VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Analysis and Design of Algorithms

Submitted by

Sameecha Sudheer (1BM20CS213)

in partial fulfillment for the award of the degree of
BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING

B.M.S. COLLEGE OF ENGINEERING
(Autonomous Institution under VTU)
BENGALURU-560019
May-2022 to July-2022

B. M. S. College of Engineering,

Bull Temple Road, Bangalore 560019

(Affiliated To Visvesvaraya Technological University, Belgaum)

Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the Lab work entitled "Analysis and Design of Algorithms" carried out by **Sameecha Sudheer (1BM20CS213),** who is bonafide student of **B. M. S. College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2022. The Lab report has been approved as it satisfies the academic requirements in respect of a Analysis and Design of Algorithms- **(**19CS4PCADA**)** work prescribed for the said degree.

Name of the Lab-Incharge : **Dr Manjunath D R**
Designation Assistant Professor
Department of CSE Department of CSE
BMSCE, Bengaluru BMSCE, Bengaluru

`

Index Sheet

17	Implement " Sum of Subsets" using Backtracking. " Sum of Subsets" problem: Find a subset of a given set S = {s1,s2,……,sn} of n positive integers whose sum is equal to a given positive integer d. For example, if S = {1,2,5,6,8} and d = 9 there are two solutions {1,2,6} and {1,8}. A suitable message is to be displayed if the given problem instance doesn't have a solution.	**48-49**
18	Implement " N-Queens Problem" using Backtracking.	**50-51**

Course Outcome

CO1	Ability to analyze time complexity of Recursive and Non-recursive algorithms using asymptotic notations
CO2	Ability to design efficient algorithms using various design techniques
CO3	Ability to apply the knowledge of complexity classes P, NP, and NP-Complete and prove certain problems are NP-Complete
CO4	Ability to conduct practical experiments to solve problems using an appropriate designing method and find time efficiency.

EXPERIMENT 1:

Write a recursive program to

a. Solve Towers-of-Hanoi problem b. To find GCD

a) Towers of Hanoi

Code:

SameechaS01 Create towers.c

1 contributor

21 lines (18 sloc) | 483 Bytes

```c
#include <stdio.h>


void towerOfHanoi(int n, char from_rod, char to_rod, char aux_rod)
{
        if (n == 1)
        {
                printf("\n Move disk 1 from rod %c to rod %c", from_rod, to_rod);
                return;
        }
        towerOfHanoi(n-1, from_rod, aux_rod, to_rod);
        printf("\n Move disk %d from rod %c to rod %c", n, from_rod, to_rod);
        towerOfHanoi(n-1, aux_rod, to_rod, from_rod);
}

int main()
{
        int n = 4; // Number of disks
        towerOfHanoi(n, \'A\', \'C\', \'B\'); // A, B and C are names of rods
        return 0;
}
```

Output:

```
/tmp/g7anAkvLYC.o
Enter number of disks4
The sequence of moves involved in the Tower of Hanoi are:

Move disk 1 from peg A to B
Move disk 2 from peg A to C
Move disk 1 from peg B to C
Move disk 3 from peg A to B
Move disk 1 from peg C to A
Move disk 2 from peg C to B
Move disk 1 from peg A to B
Move disk 4 from peg A to C
Move disk 1 from peg B to C
Move disk 2 from peg B to A
Move disk 1 from peg C to A
Move disk 3 from peg B to C
Move disk 1 from peg A to B
Move disk 2 from peg A to C
Move disk 1 from peg B to C
```

b)gcd:

SameechaS01 Create gcd.c

1 contributor

19 lines (15 sloc) | 347 Bytes

```c
#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;
}
```

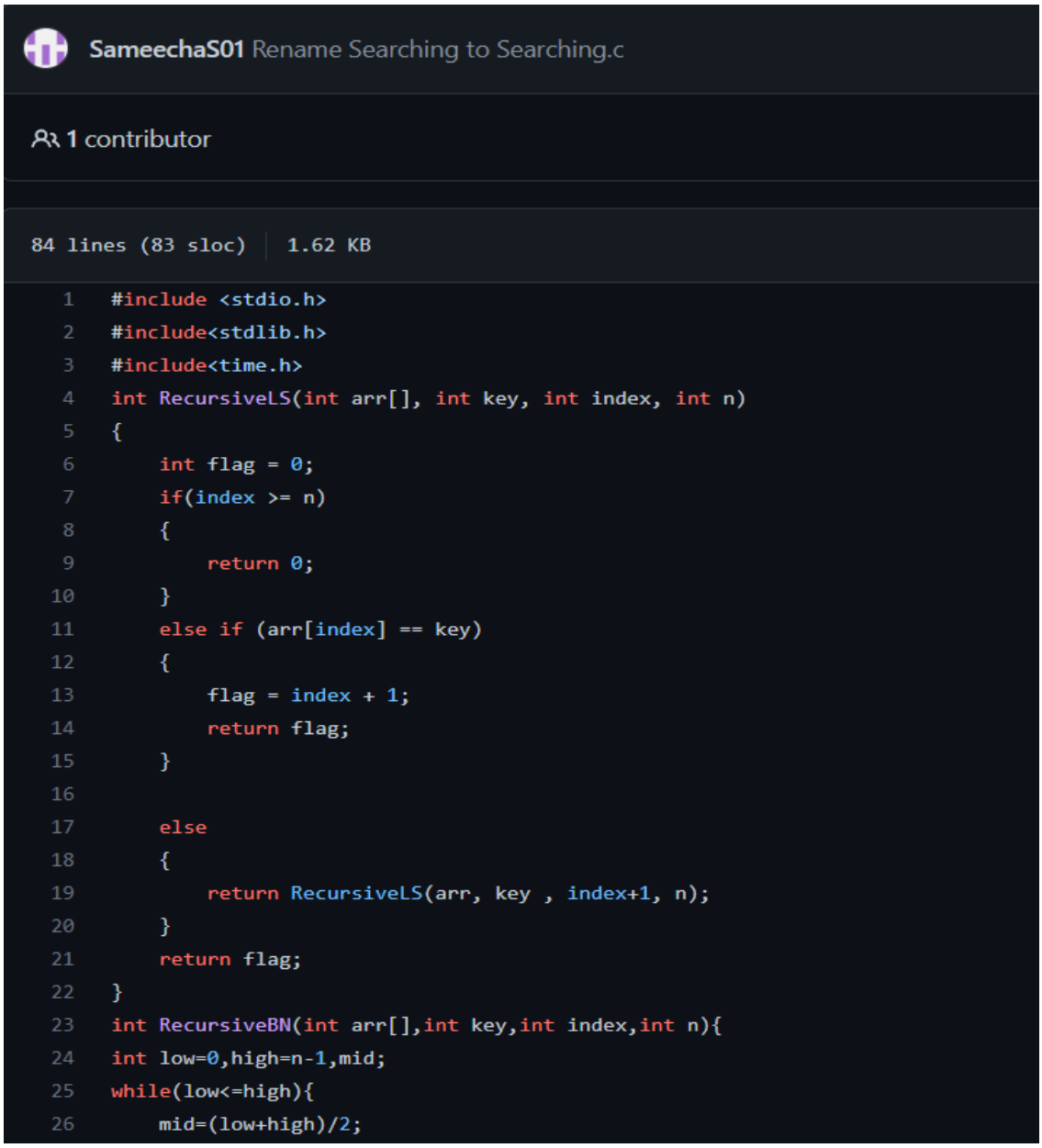
Output:

```
/tmp/g7anAkvLYC.o
Enter two integers: 5 88
G.C.D of 5 and 88 is 1
```

Experiment 2:

Implement Recursive Binary search and Linear search and determine the time required to search an element. Repeat the experiment for different values of N and plot a graph of the time taken versus N.

Code:



SameechaS01 Rename Searching to Searching.c

1 contributor

84 lines (83 sloc) | 1.62 KB

```c
#include <stdio.h>
#include<stdlib.h>
#include<time.h>
int RecursiveLS(int arr[], int key, int index, int n)
{
    int flag = 0;
    if(index >= n)
    {
        return 0;
    }
    else if (arr[index] == key)
    {
        flag = index + 1;
        return flag;
    }

    else
    {
        return RecursiveLS(arr, key , index+1, n);
    }
    return flag;
}
int RecursiveBN(int arr[],int key,int index,int n){
int low=0,high=n-1,mid;
while(low<=high){
    mid=(low+high)/2;
```

```
27      if(key==arr[mid])
28          return mid;
29      else if(key<arr[mid])
30          high=mid-1;
31      else
32          low=mid+1;
33  }
34  return -1;
35  }
36  int main()
37  {
38      int n,i,j, key , pos , m = 0, arr[5000],ch;
39      clock_t s,e;
40      while(1){
41      printf("\nEnter the choice:1.Linear search 2.Binary search\n");
42      scanf("%d",&ch);
43      switch(ch){
44      case 1:  n=500;
45      while(n<=3000){
46      for(i=0;i<n;i++){
47          arr[i];
48      }
49      key=arr[n-1];
50      s=clock();
51      pos= RecursiveLS(arr, key , 0, n);;
52      if(pos!=0)
53      printf("\n Element found ");
54      else
55      printf("element not found");
56      for(j=0;j<80000000;j++){}
57      e=clock();
58      printf("\n Time taken for %d elements is %f\n",n,((double)(e-s))/CLK_TCK);
59      n=n+500;
60      }
61      break;
```

```
        case 2:  n=500;
        while(n<=3000){
        for(i=0;i<n;i++){
            arr[i];
        }
        key=arr[n-1];
        s=clock();
        pos= RecursiveBN(arr, key , 0, n);;
        if(pos!=0)
        printf("\n Element found");
        else
        printf("element not found");
        for(j=0;j<80000000;j++){}
        e=clock();
        printf("\n Time taken for %d elements is %f\n",n,((double)(e-s))/CLK_TCK);
        n=n+500;
        }
        break;
        default: exit(0);
        }
        }
        return 0;
    }
```

EXPERIMENT 4:

Sort a given set of N integer elements using Selection Sort technique and compute its time taken. Run the program for different values of N and record the time taken to sort.

