**Lab cycle 1**

**1.Merge two sorted linked lists**

struct ListNode\* mergeTwoLists(struct ListNode\* list1, struct ListNode\* list2){

struct ListNode\* list3;

struct ListNode\* prev=NULL;

struct ListNode\* list4=NULL;

if(list1==NULL){

return list2;

}

else if(list2==NULL){

return list1;

}

else

{

if (list1->val <= list2->val){

list3=list1;

list4=list3;

list1=list1->next;

}

else if(list2->val <list1 ->val){

list3=list2;

list4=list3;

list2=list2->next;

}

while (list1 && list2){

if(list1->val<=list2->val){

list3->next=list1;

list1=list1->next;

list3=list3->next;

}

else if(list2->val < list1->val){

list3->next=list2;

list2=list2->next;

list3=list3->next;

}

}

if(list1==NULL){

list3->next=list2;

}

if(list2==NULL){

list3->next=list1;

}

return list4;

}

}

**2. Merge sorted array**

void merge(int\* nums1, int nums1Size, int m, int\* nums2, int nums2Size, int n){

if(m==0){

int j=0,k=0;

while(j<n){

nums1[k++]=nums2[j++];

}

}

else if(n==0){

}

else{

int a[m];

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

a[i]=nums1[i];

}

int i=0,j=0,k=0;

while(i<m && j<n){

if(a[i]>nums2[j]){

nums1[k++]=nums2[j++];

}

else{

nums1[k++]=a[i++];

}

}

while(i<m){

nums1[k++]=a[i++];

}

while(j<n){

nums1[k++]=nums2[j++];

}

}

}

**3. Create an array of k linked-lists lists, each linked-list is sorted in ascending order.**

***Merge all the linked-lists into one sorted linked-list and return it.***

void merge(struct ListNode\*\* lists,int l,int r){

struct ListNode \*h1=lists[l],\*h2=lists[r],\*temp=NULL,\*last=NULL,\*newn=NULL;

// if(h1==NULL)return h2;

while(h1 && h2){

if(h1->val>h2->val){

newn=(struct ListNode \*)malloc(sizeof(struct ListNode));

newn->val=h2->val;

newn->next=NULL;

h2=h2->next;

}

else{

newn=(struct ListNode \*)malloc(sizeof(struct ListNode));

newn->val=h1->val;

newn->next=NULL;

h1=h1->next;

}

if(temp==NULL){

temp=newn;

}

else

last->next=newn;

last=newn;

}

while(h1){

newn=(struct ListNode \*)malloc(sizeof(struct ListNode));

newn->val=h1->val;

newn->next=NULL;

if(temp==NULL)temp=newn;

else last->next=newn;

last=newn;

h1=h1->next;

}

while(h2){

newn=(struct ListNode \*)malloc(sizeof(struct ListNode));

newn->val=h2->val;

newn->next=NULL;

if(temp==NULL)temp=newn;

else last->next=newn;

last=newn;

h2=h2->next;

}

lists[l]=temp;

}

void mergeSort(struct ListNode\*\* lists,int l,int r ){

int mid;

if(l<r){

mid=(l+r)/2;

mergeSort(lists,l,mid);

mergeSort(lists,mid+1,r);

merge(lists,l,mid+1);

}

}

struct ListNode\* mergeKLists(struct ListNode\*\* lists, int n){

if(n==0)return NULL;

if(n==1)return lists[0];

mergeSort(lists,0,n-1);

return lists[0];}

**4. Sort an Array**

void Adjust(int a[],int i,int n)

{

int j, item;

j = 2\*i;

item = a[i];

while (j <= n)

{

if ((j < n) && (a[j] < a[j+1]))

j = j + 1;

if (item >= a[j])

break;

a[j/2] = a[j];

j = 2\*j;

}

a[j/2] = item;

}

void Heapify(int a[], int n)

{

int i;

for(i = n/2; i >= 1; i--)

Adjust(a,i,n);

}

void Heapsort(int a[], int n)

{

int i, t;

Heapify(a,n);

for( i = n; i > 1; i--)

{

t = a[i];

a[i] = a[1];

a[1] = t;

Adjust(a, 1, i-1);

