→ Data Structure for storing Chains :
l = chain length

① linked list
- Search, Delete, Insert O(l)
- LL is not cache friendly, nodes at different locations

② Dynamic Sized Arrays (Vector)
- S, D, I O(l) • cache friendly

③ Self Balancing BST (AVL Tree, Red Black Tree)
- S, D, I O(log l)
- not cache friendly

II Method

→ Using Open Addressing
- we use Single Array to handle collision
★ No. of slots in Hash Table ≥ No. of Keys to be inserted.
- Cache friendly.

Three Ways to implement Open Addressing
① Linear Probing
② Quadratic Probing
③ Double Hashing

Linear Probing : → hash(key, i) = (h(key) + i) % m

① Hash func. = key % 7 in circular manner
- in LP we linearly search for next empty slot & insert it, when collision occurs.

Searching
In hash table we go to that index (using hash func), if find element return true, otherwise linear traverse until you find empty slot or you traversed whole array

Date
Page

Chaining linear search example

50, 51, 49, 16, 56, 15, 19

0	49
1	50
2	51
3	16
4	56
5	15
6	19

48, 42, 50, 55, 53

0	42
1	50
2	55
3	
4	53
5	
6	48

Delete

- search then delete, make sure to mark it as deleted so that there is no error in searching.
- empty block नहीं होना है।

0	
1	50
2	51
3	15
4	

delete(51) →

0	
1	50
2	'deleted'
3	15
4	

→ Problem in Linear Probing:

Clustering

same index पर कई सारे element आ रहे हैं तो उस empty slot पर आए दोनों जिसको empty slot पर जिसको आना था वो आ नहीं पाएगा, cluster बन जाएगा।

To solve this problem

(i) Quadratic Probing $hash(key, i) = (h(key) + i^2) \% m$

(ii) Hash Table size = 2 × (no. of key)

(iii) Double Hashing

$hash(key, i) = (h_1(key) + i * h_2(key)) \% m$

Chaining

→ Hash Table never fills

→ Poor Cache friendly
→ extra space for links

→ Performance = $1 + \alpha$
Hash table size same
say $\alpha = 0.9$ (90% space full)
$= 1 + 0.9$
$= 1.9$
— it requires 1.9 operation for same.

- Better in Performance
- Better in Collision Handling
- Use when you don't know no. of keys

Open Addressing

Table may become full & resizing needed.

Cache friendly
extra space might be needed to achieve same performance as chaining
$= \frac{1}{1-\alpha}$
$= \frac{1}{1-0.9} = \frac{1}{0.1} = 10$

- it requires 10 operation to search, delete ..
- To achieve $\alpha = 1.9$ we have to use bigger array size
- Used when we know no. of keys in advance & can use extra space

used when we have only Keys.

→ No duplicates like Set

Unordered Set

classmate Date Page

~~Map~~

Set is based on Red Black Tree (Self Balancing BST)
Unordered Set ——— Hashing.

- unordered_set<int> s;
  Insert → s.insert(10);
- → element are stored in any order, b/z of Hashing;
- → you get different o/p on different compiler, depends upon hashing fun' used internally.

begin(), end(), size(), clear()
↓
clears all element in US

find()
- → finds if element is present or not
- → returns iterator to that element if not present points to s.end().
- → *s.find(15) → o/p → 15

count()
- → return 1 or 0

erase()
- → used to remove an element or range of element

s.erase(15)	~~s.erase~~
	auto it = s.find(10);
	s.erase(it);
		s.erase(s.begin(),
		s.begin()+3);
		• remove 3 elements

- → US uses Hashing internally
  Search, Insert, Delete = O(1) on Average b/z of Hashing
  ↓
  items are uniformly distributed

- begin(), end(), size(), empty() → O(1)
- insert(), erase(val), erase(it), find(), count() → O(1) on Average.

Unordered Map

- used to store (Key, Value) pairs
- Map is based on RBT
- Unordered Map — Hashing

- Map stores Key in Ordered form (default increasing Order)
- UM no ordering

- Map → Search, Insert, Delete O(log n)
- UM → ——— O(1) on Average

```
unordered_map<string, int> m;
```

Insert

```
m["gfg"] = 20;   // if Key is present it gives reference of
                 // value corresponding to that Key
                 // [] → member access operator

m["gfg"] = 20;

m.insert({"abc", 15});
```

**
```
for(auto x : m)
    cout << x.first << " " << x.second << endl;
```

↓ This loop can be used to traverse any container in C++

- Since no ordering in UM ∴ O/p is in any Order.