Code:

SameechaS01 Rename selection sort to selection sort.c

1 contributor

83 lines (68 sloc) | 1.43 KB

```c
#include<stdio.h>
#include<time.h>
#include<stdlib.h>
void swap(int *xp, int *yp)
{
    int temp = *xp;
    *xp = *yp;
    *yp = temp;
}

void selectionSort(int arr[], int n)
{
    int i, j, min_idx;


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

        min_idx = i;
        for (j = i+1; j < n; j++)
          if (arr[j] < arr[min_idx])
            min_idx = j;


        swap(&arr[min_idx], &arr[i]);
    }
}
```

```c
void printArray(int arr[], int size)
{
    int i;
    for (i=0; i < size; i++)
        printf("%d ", arr[i]);
    printf("\n");
}


int main()
{
    int s,ch;
    clock_t start,end;
    printf("1:Manual entry 2:Random entry\n");
    scanf("%d",&ch);
    switch(ch)
    {

    case 1:
    {printf("Enter size of array ");
    scanf("%d",&s);
    int arr[s];
    for(int i =0;i<s;i++)
    {
        scanf("%d",&arr[i]);
    }
    int n = sizeof(arr)/sizeof(arr[0]);

    start=clock();
    selectionSort(arr, n);

    printf("Sorted array 1 : \n");
```

```
63        printArray(arr,  n);
64       end=clock();
65       printf ("\nTime taken: %f", (double)(end - start)/(CLOCKS_PER_SEC));
66         break;
67        }
68
69        case 2:
70        {
71        int arr2[]={34,56,23,45,98,12};
72        int n1 = sizeof(arr2)/sizeof(arr2[0]);
73        start = clock();
74        selectionSort(arr2,n1);
75        printf("Sorted array 2 : \n");
76        printArray(arr2,n1);
77        end=clock();
78       printf ("\nTime taken: %f", (double)(end - start)/(CLOCKS_PER_SEC));
79         break;
80        }
81        }
82        return 0;
83   }
```

Output:

```
/tmp/g7anAkvLYC.o
1:Manual entry 2:Random entry
2
Sorted array 2 :
12 23 34 45 56 98

Time taken: 0.000029
```

EXPERIMENT 4:

Write program to do the following: a. Print all the nodes reachable from a given starting node in a digraph using BFS method. b. Check whether a given graph is connected or not using DFS method.

BFS Code:

SameechaS01 Rename BFS to BFS.c

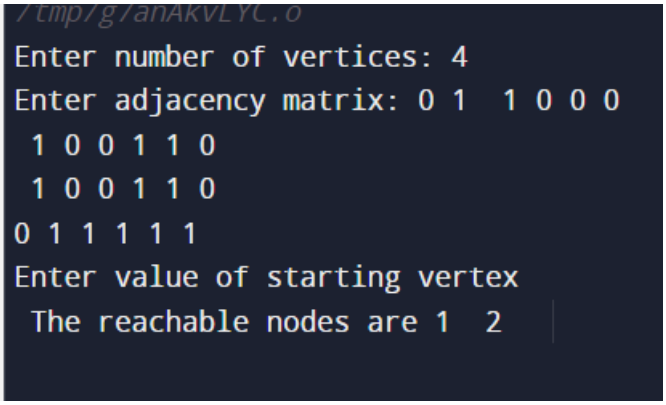
1 contributor

62 lines (60 sloc) | 731 Bytes

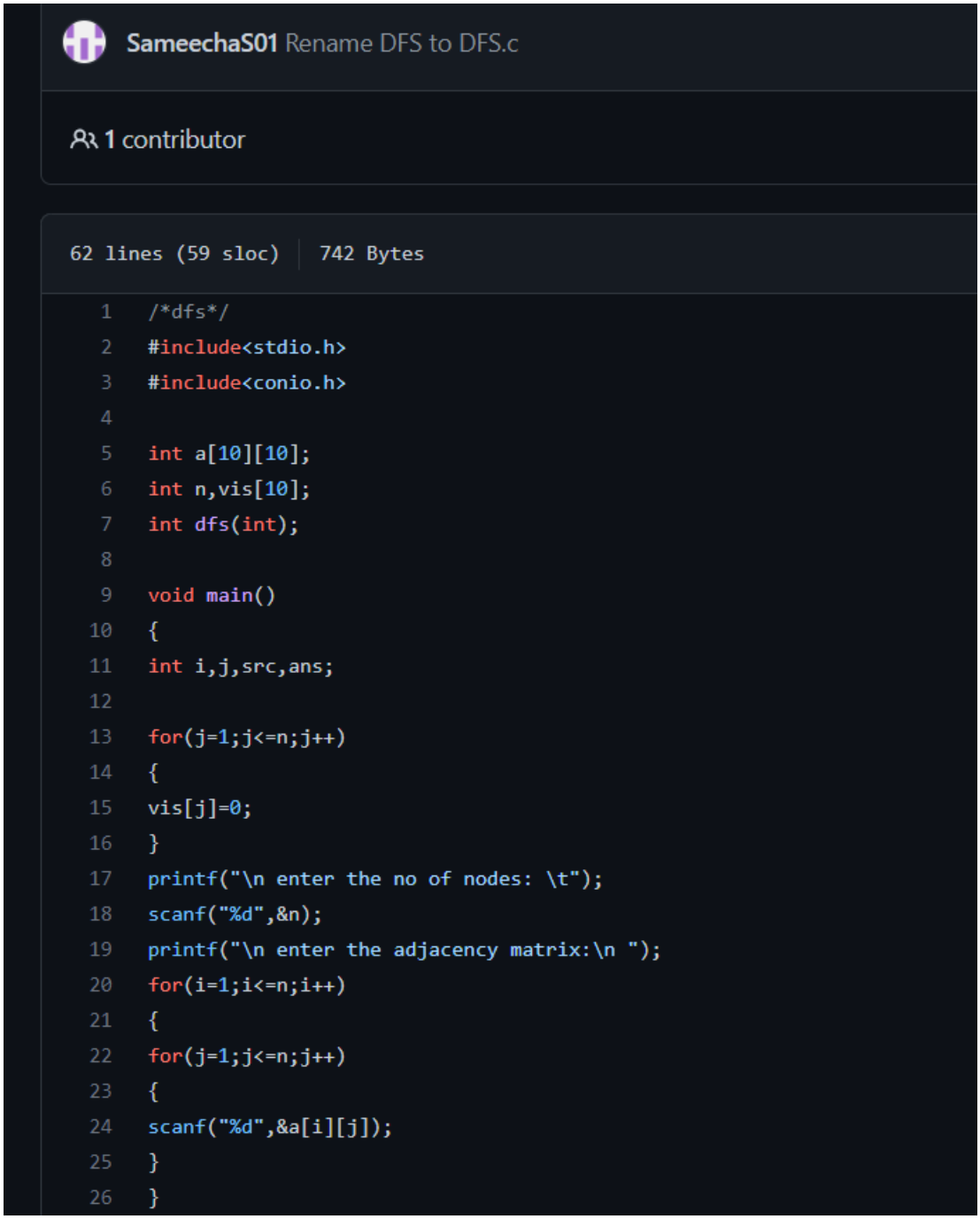
```c
/*bfs*/
#include<stdio.h>
#include<conio.h>

int a[10][10];
int n;
void bfs(int);

void main()
{
    int i,j,src;
    printf("\n enter the no of nodes:\t ");
    scanf("%d",&n);
    printf("\n enter the adjacency matrix:\n");
    for(i=1;i<=n;i++)
{
for(j=1;j<=n;j++)
{
scanf("%d",&a[i][j]);
}
}
printf("\n Enter the source node:");
scanf("%d",&src);
bfs(src);
getch();
}
```

```c
void bfs(int src)
{
int q[10],f=0,r=-1,vis[10],i,j;
for(j=1;j<=n;j++)
{
vis[j]=0;
}
vis[src]=1;
r=r+1;
q[r]=src;
while(f<=r)
{
i=q[f];
f=f+1;
for(j=1;j<=n;j++)
{
if(a[i][j]==1 && vis[j]!=1)
{
vis[j]=1;
r=r+1;
q[r]=j;
}
}
}
for(j=1;j<=n;j++)
{
if(vis[j]!=1)
{
printf("\n node %d is not reachable\n",j);
}
else
{
printf(" \n node %d is reachable \n" ,j);
}
}
```

OUTPUT:


/tmp/g7anAKvLYC.o
Enter number of vertices: 4
Enter adjacency matrix: 0 1 1 0 0 0
1 0 0 1 1 0
1 0 0 1 1 0
0 1 1 1 1 1
Enter value of starting vertex
The reachable nodes are 1 2

DFS CODE:



SameechaS01 Rename DFS to DFS.c

1 contributor

62 lines (59 sloc) | 742 Bytes

```c
/*dfs*/
#include<stdio.h>
#include<conio.h>

int a[10][10];
int n,vis[10];
int dfs(int);

void main()
{
int i,j,src,ans;

for(j=1;j<=n;j++)
{
vis[j]=0;
}
printf("\n enter the no of nodes: \t");
scanf("%d",&n);
printf("\n enter the adjacency matrix:\n ");
for(i=1;i<=n;i++)
{
for(j=1;j<=n;j++)
{
scanf("%d",&a[i][j]);
}
}
```

```
printf("\n enter the source node:\t");
scanf("%d",&src);
ans = dfs(src);
if(ans==1)
{
printf("\n graph is connected\n ");
}
else
{
printf("\n gragh is not connected\n");
}
getch();
}
int dfs(int src)
{
int j;
vis[src]=1;
for(j=1;j<=n;j++)
{
if(a[src][j]==1 && vis[j]!=1)
{
dfs(j);
}
}
for(j=1;j<=n;j++)
{
if(vis[j]!=1)
{
printf("\n node %d is not reachable\n",j);
}
else
{
printf(" \n node %d is reachable \n" ,j);
}
}
```

OUTPUT:

```
Enter number of vertices4
NEnter the adjacent matrix 0 1 1 0 0 0
0 1 1 0 0 0
 1 0 0 1 1 0
 1 0 0 1 1 0
0 1 1 0 0
1-->2

2-->3

graph is not connected
```

EXPERIMENT 5:

Sort a given set of N integer elements using Insertion Sort technique and compute its time taken.

