1. Separate chainig in hashing

vector<vector<int>> separateChaining(int hashSize,int arr[],int sizeOfArray)

{

vector<vector<int>> num(hashSize);

for(int i = 0; i<sizeOfArray; i++)

{

int key = arr[i]%hashSize;

num[key].push\_back(arr[i]);

}

return num;

}

1. Linear probing in hashing

vector<int> linearProbing(int hashSize, int arr[], int sizeOfArray)

{

vector<int> num(hashSize, -1);

for(int i = 0; i<sizeOfArray; i++)

{

int key = arr[i]%hashSize;

if(num[key]==-1 || num[key]==arr[i])

{

num[key]=arr[i];

continue;

}

int j = (key+1)%hashSize;

while(j!=key)

{

if(num[j]==-1|| num[j]==arr[i])

{

num[j]=arr[i];

break;

}

j=(j+1)%hashSize;

}

}

return num;

}

1. Quadratic probing in hashing

void QuadraticProbing(vector <int>&hash, int hashSize, int arr[], int N)

{

for(int i = 0; i<N; i++)

{

int key = arr[i]%hashSize;

if(hash[key]==-1 || hash[key]==arr[i])

{

hash[key]=arr[i];

continue;

}

int r = 1;

int j = (key+1)%hashSize;

while(j!=key)

{

if(hash[j]==-1|| hash[j]==arr[i])

{

hash[j]=arr[i];

break;

}

r++;

j=(key+r\*r)%hashSize;

}

}

return;

}

1. Count non repeated elements

int countNonRepeated(int arr[], int n)

{

set <int> s, copy;

for(int i =0;i<n;i++)

{

if(s.find(arr[i])!=s.end())

copy.insert(arr[i]);

else s.insert(arr[i]);

}

return s.size()-copy.size();

}

1. Print non repeated numbers

vector<int> printNonRepeated(int arr[],int n)

{

unordered\_set <int> s, copy;

for(int i =0;i<n;i++)

{

if(s.find(arr[i])!=s.end())

copy.insert(arr[i]);

else s.insert(arr[i]);

}

for(auto i=copy.begin();i!=copy.end();i++)

s.erase(\*i);

vector<int> v;

for(int i =0;i<n;i++)

{

if(s.find(arr[i])!=s.end())

v.push\_back(arr[i]);

}

return v;

}

1. First repeating element

int firstRepeated(int arr[], int n) {

set <int> s, cpy;

int el;

for(int i=0;i<n;i++)

{

if(s.find(arr[i])==s.end())

s.insert(arr[i]);

else

{

el =arr[i];

cpy.insert(arr[i]);

}

}

for(int i=0;i<n;i++)

{

if(cpy.find(arr[i])!=cpy.end())

return i+1;

}

return -1;

}

1. Intersection of two arrays

int NumberofElementsInIntersection(int a[], int b[], int n, int m) {

set<int> s;

int count = 0;

for(int i=0;i<n;i++)

{

s.insert(a[i]);

}

for(int i=0;i<m;i++)

{

if(s.find(b[i])!=s.end())

{

count++;

s.erase(b[i]);

}

}

return count;

}

1. Union of two arrays

int doUnion(int a[], int n, int b[], int m) {

set<int> s;

for(int i=0;i<n;i++)

{

s.insert(a[i]);

}

for(int i=0;i<m;i++)

{

s.insert(b[i]);

}

return s.size();

}

1. Hashing for pair 1

int sumExists(int arr[], int N, int sum) {

set<int> s;

for(int i=0;i<N;i++)

s.insert(arr[i]);

for(int i = 0;i<N;i++)

{

if(arr[i]\*2==sum) continue;

if(s.find(sum-arr[i])!=s.end()) return 1

}

return 0;

}

1. Hashing for pair 2

int sumExists(int arr[], int N, int sum) {

set<int> s, dup;

for(int i=0;i<N;i++)

{

if(s.find(arr[i])!=s.end())

dup.insert(arr[i]);

s.insert(arr[i]);

}

for(int i = 0;i<N;i++)

{

if(arr[i]\*2==sum) continue;

if(s.find(sum-arr[i])!=s.end()) return 1;

}

for(auto i = dup.begin();i!=dup.end();i++)

{

if((\*i)\*2==sum)

return 1;

}

return 0;

}