Unit 1

1… binary search algorithm for array of integers

Binary search is the search technique that works efficiently on sorted lists. Hence, to search an element into some list using the binary search technique, we must ensure that the list is sorted.

Binary search follows the divide and conquer approach in which the list is divided into two halves, and the item is compared with the middle element of the list. If the match is found then, the location of the middle element is returned. Otherwise, we search into either of the halves depending upon the result produced through the match.

1. Binary\_Search(a, lower\_bound, upper\_bound, val) // 'a' is the given array, 'lower\_bound' is the index of the first array element, 'upper\_bound' is the index of the last array element, 'val' is the value to search
2. Step 1: set beg = lower\_bound, end = upper\_bound, pos = - 1
3. Step 2: repeat steps 3 and 4 while beg **<**=end
4. Step 3: set mid = (beg + end)/2
5. Step 4: if a[mid] = val
6. set pos = mid
7. print pos
8. go to step 6
9. else if a[mid] **>** val
10. set end = mid - 1
11. else
12. set beg = mid + 1
13. [end of if]
14. [end of loop]
15. Step 5: if pos = -1
16. print "value is not present in the array"
17. [end of if]
18. Step 6: exit

2 bubble sort algo for array of integers

**Bubble Sort** is the simplest [sorting algorithm](https://www.geeksforgeeks.org/sorting-algorithms/) that works by repeatedly swapping the adjacent elements if they are in the wrong order. This algorithm is not suitable for large data sets as its average and worst-case time complexity is quite high.

// C program for implementation of Bubble sort

#include <stdio.h>

void swap(int\* xp, int\* yp)

{

int temp = \*xp;

\*xp = \*yp;

\*yp = temp;

}

// A function to implement bubble sort

void bubbleSort(int arr[], int n)

{int i, j;

for (i = 0; i < n - 1; i++)

// Last i elements are already in place

for (j = 0; j < n - i - 1; j++)

if (arr[j] > arr[j + 1])

swap(&arr[j], &arr[j + 1]);

}

/\* Function to print an array \*/

void printArray(int arr[], int size)

{

int i;

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

printf("%d ", arr[i]);

printf("\n");

}

// Driver program to test above functions

int main()

{

int arr[] = { 5, 1, 4, 2, 8 };

int n = sizeof(arr) / sizeof(arr[0]);

bubbleSort(arr, n);

printf("Sorted array: \n");

printArray(arr, n);

return 0;

}

3 implement a selection sort for an array of integers

**Selection sort** is a simple and efficient sorting algorithm that works by repeatedly selecting the smallest (or largest) element from the unsorted portion of the list and moving it to the sorted portion of the list. The algorithm repeatedly selects the smallest (or largest) element from the unsorted portion of the list and swaps it with the first element of the unsorted portion. This process is repeated for the remaining unsorted portion of the list until the entire list is sorted. One variation of selection sort is called “Bidirectional selection sort” which goes through the list of elements by alternating between the smallest and largest element, this way the algorithm can be faster in some cases.

1. SELECTION SORT(arr, n)
3. Step 1: Repeat Steps 2 **and** 3 **for** i = 0 to n-1
4. Step 2: CALL SMALLEST(arr, i, n, pos)
5. Step 3: SWAP arr[i] with arr[pos]
6. [END OF LOOP]
7. Step 4: EXIT
9. SMALLEST (arr, i, n, pos)
10. Step 1: [INITIALIZE] SET SMALL = arr[i]
11. Step 2: [INITIALIZE] SET pos = i
12. Step 3: Repeat **for** j = i+1 to n
13. **if** (SMALL > arr[j])
14. SET SMALL = arr[j]
15. SET pos = j
16. [END OF **if**]
17. [END OF LOOP]
18. Step 4: RETURN pos

4 insertion sort algo for an array of integers ..

insertion sort works similar to the sorting of playing cards in hands. It is assumed that the first card is already sorted in the card game, and then we select an unsorted card. If the selected unsorted card is greater than the first card, it will be placed at the right side; otherwise, it will be placed at the left side. Similarly, all unsorted cards are taken and put in their exact place.

procedure insertionSort( A : array of items )

int holePosition

int valueToInsert

for i = 1 to length(A) inclusive do:

/\* select value to be inserted \*/

valueToInsert = A[i]

holePosition = i

/\*locate hole position for the element to be inserted \*/

while holePosition > 0 and A[holePosition-1] > valueToInsert do:

A[holePosition] = A[holePosition-1]

holePosition = holePosition -1

end while

/\* insert the number at hole position \*/

A[holePosition] = valueToInsert

end for

end procedure

5 Algorithm For Merge Sort

Merge sort is a sorting technique based on divide and conquer technique. With worst-case time complexity being Ο(n log n), it is one of the most respected algorithms.

Merge sort first divides the array into equal halves and then combines them in a sorted manner.

procedure mergesort( var a as array )

if ( n == 1 ) return a

var l1 as array = a[0] ... a[n/2]

var l2 as array = a[n/2+1] ... a[n]

l1 = mergesort( l1 )

l2 = mergesort( l2 )

return merge( l1, l2 )

end procedure

procedure merge( var a as array, var b as array )

var c as array

while ( a and b have elements )

if ( a[0] > b[0] )

add b[0] to the end of c remove b[0] from b

else

add a[0] to the end of c

remove a[0] from a

end if

end while

while ( a has elements )

add a[0] to the end of c

remove a[0] from a

end while

while ( b has elements )

add b[0] to the end of c

remove b[0] from b

end while

return c

end procedure

6 Algorithm For Quick Sort

*QuickSort is a sorting algorithm based on the*[*Divide and Conquer algorithm*](https://www.geeksforgeeks.org/divide-and-conquer-algorithm-introduction/)*that picks an element as a pivot and partitions the given array around the picked pivot by placing the pivot in its correct position in the sorted array*

*/\* low  –> Starting index,  high  –> Ending index \*/*

*quickSort(arr[], low, high) {  
    if (low < high) {  
        /\* pi is partitioning index, arr[pi] is now at right place \*/  
        pi = partition(arr, low, high);  
        quickSort(arr, low, pi – 1);  // Before pi  
        quickSort(arr, pi + 1, high); // After pi  
    }  
}*

**Pseudo code for partition():**

*\* This function takes last element as pivot, places the pivot element at its correct position in sorted array, and places all smaller (smaller than pivot) to left of pivot and all greater elements to right of pivot \*/*

*partition (arr[], low, high)  
{  
    // pivot (Element to be placed at right position)  
    pivot = arr[high];*

*i = (low – 1)  // Index of smaller element and indicates the   
    // right position of pivot found so far*

*for (j = low; j <= high- 1; j++){*

*// If current element is smaller than the pivot  
        if (arr[j] < pivot){  
            i++;    // increment index of smaller element  
            swap arr[i] and arr[j]  
        }  
    }  
    swap arr[i + 1] and arr[high])  
    return (i + 1)  
}*

7 linear search algorithm

Linear search is a very simple search algorithm. In this type of search, a sequential search is made over all items one by one. Every item is checked and if a match is found then that particular item is returned, otherwise the search continues till the end of the data collection.

