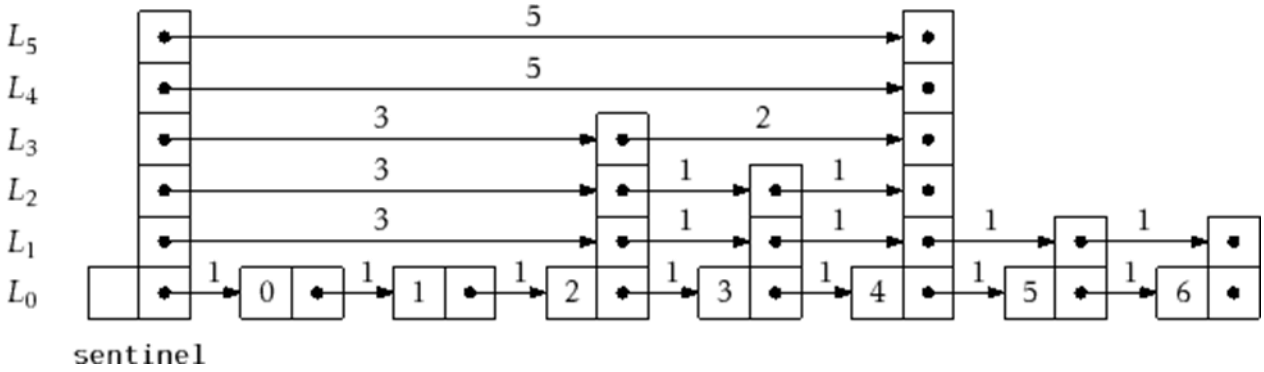
Thực tế, **Skip list** được ứng dụng trong **Competitive programming** như một sự thay thế cho **Balanced Binary Search Tree**. Về tộc độ và bộ nhớ, **Skip List** không thua gì **Balanced Binary Search Tree**, tuy nhiên lại dễ cài đặt hơn rất nhiều.

3. Indexable Skiplist

3.1. Định nghĩa

Cũng giống như việc xây dựng **Skiplist** nhưng ở đây chúng ta sẽ có thêm khái niệm cạnh (được biểu diễn bằng các số ở phía đường nối giữa 2 **Node** trên) là khoảng cách giữa hai **Node.**

Khoảng cách các cạnh ở tầng bên trên sẽ bằng tổng độ dài cạnh của các nối của các **Node** ở tầng bên dưới.



Hình 6: Minh họa Indexable Skiplist

3.2. Một số thao tác với Indexable Skiplist

Tìm kiếm một giá trị cho trước

Bắt đầu từ phần tử đầu tiên, tầng trên cùng của danh sách. Giả sử nút cần tìm mang giá trị là k ta duyệt theo quy luật:

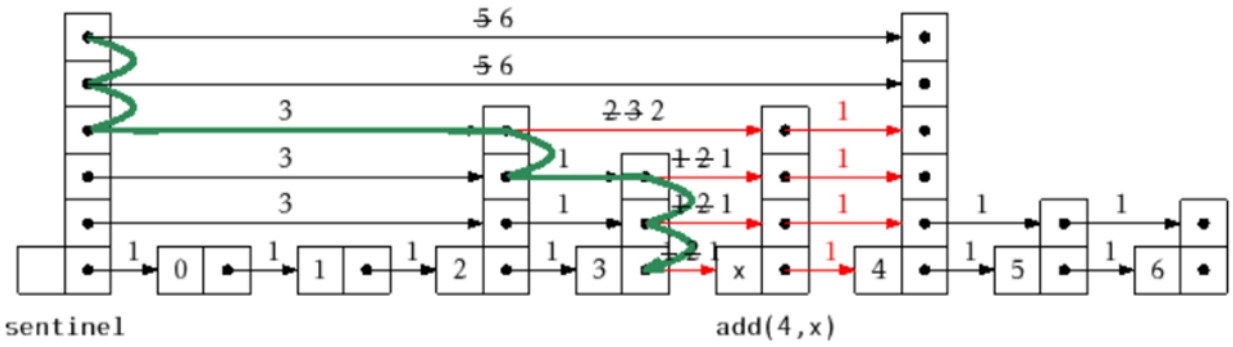
- Nhảy sang phần tử kế tiếp nếu nó khác Null và bé hơn hoặc bằng k
- Nếu không di chuyển trên cùng tầng được nữa
    - Dừng nếu ở tầng thấp nhất

