To Print all anagrams together

Sort those and store in hash if not exist else append in the form of hash chain.

To rotate an image by 90 degree get the transpose of the matrix and then reverse each row

**Graphs** :

Represented as adjacency matrix

struct graph{

int n\_v, n\_e

int degrees[MAX\_V]

int edges[MAX\_V][MAX\_EDGES];

}

//Make 0 the degrees of vertices degrees[0->MAX\_V]

void insertedge(int e1, int e2){

edges[e1][edges->degree[e1]] = e2;

edges[e2][edges->degree[e2]] = e1;

edges->degree[e1]++

edges->degree[e2]++

}

**Trees**

Heavy Path problem

/\* Main function which truncates the binary tree. \*/

struct Node \*prune(struct Node \*root, int sum)

{

// Base Case

if (root == NULL) return NULL;

// Recur for left and right subtrees

root->left = prune(root->left, sum - root->data);

root->right = prune(root->right, sum - root->data);

// If we reach leaf whose data is smaller than sum,

// we delete the leaf. An important thing to note

// is a non-leaf node can become leaf when its

// chilren are deleted.

if (root->left==NULL && root->right==NULL)

{

if (root->data < sum)

{

free(root);

return NULL;

}

}

return root;

}

//To find if a root to leaf path constitute a sum, On each node visit subtract sum-node\_value...if it reaches 0 on leaft node then there is a sum path from root to leaf.

> Number of nodes = return (size(node->left) + 1 + (size(node->right)

> Distance between 2 nodes - find LCA - dist 1st from root + dist 2nd from root - 2\*LCA

for LCA for Binary Tree - Store root to node paths in array for both nodes...compare the arrays till you find something common just before mismatch

Bottom view of tree : Store in hashmap left -- and right ++

Mirroring of tree is done using pre-order traversal

Use top-down approach to create a tree from array

Element Insertion in tree

if(element < ptr->element)

ptr->left = insertNode(ptr->left, element);

else

ptr->right = insertNode(ptr->right, element);

return ptr

if tree Equal:

return ( ptr1->element == ptr2->element && checkIfEqual(ptr1->left, ptr2->left) && checkIfEqual(ptr1->right, ptr2->right));

If foldable

return ( ( (ptr1 && ptr2) || (ptr1 == NULL && ptr2 == NULL) ) && checkIfFoldable(ptr1->left, ptr2->right) && checkIfFoldable(ptr1->right, ptr2->left));

**Level order Traversal with Q :**

int nodeLevel(struct treeStruct \*ptr, int nodeToFind)

{

int width = 0, local = 0;

int level = 0;

if(ptr->element == nodeToFind){

return 1;

}

q.push(ptr);

while(!q.empty())

{

++level;

local = q.size();

if(local > width)

width = local;

for(int i=0; i <= local - 1; i++)

{

struct treeStruct \*dummy = q.front();

if(dummy->element == nodeToFind)

return level;

q.pop();

if(dummy->left != NULL)

q.push(dummy->left);

if(dummy->right != NULL)

q.push(dummy->right);

}

}

return 0;

}

**> The distance between two nodes can be obtained in terms of lowest common ancestor. Following is the formula.**

Dist(n1, n2) = Dist(root, n1) + Dist(root, n2) - 2\*Dist(root, lca)

**> To Print left and right view of binary tree ...take this as a reference**

left/Right\_ViewUtil(root->left, level+1, max\_level);

put check if(max\_level < level ) print the node and max\_level = level in this case when further elements are encountered at that particular level it stops

**Check if a tree is BST**

Do in-order traversal and check sorting order

Another way to do the same

isBSTUtil(node->left, min, node->data-1) && // Allow only distinct values

isBSTUtil(node->right, node->data+1, max); // Allow only distinct values

To find next successor in BST..use a flag that is set to true for the element we are searching and then check that flag on next traversal

> **To find if root to leaf path equals a certain sum:**

For root to leaf path sum do subsum = sum - node\_element if subsum is 0 in case left and right are NULL then that is the answer

**> To check if trees are similar :**

return ( areTreesSimilar(ptr1->left, ptr2->left) && areTreesSimilar(ptr1->right, ptr2->right) );

> **To check a sum tree:**

if(ptr->element != (isSumTree(ptr->left, var) + isSumTree(ptr->right, var)) )

=======================================================================

Distributed hashtables are those which are decentralized and for each key a value might exists on multiple nodes/clusters

DHTs only directly support exact-match search

1- 9 gives 100 puzzle - 123-45-67+89 = 100

To store just 100 numbers from incoming flow of numbers, use a circular buffer/heap can also be used

to sort a big array with many repetitions use AVL with repetitions

Two sand timers which measure 11,7 minutes. take 11 and 7 as soon as 7 completes,, turn it down

now it will fall till first becomes 11.It now measures 11 mins till then it will have 4 at bottom and 3 at top, turn again and it will measure 4 mins again - 7=4+4 = 15

