ADA
observation book

INDEX

NAME: Agneya DA STD.: SEC.: 4-A ROLL NO.:

S. No.	Date	Title	Page No.	Teacher's Sign / Remarks
1.	30-4-24	Exploring Leet code: Reversing Linked List	1	
2.	7-5-24	Decoding String	2-4	
3.	14-5-24	DFS topological sort & source removal topological sort	5-11	
4	4-6-24	Merge sort	12-14	
5	11-6-24	Quick Sort	15-16	
6	18-6-24	Bottom-up Heap construction	17-18	
7	25-6-24	Floyd-Warshall algorithm	19-20	
8	9-7-24	Prim's, Kruskal's, Dijkstra's algorithm	24-31	
9	16-7-24	Fractional knapsack	32-34	
10	16-7-24	N-Queen's problem	35-37	

Date 30-4-2024
Page 1

1 Reversing Linked List

Given the head of a singly linked list, reverse the list, and return the reversed list

```
= struct ListNode* reverseList (struct ListNode* head){
  struct ListNode *current, *next=NULL, *prev=NULL;
  current = head;
  while (current!=NULL){
     next = current->next;
     current->next = prev;
     prev = current;
     current = next;
  }
  head = prev;
  return head;
}
```

Decoding String

```
/
#define MAX_SIZE_STR_OUT 1000
char *helper(char *str_in, int *idx_in);
char * decode_string(char *s){
    if(!s)return NULL;
    char str_out[MAX_SIZE_STR_OUT];
    char * str_new;
    int idx_out=0, idx=0;
    while(s[idx]!='\0'){
        while((s[idx]>='a'&& s[idx]<='z') ||
    (s[idx]>='A' && s[idx]<='Z')) str_out[idx_out++]=s[idx++];
        str_new = helpers(s,&idx);
        strcpy(&str_out[idx_out], str_new);
        idx_out += strlen(str_new);
        free(str_new);
    }
    str_out[idx_out++]=0
    char *str_ret = (char*) malloc(idx_out);
    strcpy(str_ret, str_out);
    return str_ret;
}
char *helper(char *str_in, int *idx_in){
    if(!str_in) return NULL;
    char * str_new;
    int idx_out=0, number=0, new_number=0;

    while(str_in[*idx_in]!='\0'){
        if((str_in[*idx_in]>='A' && str_in[*idx]<='Z'))
            str_out[idx_out++]= str_in[*idx_in];
    else if (str_in[*idx_in]>='0' && str_in[*idx_in]<='9')
    {
```

```
new_number = 0;
if (!number){
  while (str_in[(*idx_in)]>='0' && str_in[*idx_in]<='9')
{
  new_number = 0;
  if (!number){
    while (str_in[(*idx_in)]>='0'&&str_in[(*idx_in]
    <='9')
    new_number = new_number*10 + (str_in[(*idx_in]-'0');
    }
    number = new_number;
  }
  else{
    str_new = helper(str_in, idx_in);
  strcpy(&str_out[idx_out], str_new);
  idx_out += strlen(str_new);
  (*idx_in)--;
  free(str_new);
  }
}
else if (str_in[*idx_in]==']'){
  for(int i=0; i<number; i++)
    memcpy(&str_out[i*idx_out], str_out, idx_out);
  idx_out = number*idx_out+1;
  idx_out--;
  str_out[idx_out] = 0;
  (*idx_in)++;
  return str_out;
  }
  (*idx_in)++;
}
str_out[idx_out] = 0;
return str_out; }
```

Output:

(1) s = "3[a]2[bc]"