find()
- If key not present it point to m.end()
- returns iterator to that key.

```
auto it = m.find("gg");
if (it != m.end())
{
    cout<< it -> first << " " << it -> second;
}
```

arrow operator (iterator is an address ∴ we need to use AO)

- at() function
- If key is not present in the map, it throws out_of_range error.

eg:
```
m["hi"] = 1;
m["u"] = 2;
m["hello"] = 3;
cout<< m.at("hi");      O/p → 1
cout<< m.at("hey");     O/p → out_of_range
```

- Traverse through UM

```
for(auto it=m.begin(); it != m.end(); it++)
    cout<<(it -> first) << " " << (it -> second) << endl;
```

- count()
returns 1 or 0

size()

erase()
- m.erase("gg")
- it removes (Key, Value) pair corresponding to that Key
- m.erase(m.begin(), m.end());

- begin(), end(), size(), empty → O(1) in Worst Case
- count(), find(), [], at, erase(key), insert() → O(1) on Average search time.

(I) Count distinct Element T(n) → O(n) S(n) → O(n)

I/P a[] = {15,12,13,12,13,13,18}
O/P → 4

```
int f(int a[], int n)
{
    unordered_set<int> s;
    for(int i=0;i<n;i++)
        s.insert(a[i]);
    return s.size();
}
```

```
int f(int a[], int n)
{
    unordered_set<int> s(a, a+n);
    return s.size();
}
```

(II) Frequency of Array Element :

a[] = {10,12,10,15,10,20,12,12}

O/P → 10 3
12 3
15 1
20 1

```
int f(int a[], int n)
{
    unordered_map<int,int> m;
    for(int x : arr)
        m[x]++;
    for(auto e : m)
        cout << e.first << " " << e.second << endl;
}
```

Syntax ★

e → 1st pair ka

(III) Subarray with 0 Sum :

I/P → 4 2 -3 1 6

```
bool f(int a[], int n)
{
    unordered_set<int> m;
    int sum=0;
    for(int i=0; i<n; i++)
    {
        sum+=a[i];
        if(sum==0)
            return 1;
        else
        {   if( m.find(sum)!=m.end())
                return 1;
            else
                m.insert(sum);
        }
    } return 0;
}
```

(IV) Subarray with given Sum → print first occuring pair

I/P → {1, 2, 3, 7, 5, 6} s = 15

O/P → 3 5 (return position of indexes)

a) Sliding Window O(n), O(1)

b) Hashing O(n), O(n) + prefix Sum
Gets TLE

```
vector<int> f(int a[], int n, long long s)
{
    vector<int> ans;
    for(int i=1; i<n; i++)
        a[i] = a[i] + a[i-1];          // prefix sum
    unordered_map<int,int> m;
    for(int i=0; i<n; i++)
    {
        if(a[i] == s)                  // subarray starts from 0 index
        {
            ans.push_back(1);
            ans.push_back(i+1);
            return ans;
        }
        else
        {
            if(m.find(a[i]-s) != m.end())
            {
                ans.push_back(m[a[i]-s]+1);
                ans.push_back(i+1);
                return ans;
            }
            else
                m[a[i]] = i+1;
        }
        ans.push_back(-1);
        return ans;
    }
}
```

(V) Longest SubArray with Sum K ;

a[] = {10, 5, 2, 7, 1, 9}

K = 15

O/P = 4

longest subarray Question में
सभी possible answer निकालने हैं।

```
int f(int a[], int n, int k)
{
    for(int i=1; i<n; i++)
        a[i] += a[i-1];
    unordered_map<int,int> m;
    int ans = 0;
    for(int i=0; i<n; i++)
    {
        if(a[i] == k) // subarray start from 0 index
            ans = i+1;
        if(m.find(a[i]) == m.end())
            m[a[i]] = i;
        if(m.find(a[i]-k) != m.end())
            ans = max(ans, i - m[a[i]-k]);
    }
    return ans;
}
```

prefix sum

यदि हमें पहले मिला
no palm

Same as example

- कोई भी नया a[i] मिलेगा, उसे हम map में store करेंगे
- अगर array को change नहीं करना चाहते तो एक variable pre_sum = 0; बना लो।
  for loop के अंदर pre_sum calculate करो
  like pre_sum += a[i];
  और simply replace a[i] → pre_sum in above program.