Anagrams

Sorting and counting the characters..we can use one array in which we increment one string and decrement for other. After operation, check if array is filled with 0s

> To print 1 - 100

Either create 100 class objects or use recursive templates

> To check if a string anagram can be palindrome

Check if length is even then every character occurs even number of times

while if the length is odd, then one except one should appear even number of times

>Fastest way to check 2 anagrams

is to map each character with prime number ad find the multiplied result of both strings

if they are equal then it is or you can take a map / Character array[26] / Sorting

> To find row wit hmax 1's

First find the indes of 1 in first row and then update that index accordinly by going LHS and put max\_row\_index that row in case left entry

of original index is 1

> To order one array based on another array

use a1 and put its values in hashmap with <ey, value> --> <number, occurences>

now traverse a2, search for that in hashmap if its found put that in p/p array and remove it from hashmap and rest of elements are sorted and appened to the o/p array

**Recursion**

word break problem

if (dictionaryContains( str.substr(0, i) ) &&

wordBreak( str.substr(i, size-i) ))

return true;

#include<stdio.h>

/\* function to multiply two numbers x and y\*/

int multiply(int x, int y)

{

/\* 0 multiplied with anything gives 0 \*/

if(y == 0)

return 0;

/\* Add x one by one \*/

if(y > 0 )

return (x + multiply(x, y-1));

/\* the case where y is negative \*/

if(y < 0 )

return -multiply(x, -y);

}

**Tower of Hanoi**

Solution of Tower of Hanoi

(Recur function)

TOH(int n, fromRod, toRod, auxRod)

1> Move (n-1) disks from Source to Auxuliary (n-1, fromRod, auxRod, toRod);

2> Move nth disk from source to destination //just print

3> Move (n-1) disk from auxiliary to source (n-1, auxRod, toRod, fromRod);

**BackTracking**

Backtracking is a method of exhaustive search using divide and conquer approaches

Applications : Generating all permutations of a bit string

A[0....n] {

A[n-1] = 0;

Func(n-1);

A[n-1] = 1;

Func(n-1);

}

=======================================================================

1. Print the left boundary in top-down manner.

2. Print all leaf nodes from left to right, which can again be sub-divided into two sub-parts:

…..2.1 Print all leaf nodes of left sub-tree from left to right.

…..2.2 Print all leaf nodes of right subtree from left to right.

3. Print the right boundary in bottom-up manner.

We need to take care of one thing that nodes are not printed again. e.g. The left most node is also the leaf node of the tree.

Flyweight design pattern is basically for memory efficiency in which objects asked for, by multiple modules are returned from a common repository, while no new objects are created if such object doesn't exists then that one is created and stored in the repo. Example can be taken as a circles of specific color stored in hashmap

Memoization is basically type of flyweight pattern

LIS -

L(i) = 1 + max { L(j) } where a[i] > a[j] else this value is 1

Sum of subarray with a given sum divided by a number X... here in this suppose starting index is A and ending index is B then the remainder for the sum from 0->A and 0->b will be the same and between them will be the numbers with a given sum so for this

1, 9, 4, 5, 2, 8, 7, 11

sum array

1, 10 , 14, 19, 21, 29, 36, 47

now suppose X is 4 then mod array is

1 , 2, 2, 3, 1, 1, 0, 3

now create a hashmap with the <numbers\_given\_above, frequency>

total will be freq1 ! + freq2 ! + freq3 !... freqn !

here it will be - 0! + 3! + 1! + 1! = 6

quickselect is a selection algorithm to find the kth smallest element in an unordered list

**BitSets**

To check the divisibility by x

for 2 : check if rightmost bit is 1 or not

for 3 : Total odd number of bits – even number of bits should be divisible by 3. Do it recursively

for 4 : Total number of set bit should be 1 and total 0's from left should be even

for 5 : ?

for 6 : Check if divisible by 2 and 3

for 7 : recursively check (n <<3) – n

for 8 : check if last 3 digits are 0

To find out number of set bits by

doing n = n&(n-1) 'x' times till n becomes 0 where x is number of bits set.

To find next divisible number by 2, find if the number has only 1 set bit by using (n !(n&n-1)) , if it is then that is the number, else right shift one-by-one and get the count. Then 1<<count is the number.