}

}

int\* sortArray(int\* nums, int numsSize, int\* returnSize){

int a[numsSize+1];

for(int i=0,j=1;i<numsSize;i++,j++){

a[j]=nums[i];

}

\*returnSize=numsSize;

Heapsort(a,numsSize);

for(int i=1,j=0;i<numsSize+1;i++,j++){

nums[j]=a[i];

}

return nums;

}

**Lab Cycle 2**

**1. You are assigned to put some amount of boxes onto one truck. You are given a 2D array boxTypes, where boxTypes[i] = [numberOfBoxesi, numberOfUnitsPerBoxi]:**

**numberOfBoxesi is the number of boxes of type i.**

**numberOfUnitsPerBoxiis the number of units in each box of the type i.**

**You are also given an integer truckSize, which is the maximum number of boxes that can be put on the truck. You can choose any boxes to put on the truck as long as the number of boxes does not exceed truckSize.**

**Return *the maximum total number of units that can be put on the truck.***

int maximumUnits(int\*\* a, int boxTypesSize, int\* boxTypesColSize, int truckSize){

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

for(int j=0;j<boxTypesSize-i-1;j++){

if(a[j][1]<a[j+1][1]){

int c[2];

c[0]=a[j][0];

c[1]=a[j][1];

a[j][0]=a[j+1][0];

a[j][1]=a[j+1][1];

a[j+1][0]=c[0];

a[j+1][1]=c[1];

}

}

}

int m=truckSize,sum=0;

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

if(m>=a[i][0]){

sum=sum+a[i][0]\*a[i][1];

m=m-a[i][0];

}

else{

sum=sum+a[i][1]\*m;

m=0;

}

}

return sum;

}

**2. Maximum Number of Events That Can Be Attended**

class Solution {

public:

int maxEvents(vector<vector<int>>& A) {

priority\_queue <int, vector<int>, greater<int>> pq;

sort(A.begin(), A.end());

int i = 0, res = 0, n = A.size();

for (int d = 1; d <= 100000; ++d) {

while (pq.size() && pq.top() < d)

pq.pop();

while (i < n && A[i][0] == d)

pq.push(A[i++][1]);

if (pq.size()) {

pq.pop();

++res;

}

}

return res;

}

};

**3. Minimum cost to connect all points**

# include<math.h>

#include<stdio.h>

# include<stdlib.h>

#include <limits.h>

#include <stdbool.h>

int mini(int key[],bool mstSet[],int n){

int m=INT\_MAX,min\_index;

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

if(key[i]<m && mstSet[i]==false){

m=key[i],min\_index=i;

}

}

return min\_index;

}

int minCostConnectPoints(int\*\* points, int pointsSize, int\* pointsColSize){

pointsColSize=pointsSize;

int costs[pointsSize][pointsSize],parent[pointsSize],key[pointsSize];

bool mstSet[pointsSize];

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

for(int j=0;j<pointsSize;j++){

costs[i][j]=abs(points[i][0]-points[j][0])+abs(points[i][1]-points[j][1]);

}

}

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

key[i]=INT\_MAX;

mstSet[i]=false;

parent[i]=-1;

}

int count=0;

parent[0]=-1;

key[0]=0;

while(count<pointsSize-1){

int u=mini(key,mstSet,pointsSize);

mstSet[u]=true;

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

if(costs[u][v] && mstSet[v]==false && key[v]>costs[u][v]){

key[v]=costs[u][v];

parent[v]=u;

}

}

count++;

}

int mincost=0;

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

mincost+=key[i];

}

return mincost;

}

**4. Network Delay Time**

# include<limits.h>

#define INT\_MAX 1000

int networkDelayTime(int\*\* times, int timesSize, int\* timesColSize, int n, int k){

int costs[n+1][n+1];

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

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

if(i==j){

costs[i][j]=0;

}

else{

costs[i][j]=INT\_MAX;

}

}

}

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

costs[times[i][0]][times[i][1]]=times[i][2];

}

int dist[n];

bool flag[n];

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

dist[i]=costs[k][i+1];

flag[i]=false;

}

flag[k-1]=true;

int count=1;

while(count<n){

// int u=min(dist,flag,n);

int m=1000,i,min\_index=-1;

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

if(m>dist[i] && flag[i]==false){

m=dist[i];

min\_index=i;