CODE:

SameechaS01 Rename insertion sort to insertion sort.c

1 contributor

53 lines (49 sloc) | 1.21 KB

```c
#include <stdio.h>
#include<time.h>

void insertSorting(int arr[], int num)
{
    int i, j, temp;
    for (i = 1; i < num; i++) {
        temp = arr[i];
        j = i - 1;

        while(j>=0 && temp <= arr[j])
        {
            arr[j+1] = arr[j];
            j = j-1;
        }
        arr[j+1] = temp;
    }
}

void main()
{
    clock_t start, end;
    double time;
    int arr[10000],num,i;
    int choice;
```

```
27      scanf("%d",&choice);
28      switch (choice)
29      {
30      case 1:
31      printf("Enter the number of elements:\n");
32      scanf("%d",&num);
33      printf("Enter the array elements:\n");
34      for(i=0;i<num;i++){
35          scanf("%d",&arr[i]);
36      }
37
38      case 2:
39      printf("Enter the number of values : ");
40             scanf("%d", &num);
41             for(int i=0;i<num;i++){
42                 arr[i]=num-i;
43             }
44      }
45      start= clock();
46      insertSorting(arr, num);
47      end= clock();
48      printf("\nAfter sorting the array elements are: \n");
49      for (i = 0; i < num; i++)
50          printf("%d ", arr[i]);
51      time= (((double)(end-start))/CLOCKS_PER_SEC);
52      printf("\nThe time taken to sort the of elements is %f",time);
53  }
```

OUTPUT:

```
/tmp/g7anAkvLYC.o
Enter 1 for user entry & 2 for random entry2
Enter the number of values : 5
After sorting the array elements are:
1 2 3 4 5
The time taken to sort the of elements is 0.000001
```

EXPERIMENT 6:

Write program to obtain the Topological ordering of vertices in a given digraph.

CODE:

SameechaS01 Rename Topological Sorting to Topological Sorting.c

1 contributor

44 lines (34 sloc) | 795 Bytes

```c
#include <stdio.h>

int main(){
int i,j,k,n,a[10][10],indeg[10],flag[10],count=0;

printf("Enter the no of vertices:\n");
scanf("%d",&n);

printf("Enter the adjacency matrix:\n");
for(i=0;i<n;i++){
printf("Enter row %d\n",i+1);
for(j=0;j<n;j++)
scanf("%d",&a[i][j]);
}

for(i=0;i<n;i++){
        indeg[i]=0;
        flag[i]=0;
    }

    for(i=0;i<n;i++)
        for(j=0;j<n;j++)
            indeg[i]=indeg[i]+a[j][i];

```

```
25      printf("\nThe topological order is:");
26
27      while(count<n){
28          for(k=0;k<n;k++){
29              if((indeg[k]==0) && (flag[k]==0)){
30                  printf("%d ",(k+1));
31                  flag [k]=1;
32              }
33
34              for(i=0;i<n;i++){
35                  if(a[i][k]==1)
36                      indeg[k]--;
37              }
38          }
39
40          count++;
41      }
42
43      return 0;
44  }
```

OUTPUT:

```
/tmp/B21I4qPcSK.o
Enter number of nodes5
Enter adjacent matrix0 1 1 0 0 0
1 0 0 1 1 0
1 0 0 0 1 0
0 1 0 0 1 1
0 1 1 1 0 1
topological ordering:
1102095648  22051   0   0   2
Time taken to order the vertices: 0
```

EXPERIMENT 7:

Implement Johnson Trotter algorithm to generate permutations

CODE:

SameechaS01 Create johnson_trotter.c

1 contributor

118 lines (107 sloc) | 2 KB

```c
#include <stdio.h>
#include <stdlib.h>


int swap(int *a,int *b)
{
     int t = *a;
    *a = *b;
    *b = t;
}

int search(int arr[],int num,int mobile)
{
    int g;
    for(g=0;g<num;g++)
    {
        if(arr[g] == mobile)
        {
             return g;
        }
    }
    return -1;
}

int find_Moblie(int arr[],int d[],int num)
{
```

```
27      int mobile = 0;
28
29      int i;
30      for(i=0;i<num;i++)
31      {
32          if((d[arr[i]-1] == 0) && i != 0)
33          {
34              if(arr[i]>arr[i-1] && arr[i]>mobile)
35              {
36                  mobile =  arr[i];
37
38              }
39
40          }
41          else if((d[arr[i]-1] == 1) & i != num-1)
42          {
43              if(arr[i]>arr[i+1] && arr[i]>mobile)
44              {
45                  mobile = arr[i];
46
47              }
48
49          }
50
51      }
52      if(mobile == 0)
53          return 0;
54      else
55          return mobile;
56  }
```

```c
void permutations(int arr[],int d[],int num)
{
    int i;
    int mobile = find_Moblie(arr,d,num);
    int pos = search(arr,num,mobile);
    if(d[arr[pos]-1]==0)
        swap(&arr[pos],&arr[pos-1]);
    else
        swap(&arr[pos],&arr[pos+1]);
    for(int i=0;i<num;i++)
    {
        if(arr[i] > mobile)
        {
            if(d[arr[i]-1]==0)
                d[arr[i]-1] = 1;
            else
                d[arr[i]-1] = 0;
        }
    }
    for(i=0;i<num;i++)
    {
        printf(" %d ",arr[i]);
    }
}

int factorial(int k)
{
    int f = 1;
    int i = 0;
    for(i=1;i<k+1;i++)
    {
        f = f*i;
    }
    return f;
}
```

```c
int main()
{
    int num = 0;
    int i;
    int j;
    int z = 0;
    printf("Johnson trotter algorithm to find all permutations of given numbers \n");
    printf("Enter the number\n");
    scanf("%d",&num);
    int arr[num],d[num];
    z = factorial(num);
    printf("The total permutations are %d",z);
    printf("\nAll possible permutations are: \n");
    for(i=0;i<num;i++)
    {
        d[i] = 0;
        arr[i] = i+1;
        printf(" %d ",arr[i]);
    }
    printf("\n");
    for(j=1;j<z;j++)
    {
        permutations(arr,d,num);
        printf("\n");
    }
    return 0;
}
```

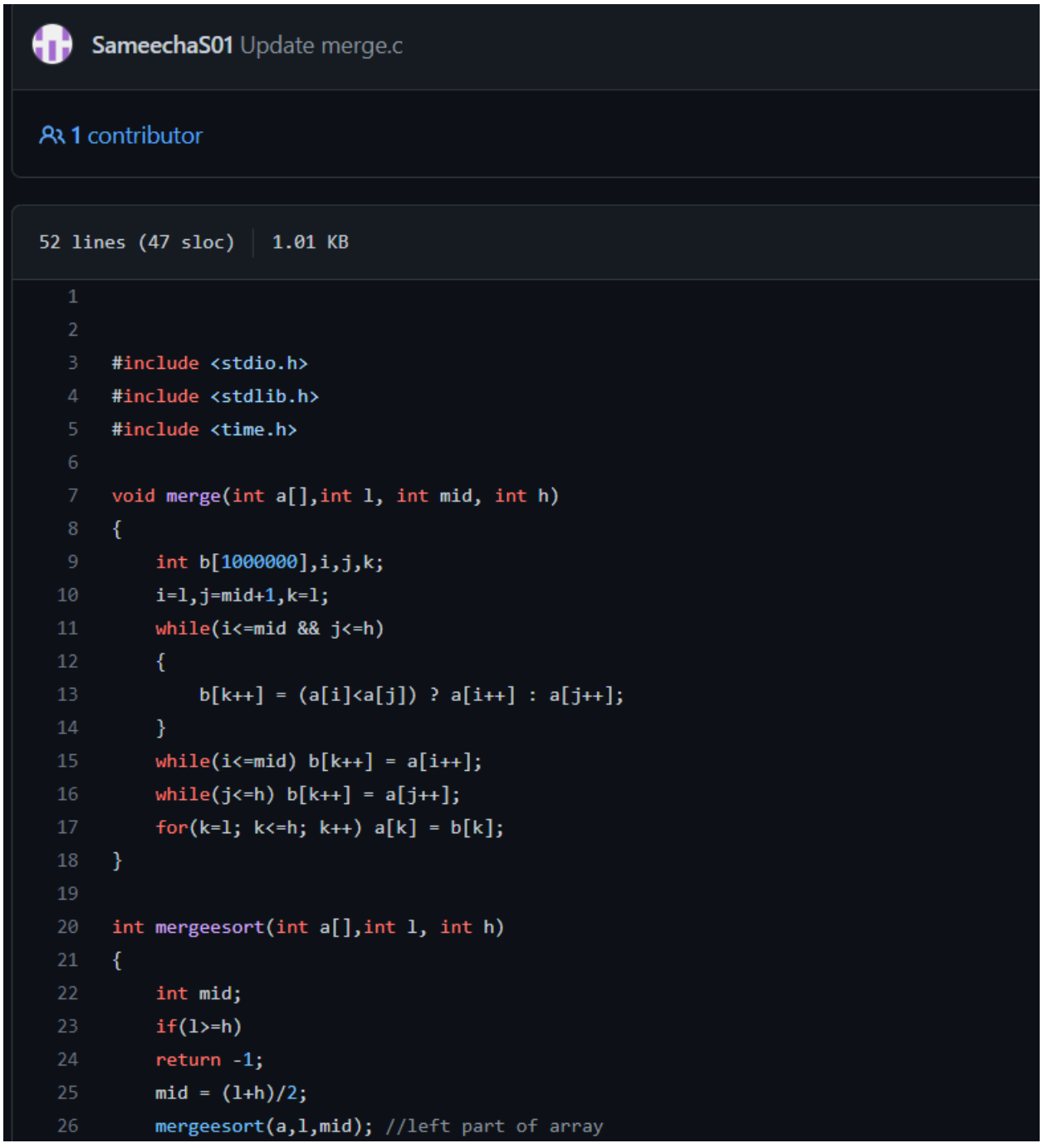
OUTPUT:

```
/tmp/OE7iSxhwkC.o
Johnson trotter algorithm to find all permutations of given numbers
Enter the number
3
The total permutations are 6
All possible permutations are:
 1  2  3
 1  3  2
 3  1  2
 3  2  1
 2  3  1
 2  1  3
```

EXPERIMENT 8:

Sort a given set of N integer elements using Merge Sort technique and compute its time taken. Run the program for different values of N and record the time taken to sort.