Swap 2 numbers

x = x^y;

y = x^y;

x = x^y;

These method will fail if both variables are same so put a check before swapping

Get the rightmost set bit - set\_bit\_no = xor & ~(xor-1)

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

{

if(arr[i] & set\_bit\_no)

\*x = \*x ^ arr[i]; /\*XOR of first set \*/

else

\*y = \*y ^ arr[i]; /\*XOR of second set\*/

}

Detect if two integers have opposite signs XOR the 2 numbers if they have opposite signs the result will have lftmost bit as 1 hence <0

Max profit

fill sum array in reverse direction with each entry filled with the max element on its right

now traverse from left->right and get sum[i] - a[i] and add them to get max profilt

There is a pattern in binary representation of the number that can be used to find if number is a multiple of 3. If difference between count of odd set bits (Bits set at odd positions) and even set bits is multiple of 3 then is the number

> Divide a number by 7 ------------- ( n<<3 ) - n

> To find the position of rightmost bit do 2'complement then & it with original number that will be the answer or (n&(!(n-1)));

unsigned **int** i = 1;

**char** \*c = (**char**\*)&i;

**if** (\*c)

**printf**("Little endian");

**else**

**printf**("Big endian");

// Bitwise operator based function to check divisibility by 9

**bool** isDivBy9(**int** n)

{

// Base cases

**if** (n == 0 || n == 9)

**return** **true**;

**if** (n < 9)

**return** **false**;

// If n is greater than 9, then recur for [floor(n/9) - n%8]

**return** isDivBy9((**int**)(n>>3) - (**int**)(n&7));

}

Align the most-significant ones of N and D.

1. Compute t = (N - D);.
2. If (t >= 0), then set the least significant bit of Q to 1, and set N = t.
3. Left-shift N by 1.
4. Left-shift Q by 1.
5. Go to step 2.

Inverse a bit string n^(~0)

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

{

b <<=1; // left shift b

b |= a & 0x1; //get unit bit

a >>= 1; // right shift a

}

Another trick to reverse a number

run loop from i = 0 to 16 (sizeof(data\_type) \* 4 )

check by right shifting number by & 1 | check by right shifting number by j & 1

set\_bit\_no = xor & ~(xor-1); ( rightmost set bit )

XOR of two different numbers x and y results in a number which contains set bits at the places where x and y differ. So if x and y are 10…0100 and 11…1001, then result would be 01…1101.

So the idea is to XOR all the elements in set. In the result xor, all repeating elements would nullify each other. The result would contain the set bits where two non-repeating elements differ.

Now, if we take any set bit of the result xor and again do XOR of the subset where that particular bit is set, we get the one non-repeating element. And for other non-repeating element we can take the subset where that particular bit is not set.

Find minimum using

y ^ ((x ^ y) & -(x < y))

Max

x ^ ((x ^ y) & -(x < y));

/\* This function will return n % d.

   d must be one of: 1, 2, 4, 8, 16, 32, … \*/

unsigned **int** getModulo(unsigned **int** n, unsigned **int** d)

{

**return** ( n & (d-1) );

}

2's complement of a number is 1's complement + 1

A number is divisible by 4 if it has only 1 set bit and number of 0's are even

Multiply a number by 3.5 :

get x\*3.5 by adding 2\*x, x and x/2. To calculate 2\*x, left shift x by 1 and to calculate x/2, right shift x by 2.

or you can do (8n – n ) /2

((n<<3) – n )>>1

To add 1 to a number x (say 0011000111), we need to flip all the bits after the rightmost 0 bit (we get 0011000000). Finally, flip the rightmost 0 bit also (we get 0011001000) and we are done

To flip a number use XOR operator

/\* Flip all the set bits until we find a 0 \*/

while( x & m )

{

x = x^m;

m <<= 1;

}

check divisibility by 8

if (((x >> 3) << 3) == x) then done;

Boolean Array Puzzle

Given an array of 2 elements one is having value 0 and another one having value 1

make both 0.

Solution : a[ a[1] ] = a[ a[0] ]

=======================================================================

Heaps

To calculate median from a running list of numbers calculate top elements from top of both max and min heap both will be kinda adjacent elements in heap. So find the average of those 2.

**Linked Lists**

>Merge Sort is preferred for them because in place access is not there and not so much movement is possible

>Divide linked list using hare and tortoise method

>Use sortedMerge recursive list merge call

> For 3-sum problem...take 1st list as such..sort 2nd in inc. order and 3rd in dec. order..run the O(n^2) algorithm.

>For flattening a list use mergeSort for each column with its right side column.

> Application of circular linked list :

Used by kernel which maintains the list of active processes in the circular linked list.

Used to implement queue (ex. In producer-consumer problem)

> nth from last

Take two pointers start moving second pointer only when first one has reached nth one

> To find count of elements in circular list

Take a dummy pointer pointing to head and traverse it till you again strike the head.