- Ngược lại, nhảy xuống phần tử ngay bên dưới ở tầng tiếp theo

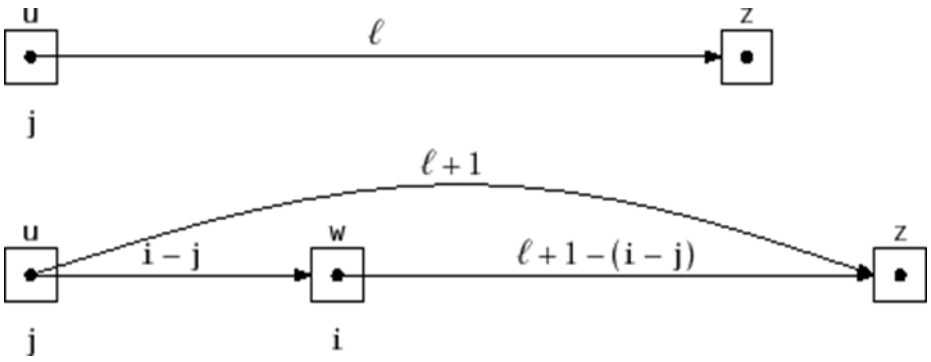
Nếu phần tử hiện tại là k thì ta trả ra kết quả k ngược lại k không tồn tại trong danh sách.

Thêm một giá trị cho trước

- Khởi tạo node w có giá trị là x và có độ cao là h
- Duyệt từ tầng cao nhất xuống và từ vị trí đầu của mỗi tầng đến vị trí kế tiếp
  - Node có giá trị kế tiếp > x và tầng node đó > h thì tăng độ dài cạnh nối lên 1
  - Node có tầng bằng h thì so sánh giá trị rồi thêm w vào trước hoặc sau node đó rồi tạo liên kết với node kế tiếp và cập nhật lại độ dài các cạnh nối
  - Đến khi tầng thấp nhất thì dừng
- Sau đó cập nhật lại số lượng node trong list +1



Hình 7: Thêm 1 giá trị trong Indexable skiplist

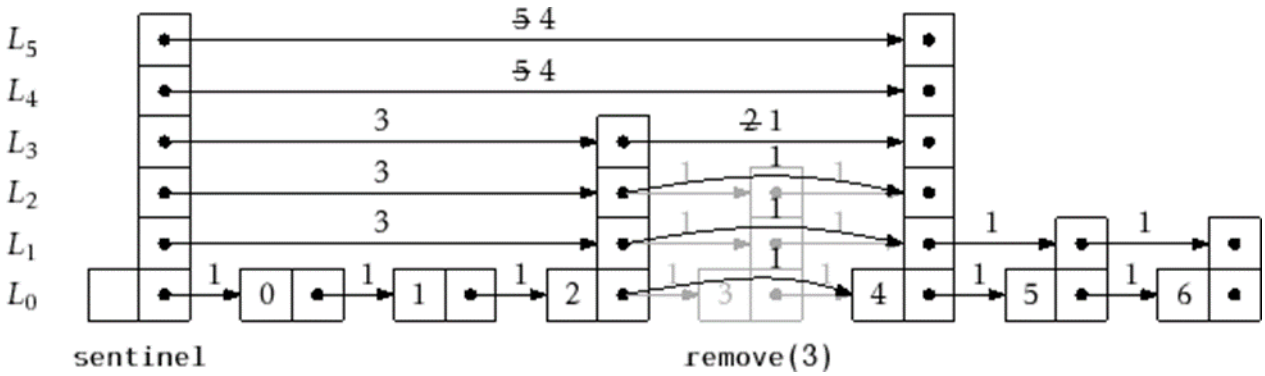


Hình 8: Thêm 1 giá trị trong Indexable skiplist

Xóa một giá trị cho trước

Giả sử ta xóa node có giá trị 3 ra khỏi list

- Tìm vị trí node trước phần tử cần xóa key1 (node 2) sau đó đánh dấu node sau nốt cần xóa (node 4) để làm key 2
- Thực hiện từ tầng dưới cùng đến tầng trên cùng
    - Xóa liên kết nốt cần xóa (node 3) với key (node 4)
    - Xóa liên kết node trước node cần xóa (node 2) với node cần xóa (node 3)
    - Cập nhật độ dài giữa 2 key là 1
    - Nếu có cạnh nào nối tới key2 thì giảm độ dài cạnh đi 1
    - Đến tầng lớn nhất thì dừng
- Giảm số lượng phần từ -1



Hình 9: Xóa phần tử trong indexable skiplist

Phần 2: Trả lời câu hỏi

1. Câu a

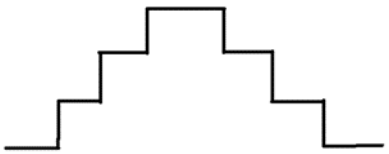
MaxLevel là giới hạn trên về level trong Skiplist. Nó có thể đảm bảo rằng level sẽ không bao giờ lớn hơn **MaxLevel** và nếu không có mức chặn này thì ta sẽ không biết được Skiplist có bao nhiêu level để có thể thao tác.