CODE:



SameechaS01 Update merge.c

1 contributor

52 lines (47 sloc) | 1.01 KB

```c

#include <stdio.h>
#include <stdlib.h>
#include <time.h>

void merge(int a[],int l, int mid, int h)
{
    int b[1000000],i,j,k;
    i=l,j=mid+1,k=l;
    while(i<=mid && j<=h)
    {
        b[k++] = (a[i]<a[j]) ? a[i++] : a[j++];
    }
    while(i<=mid) b[k++] = a[i++];
    while(j<=h) b[k++] = a[j++];
    for(k=l; k<=h; k++) a[k] = b[k];
}

int mergeesort(int a[],int l, int h)
{
    int mid;
    if(l>=h)
    return -1;
    mid = (l+h)/2;
    mergeesort(a,l,mid); //left part of array
```

```
27      mergeesort(a,mid+1,h); //right part of array
28      merge(a,l,mid,h);
29  }
30
31  int main()
32  {
33      int a[100000],n,i;
34      clock_t c;
35      printf("\nEnter the size: ");
36      scanf("%d",&n);
37      printf("\n The elements before sorting ");
38      for(i=0; i<n;i++)
39      {
40          a[i] = rand()%100;
41          printf("%d\t",a[i]);
42      }
43      c = clock();
44      mergeesort(a,0,n-1);
45      c = c - clock();
46      printf("\n The elements after sorting ");
47      for(i=0; i<n;i++)
48      {
49          printf("%d\t",a[i]);
50      }
51      printf("\n Time taken is = %lu",c/CLOCKS_PER_SEC);
52  }
```

OUTPUT:

```
/tmp/OE7iSxhwkC.o
Enter the size: 3
The elements before sorting 83  86  77
 The elements after sorting 77  83  86
Time taken is = 0
```

EXPERIMENT 9:

Sort a given set of N integer elements using Quick Sort technique and compute its time taken

CODE:

SameechaS01 Create quickSort.c

1 contributor

75 lines (38 sloc) | 762 Bytes

```
1   #include<stdio.h>
2
3   void quicksort(int number[25],int first,int last){
4
5   int i, j, pivot, temp;
6
7   if(first<last){
8
9   pivot=first;
10
11  i=first;
12
13  j=last;
14
15  while(i<j){
16
17  while(number[i]<=number[pivot]&&i<last)
18
19  i++;
20
21  while(number[j]>number[pivot])
22
23  j--;
24
25  if(i<j){
```

```
27   temp=number[i];
28
29   number[i]=number[j];
30
31   number[j]=temp;
32
33   }
34
35   }
36
37   temp=number[pivot];
38
39   number[pivot]=number[j];
40
41   number[j]=temp;
42
43   quicksort(number,first,j-1);
44
45   quicksort(number,j+1,last);
46
47   }
48
49   }
50
51   int main(){
52
53   int i, count, number[25];
54
55   printf("Enter some elements (Max. - 25): ");
56
57   scanf("%d",&count);
58
59   printf("Enter %d elements: ", count);
60
61   for(i=0;i<count;i++)
```

```
59  printf("Enter %d elements: ", count);
60
61  for(i=0;i<count;i++)
62
63  scanf("%d",&number[i]);
64
65  quicksort(number,0,count-1);
66
67  printf("The Sorted Order is: ");
68
69  for(i=0;i<count;i++)
70
71  printf(" %d",number[i]);
72
73  return 0;
74
75  }
```

OUTPUT:

```
Output

/tmp/OE7iSxhwkC.o
Enter some elements (Max. - 25): 5
Enter 5 elements: 56 55 89 21 6
The Sorted Order is:  6 21 55 56 89
```

Experiment 9:

Sort a given set of N integer elements using Quick Sort technique and compute its time taken

CODE:

SameechaS01 Create quickSort.c — Latest commit 99eaa3c 22 days ago — History

1 contributor

75 lines (38 sloc) | 762 Bytes — Raw | Blame

```c
#include<stdio.h>

void quicksort(int number[25],int first,int last){

int i, j, pivot, temp;

if(first<last){

pivot=first;

i=first;

j=last;

while(i<j){

while(number[i]<=number[pivot]&&i<last)

i++;

while(number[j]>number[pivot])

j--;

if(i<j){

temp=number[i];

number[i]=number[j];

number[j]=temp;

}

}

temp=number[pivot];

number[pivot]=number[j];

number[j]=temp;

quicksort(number,first,j-1);

quicksort(number,j+1,last);

}

}

int main(){

int i, count, number[25];

printf("Enter some elements (Max. - 25): ");

scanf("%d",&count);

printf("Enter %d elements: ", count);

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

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

quicksort(number,0,count-1);

printf("The Sorted Order is: ");

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

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

return 0;

}
```

output:

```
1:For manual entry of N value and array elements
2:To display time taken for sorting number of elements N in the range 500 to 14500
3:To exit
Enter your choice:1
Enter the number of elements: 5
Enter array elements: 45 23 89 11 60
Unsorted Array
45  23  89  11  60

Sorted array is: 11  23  45  60  89
11      23      45      60      89
Time taken to sort 5 numbers is 0.000020 Secs

1:For manual entry of N value and array elements
2:To display time taken for sorting number of elements N in the range 500 to 14500
3:To exit
Enter your choice:2

 Time taken to sort 500 numbers is 0.000812 Secs
 Time taken to sort 1500 numbers is 0.000818 Secs
 Time taken to sort 2500 numbers is 0.000813 Secs
 Time taken to sort 3500 numbers is 0.000817 Secs
 Time taken to sort 4500 numbers is 0.000825 Secs
 Time taken to sort 5500 numbers is 0.000821 Secs
 Time taken to sort 6500 numbers is 0.000823 Secs
 Time taken to sort 7500 numbers is 0.000824 Secs
 Time taken to sort 8500 numbers is 0.000843 Secs
 Time taken to sort 9500 numbers is 0.000830 Secs
 Time taken to sort 10500 numbers is 0.000831 Secs
 Time taken to sort 11500 numbers is 0.000833 Secs
 Time taken to sort 12500 numbers is 0.000832 Secs
 Time taken to sort 13500 numbers is 0.000842 Secs
 Time taken to sort 14500 numbers is 0.000842 Secs
```

Experiment 10:

Sort a given set of N integer elements using Heap Sort technique and compute its time taken.

CODE:

SameechaS01 Create heapsort.c — Latest commit

1 contributor

87 lines (77 sloc) | 2.08 KB

```c
//Heap Sort
#include <stdio.h>
#include<time.h>
#include<stdlib.h>

void swap(int* a, int* b) {
        int temp = *a;
        *a = *b;
        *b = temp;
}

void heapify(int arr[], int n, int i) {
        int largest = i;
        int left = 2 * i + 1;
        int right = 2 * i + 2;
        if (left < n && arr[left] > arr[largest])
                largest = left;
        if (right < n && arr[right] > arr[largest])
                largest = right;
        if (largest != i) {
                swap(&arr[i], &arr[largest]);
                heapify(arr, n, largest);
        }
}
```

```c
void heapSort(int arr[], int n) {
        for (int i = n / 2 - 1; i >= 0; i--)
                heapify(arr, n, i);
        for (int i = n - 1; i >= 0; i--) {
                swap(&arr[0], &arr[i]);
        heapify(arr, i, 0);
        }
}

int main() {
        int arr[15000], n, i, j, ch, temp;
    clock_t start, end;
    while(1) {
    printf("\n1:For manual entry of N value and array elements");
    printf("\n2:To display time taken for sorting number of elements N in the range 500 to 14500");
    printf("\n3:To exit");
    printf("\nEnter your choice:");
    scanf("%d", &ch);
    switch(ch) {
        case 1: printf("\nEnter the number of elements: ");
        scanf("%d", &n);
        printf("\nEnter array elements: ");

        for(i = 0; i < n; i++) {
            scanf("%d", &arr[i]);
        }
        start = clock();
        heapSort(arr, n);
        end = clock();
        printf("\nSorted array is: ");

        for(i = 0; i < n; i++)
            printf("%d\t", arr[i]);

        printf("\n Time taken to sort %d numbers is %f Secs", n, (((double)(end - start))/CLOCKS_PER_SEC));
        break;
```

```
        break;

        case 2:
            n = 500;
            while(n <= 14500) {
            for(i = 0; i < n; i++) {
            //a[i]=random(1000);
                arr[i] = n - i;
            }
            start = clock();
            heapSort(arr, n);

            //Dummy loop to create delay
            for(j = 0; j < 500000; j++) {
                temp = 38/600;
            }
            end = clock();
            printf("\n Time taken to sort %d numbers is %f Secs", n, (((double)(end-start))/CLOCKS_PER_SEC));
            n = n + 1000;
            }
        break;

        case 3: exit(0);
    }
getchar();
}
}
```

OUTPUT:

30 lines (29 sloc) | 1.2 KB Raw Blame

```
1:For manual entry of N value and array elements
2:To display time taken for sorting number of elements N in the range 500 to 14500
3:To exit
Enter your choice:1
Enter the number of elements: 5
Enter array elements: 23 455 856 12 678
Sorted array is: 12     23      455     678     856
Time taken to sort 5 numbers is 0.000002 Secs

1:For manual entry of N value and array elements
2:To display time taken for sorting number of elements N in the range 500 to 14500
3:To exit
Enter your choice:2
Time taken to sort 500 numbers is 0.000564 Secs
Time taken to sort 1500 numbers is 0.000748 Secs
Time taken to sort 2500 numbers is 0.000964 Secs
Time taken to sort 3500 numbers is 0.001181 Secs
Time taken to sort 4500 numbers is 0.001376 Secs
Time taken to sort 5500 numbers is 0.001627 Secs
Time taken to sort 6500 numbers is 0.001671 Secs
Time taken to sort 7500 numbers is 0.001857 Secs
Time taken to sort 8500 numbers is 0.002317 Secs
Time taken to sort 9500 numbers is 0.002525 Secs
Time taken to sort 10500 numbers is 0.002804 Secs
Time taken to sort 11500 numbers is 0.003074 Secs
Time taken to sort 12500 numbers is 0.003294 Secs
Time taken to sort 13500 numbers is 0.003506 Secs
Time taken to sort 14500 numbers is 0.003840 Secs
```

Experiment 11:

Implement Warshall's algorithm using dynamic programming.