}

}

int u=min\_index;

if(u==-1){

return -1;

}

flag[u]=true;

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

if(flag[i]==false && dist[i]>dist[u]+costs[u+1][i+1])

dist[i]=dist[u]+costs[u+1][i+1];

}

count++;

}

int max=0;

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

if(max<dist[i] && dist[i]!=1000 && flag[i]==true)

max=dist[i];}

if(max!=1000)

return max;

else

return -1;}

**Lab cycle 3**

**1.Maximum profit in job scheduling**

class Solution {

public:

    vector<int> dp;

    int f(int ind, vector<pair<int, pair<int, int>>>& v) {

        if (dp[ind] != -1) return dp[ind];

        int pro = v[ind].second.second;

        int st = v[ind].second.first;

        int ft = v[ind].first;

        if (ind == 0) return dp[ind] = pro;

        int t = pro;

        for (int i = ind - 1; i >= 0; i--) {

            if (v[i].first <= st) {

                t = f(i, v) + pro;

                break;

            }

        }

        int nt = f(ind - 1, v);

        return dp[ind] = max(t, nt);

    }

    int jobScheduling(vector<int>& startTime, vector<int>& endTime, vector<int>& profit) {

        vector<pair<int, pair<int, int>>> v;

        int n = startTime.size();

        dp.assign(n, -1);

        for (int i = 0; i < n; i++) {

            v.push\_back({ endTime[i], {startTime[i], profit[i]} });

        }

        sort(v.begin(), v.end());

        return f(n - 1, v);

    }

};

**2. Create an integer array coin representing coins of different denominations and an integer amount representing a total amount of money.**

**Return *the fewest number of coins that you need to make up that amount*. If that amount of money cannot be made up by any combination of the coins, return -1.**

**You may assume that you have an infinite number of each kind of coin.**

int eval(int m,int n){

if(m==-1 && n==-1){

return -1;

}

else if(m==-1){

return n;

}

else if(n==-1){

return m+1;

}

else{

if(m+1>n){

return n;

}

else return m+1;

}

}

int coinChange(int\* a, int n, int s){

s=s+1;

int b[n][s];

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

for(int j=0;j<s;j++){

if(i==0){

if(j%a[i]==0){

b[i][j]=j/a[i];

}

else{

b[i][j]=-1;

}

}

else{

if(j<a[i]){

b[i][j]=b[i-1][j];

}

else{

int m=b[i][j-a[i]];

int n=b[i-1][j];

b[i][j]=eval(m,n);

}

}

}

}

return b[n-1][s-1];

}

**3. Given an input string s, reverse the order of the words.**

**A word is defined as a sequence of non-space characters. The words in s will be separated by at least one space.**

**Return *a string of the words in reverse order concatenated by a single space.***

**Note that s may contain leading or trailing spaces or multiple spaces between two words. The returned string should only have a single space separating the words. Do not include any extra spaces.**

#include<stdio.h>

#include<string.h>

#include<malloc.h>

char \* reverseWords(char \* s)

{

char \*stack=malloc(sizeof(char)\*strlen(s)),ch,\*s2=malloc(sizeof(char)\*(strlen(s)+1));

int top=-1,k=0,m=0;

int n=strlen(s);

for(int i=n-1;i>=0;i--)

{

ch=s[i];

if(ch==' ' && top!=-1)

{

m=m+(top+1)+1;

while(top>=0)

{

s2[k]=stack[top];

k++;

top--;

}

s2[k++]=' ';

}

else if(ch!=' ')

{

stack[++top]=ch;

}

}

if(top!=-1)

{

m=m+top+1;

while(top>=0)

{

s2[k]=stack[top];

k++;

top--;

}

}

if(s[0]==' ')

{

s2[m-1]='\0';

}

else

{

s2[m]='\0';

}

free(stack);

return s2;

}

**4. shortest path visiting all nodes**

class Solution {

public:

    using int3 = tuple<int, int, int>;

    using int2 = pair<int, int>;

    int shortestPathLength(vector<vector<int>>& graph) {

        int n = graph.size();

        set<int2> path;

        int&& allMask = (1<<n) - 1;

        if (n == 1) return 0;

        queue<int3> q;

        for (int i = 0; i < n; i++) {

            int mask = 1<<i;

            q.emplace(0, i, mask);

            path.insert({i, mask});

