4 Feb, 2019

Introduction to CSE - 203

Data Structure and Algorithm

Program → Data → ingredients
Program → Instruction → procedure

Data গুলো organized way তে না রাখলে,

⇒ time বেশি লাগবে। (খুঁজতেই almost অনেক সময় লেগে যাবে)

⇒ Space বেশি লাগবে।

⇒ Add করতে ঝামেলা (নতুন কিছু রাখতে গেলে কোথায় রাখবো সেটা একটা problem)

⇒ Replace / Delete

⇒ Retrive (Excess করতে ঝামেলা)

⇒ Search করতে সময় লাগবে।

Data Structure

- Sequential container
  - ~~Array~~ (একটা ক্রম থাকবে)
  - Array list / Vector
  - Deque
  - List
- Associative (কোনো জিনিস আছে নাকি নাই সেটার উপর depend করে) কোনো number দেয়া নাই।
  - Set / Multiset
  - Map / MultiMap.
- Abstract
  - stack
  - queue
  - priority queue.

Scanned by CamScanner

last এ কিছু একটা quick add করতে হলে আমরা array list use করি।

Deque: Double Ended que:
quickly first এও add করা যায়, last এও add করা যায়।

list: যেকোনো জায়গায় fast add করা যায়।

[Sequential container এগুলো 'add করা'র উপর ভিত্তি করে আলাদা করা হয়]

Associative:

Set: i) Unique elements [অনেক language এ Set কে Bag বলে।]
ii) No order ⟶ orderwise বসাতে হবে না, Just ডাক দিলে চলে আইসো। (search)
iii) Well defined.
* যেটা search করি, সেটাই চাই।

Map: Key দিয়ে search করবো, value টা চাবো।
Suppose: Roll ~~[illegible]~~, ~~[illegible]~~ দিয়ে ডাকবো But নাম টা চাবো।
<Key, Value>.

যেমন: $f(x) = y$
x দিলাম y পেলাম

Multi Map: [একটা key, একাধিক value]

[Array, Set এর diff: Array তে order important But Set এ সেটা important না]

Scanned by CamScanner

MultiSet: unique element ব্যাপারটা compromise করা হয়
- মানে same element অনেকগুলা থাকতে পারে।

(i) Add
(ii) Delete
] Sequential → Access করতে পারবো by index

(iii) Access (Retrieve)
(iv) Search
] Associative → access করতে পারি by key

Abstract:

কোনো একটা behaviour ঠিক করে দিলে then ওইটা দিয়ে call করলে সেটা কে বলে Abstract.

যেমন: যে আগে class এ আসবে, সে আগে আসবো behaviour ঠিক করে দিলাম: 'যে class এ আগে আসবে'

Stack: behaviour: last in first out, যে ~~আগের~~ last এ আসবে, সে আগে বের হবে।

queque: & FIFO: যে first এ আসবে, সে আগে আসবে।

priority queue: একটা priority first এ select করে দিবো (depends on me). then ওই অনুযায়ী value call করবো।

Scanned by CamScanner

যেমন: Hospital: যে সবচেয়ে critical Situation এ থাকবে, তাকে আগে দেখা হবে। (different Situation এ different priority select করতে হয়)

Stack এর ধরনে priority key ঠিক না:

Stack এ priority টা fixed (যে last এ থাকবে, তাকে first এ রাখা হবে)

But priority key এর শর্ত হলো, priority টা fixed করে দেয়া থাকবে না। নিজেরা priority ঠিক করে দেয়ার freedom থাকবে।

Arraylist

2 types: → Static: compile time এ size set করা হয়।

→ dynamic: runtime এ size set করা হয়।

অনেকগুলো instruction দিয়ে যদি আমরা একটা কাজ করি, সেটা abstraction. (PC on করা মানে power on, then log in Bla Bla Bla]

Scanned by CamScanner

Regular operation:

- Add (at the end) → push_back()
- Edit
- with element বের করা

Object oriented programming:

information আর instruction কে একসাথে লিখতে পারি।

```
struct Student
{
    int cse105;
    int cse 106;
    int sum ()
    {
        return cse105 + cse 106;
    }
}
```

function টাকে struct এর মধ্যে রাখলে দাঁড়ায় এইটা C++ এ বলা যায়, এইটা হলো class.

