**CS 351 – DAA Lab Problems**

**Lab Cycle - I**

# Divide and Conquer

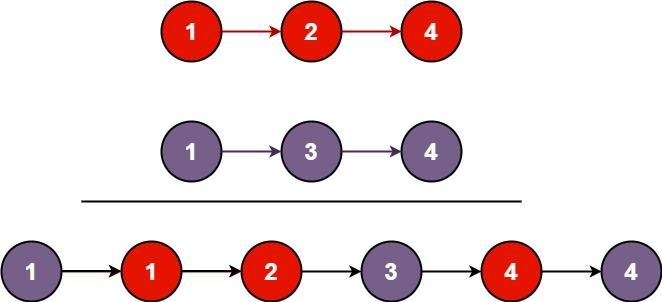
## 1. Merge Two Sorted Lists

You are given the heads of two sorted linked lists list1 and list2.

Merge the two lists in a one **sorted** list. The list should be made by splicing together the nodes of the first two lists.

Return *the head of the merged linked list*.

**Example 1:**



**Input:** list1 = [1,2,4], list2 = [1,3,4]

**Output:** [1,1,2,3,4,4]

**Example 2:**

**Input:**

list1 = [], list2 = []

**Output:**

[]

**Example 3:**

**Input:**

list1 = [], list2 = [0]

**Output:**

[0]

**Constraints:**

•

The number of nodes in both lists is in the range

]

50

[0

,

.

•

-

100 <=

Node.val

<= 100

•

Both

list1

and

list2

are sorted in

**non**

**-**

**decreasing**

order.

Link: <https://leetcode.com/problems/merge-two-sorted-lists/>

#include<stdio.h>

#include<stdlib.h>

struct ListNode

{

int val;

struct ListNode\* next;

}\*newnode,\*temp;

struct ListNode\* head=NULL;

struct ListNode\* tail=NULL;

struct ListNode\* Creation(struct ListNode\* head)

{

int ch,value;

do{

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

printf("enter the value: ");

scanf("%d",&value);

newnode->val=value;

newnode->next=NULL;

if(head==NULL)

{

head=newnode;

tail=newnode;

}

else

{

tail->next=newnode;

tail=newnode;

}

printf("enter 1 to continue, any number to exit\n");

fflush(stdin);

scanf("%d",&ch);

}while(ch==1);

return head;

}

void Display(struct ListNode\* head)

{

temp=head;

while(temp!=NULL)

{

printf("%d\n",temp->val);

temp=temp->next;

}

}

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

struct ListNode\* l1=list1;

struct ListNode\* l2=list2;

struct ListNode\* f;

struct ListNode\* n=NULL;

if(l1==NULL){

return l2;

}

else if(l2==NULL){

return l1;

}

if(l1->val<=l2->val){

n=l1;

l1=l1->next;

}else{

n=l2;

l2=l2->next;

}

f=n;

while(l1!=NULL && l2!=NULL){

struct ListNode\* t=NULL;

if(l1->val<l2->val){

t=l1;

l1=l1->next;

}else{

t=l2;

l2=l2->next;

}

f->next=t;

f=t;

}

if(l1!=NULL){

f->next=l1;

}else if(l2!=NULL){

f->next=l2;

}

return n;

}

int main()

{struct ListNode\* h1=NULL;

struct ListNode\* h2=NULL;

printf("enter linked list 1\n");

struct ListNode\* l1= Creation(h1);

printf("linked list 1\n");

Display(l1);

printf("enter linked list 2\n");

struct ListNode\* l2=Creation(h2);

printf("linked list 2\n");

Display(l2);

struct ListNode\* r= mergeTwoLists(l1,l2);

printf("linked list after merge sort\n");

Display(r);

}

## 2.Merge Sorted Array

You are given two integer arrays nums1 and nums2, sorted in **non-decreasing order**, and two integers m and n, representing the number of elements in nums1 and nums2 respectively.

**Merge** nums1 and nums2 into a single array sorted in **non-decreasing order**.

The final sorted array should not be returned by the function, but instead be *stored inside the array* nums1. To accommodate this, nums1 has a length of m + n, where the first m elements denote the elements that should be merged, and the last n elements are set to 0 and should be ignored. nums2 has a length of n.