2. Câu b

Level của một node chuẩn bị được thêm vào một skiplist cho trước được chọn dựa trên **giá trị** của node vì để nó nhảy sang các node kế tiếp hoặc nhảy xuống một tầng(level) trong **Skip list** ta cần so sánh giá trị của node để thực thi.

3. Câu c

- **Thuật toán**
  - MAX_LEVEL = max (1, log2(<kích thước sau khi thêm>)
  - Nếu Value > Max hoặc Value < Min (Min, Max là cực trị của Skiplist hiện tại khi chưa thêm)
    - Level = Value mod MAX_LEVEL
  - Nếu không thì
    - Tính double average = (Min + Max) / 2
    - Level = $(MAX\_LEVEL + 1) \cdot cos\left(\frac{|Value - average|}{2|Value - average| + MAX\_LEVEL}\pi\right)$
- **Lý do lựa chọn thuật toán**
  - Dựa trên ý tưởng ban đầu của Skiplist số phần tử ở level i = ½ level i + 1, cố gắng xây dựng Skiplist có dạng bậc thang như hình



Hình 10: Ý tưởng xây dựng thuật toán xác định level

  - Vì vậy ta mong muốn những phần ở càng gần giá trị trung bình thì level càng cao, từ đó sử dụng hàm MAX_LEVEL*cos(f(x)) - hàm này thỏa tính chất $0 \leq$ level $\leq$ MAX_LEVEL, x là khoảng cách giữa value và giá trị trung bình. Ta chỉ cần xây dựng hàm f nghịch biến sao cho thì $0 \leq f(x) \leq Pi / 2$.
  - Qua tính toán thì tìm được hàm như trên, nhưng lại phát sinh vấn đề: Nếu chèn liên tục các phần tử nằm ngoài khoảng cực trị thì thuật toán cho ra level bị trùng

lặp nhiều. Vậy nên chia ra 2 trường hợp như trên, và xử lý trường hợp ngoài khoảng cực trị bằng cách lấy module MAX_LEVEL để có tính ngẫu nhiên.

Phần 3: Trả lời về độ phức tạp Big O

1. Câu a

Thao tác	Sorted Singly Linklist	Skiplist	Indexable Skiplist
Tìm kiếm vị trí của một giá trị cho trước	O(n)	Olog(n)	Olog(n)
Thêm 1 giá trị vào danh sách	O(n)	Olog(n)	Olog(n)
Xóa 1 giá trị khỏi danh sách	O(n)	Olog(n)	Olog(n)

Bảng 1: Bảng BigO

2. Câu b

(1) Phép toán tích cực là phép so sánh giá trị tìm kiếm với lần lượt các phần tử, worst case và average case là O(n)

(2) Phép toán tích cực là so sánh để tìm vị trí đúng của phần tử mới. Giống như (1) là O(n)

(3) Cần phải tìm kiếm vị trí của phần tử cần xóa nên tương tự (1) cũng là O(n)

(4) **Skip list** có log(n) levels và mỗi level chỉ cần nhiều nhất một step để tìm vì trong 2 step trong một level chúng ta luôn có một giá trị ở giữa và nằm ở level thấp hơn. Như vậy ở bất kỳ level nào chúng ta sẽ cần nhiều nhất một step để tìm kiếm hay BigO là O(log(n)).

(5) Để chèn thì tương tự như (4) ta cần tìm kiếm và chèn như linklist (mất O (1)) vì thế BigO của nó là O(log(n)).

(6) Để xóa thì tương tự như (4) ta cần tìm kiếm và chèn như linklist (mất O (1)) vì thế BigO của nó là O(log(n)).

(7) Cấu trúc của **Indexable skiplist** tương tự như **Skiplist** vì vậy khi tìm kiếm BigO của nó là O(log(n)).

(8) Để chèn thì ta cần tìm kiếm và chèn như linklist (mất O (1)) vì thế BigO của nó là O(log(n)).

(9) Để xóa thì ta cần tìm kiếm và chèn như linklist (mất O (1)) vì thế BigO của nó là O(log(n)).

3. Câu c

- (1), (4) và (7)
    - Ở (1) do list có n phần tử và muốn duyệt tìm kiếm phần tử nào ta cũng phải duyệt từ phần tử thứ nhất đến n nên độ phức tạp là O(n)
    - Ở (4) và (7) do nó có các bước nhảy bỏ qua một số phần tử trong danh sách nên việc tìm kiếm sẽ nhanh hơn và độ phức tạp của nó sẽ chỉ còn O(log(n))
- (2), (5) và (8)
    - Skip list = Indexable Skiplist > Sorted Singly Linkedlist
    - Vì đối với **Sorted Singly Linkedlist** ta cần phải tìm đúng vị trí của list. Đây là tìm kiếm tuyến tính vì thế cần O(n) nhưng với với **Skip list** và **Indexable Skiplist** điều này là Olog(n) bởi nó đã được giảm bớt nhờ cấu trúc nhảy node.