Scanned by CamScanner

```
# include <stdio.h>
# define SIZE 100

struct Arraylist
{
    int pos = 0;
    int array[SIZE];
    void push-back (int x)
    {
        array[pos] = x;
        pos++;
    }

    int get (int index)
    {
        return array[index];
    }

    int search (int val)
    {
        for (i=0; i< pos; i++)
        {
            if (array[i] == val)
                return 1;
        }
        return -1;
    };
```

Scanned by CamScanner

```
int main ()
{
    arraylist a;
    a. pushback (10);
    a. pushback (20);
    a. pushback (30);

if (a. search (20) == 1)
        printf ("found \n");
else
        printf ("Not found \n");
}


~~find_min()~~
int find_min ()
{
    ~~for (a=0~~
    int min = array [0];
    for (i=1; i < pos ; i++)
    {
        if (array [i] <= min)
            min = array [i];
    }
    return min;
```

Scanned by CamScanner

Analyzing Runtime:

```
# include <stdio.h>
int main ()
{
    printf ("  ");
    printf ("  ");
}
```

→ unit

২ unit time লাগবে। unit মানে instruction. মানে দুটি instruction পার হইতে হবে।

```
for (int i = 1;  i<=n ; i++)
        printf ("  ");
```

n টি unit এইখানে।

মানে মোট n+2 unit.

আবার,

```
for (int i = 1;  i <= n ; i++)
    for (int i = 1,  i <= n ; i++)
    {
        printf ("  ");
    }   printf ("  ");
```

program 1

n^2 unit লাগবে।

Scanned by CamScanner

মানে, for loop একেবারে power বাড়ায়।
normal instruction 1 করে increment করে।

→ Big o notation (time complexity measure করে)

O(n)

→ order যত কম হবে, তত দ্রুত কাজ করবে।

O(n^2) → order বেশি, তাই খারাপ কাজ করবে।

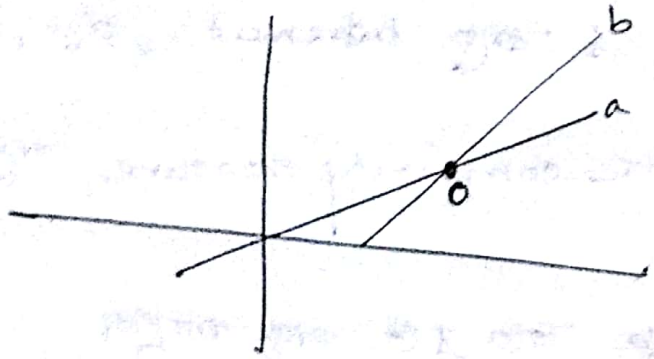
দ্রুত কাজ করা যায় মানে অল্প order এ কাজ করা যায়।

program 1 এর time complexity (2*n^2)
($2n^2$)

কারণ, ~~for loop~~ for এর জন্য 2n.
বাইরের for এর জন্য: $2n^2$.

Scanned by CamScanner

5 February, 2019 :



a একটা graph, b আরেকটা graph. a graph টা o point এর পর থেকে, কখনোই b এর উপরে যাবে না, মানে, O notation এর কাজ হচ্ছে, কোনো program lowest বিন্দুর পর আর বাড়বে না.

যেমন, যদি কোনো program এর complexity $3n+2$. $3n+2$ কখনোই $4n$ এর চেয়ে খারাপ হবে না,

$$3n+2 \le cn \longrightarrow c=4$$

$$\therefore \underbrace{3n+2}_{f(n)} \le \underbrace{4n}_{g(n)}$$

এরকম relation কে,

$$f(n) \in O(g(n))$$

$$3n+2 \in O(n)$$

$\therefore 3n+2 \le 4n$

$\Rightarrow 2 \le n$

$\therefore n \ge 2. \longrightarrow 2 = n_0$ n এর মান lowest যত হতে পারে,

Scanned by CamScanner

$O(1)$ ⟶ function টি n এর উপর depend করে না

$O(n)$ ⟶ n এর মান যত বাড়বে, performance তত খারাপ হবে।

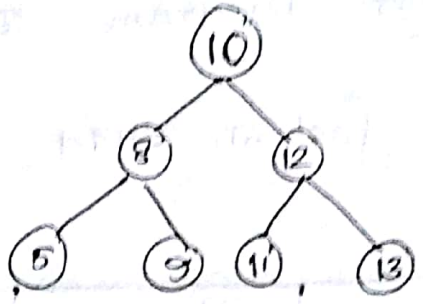
Big O notation এ আমরা worst case complexity use করবো মানে মানো

$O(\log_2 n)$

$O(n \log_2 n)$

$O(n^2)$

$O(n^3)$



আমি Binary Search করি, first এ শুরু করবো 10 থেকে। আমি জানি, ব ডানপাশের 10 এর চেয়ে বড় কোনো সংখ্যা থাকবে না। So আমরা node টা বাদ দিয়ে দিলাম। আসলাম 8 এ, আমরা জানি, বাম পাশে কখনোই 8 এর ছোট কিছু থাকবে না, So, node আগের 1টা বাদ দিয়ে দিলাম।

So, প্রত্যেক step এ node কমছে 1 এইটা logarithmic relation. কতবার চলবে loop? যতবার first element টাকে 2 দিয়ে ভাগ দেয়া যাবে।

So, $\log_2 n$ হচ্ছে, যখন, $i \leq n/2$.