**Example 1:**

**Input:** nums1 = [1,2,3,0,0,0], m = 3, nums2 = [2,5,6], n = 3

**Output:** [1,2,2,3,5,6]

**Explanation:** The arrays we are merging are [1,2,3] and [2,5,6].

The result of the merge is [1,2,2,3,5,6] with the underlined elements coming from nums1.

**Example 2:**

**Input:**

nums1 = [1], m = 1, nums2 = [], n = 0

**Output:**

[1]

**Explanation:**

The arrays we are merging are [1] and [].

The result of the merge is [1].

**Example 3:**

**Input:**

nums1 = [0], m = 0, nums2 = [1], n = 1

**Output:**

[1]

**Explanation:**

The arrays we are merging are [] and [1].

The result of the merge is [1].

Note that because m = 0, there are no elements in nums1. The 0 is only there to ensure the

merge result can fit in nums1.

**Constraints:**

•

nums1.length == m + n

•

nums2.length == n

•

0

<=

m, n <= 200

•

1

<= m + n

<= 200

•

-

10

9

<=

nums1[i], nums2[j] <=

10

9

Link: <https://leetcode.com/problems/merge-sorted-array/>

#include<stdio.h>

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

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

int a[m+n];

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

if(\*(nums1+i)<\*(nums2+j)){

a[k++]=\*(nums1+(i++));

}

else{

a[k++]=\*(nums2+(j++));

}

}

for(;i<m;i++){

a[k++]=\*(nums1+i);

}

for(;j<n;j++){

a[k++]=\*(nums2+j);

}

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

\*(nums1+i)=a[i];

}

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

printf("%d ",\*(nums1+i));

}

}

int main(){

int i,j,k,n,m,t=0,s=0,a[10],b[10];

printf("enter m");

scanf("%d",&m);

printf("enter n");

scanf("%d",&n);

int c[m+n];

printf("enter elements of 1 array\n");

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

scanf("%d",&a[i]);

}

printf("enter elements of 2 array\n");

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

scanf("%d",&b[j]);

}

printf(" array 1 is\n");

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

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

}

printf("\narray 2 is\n");

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

printf("%d ",b[j]);

}

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

if(a[i]!=0){

t=t+1;

}

}

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

if(b[i]!=0){

s=s+1;

}

}

printf("\nsorted array");

merge(a,m,t,b,n,s);

}

## 3..Merge k Sorted Lists

You are given 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.*

**Example 1:**

**Input:**

lists = [[1,4,5],[1,3,4],[2,6]]

**Output:**

[1

,1,2,3,4,4,5,

6]

**Explanation:**

The linked

-

lists are:

[

1

-

>4

-

,

>5

1

-

>3

-

,

>4

2

-

>6

]

merging them into one sorted list:

1

-

>1

-

>2

-

>3

-

>4

-

>4

-

>5

-

>6

**Example 2:**

**Input:**

lists = []

**Output:**

[]

**Example 3:**

**Input:**

lists = [[]]

**Output:** []

**Constraints:**

* k == lists.length
* 0 <= k <= 104
* 0 <= lists[i].length <= 500
* -104 <= lists[i][j] <= 104
* lists[i] is sorted in **ascending order**.
* The sum of lists[i].length will not exceed 104.

Link: <https://leetcode.com/problems/merge-k-sorted-lists/>

#include<stdio.h>

#include<stdlib.h>

struct ListNode

{

int val;

struct ListNode\* next;

}\*newnode,\*temp;

struct ListNode\* head=NULL;

struct ListNode\* tail=NULL;

struct ListNode\* Creation(struct ListNode\* head)

{

int ch,value;

do{

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

printf("enter the value: ");

scanf("%d",&value);

newnode->val=value;

newnode->next=NULL;

if(head==NULL)

{

head=newnode;

tail=newnode;

}

else

{

tail->next=newnode;

tail=newnode;

}

printf("enter 1 to continue, any number to exit\n");

fflush(stdin);

scanf("%d",&ch);

}while(ch==1);

return head;

}

void Display(struct ListNode\* head)

{

temp=head;

while(temp!=NULL)

{

printf("%d ",temp->val);

temp=temp->next;

}

}

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

struct ListNode\* l1=list1;

struct ListNode\* l2=list2;

struct ListNode\* f;

struct ListNode\* n=NULL;

if(l1==NULL){

return l2;

}

else if(l2==NULL){

return l1;