Phần 4: Trình bày hướng giải quyết ở phần 1.2

Để thực hiện yêu cầu đề bài ta sử dụng 2 **struct** sau:

```
struct Column {
	int value;
	vector<Column*> next_cell;

};
```

```
struct Skiplist {
	int MAX_LEVEL;
	int current_size;
	Column* head;
	Column* tail;

};
```

- **Column** giống như một node, nhưng chứa mảng con trỏ next cho từng level.
- **Skiplist** là một danh sách các Column liên kết với nhau.

Sau khi đã có 2 **Struct** như trên ta sẽ code những hàm bổ trợ sau:

```
Column* createColumn(int key); //
void createSkiplist(Skiplist& s); // initialize
bool empty(Skiplist s); // checks if Skiplist is empty
Column* lowerBound(Skiplist s, int key); // finds a column that has the bigest
value but lower than key.

Column* find(Skiplist s, int key);
int insert(Skiplist& s, int key);
bool erase(Skiplist& s, int key); // returns true if it is able to delete key


void deleteSkiplist(Skiplist& s); // clears data
```

- Đối với ***createSkiplist()*** ta sẽ tạo liên kết head->tail ở index 0 của vector next_cell, hàm ***createColumn()*** sẻ chỉ cấp phát một Column có value là key.
- Với hàm ***empty()*** ta chỉ đơn giản kiểm tra liên kết level 1 (index 0) của **head** có nối với **tail** hay không.
- Với hàm ***lowerBound()*** tìm kiếm Column chứa phần tử có giá trị lớn nhất nhưng nhỏ hơn key, Column kế tiếp có thể chưa phần tử key.
- Hàm ***find()*** sẽ kết hợp với ***lowerBound()*** tìm và trả về địa chỉ Column, nếu không có thì trả về **NULL**
- Hàm ***insert()*** được chia làm 3 bước:
    - Kiểm tra value đã tồn tại chưa, nếu đã tồn tại thì trả về -1và thoát khỏi hàm
    - Chọn level theo công thức trong đề

    - Chèn vào từng level nhỏ hơn hoạc bằng level vừa tính
- Hàm ***erase()*** chia làm 3 bước:
    - Kiểm tra giá trị tồn tại chưa. Nếu không tồn tại trả về **false** và thoát khỏi hàm
    - Xóa cột chứa giá trị cần xóa khỏi SkipLists bằng cách nối từng liên kết giữa các Cell liền trước và liền sau nó trên từng tầng
    - Xóa cột chứa giá trị cần xóa để giải phóng bộ nhớ.
- **Lệnh 1**: Sử dụng hàm ***insert()*** để chèn từng phần tử được nhập từ bàn phím
- **Lệnh 2:** Đi từ level cao nhất đến level 0, in các phần tử của 1 level như một linklist thông thường
- **Lệnh 3**: Trước tiên ta sẽ đếm số phần tử cùng kết thúc trong 1 level, sau đó tính số phần tử cho từng level bằng cách cộng gộp, số phần tử level sẽ là tổng của số phần tử level cao hơn với các phần tử kết thúc trong level đó.
- **Lệnh 4:** Ta sẽ duyệt tử level cao nhất, lưu trữ lại các Column đã đi qua bằng 1 string, đến khi kết thúc quá trình duyệt nếu đến được Column chứa phần tử cần tìm thì in ra đường đi.
- **Lệnh 5:** Sử dụng hàm ***insert()*** bên trên để chèn và nhận về level của phần tử vừa chèn.
- **Lệnh 6:** Sử dụng hàm ***erase().***

TÀI LIỆU THAM KHẢO

[1] "GeeksforGeeks," [Online]. Available: https://www.geeksforgeeks.org/skip-list/?ref=rp. [Accessed 2021].

[2] "GeeksforGeeks," [Online]. Available: https://www.geeksforgeeks.org/skip-list-set-2-insertion/?ref=rp. [Accessed 2021].

[3] GeeksforGeeks. [Online]. Available: https://www.geeksforgeeks.org/skip-list-set-3-searching-deletion/?ref=rp. [Accessed 2021].

[4] "UMBC," [Online]. Available: https://www.csee.umbc.edu/courses/undergraduate/341/fall01/Lectures/SkipLists/skip_lists/skip_lists.html. [Accessed 2021].

[5] V. c. P. Hoàng, "vnoi," [Online]. Available: https://vnoi.info/wiki/algo/data-structures/Skip-Lists.md. [Accessed 2021].