$\therefore$ $n \log_2 n$ হবে, যখন বাইরে আরেকটা for loop থাকবে।

Scanned by CamScanner

Regular arrays in C

* No bound Checking → in C.

- কারণ আছে, array র size 10 হওয়াতে, p[15] লিখলে compiler error দেয় না, কারণ প্রত্যেকবার Bound checking করলে slow হবে program.

⇒ push_back fast না, কারণ বারবার Capacity check করতে হয়।

But

	Static	Dynamic
Add	O(1)	O(1)
Search	O(n)	O(n)
Delete	O(1)	O(1)
Retrieve by index	O(1)	O(1)
Retrieve by key	O(n)	O(n)

→ push back function. Though normal দিক fast না. But I দিয়ে আমরা time measure করি না, instruction measure করি, So O(1) দেখে time দিব বোঝা সম্ভব না।

* size must be define in compile time. (static এ এটা problem But dynamically prob solved)
* Continuous memory location. (এটা এটা array তে problem)
* Homogeneous elements
* Zero based indexing
* We can fast and simplest Data Structure.

Scanned by CamScanner

13

pointer address store করে। continuous memory যদি না চায়, তাহলে এমন একটা mechanism লাগাবে, যেটা প্রত্যেকবার, এর পরের element এর address কে point করে দেখাবে। So, আমরা প্রত্যেকটা element এর জন্য একটা structure নিবো যেটার member হবে, একটা int value, আর একটা pointer, যেটা always ওই structure এর address কে point করবে। So, continuous না হলেও problem নাই। address টা point করতে পারবে pointer দিয়ে।

Scanned by CamScanner

pointer address store করে। continuous memory যদি না চাই, তাহলে এমন একটা mechanism লাগবে, যেটা প্রত্যেকবার, এর পরের element এর address কে point করে দেখাবে। So, আমরা প্রত্যেকটা element এর জন্য একটা structure নিবো যেটার member হবে, একটা int value, আর একটা pointer, যেটা always ওই structure এর address কে point করবে। So, continuous না হলেও problem নাই। address টা point করতে থাকবে pointer দিয়ে।

11 February, 2019

Single Linked List

কোনো array list এ last এ add করা faster, মানে, last এ delete করাও faster.

যদি মাঝখান থেকে delete করতে চাই, তাহলে order n.

But order 1 এই কাজ করা সম্ভব।

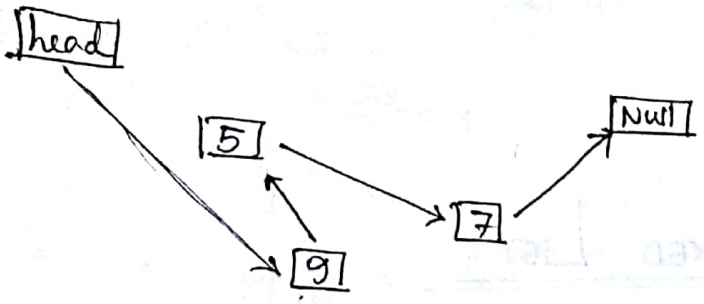
সেই index টা delete করবো, সেটাকে last element দিয়ে replace করবো, আর size 1 কমায়ে দিবো।

Scanned by CamScanner

পাশাপাশি ৩টি ঘর খালি থাকতে হবে array[3] নিলে। But যদি খালি থাকে, তাহলে, dynamically একটা list তৈরি করবো। যেটাতে, 1st element add করলাম, যেই address এ value টি থাকলো সেটাকে একটা pointer point করে রাখবে।

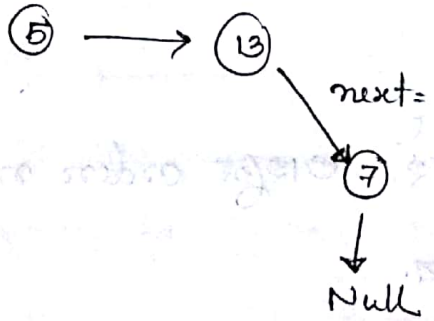
2nd element নিলে, অন্য জায়গায় dynamically একটি জায়গা রাখবে and pointer টি তখন ওই জায়গাকে point করবে।

এভাবে linked list create হয়।



push_front (order 1)
push_back (order n)