(VI) Longest Subarray of Equal 0's & 1's :

a[] = 0 1 1 0 0 1 0 0 0

o/p → 6

TC: O(n), SC: O(n)

logic → replacing 0 → -1
now finding longest subarray sum equal 0.

```
int f(int a[], int n)
{
    for(int i=0; i<n; i++)
        if(a[i]==0)
            a[i] = -1;
    unordered_map<int,int> m;
    int ans=0, sum=0;
    for(int i=0; i<n; i++)
    {
        sum += a[i];
        if(sum==0)
            ans = i+1;
        if( m.find()!=m.end())
            ans = max(ans, i-m[sum]);
        if( m.find(sum)==m.end())
            m[sum]=i;
    }
    return ans;
}
```

(VII) Longest Span in two Binary Array:

a[] = {0,1,0,0,0,0}
b[] = {1,0,1,0,0,1}

o/p → 4

Given 2 Binary Array → find length of longest common subarray with equal sum

a[i]+a[i+1]+ - - - = b[i]+b[i+1]

logic → subtract a-b

```
bool a,b;
a=0;
b=0
cout<<a-b
```

a=1		a=1	a=0
b=0		b=1	b=1
cout<<a-b		a-b	a-b
0	1	0	-1

this output comes only when a,b is either 0 or 1
if a=6 ans. wrong

```
int f(bool a[], bool b[], int n)
{
    int c[n];
    for(int i=0;i<n;i++)
        c[i] = a[i] - b[i];
    unordered_map<int,int> m;
    int ans=0, sum=0;
    for(int i=0;i<n;i++)
    {
        sum += c[i];
        if(sum == 0)
            ans = i+1;
        if(m.find(sum) == m.end())
            m[sum] = i;
        if(m.find(sum) != m.end())
            ans = max(ans, i-m[sum]);
    }
    return ans;
}
```

Can be done using Sliding Window also

(VIII) Longest Consecutive Subsequence :

I/p a[] = {9, 3, 4, 2, 10, 5, 13, 16}

o/p → 4 (2, 3, 4, 5)

Naive → Sorting

Efficient → Hashing (O(n), O(n))

O(N) , O(N)

```
int f(int a[], int n)
{
    unordered_set<int> s;
    for(int i=0; i<n; i++)
        s.insert(a[i]);
    int ans=1;
    for(auto x: s)
    {
        if(s.find(x-1) != s.end())
            continue;
        else
        {   int j=1, k=1;
            while(s.find(x+j) != s.end())
            {   k++;
                j++;
            }
            ans = max(ans, k);
        }
    }
    return ans;
}
```

x
↓
1 2 3 4 5 - - -

अगर (x-1) है तो उसके ans में मिल जाएगा
∴ continue

(IX) Longest Substring without Repeating Characters :

s = "geeksforgeeks"

O/p → 7

s consist of letters, digits, symbols & spaces.

TC → O(|s|) SC → O(k) k → constant

अगर और DS use करोगे TLE आएगा

I

set, map

256 ASCII character

```
int f(string s)
{
    vector<int> v(256);
    int j=0, ans=0;
    for(int i=0;i<s.length();i++)
    {
        if(v[s[i]])
        {
            while( j<i && s[i]!=s[j])
            {
                v[s[j]]--;
                j++;
            }
            j++;
        }
        v[s[i]]=1;           // eebc  ये statement हम
        ans= max (ans, i-j+1);  //  नीचे लिखेंगे।
    }
    return ans;
}
```

(j on first e, i on second e)

eg) abebc (j at a, i at second b)

v | -- | 1 | 1 | -- | 1 | (97 → a, b, e)

v | -- | 1 | 1 | -- | 1 | -- | (97 → a, b, e)

हम तब तक character निकालेंगे जब तक s[i]!=s[j], फिर simply एक j++

when i=3 v[s[3]]==1
go in if then while loop executes

a b e b c (j, i)

a removed

s[i]!=s[j] false
while loop के बाहर
j++

→ जब alphabet का है string

vector<int> v(26); string s;

v[s[i]-'a']++ यह करो

~~जब~~ ASCII value of s[i] − ASCII value of 'a'

say s[i] = 'c'

99 − 97 = 2 v[| | 1 |]
2

→ जब number का हो	जब सारे ascii character हो
v[s[i]-'0']++	simply v[s[i]]++;

(X) Smallest distinct Window :

aabcbcdbca

o/p → 4

find smallest window length that contains all the characters of string atleast once.

```
int f(string s)
{
    map<char,int> m;
    unordered_set<char> us;
    for(int i=0;i<s.length();i++)
        us.insert(s[i]);
    int j=0, ans=s.length();
    for(int i=0 ; i<s.length ; i++)
    {
        m[s[i]]++;
        if(m.size()==us.size())   // ans exist अब
        {                         उसको optimize करने
                                  जा रहे
```

```
        while(m[s[j]]>=1)
        {
            m[s[j]]--;
            j++;
        }
        ans=min(ans, i-j+1);
    }
}
return ans;
}
```