procedure linear\_search (list, value)

for each item in the list

if match item == value

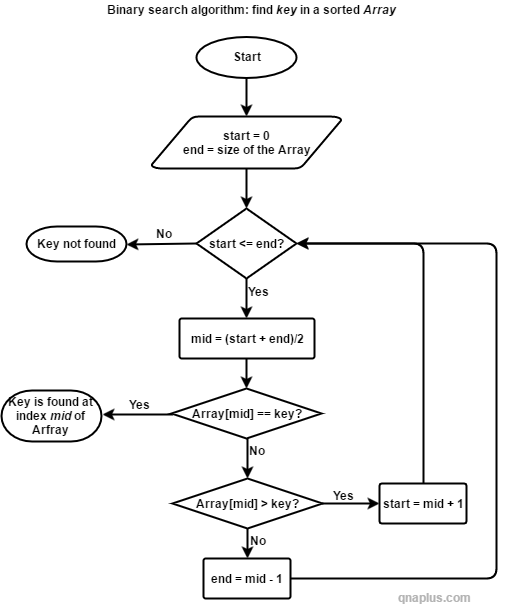
return the item's location

end if

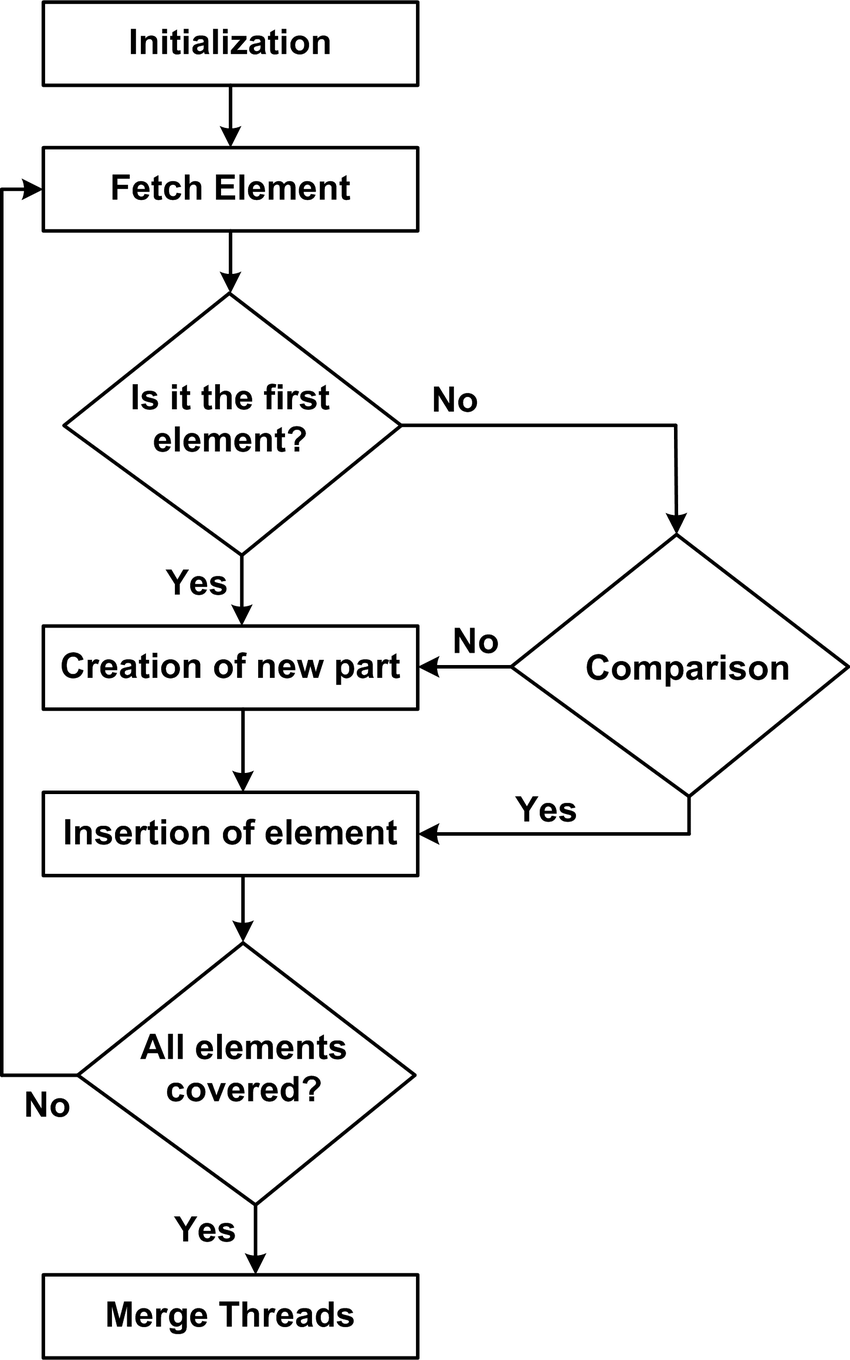
end for

end procedure

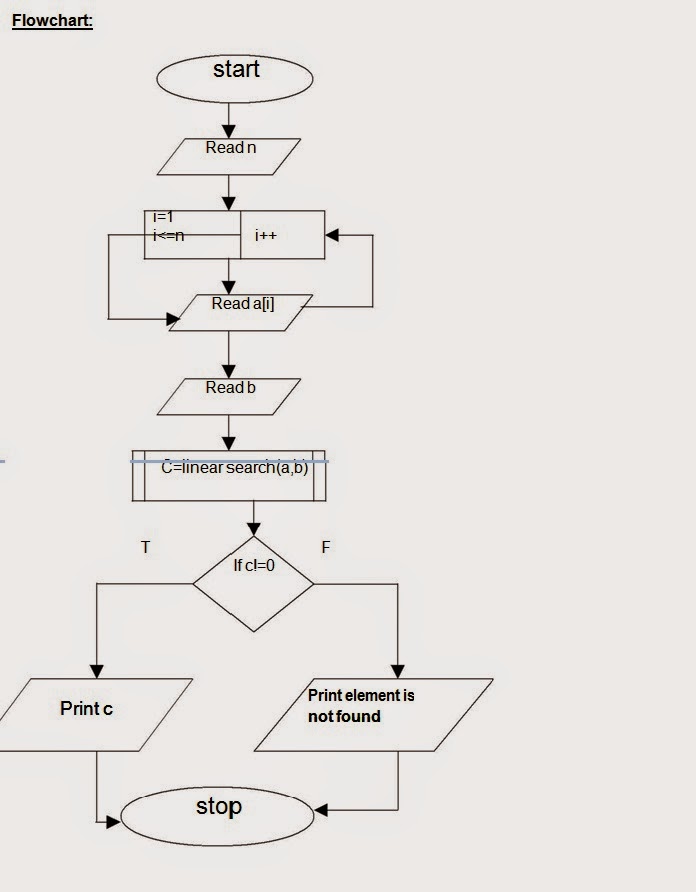
8 Flowchart For Binary Search Algorithm



.

9 Flowchart For Merge Sort 

10 Flowchart For linear Search

[](http://3.bp.blogspot.com/-bXNQW26cETQ/VJUwRumsBvI/AAAAAAAACPI/IfGB_IO_h0c/s1600/FLOWER.jpg)

Unit-2 Divide And Conquor

1.Recursive algorithm for factorial integer using divide and conquer .

// C++ program to find

// factorial of given number

#include <iostream>

using namespace std;

// Function to find factorial

// of given number

unsigned int factorial(unsigned int n)

{

if (n == 0 || n == 1)

return 1;

return n \* factorial(n - 1);

}

// Driver code

int main()

{

int num = 5;

cout << "Factorial of "

<< num << " is " << factorial(num) << endl;

return 0;

}

2.Recursive Algorithm For Fibonacci Series Using Divide And Conquer.