মাঝখানে 13 ঢুকাতে চাই,



13 7 কে point করবে, then, 5 এখন 13 কে point করবে। 13 কে delete করতে হবে।

5 এখন point করবে 13 এর next এর next কে। But এটা করলেও

push-front এর ক্ষেত্রে:

linked list এ, head point করে থাকবে 1st element কে, 1st elem. point করে থাকবে 2nd কে --- এভাবে last পর্যন্ত চলবে।

আমি যদি first এ element add করতে চাই, তাহলে head কে ওই নতুন element কে point করাবো দিবো, আর ওই নতুন element point করবে 1st element কে, বাকিগুলোতে আর হাত দেয়া লাগলো না, তাই push-front ( ) এর order 1.

push back এর ক্ষেত্রে:

last এর element এর address জানে (n-1) element, (n-1) এর element জানে (n-2). এভাবে ~~প্রত্যেকের~~ last element এর address জানতে হলে প্রত্যেকের address ই জানতে হবে। Then সবার মাধ্যমে last element কে access করতে হবে।
তাই push_back এর order (n)

Scanned by CamScanner

	array	linked list
access	$O(1)$	$O(n)$
Search	$O(n)$	$O(n)$
Insertion	$O(n)$	$O(1)$
Deletion	$O(n)$	$O(1)$

ADT: Abstract Data type.

abstraction হচ্ছে যা কাজ নিজেরটে হবে, কিন্তু সেটা দেখতে পারবো না, এই কাজটা করি Structure দিয়ে। এই Structure এ থাকবে একটা value আর একটা pointer. সেটা ওই structure কেই point করবে।

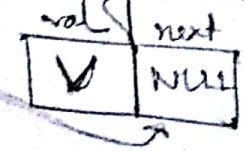
```
struct Node
{
    int val;
    Node * next;
};

Struct LinkedList
{
    ~~struct~~ Node * head = NULL; struct Node.

    void push_front (int v)
    {
     Node *temp = new Node;
     (*temp).val = v;
     temp -> next = NULL;
     if (head == NULL)
     {
        head = temp;
     }
```

head দিয়ে একটা pointer সেটা first element কে point করে দিবে। element হচ্ছে struct Node.

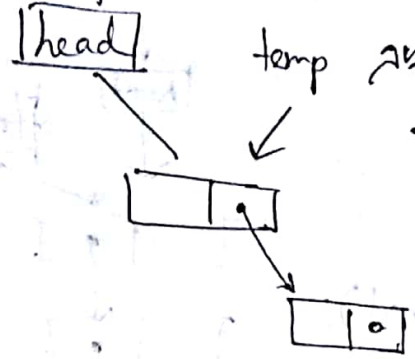
Node type এর pointer create করলাম, new দিয়ে Node এর memory allocate করলাম



Scanned by CamScanner

NULL হচ্ছে RAM এর first ~~ele~~ ঘর টা NULL একটা address.

temp সেই Box টাকে point করে আছে, head সেই Box কে point করবে।



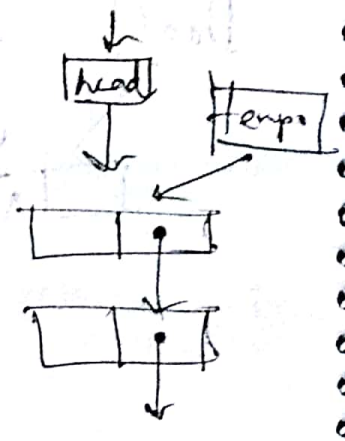
temp এখন, temp কে ঢুকাইতে চাইলে, head কে direct temp কে point করাইতে পারবো না, কারণ তাহলে head এর নিজের element গুলা হারায়ে যাবে। তাহলে, temp point করতে head যাকে point করতে দিলাম, তাকে, Then, head, temp কে point করবে।

```
∴ else
  {
      temp -> next = head;
      head = temp;
  }
}
```

new দিয়ে memory create করলাম কারণ নরমালই Node নিলে এভাবে create করলে, এর scope ওই function এর মধ্যেই শেষ হয়ে যাবে। new দিয়ে create করলে, dynamically create করলাম address টা, তাহলে এটা আমি destroy না করা পর্যন্ত free হবে না।

Scanned by CamScanner

```
void display()
{
    if (head == NULL)
            printf ("No element!\n");
    else
    {
        Node * tempo = head;
        while (tempo != NULL)
        {
            printf ("%d ", tempo->val);
            tempo = tempo -> next;
        }
        printf ("\n");
    }
}
```



first এর element কে point করানোর জন্য tempo নামে একটি new pointer নিলাম, যেটার ~~point~~ next point করলাম temp. এর next কে, প্রথমে print করে নিলাম, then temp = temp->next এভাবে শেষ পর্যন্ত যাবে

```
void dispose ()
{
    ~~Node * temp = head;~~
    while (temp != NULL)
    {
        ~~head = head -> next;~~
        ~~temp~~ Node *temp = head;
            heaad = head -> next;
            ~~tempo~~ delete temp;
    }
    printf ("All elements are cleared!");
}
```