★★ Hard

(XI) Minimum Window Substring

Given two string a,b
return minum window substring of a such that every character of b(including duplicates) is included in the window.

a="ADOBECODEBANC" b="ABC"
op: "BANC"

a,b consist of lowercase & uppercase alphabets

```
string f(string a, string b)
{
    unordered_map<char, int> m;
    for(int i=0; i<b.size(); i++)
        m[b[i]]++;
    int j=0, count=0; ans=0;
    count = m.size();
    string s=a;
```

count variable very important logic

s = abcdef (0 1 2 3 4 5) s.substr(1, 4)
o/p → bcde 1 index से 4 characters लेना

```
for(int i=0; i<a.length; i++)
{
    // if(m.find(a[i]) == m.end()) continue;  no need to write
    if(m.find(a[i]) != m.end())
    {
        m[a[i]]--;
        if(m[a[i]] == 0)
            count--;
    }
    if(count == 0)
    {
        while(count == 0)
        {
            if(m.find(a[j]) != m.end())
            {
                ** if(m[a[j]] == 0) break;
                m[a[j]]++;
                if(m[a[j]] == 1)
                {
                    count++;
                    continue;
                }
            }
        } j++;
        string s1 = a.substr(j, i-j+1);
        if(s1.length() < s.length())
            s = s1;
        ans++;
    }
}
if(ans == 0) return "";  // यहाँ ans exist करेगा तो ans++
return s;
}
```

answer को optimize करते हुए

Eg: a = "abbbceabfc" b = "abc"

map a→1, b→1, c→1

जब i = 4 then map → a→0, b→(-2), c→0 → -2 मतलब 2 'b' extra है

→ break चलेगा

जब i = 6 then map → a→-1, b→-2, c→0

अब while चलेगा और a→0 b→1 कर देगा, जैसे ही b→1 होगा count++ ∴ count = 1

Point to ~~Remember~~ Remember

Longest Subarray
(Subarray, Substring) having distinct element
Window size (given (K))
Find smallest Window
longest वाले Question
O(N) TC में हो
O(256) → vector<int> v(256)

Think of Hashing, 2-Pointer, Sliding Window

unordered_map<char, int> m → when key, value need to be stored
use map, set etc.

(XII) Sorting Element of an Array by Frequency:

{5,4,6,5,4,2,2,2} o/p: 2 2 2 4 4 5 5 6

{8,8,8,4,2,6,6} o/p: 8 8 8 6 6 2 4

5→2
2→3
4→2
6→1

2→3
4→2
5→2
6→1

```
static bool comp(pair<int,int>&a, pair<int,int>&b)
{
    if(a.second == b.second)
        return a.first < b.first;
    return a.second > b.second;
}
vector<int> f(int a[], int n)
{
    vector<pair<int,int>> v;
    unordered_map<int,int> m;
    for(int i=0;i<n;i++)
        m[a[i]]++;
    for(auto x:m)
        v.push_back(x);
    sort(v.begin(), v.end(), comp);
    vector<int> ans;
    int i=0;
    while(i<v.size())
    {
        while(v[i].second)
        {
            ans.push_back(v[i].first);
            v[i].second--;
        }
        i++;
    }
    return ans;
}
```

] Same the method

When you erase element in vector, vector is rearranged causing TC → O(n)

classmate
Date
Page

Pair pair<int, int> p;
to access → p.first , p.second
vector<pair<int, int>> v;
If v.size() == n say then space complexity of v is O(n)
pair taken O(1) constant space

(XIII) Count occurrence more than n/k
[10, 10, 20, 20, 10, 20, 20, 30, 30] n=8 k=4
o/p → 20, 30

n=10
k=3
10/3 = 3.3
floor(3.3)
3 से ज्यादा

a) Sorting

b) Map
```
int f(int a[], int n, int k)
{
    unordered_map<int, int> m;
    int ans=0;
    for(int i=0; i<n; i++)
        m[a[i]]++;
    for(auto x:m)
        if(x.second > (n/k))
            ans++;
    return ans;
}
```

c) O(nk) SC → O(k) Moore

String

- sequence of characters
- ASCII Table (American Standard Code for Information Interchange)

0-128 → 0-127 ASCII (standard characters)
128-255 extended ASCII (symbols like ⊥ Å ▮ √ ∫ ≥)

	characters
65-90	A-Z
97-122	a-z
48-57	0-9
0	NULL
32	space

```
int x = 'a';
cout << x;   o/p → 97
```

```
char x = 'a';
cout << (int) x;   o/p → 97
```

String

(i) First method to create string (C-style string)

```
char str[] = "gfg";
cout << str;
```

o/p → gfg → sizeof(str) → 4

(ii) using object of string class (C++ style)

Note: str[] = "gfg"

g	f	g	\0

→ whenever we use double quotes special character ('\0') is inserted automatically

str[6] = "gfg"

g	f	g	\0	\0	\0

sizeof(str) → 6

C-style (also valid for C++)
when \0 is encountered it stops.

