# DATA STRUCTURE

## LINKED LIST

### Định nghĩa

Gồm có 1 con trỏ head ở đầu và nhiều nút Node có cấu trúc

Dữ liệu: data

1 con trỏ “next” trỏ đến nút kế tiếp

struct Node {

Node\* next;

T data;

};

### Insert

insertHead(int data);

insertTail(int data);

insertAfter(int index, int data);

\* Thêm vào đầu DS:

+ tao 1 newNode, cho data vào

+ cắt bỏ liên kết head và nút đầu

+ Nối newNode->next vào nút đầu

\*Thêm vào cuối DS:

+ tao 1 newNode, cho data vào

+ tìm đến nút cuối, nối nút cuối với nút mới

+ nút mới => NULL

\*Thêm vào sau phần tử có chỉ số index

+ tao 1 newNode, cho data vào

+ duyệt từ đầu, tìm đến nút thứ index -1

+ next trỏ tới newNode, newNode -> next trỏ nút index

### Delete

deleteHead();

deleteTail();

\* xóa đầu:

+con trỏ phụ temp = head;

+head = head->next

+ xóa temp;

\* xóa đuôi

+duyệt tìm đến, nút cuối – 1

+ con trỏ phụ temp = head->next;

+ head->next = NULL

+ xóa temp

## STACK

Last in first out

Dùng DSLK chỉ làm việc với phần tử đầu: thay head = top

Có pop() : lấy ra ở đầu Stack => tương tự deleteHead();

Push(): đẩy vào đầu Stack => tương tự insertHead();

## QUEUE

First in first out

Dùng DSLK làm việc với cả phần tử đầu và cuối: thay head=> 2 con trỏ first và last để dễ quản lí vào ra

Node\* first;

Node\* last;

Thao tác:

Void enqueue(int data);

Int dequeue();

### Enqueue

Thêm vào cuối danh sách

+ tao 1 newNode, cho data vào

+newNode->next = NULL

+last->next = newNode

+last = last->next;

### Dequeue

Sử dụng first như head của DSLK

=> tương tự deleteHead(); => có trả về giá trị lấy ra

## BINARY SEARCH TREE

### Định nghĩa

Cây gồm : nhiều nút ghép lại và có 1 nút làm gốc

Mỗi nút có cấu trúc : có 1 khóa, 1 con trỏ sang trái của nút, 1 sang phải

Struct TreeNode {

int value;

TreeNode \*left;

TreeNode \*right;

}

### Insert

void insert(int key);

void insert(int key, TreeNode\* root);

+ nếu cây rỗng

+tạo nút -> thêm data, left, right = NULL;

+ gốc => nút mới

+ nếu ko:

+xét key < root->value =>đi sang trái () (1)

+ nếu ko có con trái làm giống cây rỗng

+làm lại bước (1)

+tương tự xét key >= root->value => sang phải

### Tìm Min

TreeNode\* findMin();

Đi theo full bên trái

### Tìm Max

TreeNode\* findMax();

Đi theo full bên phải

### Tìm Nút

TreeNode\* search(int key, TreeNode\* root);

Tìm nút với con trỏ đi từ nút tham số đầu vào

+ nếu value = key => return

+ nếu nhỏ hơn => đi trái

+ lớn hơn đi phải

### Tìm parent

TreeNode\* parent(TreeNode\* node);

Duyệt tới điểm có value = node->value, lưu nút trước => return

### Tìm successor

Là nút có khóa nhỏ nhất còn lớn hơn khóa nút đang xét

+ nếu nút có con phải => findMin con phải

+ nếu nút ko có con phải

Successor sẽ là tổ tiên của ( x hoặc tổ tiên của x) đầu tiên có con trái

//duyệt tới hết gốc

//gán p = cha của x

//nếu p -> con trái là x hoặc tổ tiên của x => ok

While p!= NULL and p->left!= x do

X = p;

P = parent(p);

endwhile

### Tìm predcessor

Tương tự

### Xóa nút

TreeNode\* deleteNode(int key, TreeNode\* root);

Xóa nút bắt đầu duyệt từ root

+root = NULL ko xóa đc => NULL

Else

+ key < root->value => xóa nút bắt đầu từ root->left

+ key > root->value => xóa nút bắt đầu từ root->right

Else

+Tìm được nút

+nếu nút là lá => nối nút cha trái hoặc phải vào NULL;