Scanned by CamScanner

dispose() call করার পর সব value গুলো delete হয়ে যাবে।

18 February, 2019

arraylist → memory continuous থাকতে হয়।
→ front এ element add করতে গেলে, time complexity হয় n. But linked list এ (1) complexity তে first এ add করা যায়।

first থেকে delete করতে হবে — pop_front function:

⇒ temp pointer নিবো যেটা first element কে point করবে। মানে head কে।

⇒ head point করবে head এর next কে।

⇒ then temp কে dispose করে দিবো।

কোন element টা delete করলাম সেটা return করবো

```
int {pop_front}()
{
    if (head == NULL)
            printf ("Linked list is empty");

    else {
        Node * temp = head;
        head = head -> next;
        int ret = temp -> value;
        delete temp;
        return ret;
    }
```

head এর next আর temp এর next same. So, temp->next ও লিখতে পারতাম

[নতুন একটা int নিলাম যেটা এখন temp এর value কে store করবে]

Scanned by CamScanner

Stack using Linked List

Peek(), size, isempty() ⟶ ৩টি function: order হবে হবে 1.

Stack বানাবো linked list দিয়ে,
element রাখবো push-front দিয়ে
element বের করবো pop-front দিয়ে,

```
class Stack
{
    Linkedlist l1;
    public:
        int n = 0;  ⟶ initial element.
        void push (int v)
        {
            n++;
            l1. push_front (v);  ⟶ element বাড়বে।
        }

        ~~int pop ()~~
        ~~{ n--; return l1. pop_front ();  ⟶ element কমবে।~~
        ~~}~~

        int pop ()
        {
            if (n= =0)
                printf ("stack is empty");
            else
```

Scanned by CamScanner

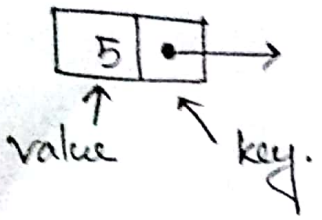
```
        {  n--;
           return  l1.pop_front();
        }
    }
int size()
    return n;
int peek()
    {
        return  l1.get_front_element();
    }
int isEmpty()
    {
        return !n;
    }
void display()
    {
        l1.display()
    }
};
```

n এর মান zero হলে isEmpty true অর্থাৎ মানে 1.
n এর মান nonzero হলে isEmpty false অর্থাৎ
তাই !n দিলাম

Scanned by CamScanner

linked list এ Search করবো এখন।

~~head~~




```
int search (int n)
{
    while (temp != null)
    {
        if (temp -> val == n)
            return temp;
        else
            temp = temp -> next;
    }
    return temp;
}
```

temp যদি NULL এ point করে থাকে, তাহলে temp→value পাবে না। So, program crash করবে।

temp যেই Node টাকে point করে রাখলো সেটার address return করবে।

```
add_first (value 1, value 2)
{
    Node *ptr = Search (v1);
    if (ptr == NULL)
        return
    else
    {
1. allocate memory of new value;
2. ~~temp~~ Node *temp create;
    temp -> value = value 2;
```

value 1 এর পরে value 2 কে বসাতে হবে।

1. allocate memory of new value; → new Node তৈরি।

Scanned by CamScanner

নতুন temp নিবো.

~~temp = ptr -> next;~~

temp -> next = ptr -> next;

ptr -> next = temp;

Delete (Value) value টা খুঁজে বের করে delete করে দিলে পরের element গুলো হারিয়ে যাবে।

So, খুঁজে বের করবো পরের element না।

1. If head is value, delete head;
2. Else temp = head.

```
While (temp -> next != NULL)
        If (temp -> next -> val == value)
                Keep temp2 = temp -> next
                link temp->next = temp -> next -> next /
                                temp2 -> next
                delete temp2;
        temp = temp -> next;
```

Scanned by CamScanner

19 February, 2019

Search and delete Code.

```
Node * search (int v)
{
    Node * temp = head;
    while (temp != NULL)
    {
        if (temp -> val == v)
            return temp;
        else
        {
            temp = temp ->next
        }
    }
    return temp;
```

NULL ও return করা যাবে। কারণ এই return টা তখন call হবে যখন temp = NULL হয়ে যাবে।

```
int search (int v)
{
    Node * temp = head;
    int count = 0;
    while (temp != NULL)
    {
        if (temp -> val == v)
            return count;
        else
            temp = temp -> next;
```