CODE:

main ▾ ADA_LAB / warshal.c — Go to file

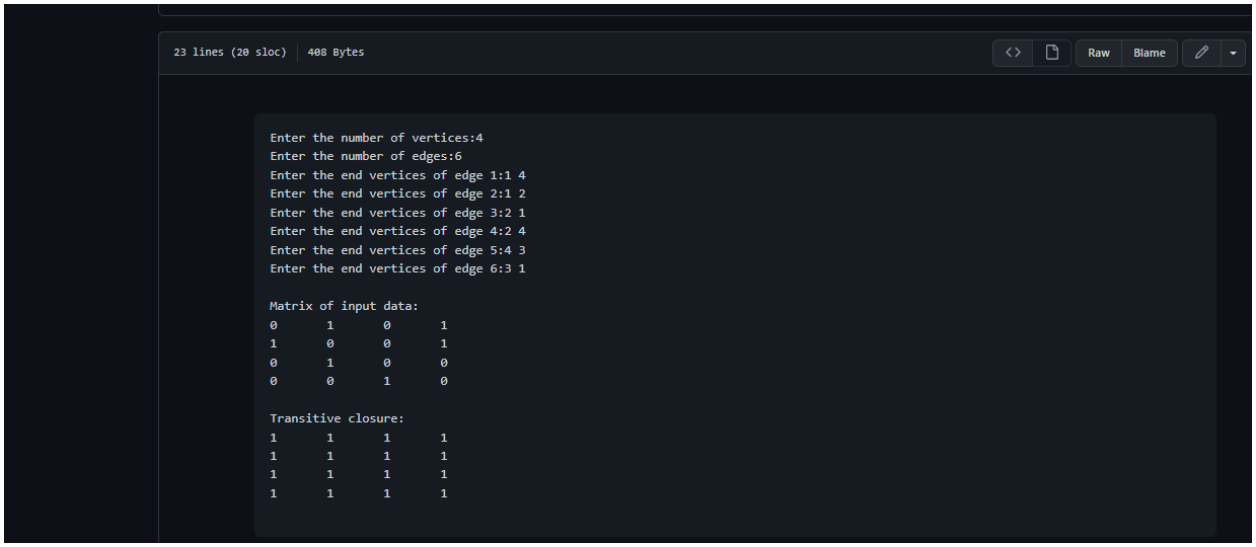
SameechaS01 Create warshal.c — Latest commit e79da03 13 days ago — History

1 contributor

70 lines (61 sloc) | 1.05 KB — Raw | Blame

```c
#include <stdio.h>

int min(int,int);
void floyds(int p[10][10],int n)
{
 int i,j,k;
 for(k=1;k<=n;k++)
  for(i=1;i<=n;i++)
   for(j=1;j<=n;j++)
    if(i==j)
     p[i][j]=0;
    else
     p[i][j]=min(p[i][j],p[i][k]+p[k][j]);
}

int min(int a,int b)
{
 if(a<b)
  return(a);
 else
  return(b);
}

void main()
{
 int p[10][10],w,n,e,u,v,i,j;;
 printf("\n Enter the number of vertices:");
 scanf("%d",&n);
 printf("\n Enter the number of edges:\n");
 scanf("%d",&e);
 for(i=1;i<=n;i++)
 {
  for(j=1;j<=n;j++)
   p[i][j]=999;
 }
 for(i=1;i<=e;i++)
 {
  printf("\n Enter the end vertices of edge%d with its weight \n",i);
  scanf("%d%d%d",&u,&v,&w);
  p[u][v]=w;
 }

 printf("\n Matrix of input data:\n");
 for(i=1;i<=n;i++)
 {
  for(j=1;j<=n;j++)
   printf("%d \t",p[i][j]);
  printf("\n");
 }

 floyds(p,n);

 printf("\n Transitive closure:\n");
 for(i=1;i<=n;i++)
 {
  for(j=1;j<=n;j++)
   printf("%d \t",p[i][j]);
  printf("\n");
 }

 printf("\n The shortest paths are:\n");
 for(i=1;i<=n;i++)
  for(j=1;j<=n;j++)
  {
   if(i!=j)
    printf("\n <%d,%d>=%d",i,j,p[i][j]);
  }

}
```

OUTPUT:


23 lines (20 sloc) 408 Bytes
Raw
Blame
Enter the number of vertices:4
Enter the number of edges:6
Enter the end vertices of edge 1:1 4
Enter the end vertices of edge 2:1 2
Enter the end vertices of edge 3:2 1
Enter the end vertices of edge 4:2 4
Enter the end vertices of edge 5:4 3
Enter the end vertices of edge 6:3 1
Matrix of input data:
0 1 0 1
1 0 0 1
0 1 0 0
0 0 1 0
Transitive closure:
1 1 1 1
1 1 1 1
1 1 1 1
1 1 1 1

Experiment 12:

Implement 0/1 Knapsack problem using dynamic programming.

CODE:

main ▾ ADA_LAB / knapsack.c Go to file ···

SameechaS01 Create knapsack.c Latest commit 7933199 13 days ago History

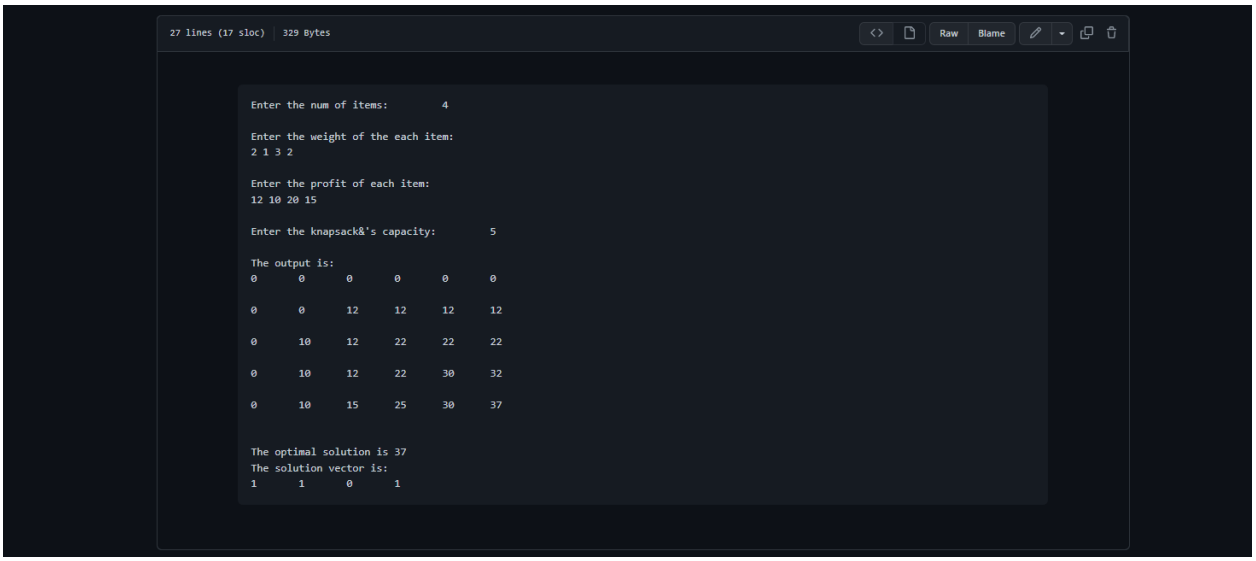
1 contributor

55 lines (49 sloc) | 865 Bytes Raw Blame

```c
#include <stdio.h>

int max(int a, int b)
{
  if(a>b)
  return a;
  else
  return b;
}

int knapsack(int W, int wt[], int val[], int n)
{
  int i,w;
  int K[n+1][W+1];
  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] = 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];
}

int main()
{
  int i,n,val[20],wt[20],w;
  printf("\nEnter number of items: ");
  scanf("%d",&n);
  printf("\nEnter value and weight of items: ");
```

```c
  printf("\nEnter value and weight of items: ");
  for(i=0; i<n; ++i){
    scanf("%d%d",&val[i],&wt[i]);
  }
  printf("\nEnter size of knapsack: ");
  scanf("%d",&W);
  printf("%d",knapsack(W,wt,val,n));
  return 0;
}
/*
Enter number of items: 3

Enter value and weight of items:
100 20
50 10
150 30

Enter size of knapsack: 50
250

*/
```

OUTPUT:


27 lines (17 sloc) | 329 Bytes
Raw
Blame
Enter the num of items: 4
Enter the weight of the each item:
2 1 3 2
Enter the profit of each item:
12 10 20 15
Enter the knapsack&'s capacity: 5
The output is:
0 0 0 0 0 0
0 0 12 12 12 12
0 10 12 22 22 22
0 10 12 22 30 32
0 10 15 25 30 37
The optimal solution is 37
The solution vector is:
1 1 0 1

Experiment 13:

Implement All Pair Shortest paths problem using Floyd's algorithm.