}

if(l1->val<=l2->val){

n=l1;

l1=l1->next;

}else{

n=l2;

l2=l2->next;

}

f=n;

while(l1!=NULL && l2!=NULL){

struct ListNode\* t=NULL;

if(l1->val<l2->val){

t=l1;

l1=l1->next;

}else{

t=l2;

l2=l2->next;

}

f->next=t;

f=t;

}

if(l1!=NULL){

f->next=l1;

}else if(l2!=NULL){

f->next=l2;

}

return n;

}

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

struct ListNode\* temp=NULL;

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

temp=mergeTwoLists(temp,lists[i]);

}

return temp;

}

int main()

{int n;

printf("enter n value");

scanf("%d",&n);

struct ListNode\* l[3]={NULL};

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

struct ListNode\* h1=NULL;

printf("\nenter %dlist\n",i+1);

l[i]=Creation(h1);

printf("first list\n");

Display(l[i]);

}

struct ListNode\* p=mergeKLists(l,n);

printf("\nafter sorting ");

Display(p);}

## 4..Sort an Array

Given an array of integers nums, sort the array in ascending order and return it.

You must solve the problem **without using any built-in** functions in O(nlog(n)) time complexity and with the smallest space complexity possible.

**Example 1:**

**Input:**

nums = [5,2,3,1]

**Output:**

[1

,2,3,

5]

**Explanation:**

After sorting the array, the positions of some numb

ers are not changed (for

example, 2 and 3), while the positions of other numbers are changed (for example, 1 and 5).

**Example 2:**

**Input:**

nums = [5,1,1,2,0,0]

**Output:**

[0

,0,1,1,2,

5]

**Explanation:**

Note that the values of nums are not necessairly unique.

**Constraints:**

|  |  |
| --- | --- |
| 1 <= nums.length <= 5 \* 104 |  |
| -5 \* 104 <= nums[i] <= 5 \* 104 | |

•

•

Link: <https://leetcode.com/problems/sort-an-array/>

#include <stdio.h>

void merge(int arr[],int p,int q,int r){

int m=q-p+1;

int n=r-q;

int A[m],B[n];

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

A[i]=arr[p+i];

}

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

B[i]=arr[q+1+i];

}

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

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

if(A[i]<B[j]){

arr[k++]=A[i++];

}

else{

arr[k++]=B[j++];

}

}

for(;i<m;i++){

arr[k++]=A[i];

}

for(;j<n;j++){

arr[k++]=B[j];

}

}

void mergesort(int arr[],int low,int high){

if(low<high){

int mid=(high+low)/2;

mergesort(arr,low,mid);

mergesort(arr,mid+1,high);

merge(arr,low,mid,high);

}

}

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

mergesort(nums,0,numsSize-1);

\*returnSize=numsSize;

return nums;

}

int main()

{int n;

printf("enter n");

scanf("%d",&n);

int b[30];

int a[30]={0};

printf("enter elements");

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

scanf("%d",&b[i]);

}

int\* c=sortArray(b,5,a);

printf("after sorting\n");

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

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

}

}

# Lab Cycle - II

## Greedy Programming

### 1. Maximum Units on a Truck

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 | | . | |
| • | numberOfUnitsPerBoxi | | is 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.*

**Example 1:**

**Input:**

boxTypes = [[1,3],[2,2],[3,1]], truckSize = 4

**Output:**

8

**Explanation:**

There are:

-

1

box of the first type that contains 3 units.

-

2

boxes of the second type that contain 2 units each.

-

3

boxes of the third type that contain 1 unit each.

You can take all the boxes of the first and second types, and one box

of the third type.

The total number of units will be = (1 \* 3) + (2 \* 2) + (1 \* 1) = 8.