#include <bits/stdc++.h>

// Fibonacci Series using Recursion

using namespace std;

int fib(int n)

{

if (n <= 1)

return n;

return fib(n - 1) + fib(n - 2);

}

int main()

{

int n = 9;

cout << fib(n);

getchar();

return 0;

}

3 . Apply Divide And Conquer For Multiplication Of Two Integer.

Large Integer Multiplication is a common procedure in computer-assisted problem solving. Multiplying big numbers is not only difficult, but also time-consuming and error-prone.

**Algorithm** DC\_DUMB\_MULTIPLICATION(A, B)

where A = an-1…a1a0,

B = bn-1... b1b0 //

Output : Multiplication of A and B as C, i.e. C = A \* // Description : Perform multiplication of large numbers using divide and conquer strategy. //

Input : Number A and B,

B

4 divide and conquer for sorting a linked list

/ C++ code for linked list merged sort

#include <bits/stdc++.h>

using namespace std;

// Link list node

class node {

public:

    int data;

    node\* next;

};

// Merging two sorted lists

node\* MergeSortedList(node\* lst1, node\* lst2)

{

    node\* result = NULL;

    // Base Cases

    if (lst1 == NULL)

        return (lst2);

    else if (lst2 == NULL)

        return (lst1);

    // recursively merging two lists

    if (lst1->data <= lst2->data)

{

        result = lst1;

        result->next = MergeSortedList(lst1->next, lst2);

    }

    else {

        result = lst2;

        result->next = MergeSortedList(lst1, lst2->next);

    }

    return result;

}

// Splitting two into halves.

// If the size of the list is odd, then extra element goes in the first list.

void SplitList(node\* source, node\*\* front, node\*\* back)

{

    node\* ptr1;

    node\* ptr2;

    ptr2 = source;

    ptr1 = source->next;

    // ptr1 is incrmented twice and ptr2 is icremented once.

    while (ptr1 != NULL) {

        ptr1 = ptr1->next;

   i

// Splitting two into halves.

// If the size of the list is odd, then extra element goes in the first list.

void SplitList(node\* source, node\*\* front, node\*\* back)

{

    node\* ptr1;

    node\* ptr2;

    ptr2 = source;

    ptr1 = source->next;

    // ptr1 is incrmented twice and ptr2 is icremented once.

    while (ptr1 != NULL) {

        ptr1 = ptr1->next;

        tnode = tnode->next;

    }

}

// Push function for inserting nodes in the list.

void push(node\*\* thead, int new\_data)

{

    node\* new\_node = new node();

    new\_node->data = new\_data;

    new\_node->next = (\*thead);

    (\*thead) = new\_node;

}

// Driver Program.

int main()

{

  // Empty list

    node\* res = NULL;

    node\* MyList = NULL;

  // List: 10->4->15->1->2->12->54

    push(&MyList, 54);

    push(&MyList, 12);

    push(&MyList, 2);

    push(&MyList, 1);

    push(&MyList, 15);

    push(&MyList, 4);

    push(&MyList, 10);

    cout << "Unsorted Linked List: ";

    printList(MyList);

    cout << "\n";

    MergeSort(&MyList);

    cout << "Sorted Linked List: ";

    printList(MyList);

    return 0;

}

5.

// A divide and conquer program in C++

// to find the smallest distance from a

// given set of points.

#include <bits/stdc++.h>

using namespace std;

// A structure to represent a Point in 2D plane

class Point

{

public:

int x, y;

};

/\* Following two functions are needed for library function qsort().

Refer: http://www.cplusplus.com/reference/clibrary/cstdlib/qsort/ \*/

// Needed to sort array of points

// according to X coordinate

int compareX(const void\* a, const void\* b)

{

Point \*p1 = (Point \*)a, \*p2 = (Point \*)b;

return (p1->x - p2->x);

}

// Needed to sort array of points according to Y coordinate

int compareY(const void\* a, const void\* b)

{

Point \*p1 = (Point \*)a, \*p2 = (Point \*)b;

return (p1->y - p2->y);

}

// A utility function to find the

// distance between two points

float dist(Point p1, Point p2)

{

return sqrt( (p1.x - p2.x)\*(p1.x - p2.x) +

(p1.y - p2.y)\*(p1.y - p2.y)

);

}

// A Brute Force method to return the

// smallest distance between two points

// in P[] of size n

float bruteForce(Point P[], int n)

{

float min = FLT\_MAX;

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

for (int j = i+1; j < n; ++j)

if (dist(P[i], P[j]) < min)

min = dist(P[i], P[j]);

return min;

}

// A utility function to find

// minimum of two float values

float min(float x, float y)

{

return (x < y)? x : y;

}

// A utility function to find the

// distance between the closest points of

// strip of given size. All points in

// strip[] are sorted according to

// y coordinate. They all have an upper

// bound on minimum distance as d.

// Note that this method seems to be

// a O(n^2) method, but it's a O(n)

// method as the inner loop runs at most 6 times

float stripClosest(Point strip[], int size, float d)

{

float min = d; // Initialize the minimum distance as d

qsort(strip, size, sizeof(Point), compareY);

// Pick all points one by one and try the next points till the difference

// between y coordinates is smaller than d.

// This is a proven fact that this loop runs at most 6 times

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

for (int j = i+1; j < size && (strip[j].y - strip[i].y) < min; ++j)

if (dist(strip[i],strip[j]) < min)

min = dist(strip[i], strip[j]);

return min;

}

// A recursive function to find the

// smallest distance. The array P contains

// all points sorted according to x coordinate

float closestUtil(Point P[], int n)

{

// If there are 2 or 3 points, then use brute force

if (n <= 3)

return bruteForce(P, n);

// Find the middle point

int mid = n/2;

Point midPoint = P[mid];

// Consider the vertical line passing

// through the middle point calculate

// the smallest distance dl on left

// of middle point and dr on right side

float dl = closestUtil(P, mid);

float dr = closestUtil(P + mid, n - mid);

// Find the smaller of two distances

float d = min(dl, dr);

// Build an array strip[] that contains

// points close (closer than d)

// to the line passing through the middle point

Point strip[n];

int j = 0;

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

if (abs(P[i].x - midPoint.x) < d)

strip[j] = P[i], j++;

// Find the closest points in strip.

// Return the minimum of d and closest

// distance is strip[]

return min(d, stripClosest(strip, j, d) );

}

// The main function that finds the smallest distance

// This method mainly uses closestUtil()

float closest(Point P[], int n)

{

qsort(P, n, sizeof(Point), compareX);

// Use recursive function closestUtil()

// to find the smallest distance

return closestUtil(P, n);

}

// Driver code

int main()

{

Point P[] = {{2, 3}, {12, 30}, {40, 50}, {5, 1}, {12, 10}, {3, 4}};

int n = sizeof(P) / sizeof(P[0]);

cout << "The smallest distance is " << closest(P, n);

return 0;

}

6implement dandc for findinh kth smallest element in an unsorted array.