CODE:

main ADA_LAB / floyd.c Go to file

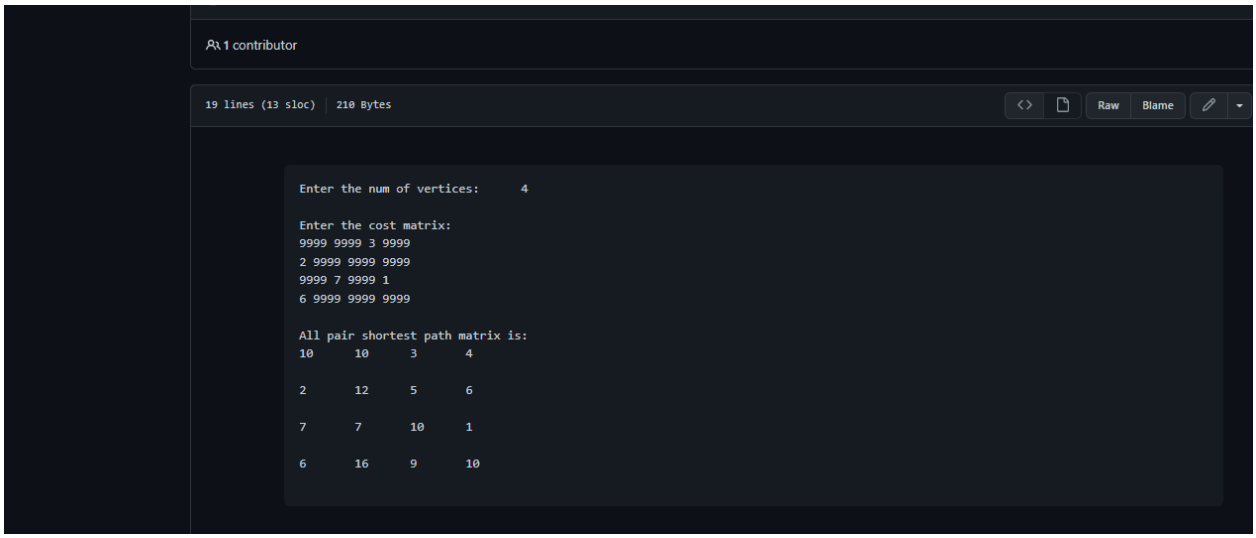
SameechaS01 Create floyd.c Latest commit dfba499 13 days ago History

1 contributor

70 lines (61 sloc) 1.05 KB Raw Blame

```c
#include <stdio.h>

int min(int,int);
void floyds(int p[10][10],int n)
{
 int i,j,k;
 for(k=1;k<=n;k++)
  for(i=1;i<=n;i++)
   for(j=1;j<=n;j++)
    if(i==j)
     p[i][j]=0;
    else
     p[i][j]=min(p[i][j],p[i][k]+p[k][j]);
}

int min(int a,int b)
{
 if(a<b)
  return(a);
 else
  return(b);
}

void main()
{
 int p[10][10],w,n,e,u,v,i,j;;
 printf("\n Enter the number of vertices:");
 scanf("%d",&n);
 printf("\n Enter the number of edges:\n");
 scanf("%d",&e);
 for(i=1;i<=n;i++)
 {
  for(j=1;j<=n;j++)
   p[i][j]=999;
 }

 for(i=1;i<=e;i++)
 {
  printf("\n Enter the end vertices of edge%d with its weight \n",i);
  scanf("%d%d%d",&u,&v,&w);
  p[u][v]=w;
 }

 printf("\n Matrix of input data:\n");
 for(i=1;i<=n;i++)
 {
  for(j=1;j<=n;j++)
   printf("%d \t",p[i][j]);
  printf("\n");
 }

 floyds(p,n);

 printf("\n Transitive closure:\n");
 for(i=1;i<=n;i++)
 {
  for(j=1;j<=n;j++)
   printf("%d \t",p[i][j]);
  printf("\n");
 }

 printf("\n The shortest paths are:\n");
 for(i=1;i<=n;i++)
  for(j=1;j<=n;j++)
  {
   if(i!=j)
    printf("\n <%d,%d>=%d",i,j,p[i][j]);
  }

}
```

OUTPUT:


1 contributor
19 lines (13 sloc) 210 Bytes
Raw
Blame
Enter the num of vertices: 4
Enter the cost matrix:
9999 9999 3 9999
2 9999 9999 9999
9999 7 9999 1
6 9999 9999 9999
All pair shortest path matrix is:
10 10 3 4
2 12 5 6
7 7 10 1
6 16 9 10

Experiment 14:

Find Minimum Cost Spanning Tree of a given undirected graph using Prim's algorithm.

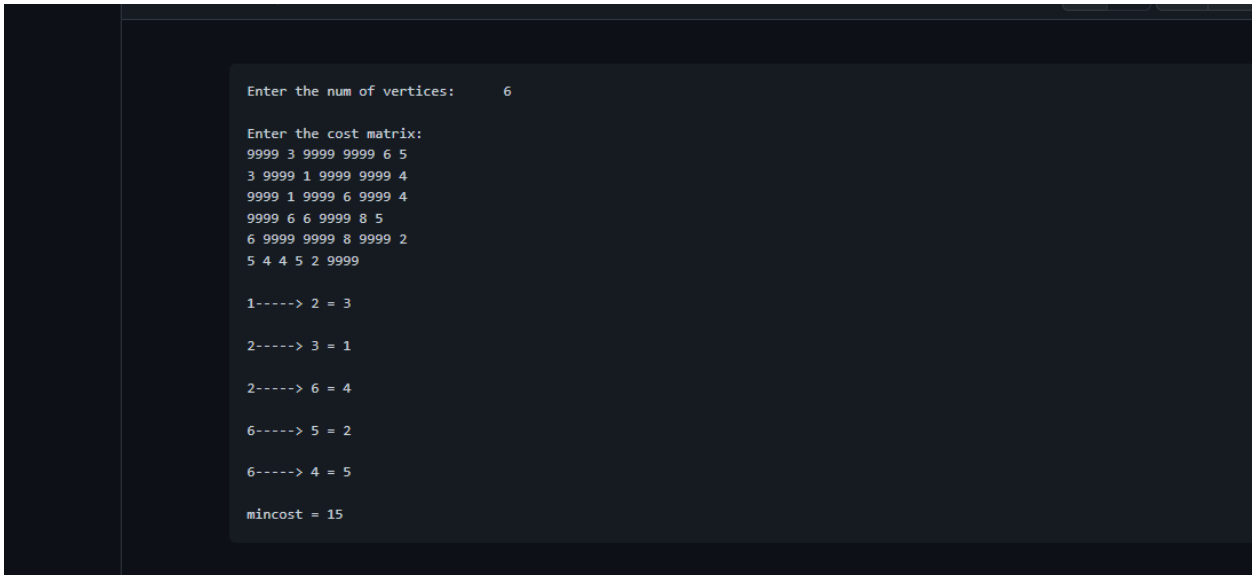
CODE:

1 contributor

51 lines (48 sloc) | 1.07 KB

```c
#include<stdio.h>
#include<conio.h>

void prims();
int c[10][10], n;

void main() {
    int i, j;
    printf("\n Enter the num of vertices: \t");
    scanf("%d", &n);
    printf("\n Enter the cost matrix: \n");
    for(i = 1; i <= n; i++) {
        for(j = 1; j <= n; j++) {
            scanf("%d", &c[i][j]);
        }
    }
    prims();
    getch();
}

void prims() {
    int i, j, u, v, min;
    int ne = 0, mincost = 0;
    int elec[10];
    for(i = 1; i <= n; i++) {
        elec[i] = 0;
    }
    elec[1] = 1;
    while(ne != n - 1) {
        min = 9999;
        for(i = 1; i <= n; i++) {
        for(j = 1; j <= n; j++) {
            if(elec[i] == 1) {
                if(c[i][j] < min) {
                    min=c[i][j];
                    u = i;
                    v = j;
                }
            }
        }
        }
        if(elec[v] != 1) {
            printf("\n%d-----> %d = %d\n", u, v, min);
            elec[v] = 1;
            ne = ne + 1;
            mincost = mincost + min;
        }
        c[u][v] = c[v][u] = 9999;
    }
    printf("\n mincost = %d", mincost);
}
```