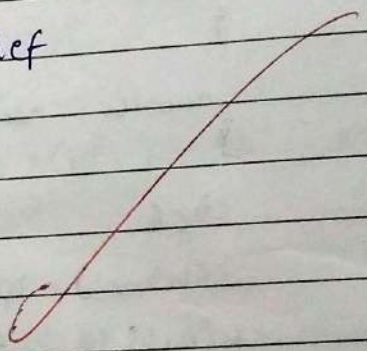
aaabcbc

(2) "3[a2[c]]"

accaccacc

(3) s = "2[abc]3[cd]ef"

abcabccdcdcdef



1) DFS Topological sort

```
#include <stdio.h>
#include<stdlib.h>
#include<stdbool.h>
struct Stack{
    int data;
    struct Stack* next;
};
struct Graph{
    int V;
    struct List *adj;
};
struct List{
    int data;
    struct List *next;
};
struct Stack *createStackNode(int data){
    struct Stack *node = (struct Stack *) malloc(sizeof
    (struct Stack));
    node->data = data;
    node->next = NULL;
    return node;
}
struct Graph *createGraph(int V){
    struct Graph* node = (struct Graph* ).malloc
                    (sizeof(struct Graph));
    node->adj = (struct List*) malloc(V* sizeof(struct List));
    node->V = V;
    for(int i=0; i<V; i++){
        node->adj[i].next=NULL;
    }
    return node;
```

```
}

void addEdge(struct Graph* graph, int v, int w){
    struct List* newNode = createListNode(w);
    newNode->next = graph->adj[v].next;
    graph->adj[v].next = newNode;
}
void topologicalSortUtil(struct Graph* graph, int v, bool
  visited[], struct stack** stack){
    visited[v] = true;
    struct List* current = graph->adj[v].next;
    while(current != NULL){
        int adjacentVertex = current->data;
        if(!visited[adjacentVertex]){
            topologicalSortUtil(graph, adjacentVertex,
            visited, stack);
        }
        current->next = *stack;
        *stack = newNode;

void topologicalSort(struct Graph* graph)
{
    struct stack* stack = NULL;
    bool* visited = (bool*) malloc(graph->V* sizeof(bool
    for(int i=0; i<graph->V; i++){
        visited[i] = false;
    }
    for(int i=0; i<graph->V; i++){
        if(!visited[i]){
            topologicalSortUtil(graph, i, visited, &stack);
        }
}
```

```
while (stack != NULL){
    printf("%d", stack->data);
    struct stack* temp = stack;
    stack = stack->next;
    free(temp);
}
free(visited);
free(graph->adj);
free(graph);
}

int main()
{
  struct Graph* g = createGraph(6);
  addEdge(g,5,2);
  addEdge(g,5,0);
  addEdge(g,4,0);
  addEdge(g,4,1);
  addEdge(g,2,3);
  addEdge(g,3,1);
  printf("Topological sorting order: ");
  topologicalSort(g);
  return 0;
}
```

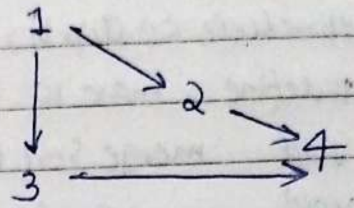
output:
Topological Sorting order: 542310

6/6/24

2) Source Removal topological sort

```
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main(){
  int n = 4, i = 0, j = 0, k = 0;
  int adjacency_matrix[][4] = {
     {0, 1, 1, 0},
     {0, 0, 0, 1},
     {0, 0, 0, 1},
     {0, 0, 0, 0}};
  int indegrees[n];
  bool visited[n];
  memset(&indegrees, 0, sizeof(*indegrees) * n);
  memset(&visited, false, sizeof(*visited) * n);
  for (i=0; i<n; i++){
    for (j=0; j<n; j++){
      indegrees[i] += adjacency_matrix[j][i];
    }
  }
  for (j=0; i<n; i++){
    for (j=0; j<n; j++){
      if (!indegrees[j] && !visited[j]){
        visited[j] = true;
        printf("%d", j+1);
        for (k=0; k<n; k++){
          if (adjacency_matrix[j][k]){
            adjacency_matrix[j][k] = 0;
            indegrees[k]--;
          }
        }
```

```
            break;
        }
      }
    }
    return 0;
}
```



Output:

1 2 3 4

G(P)sir

Merge Sort

```
#include <stdio.h>
#include <stdlib.h>
#define max 10
void mergeSort(int A[], int);
void merge(int [], int [], int [], int, int);

void main(){
   int n, i=0, A[max];
   printf("Enter the value of n: ");
   scanf("%d",&n);
   printf("Enter the array to be sorted: \n");
   for(i=0; i<n; i++){
      scanf("%d", A[i]);
   }
   mergeSort(A, n);
   printf("Sorted array is: ");
   for(i=0; i<n; i++){
      printf("%d", A[i]);
   }
}

void mergeSort(int A[], int n){
  if(n>1){
      int mid = n/2;
      int *B = (int*)malloc(mid * sizeof(int));
      int *C = (int*)malloc((n-mid) * sizeof(int));
      for(int i=0; i<mid; i++){
         B[i] = A[i];
      }
      for(int i= mid; i<n; i++){
         C[i-mid] = A[i];
      }
```

not yet completed

```
    mergeSort(B, mid);
    mergeSort(C, n-mid);
    merge(B, C, A, mid, n-mid);
    free(B);
    free(C);
  }
}

void merge(int B[], int C[], int A[], int p, int q){
  int i=0, j=0, k=0;
  while((i<p)&&(j<q)){
    if(B[i]<C[j]){
      A[k]=B[i];
      i++;
    }
    else{
      A[k]=C[j];
      j++;
    }
    k++;
  }
  if(i==p){
    while(j<q){
      A[k++]=C[j++];
    }
  }
  else{
    while(i<p){
      A[k++]=B[i++];
    }
  }
}
```

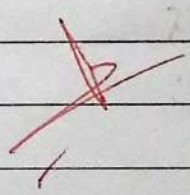
Outputs:

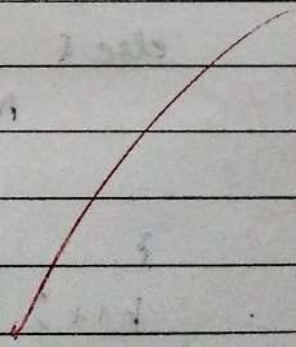
Enter the value of n: 10

Enter the array to be sorted:

9 1 2 3 6 7 4 8 10 5

Sorted array is: 1 2 3 4 5 6 7 8 9 10





Date 11/6/2024
Page 13

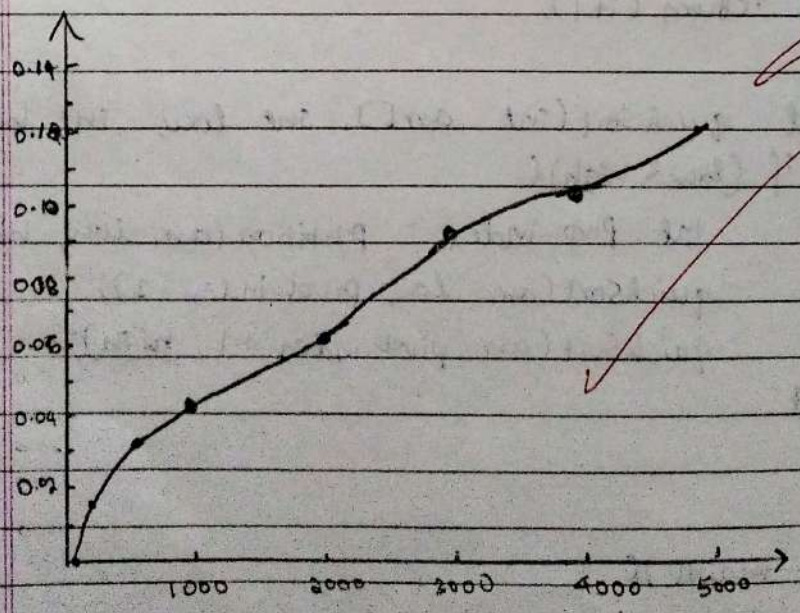
Quick Sort

```
#include<stdio.h>
#include<time.h>
#define MAX 10000
void swap(int *n1, int *n2){
    int temp;
    temp = *n1;
    *n1 = *n2;
    *n2 = temp;
}
int partition(int arr[], int low, int high){
   int pivot = arr[high];
   int i = low-1, j;
   for(j=low; j<high; j++){
       if(arr[j]<pivot){
         i++;
         swap(&arr[i], &arr[j]);
       }
   }
   swap(&arr[i+1], &arr[high]);
   return (i+1);
}
void quicksort(int arr[], int low, int high){
  if (low<high){
      int pivot_index = partition(arr, low, high);
      quicksort(arr, low, pivot_index-1);
      quicksort(arr, pivot_index+1, high);
  }
}

int main(){
  clock_t t;
  int n, arr[MAX];
```

```
printf("Enter the array elements: ");
for(int i=0; i<n; i++){
    scanf("%d", &arr[i]);
}
t = clock();
quickSort(arr, 0, n-1);
for(int i=0; i<n; i++)
printf("%d", arr[i]);
t = clock() - t;
double time_taken = ((double)t)/CLOCKS_PER_SEC;
printf("The program took %f seconds to execute",
time_taken);
return 0;
}
```