// C# program to implement

// the above approach

using System;

using System.Collections.Generic;

class GFG {

// Function to find the kth smallest array element

static int kthSmallest(int[] arr, int N, int K)

{

// For finding min element we need (Max

// heap)priority queue

List<int> pq = new List<int>();

for (int i = 0; i < K; i++) {

// First push first K elements into heap

pq.Add(arr[i]);

}

// Now check from k to last element

for (int i = K; i < N; i++) {

// If current element is < top that means

// there are other k-1 lesser elements

// are present at bottom thus, pop that element

// and add kth largest element into the heap

// till curr at last all the greater element

// than kth element will get pop off and at the

// top of heap there will be kth smallest

// element

if (arr[i] < pq[0]) {

pq.Sort();

pq.Reverse();

pq.RemoveAt(0);

// Push curr element

pq.Add(arr[i]);

}

}

// Return top of element

return pq[0];

}

// Driver's Code

public static void Main()

{

// Given array

int[] vec

= { 10, 5, 4, 3, 48, 15, 6, 2, 33, 53, 10 };

// Size of array

int N = vec.Length;

// Given K

int K = 4;

// Function Call

Console.WriteLine("Kth Smallest Element: "

+ kthSmallest(vec, N, K));

}

}

7> nd euclidian distance between points in 2d array using d and c

## Method 1

In this method, we first initialize two numpy arrays. Then, we use **linalg.norm()** of numpy to compute the Euclidean distance directly.

The details of the function can be found [here](https://www.educative.io/edpresso/what-is-the-nplinalgnorm-method-in-numpy).importing numpy

import numpy as np

#initializing two arrays

array1 = np.array([1,2,3,4,5])

array2 = np.array([7,6,5,4,3])

#computing the Euclidan distance

temp = array1 - array2

distance = np.linalg.norm(temp)

print("Euclidean Distance: ", distance)

8>

The meaning and full form of GCD is the Greatest Common Divisor. So, GCD is the greatest positive number which is a common divisor for a given set of positive numbers.

#include <stdio.h>

int main()

{

int n1, n2, i, gcd;

printf("Enter two integers: ");

scanf("%d %d", &n1, &n2);

for(i=1; i <= n1 && i <= n2; ++i)

{

// Checks if i is factor of both integers

if(n1%i==0 && n2%i==0)

gcd = i;

}

printf("G.C.D of %d and %d is %d", n1, n2, gcd);

return 0;

}

In this program, two integers entered by the user are stored in variable n1 and n2.Then, for loop is iterated until i is less than n1 and n2.

In each iteration, if both n1 and n2 are exactly divisible by i, the value of i is assigned to gcd.

When the for loop is completed, the greatest common divisor of two numbers is stored in variable gcd.

Thankyou:

Unit-3: all the problems based on dynamic programming.

1. Fibonnaci series……….

def feb(n\_)

|  |  |
| --- | --- |
|  | if n == 0: |
|  | return 0 |
|  | if n == 1: |
|  | return 1 |
|  |  |
|  | return fib(n - 1) + fib(n - 2) |
|  |  |
|  | print(fib(6)) |
|  |  |
|  | # output: 8  2>lcs problem  // Dynamic Programming implementation of LCS problem  using System;  class GFG {  // Returns length of LCS for X[0..m-1], Y[0..n-1]  static int lcs(String X, String Y, int m, int n)  {  int[, ] L = new int[m + 1, n + 1];  // Following steps build L[m+1][n+1]  // in bottom up fashion.  // Note that L[i][j] contains length of  // LCS of X[0..i-1] and Y[0..j-1]  for (int i = 0; i <= m; i++) {  for (int j = 0; j <= n; j++) {  if (i == 0 || j == 0)  L[i, j] = 0;  else if (X[i - 1] == Y[j - 1])  L[i, j] = L[i - 1, j - 1] + 1;  else  L[i, j] = max(L[i - 1, j], L[i, j - 1]);  }  }  return L[m, n];  }  // Utility function to get max of 2 integers  static int max(int a, int b) { return (a > b) ? a : b; }  // Driver code  public static void Main()  {  String S1 = "AGGTAB";  String S2 = "GXTXAYB";  int m = S1.Length;  int n = S2.Length;  Console.Write("Length of LCS is"  + " " + lcs(S1, S2, m, n));  }  } |

3>Knapsack Problem

// A Dynamic Programming based solution for

// 0-1 Knapsack problem

using System;

class GFG {

// A utility function that returns

// maximum of two integers

static int max(int a, int b) { return (a > b) ? a : b; }

// Returns the maximum value that

// can be put in a knapsack of

// capacity W

static int knapSack(int W, int[] wt, int[] val, int n)

{

int i, w;

int[, ] K = new int[n + 1, W + 1];

// Build table K[][] in bottom

// up manner

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

for (w = 0; w <= W; w++) {

if (i == 0 || w == 0)

K[i, w] = 0;

else if (wt[i - 1] <= w)

K[i, w] = Math.Max(

val[i - 1]

+ K[i - 1, w - wt[i - 1]],

K[i - 1, w]);

else

K[i, w] = K[i - 1, w];

}} return K[n, W];

}

// Driver code

static void Main()

{

int[] profit = new int[] { 60, 100, 120 };

int[] weight = new int[] { 10, 20, 30 };

int W = 50;

int n = profit.Length;

Console.WriteLine(knapSack(W, weight, profit, n));

}}

4> rod cutting problem dynamic programming

***include*** *<stdio.h>*

***#define MAX(x, y) (((x) > (y)) ? (x) : (y))***

***const******int*** *INF = 100000;*

***int*** *r[5+1];*

***void*** *init\_r() {*

***int*** *i;*

*r[0] = 0;*

***for****(i=1; i<=5; i++) {*

*r[i] = -1\*INF;*

*}*

*}*

***int*** *top\_down\_rod\_cutting(****int*** *c[],* ***int*** *n) {*

***if*** *(r[n] >= 0) {*

***return*** *r[n];*

*}*

***int*** *maximum\_revenue = -1\*INF;*

***int*** *i;*

***for****(i=1; i<=n; i++) {*

*maximum\_revenue = MAX(maximum\_revenue, c[i] + top\_down\_rod\_cutting(c, n-i));*

*}*

*r[n] = maximum\_revenue;*

***return*** *r[n];*

*}*

***int*** *main() {*

*init\_r();*

*// array starting from 1, element at index 0 is fake*

***int*** *c[] = {0, 10, 24, 30, 40, 45};*

*printf("%d\n", top\_down\_rod\_cutting(c, 5));*

***return*** *0;*

*}*

5>

using System;

class GFG

{

    // Matrix Ai has dimension p[i-1] x p[i]

    // for i = 1..n

    static int MatrixChainOrder(int[] p, int n)

    {

  /\* For simplicity of the program, one

        extra row and one extra column are

        allocated in m[][]. 0th row and 0th

        column of m[][] are not used \*/

        int[, ] m = new int[n, n];

        int i, j, k, L, q;

        /\* m[i, j] = Minimum number of scalar

        multiplications needed

        to compute the matrix A[i]A[i+1]...A[j]

        = A[i..j] where dimension of A[i] is

        p[i-1] x p[i] \*/

        // cost is zero when multiplying

        // one matrix.

        for (i = 1; i < n; i++)

            m[i, i] = 0;

  // L is chain length.

        for (L = 2; L < n; L++)

        {

            for (i = 1; i < n - L + 1; i++)

            {

                j = i + L - 1;

                if (j == n)

                    continue;

                m[i, j] = int.MaxValue;

                for (k = i; k <= j - 1; k++)

                {

                    // q = cost/scalar multiplications

                    q = m[i, k] + m[k + 1, j]

                        + p[i - 1] \* p[k] \* p[j];

                    if (q < m[i, j])

                        m[i, j] = q;

                }

     }

        }

        return m[1, n - 1];

    }

    // Driver code

    public static void Main()

    {

        int[] arr = new int[] { 1, 2, 3, 4 };

        int size = arr.Length;

        Console.Write("Minimum number of "

+ "multiplications is "

                      + MatrixChainOrder(arr, size));

    }

}

6>edit distance problem in c .

iven two strings str1 and str2 and below operations that can be performed on str1. Find minimum number of edits (operations) required to convert ‘str1’ into ‘str2’.