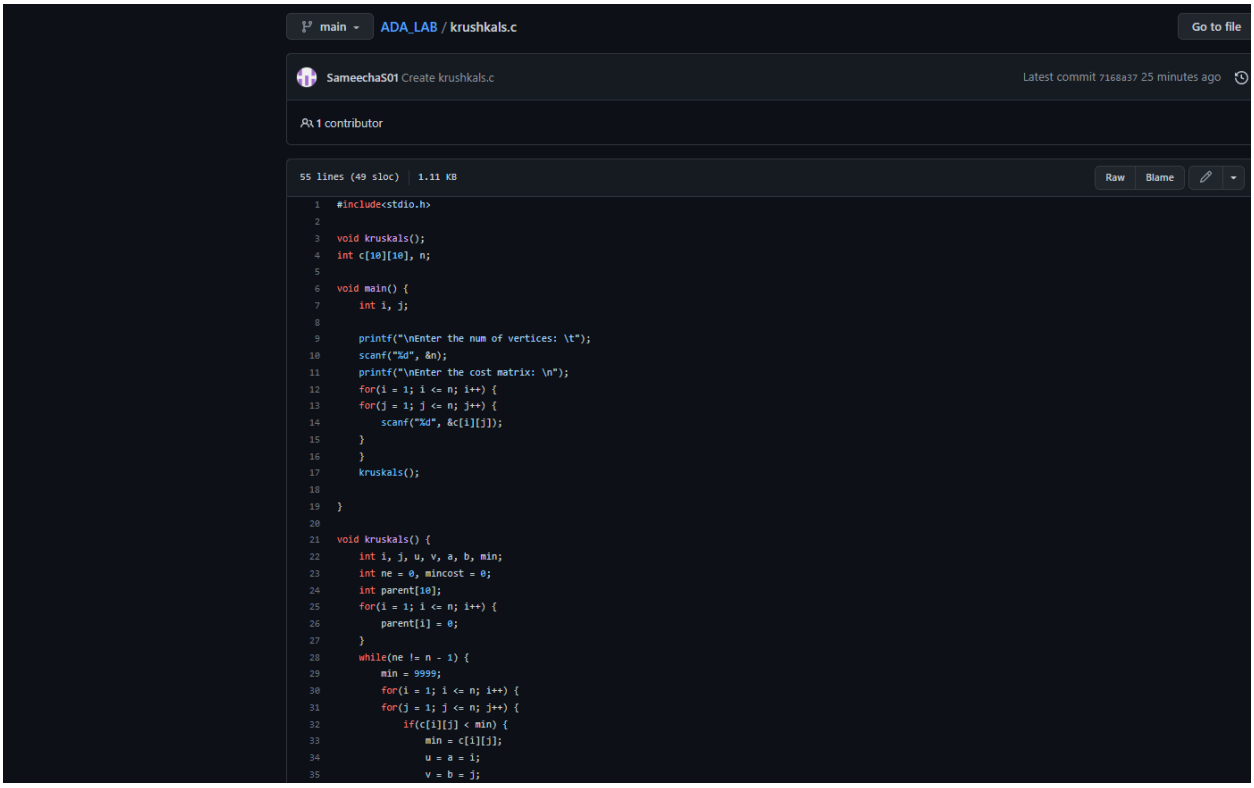
OUTPUT:


Enter the num of vertices: 6
Enter the cost matrix:
9999 3 9999 9999 6 5
3 9999 1 9999 9999 4
9999 1 9999 6 9999 4
9999 6 6 9999 8 5
6 9999 9999 8 9999 2
5 4 4 5 2 9999
1-----> 2 = 3
2-----> 3 = 1
2-----> 6 = 4
6-----> 5 = 2
6-----> 4 = 5
mincost = 15

Experiment 15:

Find Minimum Cost Spanning Tree of a given undirected graph using Kruskals algorithm.

CODE:



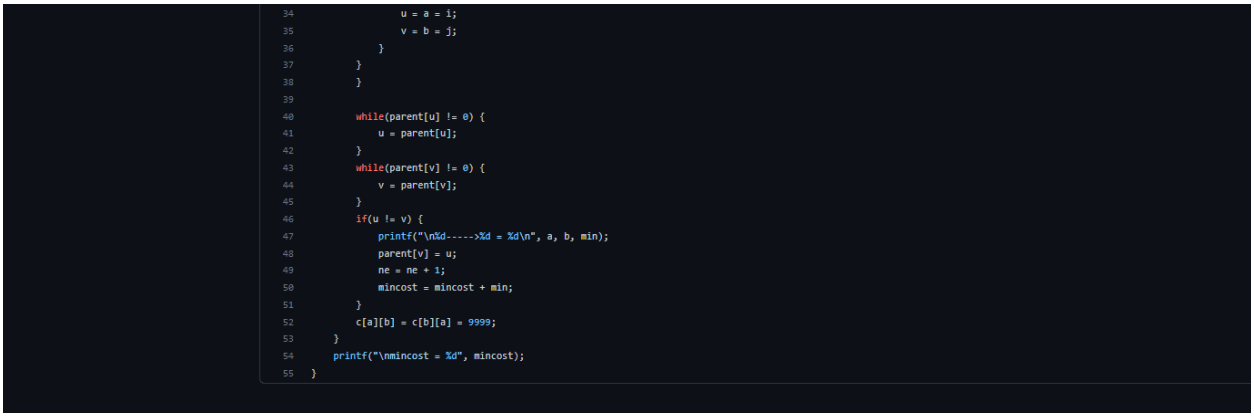
main ADA_LAB / krushkals.c Go to file

SameechaS01 Create krushkals.c Latest commit 7168a37 25 minutes ago

1 contributor

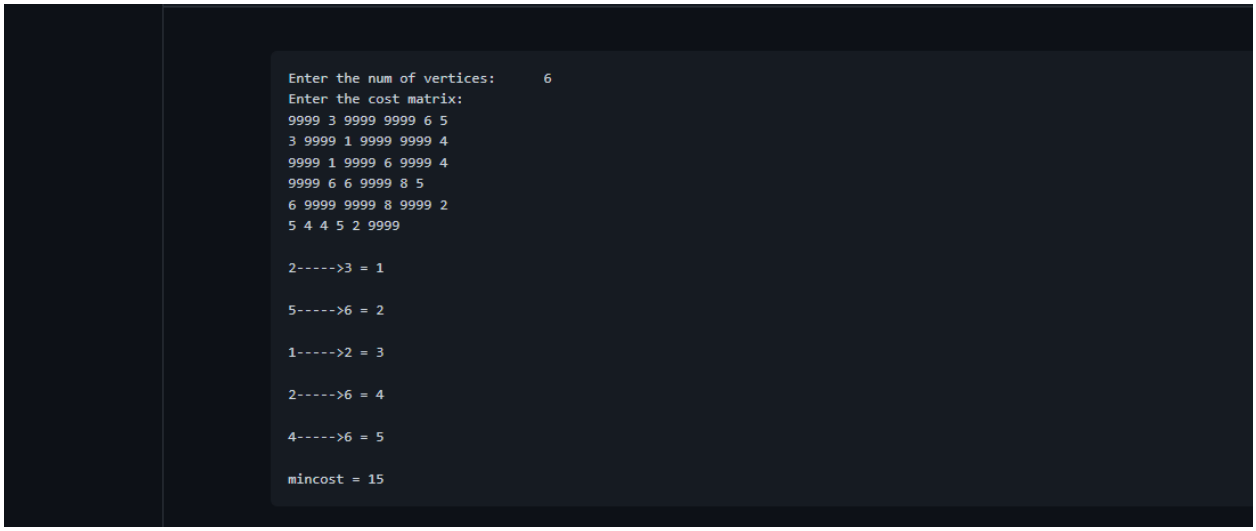
55 lines (49 sloc) 1.11 KB Raw Blame

```c
#include<stdio.h>

void kruskals();
int c[10][10], n;

void main() {
    int i, j;

    printf("\nEnter the num of vertices: \t");
    scanf("%d", &n);
    printf("\nEnter the cost matrix: \n");
    for(i = 1; i <= n; i++) {
    for(j = 1; j <= n; j++) {
        scanf("%d", &c[i][j]);
    }
    }
    kruskals();

}

void kruskals() {
    int i, j, u, v, a, b, min;
    int ne = 0, mincost = 0;
    int parent[10];
    for(i = 1; i <= n; i++) {
        parent[i] = 0;
    }
    while(ne != n - 1) {
        min = 9999;
        for(i = 1; i <= n; i++) {
        for(j = 1; j <= n; j++) {
            if(c[i][j] < min) {
                min = c[i][j];
                u = a = i;
                v = b = j;
```



```c
                u = a = i;
                v = b = j;
            }
        }
        }

        while(parent[u] != 0) {
            u = parent[u];
        }
        while(parent[v] != 0) {
            v = parent[v];
        }
        if(u != v) {
            printf("\n%d----->%d = %d\n", a, b, min);
            parent[v] = u;
            ne = ne + 1;
            mincost = mincost + min;
        }
        c[a][b] = c[b][a] = 9999;
    }
    printf("\nmincost = %d", mincost);
}
```

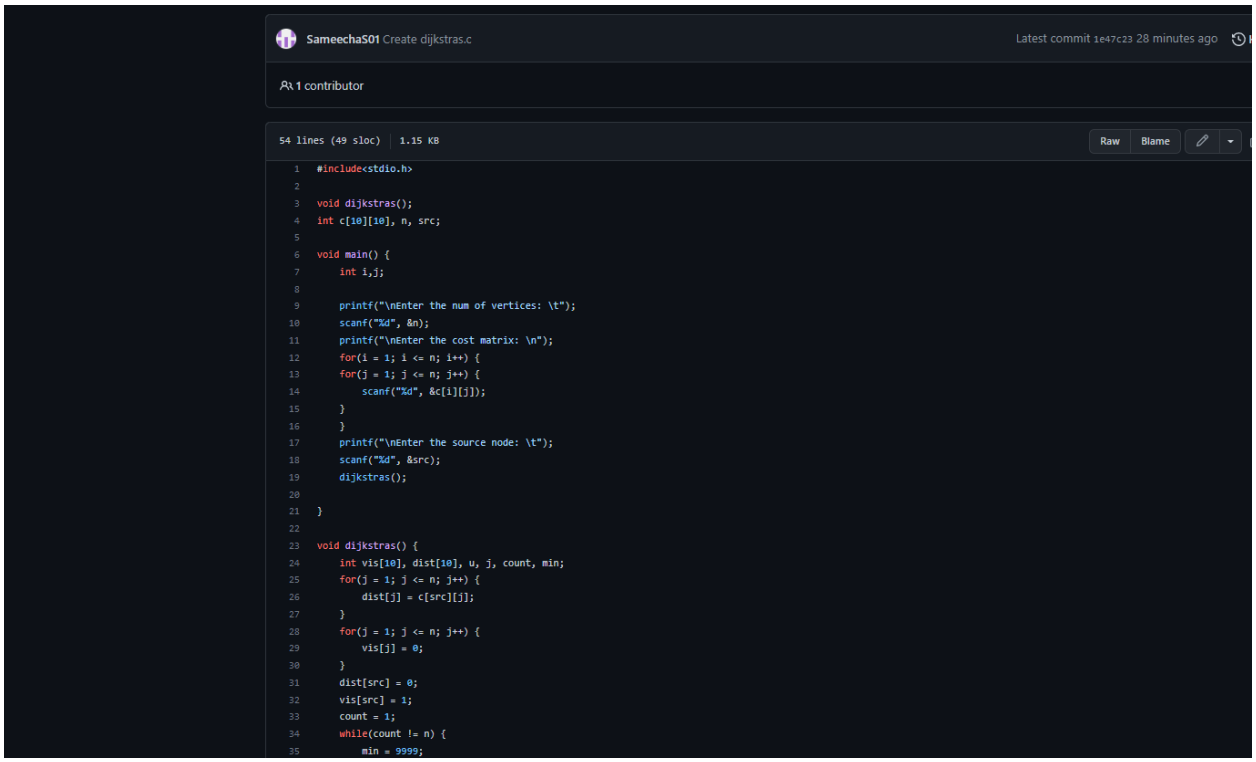
OUTPUT:


Enter the num of vertices: 6
Enter the cost matrix:
9999 3 9999 9999 6 5
3 9999 1 9999 9999 4
9999 1 9999 6 9999 4
9999 6 6 9999 8 5
6 9999 9999 8 9999 2
5 4 4 5 2 9999
2----->3 = 1
5----->6 = 2
1----->2 = 3
2----->6 = 4
4----->6 = 5
mincost = 15

Experiment 16:

From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm.