Output:

Enter input size: 10
Enter the array elements: 9 3 5 4 16 10 2 7 8
Sorted array:
1 2 3 4 5 6 7 8 9 10



Bottom-up Heap construction

Date 18-6-2024
Page 17

```
#include<stdio.h>
#define MAX 20
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 l = 2*i+1;
    int r = 2*i+2;
    if(l<N && arr[l] > arr[largest]){
        largest = l;
    }
    if(r<N && arr[r] > arr[largest]){
        largest = r;
    }
    if(i!= largest){
        swap(&arr[i], &arr[largest]);
        heapify(arr, N, largest);
    }
}

void buildHeap(int arr[], int N){
    int startIdx = (N/2) - 1;
    for(int i= startIdx; i>=0; i--){
        heapify(arr, N, i);
    }
}

void printHeap(int arr[], int N){
    printf("Array representation of Heap is: \n");
```

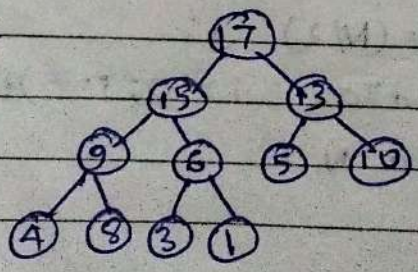
```
    for (int i=0; i<N; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

int main(){
    int N, arr[MAX];
    printf("Enter the value of N: ");
    scanf("%d", &N);
    printf("Enter the array: ");
    for(int i=0; i<N; i++){
        scanf("%d", &arr[i]);
    }
    buildHeap(arr, N);
    printHeap(arr, N);
    return 0;
}
```

Output:

Enter the value of N: 11
Enter the array: 1 3 5 4 6 13 10 9 8 15 17
Array representation of Heap is:
17 15 13 9 6 5 10 4 8 3 1

18/6/24

Heap:



Floyd - Warshall Algorithm

Date 25-6-2024
Page 19

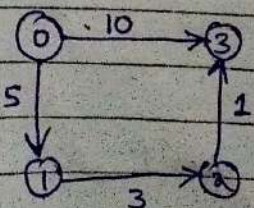
```
#include<stdio.h>
#include<limits.h>
int INF = 1e5;
void printSolution(int V, int dist[V][V]){
    printf("The following matrix shows the shortest
    distances between every pair of vertices (-1: infinity)
    :\n");
    for(int i=0; i<V; i++){
        for(int j=0; j<V; j++){
            if(dist[i][j]==INF)
                printf("-1");
            else
                printf("%d ", dist[i][j]);
        }
        printf("\n");
    }
}

void floydWarshall(int V, int graph[V][V]){
    int dist[V][V], i, j, k;
    for(i=0; i<V; i++)
        for(j=0; j<V; j++)
            dist[i][j] = graph[i][j];
    for(k=0; k<V; k++){ // Middle vertex
        for(i=0; i<V; i++){ // start vertex
            for(j=0; j<V; j++){ // destination vertex
                if(dist[i][k] + dist[k][j] < dist[i][j])
                    dist[i][j] = dist[i][k] + dist[k][j];
            }
        }
    }
    printSolution(V, dist);
}
```

```
int main(){
    int V;
    printf("Enter no of vertices: ");
    scanf("%d", &V);
    int graph[V][V];
    printf("Enter weighted adjacency matrix (Enter -1
    for inf): \n");
    for(int i=0; i<V; i++){
        for(int j=0; j<V; j++){
            scanf("%d", &graph[i][j]);
            if(graph[i][j]==-1) graph[i][j]=INF;
        }
    }
    floydWarshall(V, graph);
    return 0;
}
```