> To remove cycle.

Point slow pointer back to head and move both, they will meet at looping node

>Find if 2 linked lists are Merging

Use Hashtable **or** Use array storing pointer of both lists and finding repeated elements(pointers) **or**

use 2 stacks:

> check top, if equal, pop and throw: the first top's of stacks which are not matching gves the result

> Merge 2 sorted linked lists

Merge(struct linked\* a, struct linked \*b){

struct linked\* result = NULL;

if(a == NULL)

return a;

if(b == NULL)

return b;

if(a->data < b->data) {

result = a;

result->next = Merge(a->next, b)

}

else {

result = b;

result->next = Merge(a, b->next);

}

return result;

> Reverse in pairs

Store the element head->next->next which will be passed to the recursive call and reverse the first two

Node- 1

Node-2

Node-3

Save this node

> Clone linked list with next and arbit pointers

**Method-1**

Duplicate each node and put that node in between that original node and next of that original node

then do this

original->next->arbit = original->arbit->next

Its nothing just modifying the arbit pointer of interleaving duplicated list

Then restore the next pointer of original list

original->next = original->next->next;

duplicate->next = duplicate->next->next;

**Method-2**

Store the node and its next pointer to the array, next pointer of original linked list will now point to the corresponding copy linked list node and arbit pointer of copy linked list will point to corresponding original linked list.

Then,

copy->arbit = copy->arbit->arbit->next;

copy = copy->next;

restore next pointers of original linked list

**Method-3**

Hashmap approach

Store the one-to-one mapping of original and duplicated list nodes in a map. Then,

copy->arbit = orig->arbit->second;

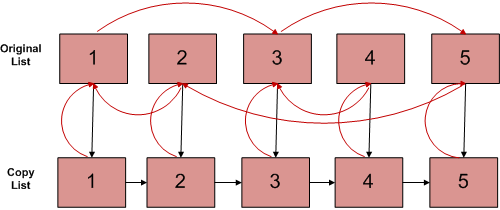


Image : arbit.gif

**Stacks**

> Find GetMin() in O(1) time using stacks

We can find the solution of it using 2 stacks – mainStack and minStack , Main Stack will be the normal stack while min stack will contains 1-to-1 elements where each position represents the minimum elements below that position. So While pushing element if that element is < the current minElement on stack we push that on minStack else we push a new copy of the top of minStack.

> Reverse a stack using only push() and pop()

Use recursion where each recursive call pops the element and store that in function stack. While returning just after the recur() call push the element back on stack, do it using another recursive function InsertAtBottom() which call itself to expel out elements till stack is empty. It then pushes that element in base case and then pushes the other elemet as a push(temp) operation placed after the recursive call.

> Infuse 3 stacks in an Array

**Method-1**

Start 2 stacks a both ends and third in middle, in case it is going to hit in top of 2, shift it to the left according to equal partitioning so that both ends of the middle stack has equal spaces

**Method-2**

All stacks are implemented as above but element insertion to the middle stack happens in alternating order.

> Given a string of S's and X's check if sequence is permissible or not

For this traverse the sequence and at any moment of time if number of X's > number of S's then the sequence is not permissible

> Find maximum rectangle area in a histogram

This can be done using Divide and Conquer technique where max area can be :

for that given subarray -> minElement\*Number\_Of\_Elements

or Max value on left side

or Max value on right side

Using Stack of indices

if stack is empty or current hist[i] > hist[top] push it on stack

else

get hist[tp] and get max area = hist[tp] \* (i – tp – 1)

**Queues**

> To implement Q using 2 stacks.

Make s1 for EnQ and s2 for DeQ,

EnQ :

Add data to s1

DeQ :

if(s2 ! Empty)

pop and return top;

else

push(s2) and pop(s1) for all elements

> One stack using 2 Qs

**Arrays**

MAX Sub SEQUENCE SUM =

max[0] = A[0];

max[1] = A[1];

max[i] = max{ A[i-1], A[i-2] + A[i] }

To merge k sorted arrays use min heap putting all elements at once then picking min element

Snake and ladder game can be implemented using a 2-D array... to find min nodes use shortest path.

**Strings**

**Longest Increasing Subsequence**

initialize all to 0 for i,0 and 0,j

if A[i] == B[j]

LCS[i, j] = 1 + LCS[i-1, j-1];

else

LCS[i, j] = max{ LCS(i-1, j) , LCS(i, j-1) }

**Longest Palindromic suubsequence**

if( A[i] == A[j])

LCS[i, j] = LCS[i-1, j-1] + 2

else if i == j

LCS[i, j] = 1

else

LCS[i, j] = max{ LCS[i-1, j] , LCS[i, j-1]}