**Example 2:**

**Input:**

boxTypes = [[5,10],[2,5],[4,7],[3,9]], truckSize = 10

**Output:**

91

**Constraints:**

* 1 <= boxTypes.length <= 1000
* 1 <= numberOfBoxesi, numberOfUnitsPerBoxi <= 1000
* 1 <= truckSize <= 106

Link: <https://leetcode.com/problems/maximum-units-on-a-truck/>

#include <stdio.h>

#include<stdlib.h>

void maximumUnits(int\*\* boxTypes, int boxTypesSize,int boxTypesColSize, int truckSize){

int i,j,a[boxTypesSize],b[boxTypesSize];

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

a[i]=boxTypes[i][0];

b[i]=boxTypes[i][1];

}

int s=0,t,p;

while(truckSize!=0){

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

int max=0;

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

if(max<b[i]){

max=b[i];

p=i;

}

}

if(a[p]<=truckSize){

s=s+a[p]\*max;

b[p]=0;

truckSize=truckSize-a[p];

}

else{

s=s+b[p]\*truckSize;

truckSize=0;

}

i=boxTypesSize;

}

}

printf("%d",s);

}

int main(){

int r,c,t,i,j

printf("enter r c t");

scanf("%d%d%d",&r,&c,&t);

int\*\* arr = (int\*\*)malloc(r\*sizeof(int));

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

{ arr[i] = (int\*)malloc(c\*sizeof(int));

}

printf("enter elements");

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

{ for (j = 0; j < c; j++)

{ scanf("%d",&arr[i][j]);

}

}

printf("maximum total number of units: ");

maximumUnits(arr,r,c,t);

}

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

You are given an array of events where events[i] = [startDayi, endDayi]. Every event i starts at startDayi and ends at endDayi.

You can attend an event i at any day d where startTimei <= d <= endTimei. You can only attend one event at any time d.

Return *the maximum number of events you can attend*.

**Example 1:**

**Input:**

events = [[1,2],[2,3],[3,4]]

**Output:**

3

**Explanation:**

You can attend all the three events.

One way to attend them all is as shown.

Attend the first event on day 1.

Attend the second event on day 2.

Attend the third event on day 3.

**Example 2:**

**Input:**

events= [[1,2],[2,3],[3,4],[1,2]]

**Output:**

4

**Constraints:**

•

<= events.length <=

1

10

5

•

events[i].length == 2

•

1

<= startDay

i

<=

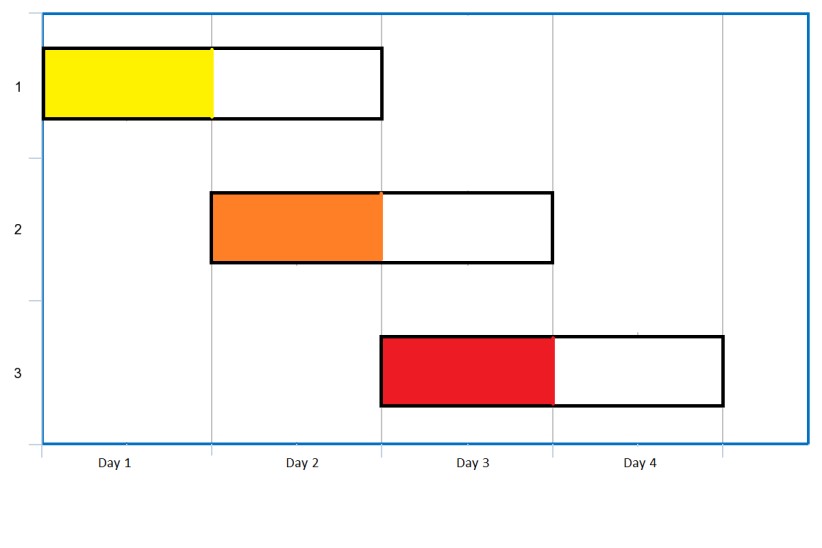
endDay

i

<=

10

5



Link: <https://leetcode.com/problems/maximum-number-of-events-that-can-be-attended/>

void merge(int \*\*a,int l,int m,int r)

{

int x=m-l+1,y=r-m,i,j,k;

int \*\*c=(int\*\*)malloc(x\*sizeof(int\*));

int \*\*d=(int\*\*)malloc(y\*sizeof(int\*));

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

{

c[i]=(int\*)malloc(2\*sizeof(int));

c[i]=a[l+i];

}

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

{

d[i]=(int\*)malloc(2\*sizeof(int));

d[i]=a[m+1+i];

}

i=0,j=0,k=l;

while(i<x && j<y)

{

if(c[i][0]<=d[j][0])

a[k]=c[i],i++;

else

a[k]=d[j], j++;

k++;

}

for(;i<x;i++,k++)

a[k]=c[i];

for(;j<y;j++,k++)

a[k]=d[j];