1. Insert
2. Remove
3. Replace
4. using System;
6. class GFG {
7. static int min(int x, int y, int z)
8. {
9. if (x <= y && x <= z)
10. return x;
11. if (y <= x && y <= z)
12. return y;
13. else
14. return z;
15. }
17. static int editDistDP(String str1, String str2, int m,
18. int n)
19. {
20. // Create a table to store
21. // results of subproblems
22. int[, ] dp = new int[m + 1, n + 1];
24. // Fill d[][] in bottom up manner
25. for (int i = 0; i <= m; i++) {
26. for (int j = 0; j <= n; j++) {
27. // If first string is empty, only option is
28. // to insert all characters of second string
29. if (i == 0)
31. // Min. operations = j
32. dp[i, j] = j;
34. // If second string is empty, only option is
35. // to remove all characters of second string
36. else if (j == 0)
38. // Min. operations = i
39. dp[i, j] = i;
41. // If last characters are same, ignore last
42. // char and recur for remaining string
43. else if (str1[i - 1] == str2[j - 1])
44. dp[i, j] = dp[i - 1, j - 1];
46. // If the last character is different,
47. // consider all possibilities and find the
48. // minimum
49. else
50. dp[i, j] = 1
51. + min(dp[i, j - 1], // Insert
52. dp[i - 1, j], // Remove
53. dp[i - 1,
54. j - 1]); // Replace
55. }
56. }
58. return dp[m, n];
59. }
60. // Driver code
61. public static void Main()
62. {
63. String str1 = "sunday";
64. String str2 = "saturday";
65. Console.Write(editDistDP(str1, str2, str1.Length,
66. str2.Length));
67. }
68. }

37> coin change problem using dp……

/\* We have input values of N and

an array Coins that holds all of

the coins. We use data type of

long because we want to be able

to test large values without

integer overflow\*/

using System;

public class getWays

{

static long getNumberOfWays(long N, long[] Coins)

{

// Create the ways array to 1 plus the amount

// to stop overflow

long[] ways = new long[(int)N + 1];

// Set the first way to 1 because its 0 and

// there is 1 way to make 0 with 0 coins

ways[0] = 1;

// Go through all of the coins

for (int i = 0; i < Coins.Length; i++)

{

// Make a comparison to each index value

// of ways with the coin value.

for (int j = 0; j < ways.Length; j++)

{

if (Coins[i] <= j)

{

// Update the ways array

ways[j] += ways[(int)(j - Coins[i])];

}

}

}

// return the value at the Nth position

// of the ways array.

return ways[(int)N];

}

static void printArray(long[] coins)

{

foreach (long i in coins)

Console.WriteLine(i);

}

// Driver code

public static void Main(String []args)

{

long []Coins = { 1, 5, 10 };

Console.WriteLine("The Coins Array:");

printArray(Coins);

Console.WriteLine("Solution:");

Console.WriteLine(getNumberOfWays(12, Coins));

}

}

8> maximum sum subarray…………………..

//Approach 2  
//Runtime: 6ms  
//Memory usage: 38.6MBclass Solution {  
 public int maxSubArray(int[] nums) {  
 int[] sum = new int[nums.length];  
 sum[0] = nums[0];  
 int max = nums[0];  
 for(int i=1; i<nums.length; i++){  
 sum[i] = Math.max(sum[i-1]+nums[i], nums[i]);  
 max = Math.max(max, sum[i]);  
 }  
 return max;  
 }  
}

9.longest increasing subsequence

#include<stdio.h>

#include<string.h>

int LCS( char \*A, char \*B, int x, int y )

{

if (x == 0 || y == 0)

return 0;

if (A[x-1] == B[y-1])

return 1 + LCS(A, B, x-1, y-1);

else

return max(LCS(A, B, x, y-1), LCS(A, B, x-1, y));

}

int max(int m, int n)

{

return (m > n)? m : n;

}

int main()

{

char A[] = "XY";

char B[] = "XPYQ";

int x = strlen(A);

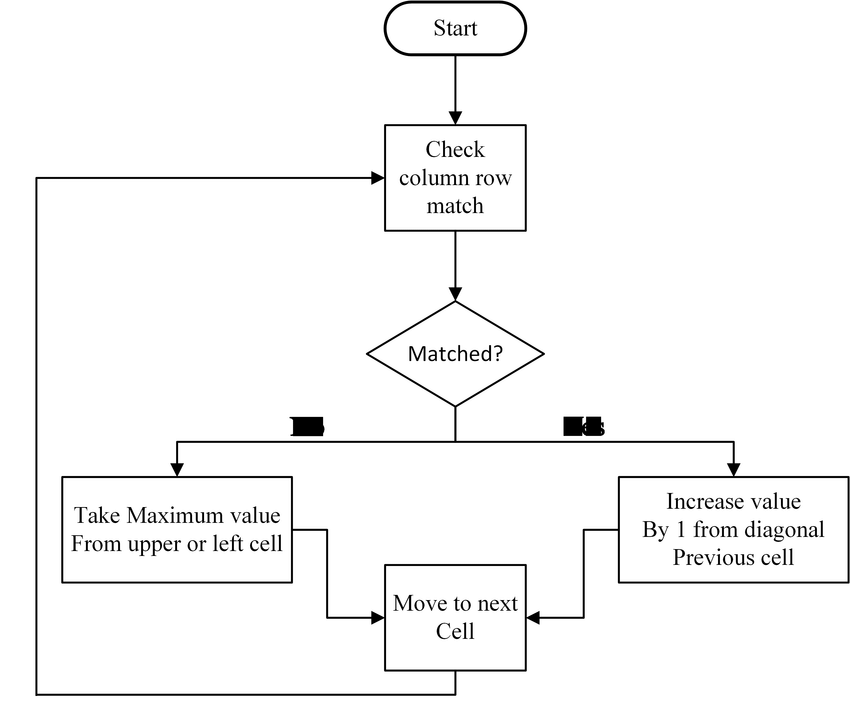
int y = strlen(B);

printf("Length of LCS is %d", LCS( A, B, x, y ));

return 0;

}

40. flowchart for longest common substring problem ..