Scanned by CamScanner

```
        count++;
    }
}
return -1;
}
```

→ index return করবে এই function টা

Delete করবো element:

- যেই element টাকে delete করবো, তার আগের index টা খুঁজে বের করতে হবে. cz ওইটা দিয়ে তার পরের element টাকে ধরতে হবে

temp দিয়ে আমরা value টার আগের element কে ধরবো।

temp 2 দিয়ে ওই element টাকেই ধরবো।

temp2 এর next কে এখন ধরবে temp. আর temp 2 কে delete করে দিবো

```
void deleted(int value)
if (head == NULL)
        cout << " No element here!!! " << endl;
else if (head -> val == value)
        pop_front();
else { Node * temp = head;
    while(tem -> next != NULL)
        {
```

[মানে, প্রথম element টাই যদি value হয়, তাহলে pop_front দিয়ে delete করে দিবো]

Scanned by CamScanner

```
        if (temp->next->val == value)
        {
            Node * temp2 = temp->next;
            temp->next = temp->next->next;
            delete temp2;
        }
        temp = temp->next;
    }
}
```

→ আচ্ছা, যদি delete করতে পারে, তাহলে 1 return করবো।
যদি delete না করতে পারে, তাহলে 0 return করবো।

```
bool ~~void~~ deleted (int value)
{
    if (head == NULL)
        return 0;
    else if (head->val == value)
        return 1;  → pop_front();
    else
    {
        Node * temp = head;
        while (temp->next != NULL)
        {
            if (temp->next->val == value)
            {
                Node * temp2 = temp->next;
                temp->next = temp->next->next;
                delete temp2;
                return 1;
            }
            temp = temp->next;
        }
    }
    return 0;
```

Scanned by CamScanner

25 February, 2019

Primitive Data types

Char → 1 byte → 8 bit.

1st bit টি MSB → Most Significant Bit. → Sign Bit.

বাকি 7 Bit এ আমরা মোট 2^7 টি Symbol রাখতে পারবো, 0 – 127 টি character.

সামনে 0 কে + , 1 কে – বোঝাবে,

Problem হচ্ছে, +0, –0 চলে আসে যেটার আসলে কোনো individual মানে নাই।

তাই, সামনে sign Bit হিসেবে যখন 1 থাকবে, তখন সেটাকে সরাসরি minus চিন্তা না করে, আমরা এটাকে 2's complement করবো

যেমন

1 00

↓

sign Bit ∴ –0 হওয়ার কথা।

But

100

011 → 1's complement

+1

2's Complement ← 100 → 4

∴ 4 answer, যেহেতু সামনে 1 দিলাম, তাই ধরে নিচ্ছি –4.

∴ –0 হয়ে গেল –4.

Scanned by CamScanner

মানে, এখন আমি −4 থেকে +3 পর্যন্ত কাজ করতে পারবো,

∴ ২টি ঘর ⟶ −2 — +1
৩টি ঘর ⟶ −4 — +3
৪টি ঘর ⟶ −8 — +7

number হিসেবে ব্যবহার করবো যেখানে আমার জায়গা short.

unsigned [মানে sign bit নাই]

∴ 4 byte এর পুরাটাই use হবে। ∴ 0−255 represent করতে পারবে।

signed মানে 3 byte use করতে পারবো,

ANSII table এ unsigned code use করা হয়,

∴ 0−255 টি code এর against এ 256 টি character থাকে, 8 bit Data ∴ $2^8 = 256$. characters.

Unicode ⟶ 2^{16} use করে সব language include করা হয়,

এখন, 32 bit এর জন্য,

unsignet bit এর ক্ষেত্রে, range ⟶ $0 - (2^{32} - 1)$

এখন, sign bit হয়ে গেলে, number এর সংখ্যা অর্ধেক হয়ে যাবে, ∴ range $0 - \left(\frac{2^{32}}{2} - 1\right)$ ∴ $0 - (2^{31} - 1)$

Scanned by CamScanner

এখন, $2^{32}-1$ এর সবগুলো জায়গা use হয়ে যায়, যদি,
তারপর আবার i++ করলাম।

তাহলে → i) gurbage আসবে না, cz, এরপরে আর কোনো ঘরই নাই।

∴ last $(2^{32}-1)$ এর Binary হলো, 01111111111 ..... 32টি
↳ positive.