        }

        while (!q.empty()) {

            auto [d, i, mask] = q.front();

            q.pop();

            if (mask==allMask) return d;

            for (int j : graph[i]) {

                int jMask= mask|(1<<j);

                int2 edge={j, jMask};

                if (path.count(edge)==0) {

                    path.insert(edge);

                    q.emplace(d+1, j, jMask);

                }

            }

        }

        return -1;

    }

};

**5. There is a robot on an m x n grid. The robot is initially located at the top-left corner (i.e., grid[0][0]). The robot tries to move to the bottom-right corner (i.e., grid[m - 1][n - 1]). The robot can only move either down or right at any point in time.**

**Given the two integers m and n, return *the number of possible unique paths that the robot can take to reach the bottom-right corner*.**

**The test cases are generated so that the answer will be less than or equal to 2 \* 109.**

int uniquePaths(int m, int n){

if(m==1 && n==1)return 1;

int a[m][n];

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

{

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

{

if(j==n-1 && i==m-1) a[i][j]=0;

else if(j==n-1 || i==m-1) a[i][j]=1;

else a[i][j]=0;

}

}

for(int i=m-2;i>=0;i--){

for(int j=n-2;j>=0;j--){

a[i][j]=a[i][j+1]+a[i+1][j];

}

}

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

for(int j=0;j<n;j++){

printf("%d ",a[i][j]);

}

printf("\n");

}

return a[0][0];

}

**Lab cycle 4**

**1. The n-queens puzzle is the problem of placing n queens on an n x n chessboard such that no two queens attack each other.**

**Given an integer n, return *all distinct solutions to the n-queens puzzle*. You may return the answer in any order.**

**Each solution contains a distinct board configuration of the n-queens' placement, where 'Q' and '.' both indicate a queen and an empty space, respectively.**

class Solution {

public:

bool canPlace(int k,int i,vector<int>&p){

for(int j=0;j<k;j++){

if(p[j]==i || (abs(k-j)==abs(i-p[j]))){

return false;

}//kth row ith column ,jth row p[j]

}

return true;

}

void nQueens(int k,int n,vector<vector<string>>&v,vector<int>&p){

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

if(canPlace(k,i,p)){

p[k]=i;

if(k==n-1){

vector<string>a;

for(int j=0;j<n;j++){

string s="";

for(int q=0;q<n;q++){

if(p[j]==q){

s+='Q';

}

else s+='.';

}

a.push\_back(s);

}

v.push\_back(a);

}

else{

nQueens(k+1,n,v,p);

}

}

}

}

vector<vector<string>> solveNQueens(int n) {

vector<vector<string>>v;

vector<int>p(n,0);

nQueens(0,n,v,p);

return v;

}

};

**2. Longest substring without repeating characters**

#include<stdio.h>

#include<string.h>

#include<malloc.h>

int isin(char ch,char \* a,int k)

{

int n=strlen(a);

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

{

if(ch==a[i])

return 1;

}

return 0;

}

int lengthOfLongestSubstring(char \* s){

int n=strlen(s),max=0;

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

char \*a=malloc(sizeof(char)\*n);

int k=0;

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

{

if(!isin(s[j],a,k)){

a[k]=s[j];

k++;

if(j==n-1){

//a[k]='\0';

//printf("%s ",a);

if(k>max)

max=k;

break;

}

}

else{

free(a);

if(max<k)max=k;

break;

}

}

}

return max;

}

**3. Longest common prefix**

#include<malloc.h>

#include<string.h>

#include<stdbool.h>

char \* longestCommonPrefix(char \*\* s, int strsSize){

bool flag=true;

// printf("%dvnm",strsSize);

int n=strsSize,j=0;

// bool flag=true;

while(flag && n!=1){

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

if(s[i][j]=='\0' || s[i+1][j]=='\0')

{

s[0][j]='\0';

flag=false;

break;

}

else if(s[i][j]!=s[i+1][j])

{

// flag=true;

flag=false;

s[0][j]='\0';

break;

}

// else {

// flag=false;

// s[0][j]='\0';

// break;

// }

}

j++;

}

return s[0];

}