Lab - 1

02-05-24

(Q) Leet code - 448 - Find all numbers disappeared in an array.

```
int *findDisappearedNumbers(int *nums, int numsSize,
                            int *returnSize) {

    int temp = 0;

for (int i=0; i < numsSize; ++i) {
        temp = abs(nums[i]) - 1;
        nums[temp] = abs(nums[temp] * -1);
}

int insert_index = 0;
*returnSize = 0;

for (int i=0; i<numsSize; ++i) {
    if (nums[i] > 0) {
        ++*returnSize;
        nums[insert_index++] = i+1;
    }
}

return nums;
}
```

9/5/24

Output :-

Case 1

In : [4,3,2,7,8,2,3,1]

Out : [5,6]

Case 2

In : [1,1]

Out : [2]

Lab - 2 09-05-24

(Q) Leetcode - 103 - Binary Tree Zigzag level Order Traversal.

```
int **zigzagLevelOrder(struct TreeNode* root, int* returnSize,
                       int** returnColumnSizes) {
    struct Queue q;
    struct Result r;
    int level = 0;
    struct Pair c = {root, level};
    memset(&q, 0, sizeof(q));
    memset(&r, 0, sizeof(r));

    if (root) {
        qpush(&q, c);
    }
    while (!qempty(&q)) {
        struct Pair cur = qtop(&q);
        if (cur.level != level) {
            push(&r);
        }
        add(&r, cur.node->value);
        if (cur.node->left) {
            struct Pair n = {cur.node->left,
                             cur.level + 1};
            q.push(&q, n);
        }
        if (cur.node->right) {
            struct Pair n = {cur.node->right,
                             cur.level + 1};
            q.push(&q, n);
        }
        level = cur.level;
    }
```

```
flush (&r)

* *returnSize = r.size;
* *returnColumnSizes = r.cols;

for (i=1; i<r.size; i+=2) {
    reverse (r.r[i], r.cols[i]);
}

return r.r;
}
```

Output:-

Case 1 :- In: [3,9,20,null,null,15,7]

Out: [[3],[20,9],[15,7]]

Case 2 :- In: [[1]]

Out: [[1]]

Case 3 :- In: []

Out: []

9/5/24

(Q) Leet Code – 897 – Increasing Order Search Tree

```
Struct TreeNode* increasingBST (Struct TreeNode* root) {
    int arr[1000], i=0;
    inOrder (root, arr, &i);
    Struct TreeNode* node = NULL;
    for (int k=0; k<i; k++)
        node = newTree (node, arr[k]);
    return node;
}

Struct TreeNode* newTree (Struct TreeNode* root,
                          int val) {
    if (root == NULL)
        return newNode (val);
    if (val < root->val)
        root->left = newTree (root->left, val);
    else
        root->right = newTree (root->right, val);
    return root;
}

void inOrder (Struct TreeNode* root, int arr[],
              int *i) {
    if (root == NULL)
        return;
```

```
inOrder(root->left, arr, i);
arr[(i)++] = root->val;
inOrder(root->right, arr, i);
}
```

Output :-

Case 1 :-

root := [5, 3, 6, 2, 4, null, 8, 1, null, null, null, 7, 9]

Out := [1, null, 2, null, 3, null, 4, null, 5, null, 6, null, 7, null, 8, null, 9]

Case 2 :-

root := [5, 1, 7]

Out := [1, null, 5, null, 7]

23/5/24

(Q) "C" program to implement Topological Sort Order.

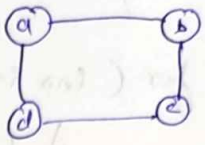
```
#include <stdio.h>
#include <stdlib.h>
#define MAX 100

int adj[MAX][MAX];
int visited[MAX];
int stack[MAX];
int top = -1;
int n;

void push(int v) {
    stack[++top] = v;
}

int pop() {
    return stack[top--];
}

void dfs(int v) {
    visited[v] = 1;
    for(int i=0; i<n; i++) {
        if(adj[v][i] && !visited[i]) {
            dfs(i);
        }
    }
    push(v);
}
```

```
void topological Sort (){
    for (int i = 0; i<n ; i++) {
        visited[i] = 0;
    }
    for ( int i=0 ; i<n ; i++) {
        if (!visited[i]){
            dfs(i);
        }
    }
    while (top != -1){
        printf ("%d", pop());
    }
    printf ("\n");
}
int main(){
    int edges, start, end;
    printf (" No of vertex :");
    Scanf ("%d", &n);
    printf(" No of Edges :");
    Scanf ("%d", &edges);
    for (int i=0; i<edges ; i++) {
        printf (" start & end :");
        Scanf ("%d %d", &start, &end);
        adj[start][end] = 1;
    }
}
```