এর সাথে ++ করলে, সংখ্যাটি হবে, 10000 ..... 32টি
মানে, negative এর highest টি আসবে।

যেমন: +3 → 011 কে +1 করলে আসে 100 (−4)

∴ rotation start হবে যতই +1 করতে থাকবো
মানে highest positive থেকে highest negative এ চলে যায়।

float কিভাবে রাখা হয়

যেকোনো সংখ্যা, যেমন,

1.25702.0

⇒ $1.\boxed{25702} \times 10^{5}$ → একে রাখা হয় exponent এ।

এটাকে রাখা হয় mantissa তে,

$(10.375)_{10}$

$10/2 = 5 + 0$
$5/2 = 2 + 1$
$2/2 = 1 + 0$
$1/2 = 0 + 1$

$\therefore (10)_{10} = (1010)_2$

$0.375 \times 2 = 0.75$
$0.75 \times 2 = 1.50$
$0.5 \times 2 = 1.00$

$\therefore (0.375)_{10} = (0.011)_2$

Scanned by CamScanner

$10.375 = (1010.011)_2$

$= 1.010011$ (exponent $+3$)

↳ Binary র ক্ষেত্রে, এইখানে সবসময় 1 ই থাকতো So, এই 1 কে আলাদা করে ধরার দরকার নাই।

∴ $.010011$

↳ এইটাকে mantissa তে রাখবো

exponent কে store করতে গেলে,

positive exponent এর জন্য 127 এর সাথে ex কে যোগ করা হয়, যদি যোগফল > 127 ∴ positive

< 127 ∴ negative

$= 127$ ∴ zero.

127 কেন আসছে, $2^8 = \frac{256}{2} = 128 - 1$ থেকে।

positive, negative মানে highest number হতে পারে 127,

∴ যোগ করে, বেশি হলে positive হবে।

" " কম হলে negative হবে।

এই 127 কে বলা হয় BIAS.

mantissa তে 23 bit থাকে, So, এর বেশি হলে, বাকি Bit গুলি show করবে না,

exponent এ 8 bit থাকে।

Scanned by CamScanner

এই system এর problem হলো,

→ 0.00 রাখা যাবে না, মানে 0 রাখতে পারবে না, কারন সে এই number টাকে 1.something বানানোর চেষ্টা করবে। But কোনোভাবেই 1 কে খুঁজে পাবে না তাই 0 রাখা যায় না।

যখন float এ sqrt(-1) বা 0/0 রাখা হয়, আসে, INFINITY NaN এ আসে →

* 15.875

$15/2 = 7 + 1$

$7/2 = 3 + 1$

$3/2 = 1 + 1$

$1/2 = 0 + 1$

$\therefore (15)_{10} = (1111)_2$

$.875 \times 2 = 1.75 + 1$

$.75 \times 2 = 0.5 + 1$

$.5 \times 2 = 1.0 + 0.1$

$(.875) = (.111)_2$

$\therefore (1111.111)_2$

$\therefore$ $1.\underbrace{111111}_{\text{mantissa}}$ exponent +3.

23টি Bit এর প্রথম 6টাতে 111111 বসবে, বাকিগুলো 0 থাকবে।

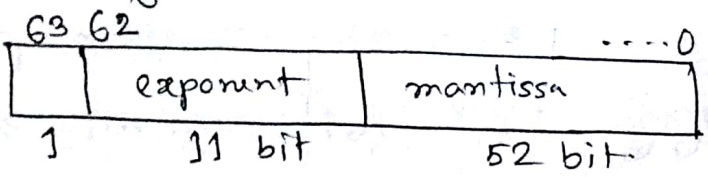
$127+3 = 130 \rightarrow 10000010$

→ এইটা থাকবে exponent এ।

Scanned by CamScanner

double এর জন্য।

64 bit শ্রেণি

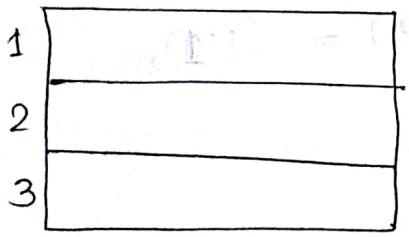


$\therefore$ এখন Bias $= \frac{2^{11}}{2} - 1 = 1023.$

mantissa — 52 bit: মানে ~~fl~~ point এরপরে আরো বেশি সংখ্যা রাখা যায়।

short int = 2 byte
long int = 4 byte / 8 byte.