- If double quotes not used compiler not put '\0'
  char str[] = {'g', 'f', 'g'};
  cout<<str; ⇒ gfg
  {'g', 'f', 'g', '\0'}
  ⇒ gfg
  video said

strlen(s);
strcmp(s1, s2); lexographic compare
strcpy(s1, s2); → if diff +ve s2 comes first in dict.
-ve s1

C++ String

- rich in library
- support operators like +, <, >, ==, <=, >=
- cache friendly since continuous location
- dynamic size
  c_str() to convert C++ string into C string

string str = "gfg"
↓ class object

str.length()	str + "xyz" → gfgxyz	s.substr()	s.find("fg"); → when first occurs of string not present → -1

getline(cin, str);
↳ stops when you press enter

getline(cin, str, 'b');
↓
stops when you press b

Traverse

```
for(char x: str)
    cout<<x;
```

Const is used when we don't want to modify string vector or pass by reference

(I) Check if the string is Subsequence of other

I/P → a = "ABCD" O/P → YES
b = "AD"

```
bool f(string a, string b)
{
    int j=0;
    for(int i=0; i<a.length; i++)
    {
        if(a[i]==b[j])
            j++;
    }
    if(j==b.length())
        return 1;
    return 0;
}
```

(2nd) ABC → "A", "B", "C", "AB", "AC", "BC", "ABC", ""

```
int n = a.length();
m = b.length();
for(int i=0; i<n && j<m; i++)
    if(a[i]==b[j])
        j++;
return (j==m);
```

(II) Check for Anagram:

- 2 two string is anagram if they are in permutation
- I/P → s1 = "listen", s2 = "silent"
  O/P → Yes

(A) Sorting

(B)

```
bool f(string a, string b)
{ if(a.length() != b.length())
    return 0;
  vector<int> v(26);
  for(int i=0; i<a.size(); i++)
  { v[a[i]]++; v[a[i]-'a']++;
    v[b[i]]--; v[b[i]-'a']--;
  }
  for(int i=0; i<26; i++)
    if(v[i]!=0)
      return 0;
  return 1;
}
```

v[s[i]] → v[s[i]-'a']

(III) leftmost Repeating Character:

I/p "geeksforgeeks" "abbcc" "abcd"

O/p → 0 (index) 1 -1

a)
```
int f(string s)
{
    vector<int> v(26);
    for(int i=0;i<s.length();i++)
        v[s[i]]++;
    for(int i=0;i<s.length();i++)
        if(v[s[i]]>1)
            return i;
    return -1;
}
```

abbccbd

1	3	2	1	0
0	1	2	3	4
a	b	-	-	-

b)
```
int f(string s)
{
    vector<int> v(26,-1);
    for(int i=0;i<s.length();i++)
    {   int ans=INTMAX;
        if(v[s[i]]==-1)
            v[s[i]]=i;
        else
            ans=min(ans,v[s[i]]);
    }
    return (ans==INTMAX)?-1:ans;
}
```

i) Traverse from backward since answer lie leftmost we have not maintain min().

```
int f(string s)
{   vector<bool> v(26,0);
    int ans=-1;
    for(int i=s.length()-1; i>=0; i--)
    {   if( v[s[i]])
            ans=i;
        else
            v[s[i]]=true;
    }
    return ans;
}
```

(IV) Leftmost Non Repeating Character:

"geeksforgeeks"	abcc
o/p: 5 ('f')	-1

a) same as a) of above Question with some modification

b)
```
int f(string s)
{
    vector<int> v(26,-1);
    for(int i=0;i<s.length();i++)
    {
        if( v[s[i]-'a'] ==-1)
            v[s[i]-'a']=i;
        else
            v[s[i]-'a']=-2;   // एक से ज्यादा बार आया उसके index पे -2.
    }
    for(int i=0;i<26;i++)
        if( v[s[i]-'a']>=0)
            ans=min(ans, v[s[i]-'a']);
    return (ans==INT_MAX)?-1:ans;
}
```