Unit-7

1. Parallel algorithm for matrix multiplication…

double **optimizedParallelMultiply**(TYPE\*\* matrixA, TYPE\*\* matrixB, TYPE\*\* matrixC, int dimension){  
  
 int i, j, k, iOff, jOff;  
 TYPE tot;  
  
 struct timeval t0, t1;  
 gettimeofday(&t0, 0); convert(matrixA, matrixB, dimension);  
 #pragma omp parallel shared(matrixC) private(i, j, k, iOff, jOff, tot) num\_threads(40)  
 {  
 #pragma omp for schedule(static)  
 for(i=0; i<dimension; i++){  
 iOff = i \* dimension;  
 for(j=0; j<dimension; j++){  
 jOff = j \* dimension;  
 tot = 0;  
 for(k=0; k<dimension; k++){  
 tot += flatA[iOff + k] \* flatB[jOff + k];  
 }  
 matrixC[i][j] = tot;  
 }  
 }  
 }

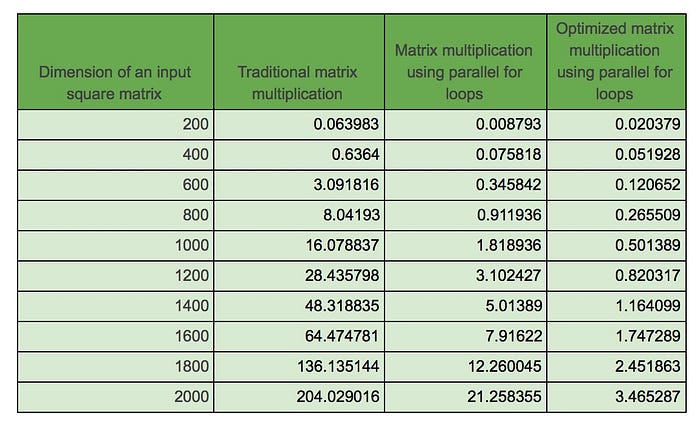
gettimeofday(&t1, 0);  
double elapsed = (t1.tv\_sec-t0.tv\_sec) \* 1.0f + (t1.tv\_usec - t0.tv\_usec) / 1000000.0f;  
  
return elapsed

Here we have launched 40 threads to do the multiplication process. Internally we have divided the workload in static manner assuming that each multiplication instruction would take same amount of time. Since we are dealing with dimensions of 200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800 and 2000, workload can be divided equally among threads.

In **omp**we have explicitly declared that **matrixC**as shared resource to avoid race conditions.

**Analyze Performance**

Following table contains time taken for each dimension at each approach.



2>develop randomized algo for k median problem

/\* C++ program to find Approximate Median using

1/2 Approximate Algorithm \*/

#include<bits/stdc++.h>

using namespace std;

// This function returns the Approximate Median

int randApproxMedian(int arr[],int n)

{

// Declaration for the random number generator

random\_device rand\_dev;

mt19937 generator(rand\_dev());

// Random number generated will be in the range [0,n-1]

uniform\_int\_distribution<int> distribution(0, n-1);

if (n==0)

return 0;

int k = 10\*log2(n); // Taking c as 10

// A set stores unique elements in sorted order

set<int> s;

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

{

// Generating a random index

int index = distribution(generator);

//Inserting into the set

s.insert(arr[index]);

}

set<int> ::iterator itr = s.begin();

// Report the median of the set at k/2 position

// Move the itr to k/2th position

advance(itr, (s.size()/2) - 1);

// Return the median

return \*itr;

}

// Driver method to test above method

int main()

{

int arr[] = {1, 3, 2, 4, 5, 6, 8, 7};

int n = sizeof(arr)/sizeof(int);

printf("Approximate Median is %d\n",randApproxMedian(arr,n));

return 0

}

1. Parallel breadth first search algorihm…

The [breadth-first-search algorithm](https://en.wikipedia.org/wiki/Breadth-first_search) is a way to explore the vertices of a graph layer by layer. It is a basic algorithm in graph theory which can be used as a part of other graph algorithms. For instance, BFS is used by [Dinic's algorithm](https://en.wikipedia.org/wiki/Dinic%27s_algorithm) to find maximum flow in a graph. Moreover, BFS is also one of the kernel algorithms in [Graph500](https://en.wikipedia.org/wiki/Graph500) benchmark, which is a benchmark for data-intensive supercomputing problems.[[1]](https://en.wikipedia.org/wiki/Parallel_breadth-first_search#cite_note-graph500-1) This article discusses the possibility of speeding up BFS through the use of [parallel computing](https://en.wikipedia.org/wiki/Parallel_computing).

**define** bfs\_sequential(graph(V,E), source s):

2 **for** all v in V **do**

3 d[v] = -1;

4 d[s] = 0; level = 1; FS = {}; NS = {};

5 push(s, FS);

6 **while** FS !empty **do**

7 **for** u in FS **do**

8 **for** each neighbour v of u **do**

9 **if** d[v] = -1 **then**

10 push(v, NS);

11 d[v] = level;

12 FS = NS, NS = {}, level = level + 1;

**define** 1\_D\_distributed\_memory\_BFS( graph(V,E), source s):

2 *//normal initialization*

3 **for** all v in V **do**

4 d[v] = -1;

5 d[s] = 0; level = 0; FS = {}; NS = {};

6 *//begin BFS traversal*

7 **while** True **do**:

8 FS = {the set of local vertices with level}

9 *//all vertices traversed*

10 **if** FS = {} for all processors **then**:

11 terminate the while loop

12 *//construct the NS based on local vertices in current frontier*

13 NS = {neighbors of vertices in FS, both local and not local vertices}

*//synchronization: all-to-all communication*

15 **for** 0 <= j < p **do**:

16 N\_j = {vertices in NS owned by processor j}

17 **send** N\_j to processor j

18 **receive** N\_j\_rcv from processor j

19 *//combine the received message to form local next vertex frontier then update the level for them*

20 NS\_rcv = Union(N\_j\_rcv)

21 **for** v in NS\_rcv **and** d[v] == -1 **do**

22 d[v] = level + 1

5> algo for mst problrm …….

// A C# program for Prim's Minimum

// Spanning Tree (MST) algorithm.

// The program is for adjacency

// matrix representation of the graph

using System;

class MST {

// Number of vertices in the graph

static int V = 5;

// A utility function to find

// the vertex with minimum key

// value, from the set of vertices

// not yet included in MST

static int minKey(int[] key, bool[] mstSet)

{

// Initialize min value

int min = int.MaxValue, min\_index = -1;

for (int v = 0; v < V; v++)

if (mstSet[v] == false && key[v] < min) {

min = key[v];

min\_index = v;

}

return min\_index;

}

// A utility function to print

// the constructed MST stored in parent[]

static void printMST(int[] parent, int[, ] graph)

{

Console.WriteLine("Edge \tWeight");

for (int i = 1; i < V; i++)

Console.WriteLine(parent[i] + " - " + i + "\t"

+ graph[i, parent[i]]);

}

// Function to construct and

// print MST for a graph represented

// using adjacency matrix representation

static void primMST(int[, ] graph)

{

// Array to store constructed MST

int[] parent = new int[V];

// Key values used to pick

// minimum weight edge in cut

int[] key = new int[V];

// To represent set of vertices

// included in MST

bool[] mstSet = new bool[V];

// Initialize all keys

// as INFINITE

for (int i = 0; i < V; i++) {

key[i] = int.MaxValue;

mstSet[i] = false;

}

// Always include first 1st vertex in MST.