}

void mergeSort(int \*\*a,int l,int r)

{

if(l<r)

{

int m=(l+r)/2;

mergeSort(a,l,m);

mergeSort(a,m+1,r);

merge(a,l,m,r);

}

}

void insert(int pq[],int n)

{

int i,ele=pq[n];

for(i=n;i>1 && pq[i/2]>ele;i=i/2)

pq[i]=pq[i/2];

pq[i]=ele;

}

void delete(int pq[],int n)

{

pq[1]=pq[n];

int j=2,ele=pq[1];

n--;

while(j<=n)

{

if(j<n && pq[j]>pq[j+1])

j++;

if(ele<=pq[j])

break;

pq[j/2]=pq[j];

j=2\*j;

}

pq[j/2]=ele;

}

int maxEvents(int\*\* events, int eventsSize, int eventsColSize){

int \*\*e=events;

int i,\*t,s=eventsSize;

mergeSort(e,0,s-1);

int pq[100001],n=0,c=0,d;

i=0;

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

{

for(;i<s && e[i][0]==d;i++)

{

n++;

pq[n]=e[i][1];

insert(pq,n);

}

for(;n!=0 && pq[1]<d;n--)

delete(pq,n);

if(n!=0)

{

delete(pq,n);

c++;

n--;

}

}

return c;

}

#include <stdio.h>

int main()

{

int r,c=2,i,j;

printf("enter r ");

scanf("%d",&r);

int\*\* arr = (int\*\*)malloc(r\*sizeof(int));

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

{ arr[i] = (int\*)malloc(c\*sizeof(int));

}

printf("enter elements");

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

{ for (j = 0; j < c; j++)

{ scanf("%d",&arr[i][j]);

}

}

int t=maxEvents(arr,r,c);

printf("max events can attend :%d",t);

}

### **3**. Min Cost to Connect All Points

You are given an array points representing integer coordinates of some points on a 2D-plane, where points[i] = [xi, yi].

The cost of connecting two points [xi, yi] and [xj, yj] is the **manhattan distance** between them: |xi - xj| + |yi - yj|, where |val| denotes the absolute value of val.

Return *the minimum cost to make all points connected.* All points are connected if there is **exactly one** simple path between any two points.

**Example 1:**

**Input:**

points = [[0,0],[2,2],[3,10],[5,2],[7,0]]

**Output:**

20

**Explanation:**

We can connect the points as

shown above to get the minimum cost of 20.

Notice that there is a unique path between every pair of points.

**Example 2:**

**Input:**

points = [[3,12],[

-

2

,5],

[

-

4,1]]

**Output:**

18

**Constraints:**

•

<= points.length

<= 1000

1

•

-

10

6

<=

x

i

, y

i

<=

10

6

•

All pairs

(

x

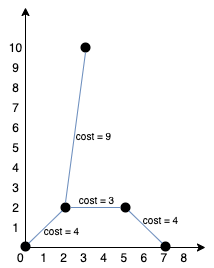
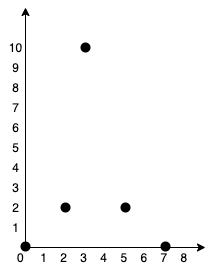
i

, y

i

)

are distinct.



Link: <https://leetcode.com/problems/min-cost-to-connect-all-points/>

#include<stdio.h>

#include<stdlib.h>

int min(int g[],int n,int a[]){

int m=100000000,t;

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

if(m>g[i] && a[i]!=1){

m=g[i];

t=i;

}

}

return t;

}

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

int g[pointsSize],a[pointsSize],r,s=0;

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

a[i]=0;

}

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

g[i]=100000000;

}

g[0]=0;

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

r=min(g,pointsSize,a);

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

int t=abs(abs(points[r][0]-points[j][0])+abs(points[r][1]-points[j][1]));

if(a[j]!=1 && t<g[j] && j!=r){

g[j]=t;

}

}

a[r]=1;

}

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

s=s+g[i];

}

printf("%d",s);

}

int main(){

int r,c=2,i,j;

printf("enter r");

scanf("%d",&r);

int\*\* arr = (int\*\*)malloc(r\*sizeof(int\*));

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

{ arr[i] = (int\*)malloc(c\*sizeof(int));

}

printf("enter elements");

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

{ for (j = 0; j < c; j++)