CODE:



SameechaS01 Create dijkstras.c — Latest commit 1e47c23 28 minutes ago

1 contributor

54 lines (49 sloc) | 1.15 KB — Raw | Blame

```c
#include<stdio.h>

void dijkstras();
int c[10][10], n, src;

void main() {
    int i,j;

    printf("\nEnter the num of vertices: \t");
    scanf("%d", &n);
    printf("\nEnter the cost matrix: \n");
    for(i = 1; i <= n; i++) {
    for(j = 1; j <= n; j++) {
        scanf("%d", &c[i][j]);
    }
    }
    printf("\nEnter the source node: \t");
    scanf("%d", &src);
    dijkstras();

}

void dijkstras() {
    int vis[10], dist[10], u, j, count, min;
    for(j = 1; j <= n; j++) {
        dist[j] = c[src][j];
    }
    for(j = 1; j <= n; j++) {
        vis[j] = 0;
    }
    dist[src] = 0;
    vis[src] = 1;
    count = 1;
    while(count != n) {
        min = 9999;
```

```c
    vis[src] = 1;
    count = 1;
    while(count != n) {
        min = 9999;
        for(j = 1; j <= n; j++) {
            if(dist[j] < min && vis[j] != 1) {
                min = dist[j];
                u = j;
            }
        }
        vis[u] = 1;
        count++;
        for(j = 1; j <= n; j++) {
            if(min + c[u][j] < dist[j] && vis[j] != 1) {
                dist[j] = min + c[u][j];
            }
        }
    }
    printf("\nThe shortest distance is: \n");
    for(j = 1; j <= n; j++) {
        printf("\n%d----->%d = %d", src, j, dist[j]);
    }
}
```

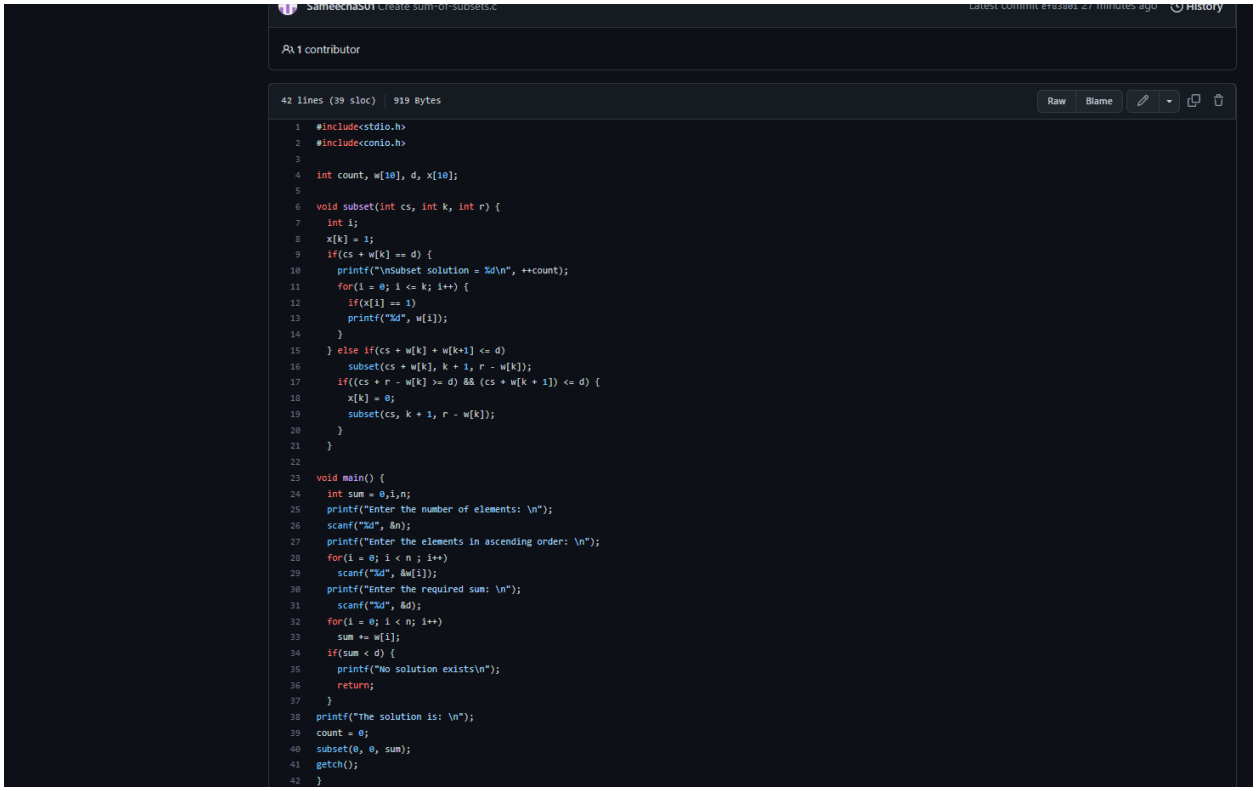
OUTPUT:


18 lines (16 sloc) | 269 Bytes
Raw
Blame
Enter the num of vertices: 5
Enter the cost matrix:
9999 3 9999 7 9999
3 9999 4 2 9999
9999 4 9999 5 6
7 2 5 9999 4
9999 9999 6 4 9999
Enter the source node: 1
The shortest distance is:
1----->1 = 0
1----->2 = 3
1----->3 = 7
1----->4 = 5
1----->5 = 9

Experiment 17:

Implement " Sum of Subsets" using Backtracking. " Sum of Subsets" problem: Find a subset of a given set S = {s1,s2,……,sn} of n positive integers whose sum is equal to a given positive integer d. For example, if S = {1,2,5,6,8} and d = 9 there are two solutions {1,2,6} and {1,8}. A suitable message is to be displayed if the given problem instance doesn't have a solution.

CODE:



1 contributor

42 lines (39 sloc) | 919 Bytes

Raw Blame

```c
#include<stdio.h>
#include<conio.h>

int count, w[10], d, x[10];

void subset(int cs, int k, int r) {
  int i;
  x[k] = 1;
  if(cs + w[k] == d) {
    printf("\nSubset solution = %d\n", ++count);
    for(i = 0; i <= k; i++) {
      if(x[i] == 1)
      printf("%d", w[i]);
    }
  } else if(cs + w[k] + w[k+1] <= d)
      subset(cs + w[k], k + 1, r - w[k]);
    if((cs + r - w[k] >= d) && (cs + w[k + 1]) <= d) {
      x[k] = 0;
      subset(cs, k + 1, r - w[k]);
    }
  }

void main() {
  int sum = 0,i,n;
  printf("Enter the number of elements: \n");
  scanf("%d", &n);
  printf("Enter the elements in ascending order: \n");
  for(i = 0; i < n ; i++)
    scanf("%d", &w[i]);
  printf("Enter the required sum: \n");
    scanf("%d", &d);
  for(i = 0; i < n; i++)
    sum += w[i];
  if(sum < d) {
    printf("No solution exists\n");
    return;
  }
printf("The solution is: \n");
count = 0;
subset(0, 0, sum);
getch();
}
```

1 contributor

11 lines (10 sloc) | 161 Bytes

```
Enter the number of elements:
5
Enter the elements in ascending order:
1 3 6 8 9 11
Enter the required sum:
The solution is:

Subset solution = 1
38
```

Experiment 18:

Implement " N-Queens Problem" using Backtracking.

CODE:

SameechaS01 Create n-queens.c — Latest commit 3b34565 29 minutes ago — History

1 contributor

40 lines (38 sloc) | 752 Bytes — Raw | Blame

```c
#include<stdio.h>
#include<conio.h>
void nqueens(int n) {
  int k, x[20], count = 0;
  k = 1;
  x[k] = 0;
  while(k != 0) {
    x[k]++;
    while(place(x, k) !=1 && x[k] <= n)
      x[k]++;
    if(x[k] <= n) {
      if(k == n) {
        printf("\nSolution is %d\n", ++count);
        printf("Queen\t\tPosition\n");
        for(k = 1; k <= n; k++)
          printf("%d\t\t%d\n", k, x[k]);
      } else {
          k++;
          x[k] = 0;
      }
    } else
        k--;
    }
  }

int place(int x[], int k) {
  int i;
  for(i = 1; i <= k - 1; i++) {
    if(i + x[i] == k + x[k] || i - x[i] == k - x[k] || x[i] == x[k])
    return 0;
  }
  return 1;
}

void main() {
  int n;
  printf("Enter the number of Queens\n");
  scanf("%d", &n);
  nqueens(n);
}
```

OUTPUT:

1 contributor

18 lines (16 sloc) | 139 Bytes

```
Enter the number of Queens
4

Solution is 1
Queen           Position
1               2
2               4
3               1
4               3

Solution is 2
Queen           Position
1               3
2               1
3               4
4               2
```