// Make key 0 so that this vertex is

// picked as first vertex

// First node is always root of MST

key[0] = 0;

parent[0] = -1;

// The MST will have V vertices

for (int count = 0; count < V - 1; count++) {

// Pick the minimum key vertex

// from the set of vertices

// not yet included in MST

int u = minKey(key, mstSet);

// Add the picked vertex

// to the MST Set

mstSet[u] = true;

// Update key value and parent

// index of the adjacent vertices

// of the picked vertex. Consider

// only those vertices which are

// not yet included in MST

for (int v = 0; v < V; v++)

// graph[u][v] is non zero only

// for adjacent vertices of m

// mstSet[v] is false for vertices

// not yet included in MST Update

// the key only if graph[u][v] is

// smaller than key[v]

if (graph[u, v] != 0 && mstSet[v] == false

&& graph[u, v] < key[v]) {

parent[v] = u;

key[v] = graph[u, v];

}

}

// Print the constructed MST

printMST(parent, graph);

}

// Driver's Code

public static void Main()

{

int[, ] graph = new int[, ] { { 0, 2, 0, 6, 0 },

{ 2, 0, 3, 8, 5 },

{ 0, 3, 0, 0, 7 },

{ 6, 8, 0, 0, 9 },

{ 0, 5, 7, 9, 0 } };

// Print the solution

primMST(graph);

}

}

6 minimum cut problem randomised algorithm

using System;

using System.IO;

using System.Collections.Generic;

class GFG {

// a structure to represent a unweighted edge in graph

public class Edge {

public int src, dest;

public Edge(int s, int d)

{

this.src = s;

this.dest = d;

}

}

// a structure to represent a connected, undirected

// and unweighted graph as a collection of edges.

public class Graph {

// V-> Number of vertices, E-> Number of edges

public int V, E;

// graph is represented as an array of edges.

// Since the graph is undirected, the edge

// from src to dest is also edge from dest

// to src. Both are counted as 1 edge here.

public Edge[] edge;

public Graph(int v, int e)

{

this.V = v;

this.E = e;

this.edge = new Edge[e];

}

}

// A structure to represent a subset for union-find

public class subset {

public int parent;

public int rank;

public subset(int p, int r)

{

this.parent = p;

this.rank = r;

}

}

// A very basic implementation of Karger's randomized

// algorithm for finding the minimum cut. Please note

// that Karger's algorithm is a Monte Carlo Randomized

// algo and the cut returned by the algorithm may not be

// minimum always

public static int kargerMinCut(Graph graph)

{

// Get data of given graph

int V = graph.V, E = graph.E;

Edge[] edge = graph.edge;

// Allocate memory for creating V subsets.

subset[] subsets = new subset[V];

// Create V subsets with single elements

for (int v = 0; v < V; ++v) {

subsets[v] = new subset(v, 0);

}

// Initially there are V vertices in

// contracted graph

int vertices = V;

// Keep contracting vertices until there are

// 2 vertices.

while (vertices > 2) {

// Pick a random edge

int i = ((int)(new Random().NextDouble() \* 10))

% E;

// Find vertices (or sets) of two corners

// of current edge

int subset1 = find(subsets, edge[i].src);

int subset2 = find(subsets, edge[i].dest);

// If two corners belong to same subset,

// then no point considering this edge

if (subset1 == subset2) {

continue;

}

// Else contract the edge (or combine the

// corners of edge into one vertex)

else {

Console.WriteLine("Contracting edge "

+ edge[i].src + "-"

+ edge[i].dest);

vertices--;

Union(subsets, subset1, subset2);

}

}

// Now we have two vertices (or subsets) left in

// the contracted graph, so count the edges between

// two components and return the count.

int cutedges = 0;

for (int i = 0; i < E; i++) {

int subset1 = find(subsets, edge[i].src);

int subset2 = find(subsets, edge[i].dest);

if (subset1 != subset2) {

cutedges++;

}

}

return cutedges;

}

// A utility function to find set of an element i

// (uses path compression technique)

public static int find(subset[] subsets, int i)

{

// find root and make root as parent of i

// (path compression)

if (subsets[i].parent != i) {

subsets[i].parent

= find(subsets, subsets[i].parent);

}

return subsets[i].parent;

}

// A function that does union of two sets of x and y

// (uses union by rank)

public static void Union(subset[] subsets, int x, int y)

{

int xroot = find(subsets, x);

int yroot = find(subsets, y);

// Attach smaller rank tree under root of high

// rank tree (Union by Rank)

if (subsets[xroot].rank < subsets[yroot].rank) {

subsets[xroot].parent = yroot;

}

else if (subsets[xroot].rank

> subsets[yroot].rank) {

subsets[yroot].parent = xroot;

}

// If ranks are same, then make one as root and

// increment its rank by one

else {

subsets[yroot].parent = xroot;

subsets[xroot].rank++;

}

}

// Driver program to test above functions

public static void Main()

{

// Let us create following unweighted graph

// 0------1

// | \ |

// | \ |

// | \ |

// | \ |

// 3------2

int V = 4, E = 5;

Graph graph = new Graph(V, E);

// add edge 0-1

graph.edge[0] = new Edge(0, 1);

// add edge 0-2

graph.edge[1] = new Edge(0, 2);

// add edge 0-3

graph.edge[2] = new Edge(0, 3);

// add edge 1-2

graph.edge[3] = new Edge(1, 2);

// add edge 2-3

graph.edge[4] = new Edge(2, 3);

// Use a different seed value for every run.

Random r = new Random();

int res = kargerMinCut(graph);

Console.WriteLine(

"Cut found by Karger's randomized algo is "

+ res);

}

Unit -6

Using greedy find knapsack problem

### Knapsack Problem Using Greedy Method Pseudocode

A pseudo-code for solving knapsack problems using the greedy method is;  
greedy fractional-knapsack (P[1...n], W[1...n], X[1..n]. M)  
/\*P[1...n] and W[1...n] contain the profit and weight of the n-objects ordered such that X[1...n] is a solution set and M is the capacity of knapsack\*/  
{

For j ← 1 to n do  
X[j]← 0  
profit ← 0 // Total profit of item filled in the knapsack  
weight ← 0 // Total weight of items packed in knapsacks  
j ← 1  
While (Weight < M) // M is the knapsack capacity

In this method, the Knapsack's filling is done so that the maximum capacity of the knapsack is utilized so that maximum profit can be earned from it. The knapsack problem using the Greedy Method is referred to as:  
Given a list of n objects, say {I1, I2,……, In) and a knapsack (or bag).  
The capacity of the knapsack is M.  
Each object Ij has a weight wj and a profit of pj  
If a fraction xj (where x ∈ {0...., 1)) of an object Ij is placed into a knapsack, then a profit of pjxj is earned.  
The problem (or Objective) is to fill the knapsack (up to its maximum capacity M), maximizing the total profit earned.