{ scanf("%d",&arr[i][j]);

}

}

printf("min cost for connecting all pairs: ");

minCostConnectPoints(arr,r,c);

}

### 4. Network Delay Time

You are given a network of n nodes, labeled from 1 to n. You are also given times, a list of travel times as directed edges times[i] = (ui, vi, wi), where ui is the source node, vi is the target node, and wi is the time it takes for a signal to travel from source to target.

|  |  |
| --- | --- |
| We will send a signal from a given node k | . Return *the* ***minimum*** *time it takes for all* |
| |  | | --- | | -1 |   *the* n *nodes to receive the signal*. If it is impossible for all the n nodes to receive the signal, return . | |

**Example 1:**

**Input:**

times = [[2,1,1],[2,3,1],[3,4,1]], n = 4, k = 2

**Output:**

2

**Example 2:**

**Input:**

times = [[1,2,1]], n = 2, k = 1

**Output:**

1

**Example 3:**

**Input:**

times = [[1,2,1]], n = 2, k = 2

**Output:**

-

1

**Constraints:**

•

<= k <= n <=

1

100

•

= 6000

<

<= times.length

1

•

times[i].length == 3

•

1

<= u

i

, v

i

<=

n

•

u

i

!=

v

i

•

0

<= w

i

<= 100

•

All the pairs

(

u

i

, v

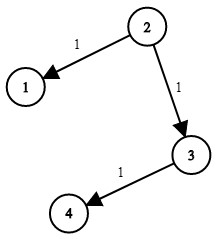
i

)

are

**unique**

. (i.e., no multiple edges.)



Link: <https://leetcode.com/problems/network-delay-time/>

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

int i,j,u,v,w,r=0;

int d[n];

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

{

d[i]=10000;

}

d[k-1]=0;

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

{

for(j=0;j<timesSize;j++)

{

u=times[j][0]-1;

v=times[j][1]-1;

w=times[j][2];

d[v]=d[v]<d[u]+w?d[v]:d[u]+w;

}

}

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

{

if(d[i]==10000)

{

return -1;

}

r=r<d[i]?d[i]:r;

}

return r;

}

#include <stdio.h>

int main()

{ int r,c,i,j,n,k;

printf("enter r c n k");

scanf("%d%d%d%d",&r,&c,&n,&k);

int\*\* arr = (int\*\*)malloc(r\*sizeof(int));

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

{ arr[i] = (int\*)malloc(c\*sizeof(int));

}

printf("enter elements");

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

{ for (j = 0; j < c; j++)

{ scanf("%d",&arr[i][j]);

}

}

int p=networkDelayTime(arr,r,c,n,k);

printf("%d",p);

}

# Lab Cycle - III

## Dynamic Programming & Graphs

**1. Coin Change**

You are given 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.

**Example 1:**

**Input:**

coins = [1,2,5], amount = 11

**Output:**

3

**Explanation:**

11

= 5 + 5 +

1

**Example 2:**

**Input:**

coins = [2], amount = 3

**Output:**

-

1

**Example 3:**

**Input:**

coins = [1], amount = 0

**Output:**

0

**Constraints:**

•

1

<= coins.length <=

12

•

1

<= coins[i] <=

2

31

-

1

•

0

<= amount <=

10

4

Link: <https://leetcode.com/problems/coin-change/>

#include <stdio.h>

int coinChange(int\* coins, int coinsSize, int amount){

int a[amount+1];

a[0]=0;

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

{

a[i]=amount+1;

}

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

{

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

{

if(coins[j]<=i)

{int x=1+a[i-coins[j]];

int y=a[i];

a[i]=x<y?x:y;

}

}

}

if(a[amount]==amount+1)

return -1;

else

return a[amount];

}

int main()

{int c[10],n,a;

printf("enter n and amount");

scanf("%d%d",&n,&a);

printf("enter coins");

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

scanf("%d",&c[i]);

}

int r=coinChange(c,n,a);

printf("min coins :%d",r);

}

### 2. Maximum Profit in Job Scheduling

We have n jobs, where every job is scheduled to be done from startTime[i] to endTime[i], obtaining a profit of profit[i].

You're given the startTime, endTime and profit arrays, return the maximum profit you can take such that there are no two jobs in the subset with overlapping time range.

If you choose a job that ends at time X you will be able to start another job that starts at time X.