+xóa nút

+nếu nút chỉ có con trái

+temp = root

+root = root->left

+xóa temp;

+ tương tự chỉ con phải

+ có 2 con

+ tìm successor của nút => findMin() con phải

+ gán key của nút = successcor-key

+ xóa successor->key, từ nút->right

## HEAP

### Định nghĩa

Gồm 1 mảng chứa các giá trị thỏa mãn

Tính chất: với mỗi nút x có parent(x) >= x;

leftChild : chỉ số số chẵn

rightChild: lẻ

parent: chia đôi

### Heapify

Khôi phục tính chất đống

Khôi phục tính chất đống cho trường hợp a[i] không thỏa mãn tính chất

Khi 2 cây con trái và con phải đều là max heapify

Và a[i] không phải là lá

function heapify(int\* a, int i, int n)

begin

left = leftChild(i);

right = rightChild(i);

integer largest = i;

//nut cuoi

if left <= n - 1 then

if a[left] < a[i] then

largest = i;

else

largest = left;

endif

endif

if right <= n -1 then

if a[right] > a[largest] then

largest = right;

endif

endif

if largest # i then

swap(a[largest], a[i]);

heapify(a, largest, n);

end

### Build Heap

Xây dựng heap với tính chất trên từ 1 mảng a[]

Ta thực hiện khôi phục tính chất đống với các nút trong, nên chỉ cần xét từ nút n/2

function buildHeap(int\* a, int i, int n)

begin

for i = n/2 downto 1 do

heapify(a, i, n);

endfor

end

### Thêm 1 nút

Sẽ thêm vào lá từ trái qua phải

Làm tương tự như enqueue() và increaseKey() của priority queue

## PRIORITY QUEUE

### Định nghĩa

Là hàng đợi nhưng với mỗi nút ta có thêm 1 khóa ưu tiên, khi đó phần tử được dequeue là phần tử có khóa lớn nhất, nếu nhiều hơn 1 phần tử có cùng khóa đó thì xuất ra theo thứ tự nút vào

### Thao tác

Xây dựng bằng heap nên cũng có các phương thức trên

void increaseKey(int i, int key);

void enqueue(int data);

int dequeue();

#### \*\* Enqueue

Cấp phát thêm 1 phần tử cho mảng

Gán phần tử a[size] = -vô cùng

Gán key bằng cách tăng key cho nút mới thêm increaseKey(size+1, key);

#### \*\*Tăng khóa cho nút thứ i

+ nếu khóa mới <= khóa hiện tại => ko làm gì

+ else

+Nếu sau khi tăng: tính chất heap bị vi phạm

//thực hiện nghịch heapify => đi lên

//nếu a[i] <= a[parent(i)] không làm gì cả

//nếu ko: swap(a[i], a[parent(i)])

//gán I = parent(i) thực hiện lại 3 bước trên

//dừng: chạm gốc

//hoặc : thỏa mãn điều kiện 1

#### \*\**dequeue*

Đưa ra phần tử có khóa lớn nhất trong heap, giảm heap size đi 1đv

Gán a[0] = min của Heap;

buildHeap(a, n);

swap(a[1], a[n-1])

lưu a[n-1] => return

n--;

heapify(a, 1, n);

# ALGORISTHM

## SORTING

Kiểm tra từ phần tử thứ 2 đến hết

Với mỗi lần ta sắp xếp lại vị trí cho phần tử cuối cùng vào dãy trước nó đã được sắp xếp

### Insertion sort

function insertionSort(a(0:n-1))

begin

for i=1 to n do

integer last <- a[i];

integer j = i;

while last < a[j-1] and j > 0 do

swap(a[j], a[j-1]);

j--;

endwhile;

endfor

end

### Selection sort

Duyệt từ đầu dãy

Chọn phần tử nhỏ nhất đưa về đầu dãy i -> n và làm tương tự với dãy

i+1 ->n

function selectionSort(a(0:n-1))

begin

for i=0 to n-1 do

min = i;

for j = i to n-1 do

if a[j] < a[min] then

min = j;

endif

endfor

//tim duoc so nho nhat

swap(a[i], a[min]);

endfor

end

### Bubble sort

Chuyển chỗ các phần tử với việc so sánh 1 phần tử với phần tử đi sau nó

Thực hiện từ đầu dãy như vậy n lần

function bubbleSort(a(0:n-1))

begin

for j = 0 to n-1 do

for i = 0 to n-2 do

if a[i] < a[i+1] then

swap(a[i], a[i+1]);

endif

endfor

endfor

end

### Merge Sort

Gồm 3 thao tác:

+ Chia dãy thành 2 dãy con

+ Sắp xếp từng dãy con 1 cách đệ quy

+ Hợp 2 dãy lại

function mergeSort(a(0:n-1), dau, cuoi)

begin

if dau < cuoi then

integer mid = (dau + cuoi)/2;

mergeSort(a, dau, mid);

mergeSort(a, mid + 1, cuoi);

merge(a, dau, mid, cuoi);

endif

end

/////////////////////////////////////////////////////////////////////////////

Đầu vào: 2 dãy đã được sắp xếp

+ Dãy 1: từ dau -> mid

+ Dãy 2: mid + 1 -> cuoi

Đầu ra: 1 dãy đã được sắp xếp (dau -> cuoi)

procedure merge(a(0:n-1), dau, mid, cuoi)

begin

tao mang phu L[]: so phan tu <- cuoi - dau + 1

integer index = dau;

while ( dau < mid) and (mid + 1 < cuoi )

if a[dau] <= a[mid + 1] then

L[index] <- a[dau];

index <- index + 1;

dau <- dau + 1;

else

L[index] <- a[mid + 1]

index <- index + 1;

mid <- mid + 1;

endif

endwhile

//1 trong 2 day het truoc

//điền cac phan tu con lai vao L[] theo thu tu

if dau = mid then

while (mid + 1 # cuoi)

L[index] = a[mid + 1];

mid <- mid + 1;

endif

=> Tương tự cho trường hợp dãy sau hết trước

end

### Quick Sort

Gồm 3 thao tác

+ Chọn phần tử chốt

+ Duyệt từ đầu dãy, đưa các phần tử nhỏ hơn chốt sang dãy trái

+ Duyệt từ đầu dãy, đưa các phần tử lớn hơn chốt sang dãy phải

+ Sắp xếp 2 dãy con 1 cách đệ quy

function quickSort(a(0:n-1), dau, cuoi)

begin

if dau < cuoi then

integer phanTuChot = partition(a, dau, cuoi);

quickSort(a, dau, phanTuChot-1);

quickSort(a, phanTuChot+1, cuoi);

endif

end

procedure partition(a, dau, cuoi)

begin

integer CHOT = (dau + cuoi)/2;

tao mang phu L[]: so phan tu <- cuoi - dau + 1

integer index = dau;

for i= "dau" to "cuoi" do

if a[i] < a[CHOT] then

L[index] = a[i];

index <- index + 1;

endif

endfor

//het day trai

L[index] <- a[CHOT]; //vi tri cua phan tu chot

integer temp <- index;

index <- index + 1;

//duyet tu dau tim so lon hon -> day phai

for i= "dau" to "cuoi" do

if a[i] >= a[CHOT] then

L[index] = a[i];

index <- index + 1;

endif

enfor

return temp;

end

### Heap Sort

Xây dựng thuật toán dựa trên cấu trúc đống

+ Ta xây dựng đống từ mảng a[] đầu vào

+ Chuyển phần tử lớn nhất a[1] về cuối

+ Giảm kích thước heap

+ Thực hiện như vậy đến khi Heap chỉ còn 1 phần tử

function quickSort(a(0:n-1))

begin

while n > 1 do

buildHeap(a, n);

swap(a[0], a[n-1]);

n <- n -1;

endwhile

end

procedure buildHeap(a(0:n-1), n)

=> Xây dựng 1 max heap từ mảng a[n]

=> Xem cấu trúc dữ liệu Heap

## SEARCHING

### Sequential Search

Duyệt toàn dãy, so sánh từng phần tử với phần tử tìm kiếm, nếu đúng trả về chỉ số