Output: Enter no. of vertices: 4
Enter weighted adjacency matrix (Enter -1 for Inf):
0 5 gjgg 10
-1 0 3 gjg
-1 -1 0 1
-1 -1 -1 0

The following matrix shows the shortest distances between every pair of vertices (-1 = Infinity):
0 5 8 9 output
-1 0 3 4
-1 -1 0 1
-1 -1 -1 0

Graph:



25/6/2024

Knapsack using dynamic programming

Date 3/7/24
Page 91

```
#include <stdbool.h>
#include <stdio.h>
#define MAX 10
int max(int, int);
void main(){
    int w[MAX], v[MAX];
    int val[MAX][MAX];
    int n, m, j=0;
    printf("Enter the number of items: ");
    scanf("%d", &n);
    printf("Enter maximum knapsack capacity: ");
    scanf("%d", &m);
bool keep[n+1][n+1];
    w[0] = 0;
    v[0] = 0;
    printf("Enter the weight of each item in
    sequence: ");
    for(i=1; i<=n; i++){
        scanf("%d", &v[i]);
    }
    for(i=1; i<=n; i++)
        for(j=0; j<=m; j++){
            if(i==0 || j==0){
                val[i][j] = 0;
            }
            else{
                if(w[i]>j){
                    val[i][j] = val[i-1][j];
                    keep[i][j] = false;
                }
                else{ bool keep[i][j];
                    val[i][j] = max(val[i-1][j], val[i-1][j-w[i]]
                                    + v[i]);
                    keep[i][j] = include;
                }
            }
        }
```

```
for (i=0; i<= n; i++){
    for (j=0; j<=m; j++){
        printf ("%d", val[i][j]);
    }
    printf("\n");
}
int k=0;
int ks[n];
for (int i=n; i>0 && j>0; i--){
    if (keep[i][j]){
        ks[k++]=i;
        j -= w[i];
    }
}
printf("Selected items: ");
for(int i=0; i<k; i++){
    printf("%d ", ks[i]);
}
printf("\n");
printf("\n Maximum value that can be stored in
this knapsack is: %d", val[n][m]);
}
int max(int a, int b){
    if (a>=b) return a;
    if (a<b) return b;
}
```

Output:

Enter number of items: 4
Enter ~~number of~~ maximum capacity: 5
Enter weight of each item in sequence: 2 1 3 2
Enter the value of each item in sequence: 12 10 20 15

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

Selected items: 1 2 4

Maximum value that can be stored in this knapsack is: 37

2/7/24

Prim's Algorithm

1.
```
#include <limits.h>
#include <stdbool.h>
#include <stdio.h>
#define V 5
int minKey(int key[], bool mstSet[]) {
    int min = INT_MAX, min_index;
    // INT_MAX = ∞
    for (int v = 0; v < V; v++)
        if (mstSet[v] == false && key[v] < min)
            min = key[v], min_index = v;
    return min_index;
}

int printMST(int parent[], int graph[V][V]) {
    printf("Edge \t Weight\n");
    for (int i = 1; i < V; i++)
        printf("%d - %d \t %d \n", parent[i], i,
        graph[i][parent[i]]);
}

void primMST(int graph[V][V]) {
    int parent[V];
    int key[V];
    bool mstSet[V];
    for (int i = 0; i < V; i++)
        key[i] = INT_MAX, mstSet[i] = false;
    key[0] = 0;
    parent[0] = -1;
    for (int count = 0; count < V-1; count++) {
        int u = minKey(key, mstSet);
        mstSet[u] = true;
        for (int v = 0; v < V; v++)
```