1. Vertex cover problrm …

// C# Program to print Vertex

// Cover of a given undirected

// graph

using System;

using System.Collections.Generic;

// This class represents an

// undirected graph using

// adjacency list

class Graph{

// No. of vertices

public int V;

// Array of lists for

// Adjacency List Representation

public List<int> []adj;

// Constructor

public Graph(int v)

{

V = v;

adj = new List<int>[v];

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

adj[i] = new List<int>();

}

//Function to add an edge

// into the graph

void addEdge(int v, int w)

{

// Add w to v's list.

adj[v].Add(w);

//Graph is undirected

adj[w].Add(v);

}

// The function to print

// vertex cover

void printVertexCover()

{

// Initialize all vertices

// as not visited.

bool []visited = new bool[V];

// Consider all edges one

// by one

for (int u = 0; u < V; u++)

{

// An edge is only picked

// when both visited[u]

// and visited[v] are false

if (visited[u] == false)

{

// Go through all adjacents

// of u and pick the first

// not yet visited vertex

// (We are basically picking

// an edge (u, v) from remaining

// edges.

foreach(int i in adj[u])

{

int v = i;

if (visited[v] == false)

{

// Add the vertices (u, v)

// to the result set. We

// make the vertex u and

// v visited so that all

// edges from/to them would

// be ignored

visited[v] = true;

visited[u] = true;

break;

}

}

}

}

// Print the vertex cover

for (int j = 0; j < V; j++)

if (visited[j])

Console.Write(j + " ");

}

// Driver method

public static void Main(String []args)

{

// Create a graph given in

// the above diagram

Graph g = new Graph(7);

g.addEdge(0, 1);

g.addEdge(0, 2);

g.addEdge(1, 3);

g.addEdge(3, 4);

g.addEdge(4, 5);

g.addEdge(5, 6);

g.printVertexCover();

}

}

1. Maximum clique problem…

### Pseudocode

Algorithm: Max-Clique (G, n, k)

S := ф

for i = 1 to k do

t := choice (1…n)

if t є S then

return failure

S := S U t

for all pairs (i, j) such that i є S and j є S and i ≠ j do

if (i, j) is not a edge of the graph then

return failure

return success

### Analysis

Max-Clique problem is a non-deterministic algorithm. In this algorithm, first we try to determine a set of **k** distinct vertices and then we try to test whether these vertices form a complete graph.

There is no polynomial time deterministic algorithm to solve this problem. This problem is NP-Complete.

1. Graph coloring algorithm …..

[Graph coloring](http://en.wikipedia.org/wiki/Graph_coloring) problem is to assign colors to certain elements of a graph subject to certain constraints.

**Vertex coloring** is the most common graph coloring problem. The problem is, given m colors, find a way of coloring the vertices of a graph such that no two adjacent vertices are colored using same color. The other graph coloring problems like ***Edge Coloring*** (No vertex is incident to two edges of same color) and ***Face Coloring***(Geographical Map Coloring) can be transformed into vertex coloring.

### Algorithm for graph coloring

Algorithm GRAPH COLORING(G, COLOR, i)

Description: Solve the graph coloring problem using backtracking

//Input: Graph G with n vertices, list of colors, initial vertex i

COLOR(1...n] is the array of n different colors

//Output: Colored graph with minimum color

f CHECK\_VERTEX(i)==1 then

if i == N then

print COLOR[1...n]

else

j <- 1

while (S<=M) do

COLOR(i+1) <- j

j +j <- 1

end

end

end

Function CHECK\_VERTEX(i)

for j <- 1to i -1 do

if Adjacent(i,j) then

if COLOR(i)==COLOR(j) then

return 0

end

end

end

return 1

**Applications of Graph Coloring:**

The graph coloring problem has huge number of applications.

***1) Making Schedule or Time Table:***Suppose we want to make am exam schedule for a university. We have list different subjects and students enrolled in every subject. Many subjects would have common students (of same batch, some backlog students, etc). *How do we schedule the exam so that no two exams with a common student are scheduled at same time? How many minimum time slots are needed to schedule all exams?* This problem can be represented as a graph where every vertex is a subject and an edge between two vertices mean there is a common student. So this is a graph coloring problem where minimum number of time slots is equal to the chromatic number of the graph.

***2)***[***Mobile Radio Frequency Assignment***](http://www.zib.de/groetschel/teaching/SS2012/GraphCol%20and%20FrequAssignment.pdf)***:*** When frequencies are assigned to towers, frequencies assigned to all towers at the same location must be different. How to assign frequencies with this constraint? What is the minimum number of frequencies needed? This problem is also an instance of graph coloring problem where every tower represents a vertex and an edge between two towers represents that they are in range of each other.

***3) Sudoku:***Sudoku is also a variation of Graph coloring problem where every cell represents a vertex. There is an edge between two vertices if they are in same row or same column or same block.

***4)***[***Register Allocation***](http://en.wikipedia.org/wiki/Register_allocation)***:***In compiler optimization, register allocation is the process of assigning a large number of target program variables onto a small number of CPU registers. This problem is also a graph coloring problem.

***5) Bipartite Graphs:***We can check if a graph is Bipartite or not by coloring the graph using two colors.

1. Maximum cut problem ….

Given a universe U of n elements, a collection of subsets of U say S = {S1, S2…,Sm} where every subset Si has an associated cost. Find a minimum cost subcollection of S that covers all elements of U.  Example:

U = {1,2,3,4,5}

S = {S1,S2,S3}

S1 = {4,1,3}, Cost(S1) = 5

S2 = {2,5}, Cost(S2) = 10

S3 = {1,4,3,2}, Cost(S3) = 3

Output: Minimum cost of set cover is 13 and

set cover is {S2, S3}

There are two possible set covers {S1, S2} with cost 15

and {S2, S3} with cost 13.

**Why is it useful?** It was one of Karp’s NP-complete problems, shown to be so in 1972. Other applications: edge covering, vertex cover Interesting example: IBM finds computer viruses (wikipedia) Elements- 5000 known viruses Sets- 9000 substrings of 20 or more consecutive bytes from viruses, not found in ‘good’ code. A set cover of 180 was found. It suffices to search for these 180 substrings to verify the existence of known computer viruses. Another example: Consider General Motors needs to buy a certain amount of varied supplies and there are suppliers that offer various deals for different combinations of materials (Supplier A: 2 tons of steel + 500 tiles for $x; Supplier B: 1 ton of steel + 2000 tiles for $y; etc.). You could use set covering to find the best way to get all the materials while minimizing cost Source: <http://math.mit.edu/~goemans/18434S06/setcover-tamara.pdf> **Set Cover is NP-Hard:** There is no polynomial time solution available for this problem as the problem is a known NP-Hard problem. There is a polynomial time Greedy approximate algorithm, the greedy algorithm provides a Logn approximate algorithm. **2-ApproxApproximate Greedy Algorithm:** Let U be the universe of elements, {S1, S2, … Sm} be collection of subsets of U and Cost(S1), C(S2), … Cost(Sm) be costs of subsets.

6 quadratic assignment problem

# Quadratic Assignment Problem

The quadratic assignment problem is amongst some of the hard problems we have in computer science today and when I mean hard, I mean it would probably take the latest high-end computers we have today years to find a solution.

|  |  |
| --- | --- |
|  | |
| imp   |  | | --- | | population\_size : int | |  | number of data we want in our list | |  |  | |  | Returns | |  | ------- | |  | List | |  | return list of chromosome | |  | """ | |  |  | |  | population = [] | |  |  | |  | for i in range (population\_size): | |  |  | |  | # create list with size == problem size and random values ranging from 0 to problem\_size | |  | x = random.sample(range(problem\_size), problem\_size) | |  |  |   ort random | |
|  | |  | |
|  | | # function to generate new population | |
|  | | def Generate\_Initial\_Population(problem\_size, population\_size) -> list: | |
|  | | """Generate list of random chromosome | |
|  | |  | |
|  | | Parameters | |
|  | | ---------- | |
|  | | problem\_size : int | |
|  | | size of the problem i.e no of location/facilities | |
|  | |  | |
| # add list x to population. The item in the second index (0) is the fitness score we'll use later |
|  | population.append([x, 0]) | |
|  |  | |
|  | return population | |

Thankyou;