### Binary search

Sắp xếp dãy theo thứ tự không giảm

function binarySearch(a(dau:cuoi), key)

begin

integer mid = (dau + cuoi)/2;

if a[mid] < key then

binarySearch(a, dau, mid-1);

else a[mid] > key then

binarySearch(a, mid+1, cuoi);

else return mid;

end

### String Search

Bài toán : 2 chuỗi pattern P : độ dài m, text T : độ dài n, với m < n, tìm các vị trí khớp thứ s với chuỗi P trong T

#### Boyer – Moore Algoristhm

Chọn phần tử tồi: là phần tử tìm thấy đầu tiên của chuỗi văn bản T không khớp với phần tử tương ứng trong chuỗi mẫu P khi duyệt từ cuối P trở về

Thực thi thuật toán

Duyệt từ đầu dãy T bằng biến s

Khi đó, xét xem trong P có xuất hiện phần tử tồi hay không

+ Nếu có

+ Tìm vị trí bên phải nhất của phần tử đó trong P

+ Nếu vị trí đó nằm bên trái so với phần tử tồi trong chuỗi T, ta chuyển được vị trí s tới đó và giảm số bước đáng kể

+ Nếu nằm bên phải thì tăng biến đếm s + 1 bình thường

+ Nếu không có

+ Tăng biến đếm lên về đến đầu dãy P và giảm được số bước đáng kể

function boyerMoore(string P, string T)

begin

m <- so phan tu P;

n <- so phan tu T;

integer s = 0;

while s<= n - m -2 do

integer j = m-1;

for j = m-1 downto "0" do

if P[j] = T [s+j] then

j--;

else if P[j] # T [s+j] then

//di chuyen sang phai den khop c phai nhat

//hoac di chuyen 1 sang phai neu, chuyen den khop c la di chuyen sang trai

s <- s + max(last(T[s+j]), P);

endif

endfor

if j = -1 then

s++;

=> output "s";

endif

endwhile

end

//Tim ky tu "c" ben phai nhat cua pattern P

procedure last(char c, string P)

begin

m<- so phan tu P;

integer i = m -1;

while i >= 0 do

if P[i] = c then

i <- i -1;

else break;

endif

endwhile

return i;

end

#### Knuth Morris Pratt Algoristhm - KMP

Prefix : nếu k<=m thì P(0:k) là prefix của P(0:m)

Suffix : nếu m >=0 P(m:q) là suffix của P(0:q)

Duyệt s trong T

Duyệt j trong P, nếu tìm thấy điểm không khớp

+ Đưa số lượng các phần tử khớp lớn nhất là prefix của P và là suffix của chuỗi T với điểm cuối của T là điểm ngay phía trước điểm đang xét => như vậy ta tăng biến đếm s và giảm được số bước đáng kể

Từ đó đưa ra khái niệm prefix function của chuỗi P: là số lượng phần tử của prefix lớn nhất của P sao cho prefix đó cũng là suffix của P (với số lượng phần tử đang xét)

function searchKMP(string P, string T)

begin

m <- so phan tu P;

n <- so phan tu T;

integer s = 0;

while s <= n-m-1 do

integer j =-1;

while j <= m-1 do

//duyet j trong P neu tim thay diem ko khop

while (P[j+1] # T[s+j]) and (j > -1) do

j = PI(j);

endwhile

if P[j] = T[s+j] then

j++;

endif

//den j = -1 van P[0] khong khop thi bo qua ko lam gi

endwhile

if j = m then

output(s- m + 1);

s++;

endif //tang tim do tim tiep

endwhile

end

//prefix function

//tim max k sao cho P(0:k) la suffix cua P(0:m-1)

//tra ve 1 mang cac so da tim duoc

procedure PI(string P)

begin

m <- so phan tu P;

tao mang integer "arr\_pi" -> so phan tu: m

arr\_pi[0] = -1;

for j=1 to m-1 do

integer k = arr\_pi[j-1];

while (k > -1) and (P[k+1] # P[j]) do

k = arr\_pi[k];

endwhile

//tim duoc k sao cho p(0:k) la suffix cua p(0:j-1)

if P[k+1] = P[j] then

k = k + 1;

endif

//tim duoc PI[j]

PI[j] = k;

endfor

return arr\_pi; //tra ve mang

end