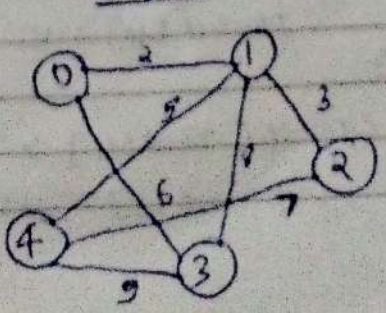
```
        if (graph[u][v] && mstSet[v] == false &&
            graph[u][v] < key[v])
            parent[v] = u, key[v] = graph[u][v];
    }
    printMST(parent, graph);
}

int main(){
    printf("Enter number of vertices: ");
    scanf("%d", &V);
    printf int graph[V][V];
    printf("Enter adjacency matrix: ");
    for(int i=0; i<V; i++){
        for(int j=0; j<V; j++){
            scanf("%d", &graph[i][j]);
        }
    }
    printMST(graph);
    return 0;
}
```

Output: Enter number of vertices: 5
Enter adjacency matrix:

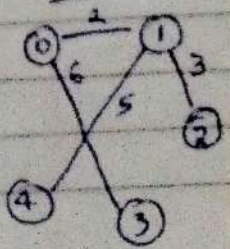
0	2	0	6	0
2	0	3	8	5
0	3	0	0	7
6	8	0	0	9
0	5	7	9	0

Edge	Weight
0 - 1	2
1 - 2	3
0 - 3	6
1 - 4	5

Graph



MST



Kruskal's Algorithm

```
#include <stdio.h>
#include <stdlib.h>
int comparator(const void* p1, const void* p2)
{
    const int(*x)[3] = p1;
    const int(*y)[3] = p2;
    return (*x)[2] - (*y)[2];
}

void makeSet(int parent[], int rank[], int n)
{
    for(int i=0; i<n; i++){
        parent[i] = i;
        rank[i] = 0;
    }
}

int findParent(int parent[], int component){
    if(parent[component] == component)
        return component;
    return parent[component] = findParent(parent,
    parent[component]);
}

void unionSet(int u, int v, int parent[], int rank[],
    int n){
    u = findParent(parent, u);
    v = findParent(parent, v);
    if(rank[u] < rank[v]){
        parent[u] = v;
    }
    else if(rank[u] > rank[v]){
        parent[v] = u;
```

```
    }
    else {
        parent[v] = u;
        rank[u]++;
    }
}
void kruskalAlgo(int n, int edge[n][3]){
    qSort(edge, n, sizeof(edge[0]), comparator);
    int parent[n];
    int rank[n];
    makeSet(parent, rank, n);
    int minCost = 0;
    printf("Following are the edges in the
    constructed MST\n");
    for(int i=0; i<n; i++){
        int v1 = findParent(parent, edge[i][0]);
        int v2 = findParent(parent, edge[i][1]);
        int wt = edge[i][2];
        if(v1 != v2){
            unionSet(v1, v2, parent, rank, n);
            minCost += wt;
            printf("%d -- %d == %d\n", edge[i][0],
              edge[i][1], wt);
        }
    }
    printf("Minimum Cost Spanning Tree: %d\n",
    minCost);
}

void main(){
    int v;
    printf("Enter number of edges");
```

```
scanf("%d",&V);
int edge[V][3];
printf("Enter start, end and weight:");
for(int i=0; i<V; i++){
    for(int j=0; j<3; j++){
        scanf("%d",&edge[i][j]);
    }
}
kruskalAlgo(V, edge);
return 0;
}
```