**Example 1:**

**Input:**

startTime = [1,2,3,3], endTime = [3,4,5,6], profit = [50,10,40,70]

**Output:**

120

**Explanation:**

The subset chosen is the first and fourth job.

Time range [1

-

3]+[3

-

+ 70.

6]

, we get profit of 120 = 50

**Example 2:**

**Input:**

startTime

[1,2,3,4,6], endTime = [3,5,10,6,9], profit = [20,20,100,

70,60]

=

**Output:**

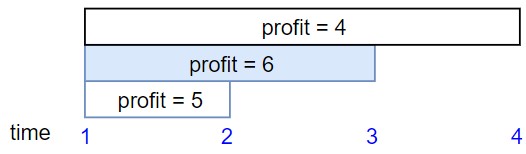
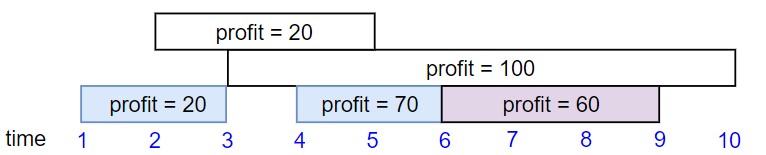
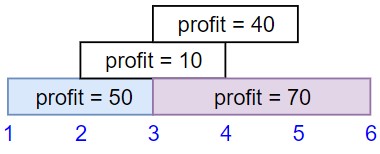
150

**Explanation:**

The subset chosen is the first, fourth and fifth job.

Profit obtained 150 = 20 + 70 + 60.

**Example 3:**



**Input:**

startTime = [1,1,1], endTime = [2,3,4], profit = [5,6

,4]

**Output:**

6

**Constraints:**

•

1

<= startTime.length == endTime.length == profit.length <= 5 \*

10

4

•

1

<= startTime[i] < endTime[i] <=

10

9

•

1

<= profit[i] <=

10

4

Link: <https://leetcode.com/problems/maximum-profit-in-job-scheduling/>

### 3. Reverse Words in a String

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.

There is **at least one** word in s.

**Example 1:**

**Input:**

s = "the s

ky is blue"

**Output:**

"blue is sky the"

**Example 2:**

**Input:**

s = " hello world "

**Output:**

"world hello"

**Explanation:**

Your reversed string should not contain leading or trailing spaces.

**Example 3:**

**Input:**

s = "a good example"

**Output:**

"example good a"

**Explanation:**

You need to reduce multiple spaces between two words to a single space in the

reversed string.

**Constraints:**

•

<= s.length <=

1

10

4

•

s

contains English letters (upper

-

case and lower

-

case), digits, and spaces

' '

.

•

Link: <https://leetcode.com/problems/reverse-words-in-a-string/>

#include <string.h>  
#include <stdio.h>  
#include<stdlib.h>  
int main () {  
   char str[80];  
   printf("enter string : ");  
   gets(str);  
   char \*token;  
   char \*a[100];  
   int i=0,c=0;  
   token = strtok(str, " ");  
   printf("string before reversing : ");  
   while( token != NULL ) {  
      printf( "%s ", token );  
      a[i++]=token;  
      c++;  
      token = strtok(NULL, " ");  
   }  
   printf("\nafter reversing the words in a string : ");  
   for(i=c-1;i>=0;i--){  
       printf("%s ",a[i]);  
   }  
  