2D Array:



Row major হলে,
1 এর পরে 2, এর পরে 3 নাম্বার array বলে একটা 1D array create করে।

$\therefore$ $(0,0) \longrightarrow b + 3 \times 0 + 0$ column number.
row number

$(0,1) \longrightarrow b + 3 \times 0 + 1$

$(0,2) \longrightarrow b + 3 \times 0 + 2$

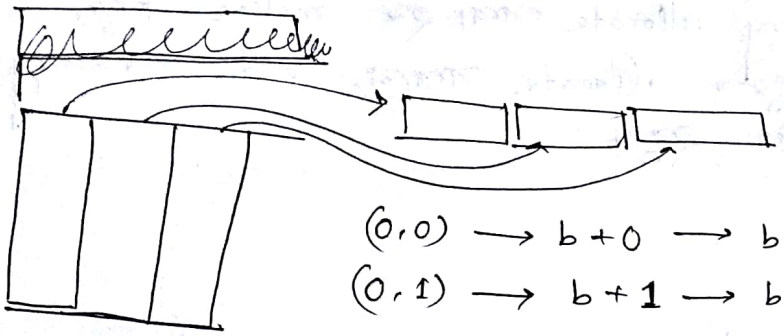
$(1,0) \longrightarrow b + 3 \times 1 + 0$

$(1,1) \longrightarrow b + 3 \times 1 + 1$

column সংখ্যা

Scanned by CamScanner

Column major :



$(0,0) \rightarrow b+0 \rightarrow b+3\times 0+0$ (row number, row, column)

$(0,1) \rightarrow b+1 \rightarrow b+3\times 0+1$

$(0,2) \rightarrow b+2 \rightarrow b+3\times 0+2$

$(1,0) \rightarrow b+3 \rightarrow b+3\times 1+0$

$(1,1) \rightarrow b+4 \rightarrow b+3\times 1+1$

int array[3][2] = {{0,1},{2,3},{4,5}}

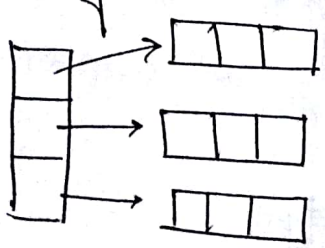
0	1
2	3
4	5

array[6] = {0,1,2,3,4,5}.

এই দুইটা same দেখাচ্ছে কারণ হচ্ছে, 2D array টাকে 1D array দিয়েই represent করা যায়।

কিন্তু double pointer হচ্ছে, 2D array'র মতই, But - পার্থক্য আছে।

** array :



একটা pointer তৈরি হয়, যেটার প্রত্যেকটা element আরেকটা array'র first element কে point করে।

Scanned by CamScanner

এখন, যেহেতু প্রত্যেকটা একেকটা pointer দিয়ে point করা, তাই, এদের memory allocate করা হয় malloc দিয়ে। আর malloc দিয়ে allocate করলে, not necessary memory continuous হবে।

26 February, 2019

Queue

Property:

1. Non-primitive linear data Structure.

Sequential → মানে একটা element এর আগের এবং পরের element নির্দিষ্ট, যেমন, tree তে একটা node এর next or node একাধিক, মানে, tree linear না, মানে,

- specific next element
- " prev "
- unique first "
- unique last "

} linear এর property.

2. Insertion / Deletation Behavious : FIFO

3. Supported operations are — enqueue
dequeue
isEmpty
isFull
makeEmpty.

Scanned by CamScanner

Queue/Stack এর size টা fixed. usually program এর একটা part এ stack/Queue use করা হয়। So. size টা জানবো

Implementation using array

* for front =
  Rear =
  Capacity =
  Length =

enqueue করার সময় দেখবো rear == capacity নাকি।
dequeue করার সময় দেখবো length = 0 নাকি।

Deque এর সময়,

Rear = (Rear + 1) mod capacity.

যদি length আর capacity র মধ্যে difference থাকে, তাহলে আমরা enque করতে পারবো।

যখন enque করবো, তখন length ++ হবে আর rear ++ হবে।

যদি rear == capacity হয়ে যায়,
তাহলে rear = 0 করে দিবো (যদি length < capacity হয়)

যখন deque করবো, তখন front++ করবো

linked list দিয়ে করতে গেলে,
- আমাদের লাগবে pop-front আর push-back.

Scanned by CamScanner

push_back (val)

tail একটা pointer নিবো যেটা last element কে point করবে।

Then নতুন node create করবো temp নামে।

tail → next = temp.

tail = temp.

নতুন একটা tail pointer নিতে হবে। for push back and ppp front.

push-back এ tail এর upgrade হচ্ছে যখন আমি একটা একটা করে element push করছি, tail ও upgrade হচ্ছে সাথে সাথে। But head upgrade হবে না।

কিন্তু pop front এর ক্ষেত্রে, head upgrade হবে যখন একটা একটা করে element pop হবে সামনে থেকে। But tail জায়গায় থাকবে। যখন, এভাবে pop করতে করতে head আর tail same element কে point করবে তখন ~~tail~~ কেও upgrade করতে হবে। কারন তখন সব element ই delete হয়ে গেছে।

Scanned by CamScanner