Output: Enter number of edges 10
Enter start, end and weight: 0 1 3 0 5 5 0 4 1
1 2 1 1 5 4 2 5 4 2 3 6 5 3 5 5 4 2 3 4 3

Following are the edges in the constructed MST

1 -- 2 == 1

5 -- 4 == 2

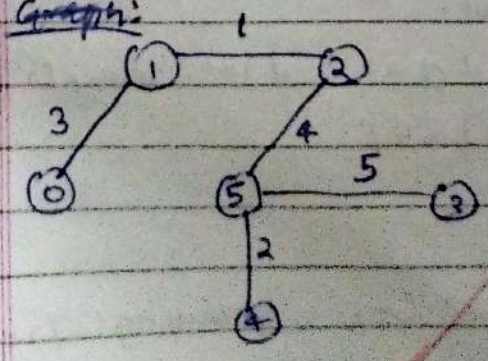
0 -- 1 == 3

2 -- 5 == 4

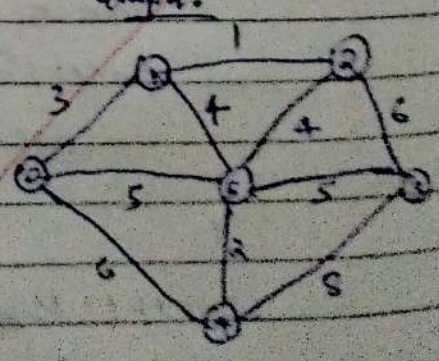
5 -- 3 == 5

Minimum cost spanning tree = 15

MST Graph:



Graph:



Dijkstra's algorithm

Date 9-7-24
Page 29

3.
```
#include <limits.h>
#include <stdbool.h>
#include <stdio.h>
int V;
int minDistance (int dist[], bool sptSet[])
{
  int min = INT_MAX, min_index;
  for(int v = 0; v < V; v++) {
    if (sptSet[v] == false && dist[v] <= min)
      min = dist[v], min_index = v;
  return min_index;
}

void printSolution (int dist[]){
  printf("Vertex \t\t Distance from Source\n");
  for(int i=0; i < V; i++)
    printf("%d \t\t\t\t %d\n", i, dist[i]);
}

void dijkstra (int graph[V][V], int src){
  int dist[V];
  bool sptSet[V];
  for (int i = 0; i< V; i++)
    dist[i] = INT_MAX, sptSet[i] = false;
  dist[src] = 0;
  for (int count = 0; count < V-1; count++) {
    int u = minDistance (dist, sptSet);
    SptSet[u] = true;
    for(int v=0; v < V; v++)
      if(!sptSet[v] && graph[u][v]
      && dist[u] != INT_MAX &&
      dist[u] + graph[u][v] < dist[v])
```

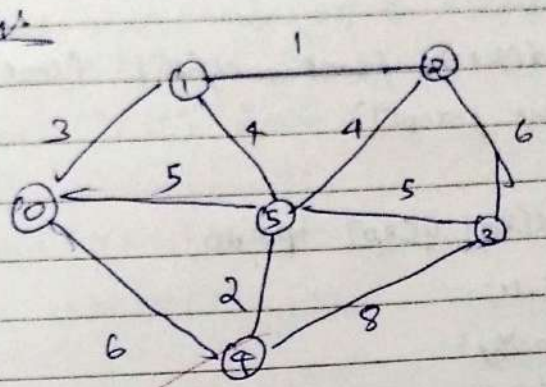
```
    dist[v] = dist[u] + graph[u][v];
  }
  printSolution(dist);
}
int main(){
  printf("Enter number of vertices: ");
  scanf("%d", &V);
  int graph[V][V];
  printf("Enter adjacency matrix:\n");
  for(int i=0; i<V; i++){
    for(int j=0; j<V; j++)
      scanf("%d", &graph[i][j]);
  }
  dijkstra(graph, 0);
  return 0;
}
```

Output: Enter number of vertices: 6
Enter adjacency matrix:

0	3	0	0	6	5
3	0	1	0	0	4
0	1	0	6	0	4
0	0	6	0	8	5
6	0	0	8	0	2
5	4	4	5	2	0

Vertex	Distance from Source
0	0
1	3
2	4
3	10
4	6
5	5

Graph:-



W.