}

or

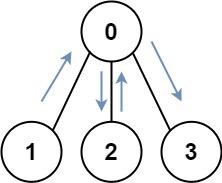
|  |
| --- |
| #include<stdio.h> |
|  | #include<stdlib.h> |
|  | #include<malloc.h> |
|  | #include<stdio.h> |
|  | #include<string.h> |
|  | char pop(char \*stack,int \*top) |
|  | { |
|  | return stack[(\*top)--]; |
|  | } |
|  | void push(char \*stack,int \*top,char ch) |
|  | { |
|  | stack[++(\*top)]=ch; |
|  |  |
|  | } |
|  | char \* reverseWords(char \* s){ |
|  | int right=strlen(s)-1; |
|  | char stack[strlen(s)]; |
|  | char \*ans=malloc(strlen(s)+1); |
|  | int top=-1; |
|  | int k=0; |
|  | for(int i=right;i>=0;i--) |
|  | { |
|  | if((s[i]==' ' && i==right)||(s[i]==' ' && i==0)) |
|  | { |
|  | continue; |
|  | } |
|  | else if(s[i]==' '&& s[i+1]==' ') |
|  | { |
|  | continue; |
|  | } |
|  | else if(s[i]==' ') |
|  | { |
|  | while(top!=-1) |
|  |  |
|  | { |
|  |  |
|  | ans[k++]=pop(stack,&top); |
|  | } |
|  |  |
|  | ans[k++]=' '; |
|  | } |
|  | else |
|  | { |
|  |  |
|  | push(stack,&top,s[i]); |
|  |  |
|  | } |
|  | } |
|  |  |
|  | while(top!=-1) |
|  | { |
|  |  |
|  | ans[k++]=pop(stack,&top); |
|  | } |
|  | if(ans[k-1]==' ') |
|  | { |
|  |  |
|  | ans[k-1]='\0'; |
|  | } |
|  | else |
|  | { |
|  |  |
|  | ans[k++]='\0'; |
|  |  |
|  | } |
|  |  |
|  | char \*p=ans; |
|  | return p; |
|  | } |
|  | void main() |
|  | { |
|  | int n; |
|  | char \*s; |
|  | printf("enter string"); |
|  | scanf("%[^\n]",s); |
|  | char \*k=reverseWords(s); |
|  | printf("The resulted string is: %s",k); |
|  | } |

### 4. Shortest Path Visiting All Nodes

You have an undirected, connected graph of n nodes labeled from 0 to n - 1. You are given an array graph where graph[i] is a list of all the nodes connected with node i by an edge.

Return *the length of the shortest path that visits every node*. You may start and stop at any node, you may revisit nodes multiple times, and you may reuse edges.

**Example 1:**



**Input:** graph = [[1,2,3],[0],[0],[0]]

**Output:** 4

**Explanation:** One possible path is [1,0,2,0,3]

**Example 2:**

**Input:**

graph = [[1],[0,2,4],[1,3,4],[2],[1,2]]

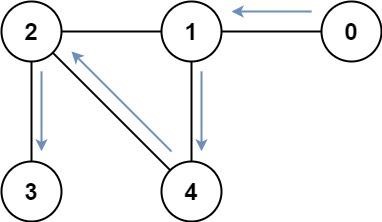
**Output:**

4

**Explanation:**

One possible path is [0,1,4,2,3]

**Constraints:**



|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| • | n == graph.length | | | | |  | |
| • • • •  • | 1 <= n <= 12 | | | |  | | |
| 0 <= graph[i].length < n | | | | | | i.   |  | | --- | | graph[b] |   , then contains a. |
| graph[i] | | does not contain | | | |
| If | graph[a] | | contains b  The input graph is always connected. | | |
|  | |

Link: <https://leetcode.com/problems/shortest-path-visiting-all-nodes/>

### 5. Unique Paths

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 | . |

**Example 1:**

**Input:**

m = 3, n = 7

**Output:**

28

**Example 2:**

**Input:**

m = 3, n = 2

**Output:**

3

**Explanation:**

From the top

-

left corner, there are a total of 3 ways to reach the bottom

-

right

corner:

1

. Right

-

>

Down

-

>

Down

2

. Down

-

>

Down

-

>

Right

3

. Down

-

>

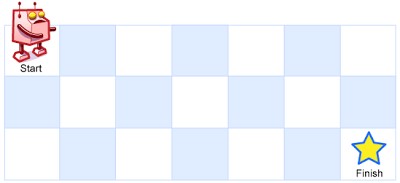
Right

-

>

Down

**Constraints:**



• 1 <= m, n <= 100

Link: <https://leetcode.com/problems/unique-paths/>

#include <stdio.h>  
int uniquePaths(int m, int n){  
int a[m][n];  
for(int i=0;i<m;i++){  
    for(int j=0;j<n;j++){  
        if(i==0 || j==0){  
            a[i][j]=1;  
        }  
        else{  
            a[i][j]=a[i-1][j]+a[i][j-1];  
        }  
    }  
}  
return a[m-1][n-1];  
  
}  
int main()  
{  
   int m,n;  
   printf("enter m n : ");  
   scanf("%d%d",&m,&n);  
   printf("number of uniquePaths are : %d",uniquePaths(m,n));  
}