9/7/24

Fractional Knapsack

```
#include<stdio.h>
void knapsack(int n, float weight[], float profit[], float capacity, int process[])
{
    float x[20], y[20], tp=0.0;
    int i,j,u;
    u = capacity;
    for(i=0;i<n;i++)
        x[i] = 0.0;
    for(i=0;i<n;i++){
        if (weight[i]>u)
            break;
        else{
            x[i] = 1.0;
            tp = tp+ profit[i];
            u = u-weight[i];
        }
    }
    if (i<n)
        x[i] = u/weight[i];
    tp = tp + (x[i]*profit[i]);
    printf("\n The result vector is :- ");
    for(i=0;i<n;i++) printf("%d\t",process[i]);
    printf("\n");
    for(i=0;i<n;i++)
        printf("%f\t",x[i]);
    printf("\nMaximum profit is:- %f",tp);
}

int main(){
    float weight[20], profit[20], capacity;
    int num,i,j;
```

```
float ratio[20], temp;
printf("Enter number of objects: ");
scanf("%d", &num);
int process[num];
for(i=0; i<num; i++) {
    process[i] = i+1;
}
printf("\n Enter the wts and profits of each
 objects: ");
  scanf for(i=0; i<num; i++)
            scanf("%f %f", &weight[i], &profit[i]);
printf("\n Enter the capacity of knapsack: ");
scanf("%f", &capacity);
for(i=0; i<num; i++) {
   ratio[i] = profit[i] / weight[i];
}
for(i=0; i<num; i++) {
   for(j=i+1; j<num; j++) {
     if(ratio[i] < ratio[j]) {
       temp = ratio[j];
       ratio[j] = ratio[i];
       ratio[i] = temp;
       temp = weight[j];
       weight[j] = weight[i];
       weight[i] = temp;
       temp = profit[j];
       profit[j] = profit[i];
       profit[i] = temp;
       temp = process[i];
       process[i] = process[j];
       process[j] = temp;
     }
   }
}
```

```
    knapsack (num, weight, profit, capacity, proces);
    return 0;
}
```

Output:

```
Enter number of objects: 7
Enter wts and profits:
1 10
3 15
5 7
4 8
1 9
3 4
2 10
Enter the capacity of knapsack: 15
The result vector is:
1   5   2   7   4   3   6
1   1   1   1   1   0.8 0
```

N - Queens

Date: 16-7-24
Page: 35

```
#define N 4
#include <stdio.h>
#include <stdbool.h>
void ps(int b[N][N]){
    for(i=0; i<N; i++){
      for(j=0; j<N; j++){
          if (b[i][j],
              printf("Q");
          else
              printf(". ");
          }
          printf("\n");
      }
}

bool issafe(int b[N][N], int r, int c){
    int i,j;
    for(i=0; i<c; i++)
      if(b[r][i])
          return false;
    for(i=r, j=c; i>=0 && j>=0; i--, j--)
      if [b[i][j])
          return false;
    for(i=0 ,j==c; j>=0 && i<N ; i++, j--){
        if(b[i][j])
              return false;

    return true;
}
```

```
bool solve (int b[N][N], int c){
  if (c >= N)
      return true
      return false;
  for (int i=0; i<N; i++){
    if (issafe(b, i, c)){
      b[i][c] = 1;
      if (solve(b, c+1))
          return true;
      b[i][c]=0;
    }
  }
  return false;
}

bool NQ(){
  int b[N][N] = {{0,0,0,0},
                 {0,0,0,0},
                 {0,0,0,0},
                 {0,0,0,0}};
  if(solve(b,0)==false){
    printf("Solution does not exist");
    return false;
  }
  ps(b);
  return true;
}
void main(){
  NQ();
}
```

Output :-

. . Q .
Q . . .
. . . Q
. Q . .