Output:-

No. of vertices : 4

No. of edges = 4



Start & end : 1 2

Start & end : 2 3

Start & end : 3 4

Start & end : 4 1

Topological sort : 1 2 3 0

(9)

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

int st[100];
int top = -1;

void degree(int adj[][20], int n) {

    int indegree[20];
    int sum = 0;

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

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

            sum = sum + adj[i][j];

        }

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

    if(indegree[i]==0){

        top++;
        st[top] = i;
```

```
int main() {

int n;
printf("No of nodes");
scanf("%d", &n);

int adj[20][20];
printf("Adjacency matrix");
scanf("
for(int i=0; i<n; i++) {
    for(int j=0; j<n; j++) {
        scanf("%d", &adj[i][j]);

    }

}

printf("Topological order:");
degree(adj, n);

return 0;

}
```

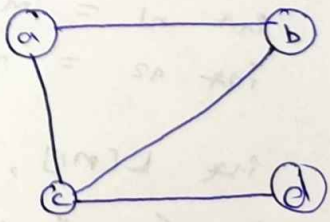
23/5/24

Output :-

No. of Nodes : 4

Enter Matrix :-

0 1 1 0
0 0 1 0
0 0 0 1
0 0 0 0



Topological order :- 0 1 2 3

(Q) Selection & Merge Sort

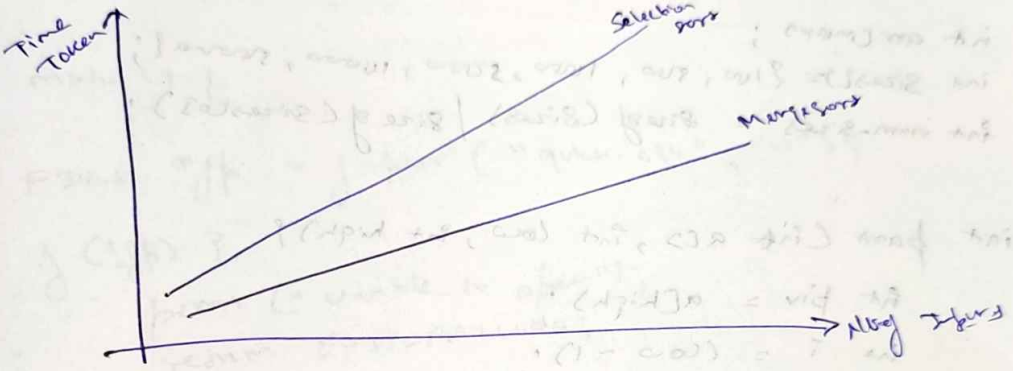
```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

void Selection_sort (int arr[], int n) {
    for (int i=0; i<n; i++) {
        int min = i;
        for (int j = i+1; j<n; j++) {
            if (arr[j] < arr[min]) {
                min = j;
            }
        }
        int temp = arr[min];
        arr[min] = arr[i];
        arr[i] = temp;
    }
}

void merge (int arr[], int l, int m, int r){
    int n1 = m-l+1;
    int n2 = r-m;

    int L[n1], R[n2];
    for (int i=0; i<n1; i++)
        L[i] = arr[l+i];
    for (int i=0; i<n2; i++)
        R[i] = arr[m+1+i];
```

```
int i=0, j=0, k=l;
while (i<n1 && j<n2) {
    if (L[i] <= R[j]) {
        arr[k++] = L[i++];
    } else {
        arr[k++] = R[j++];
    }
}
while (i<n1) {
    arr[k++] = L[i++];
}
while (j<n2) {
    arr[k++] = R[j++];
}
}

void merge_sort (int arr[], int l, int r) {
    if (l<r) {
        int m = l + (r-1)/2;
        merge_sort (arr, l, m);
        merge_sort (arr, m+1, r);
        merge (arr, l, m, r);
    }
}

void random (int arr[], int n) {
    for (int i=0; i<n; i++) {
        arr[i] = rand() % (n*10);
    }
}
```

```
double time(void (*sort)(int[], int), int arr[], int n) {
    clock_t start = clock();
    sort(arr, n);
    clock_t end = clock();
    return ((double)(end - start)) / CLOCKS_PER_SEC;
}

int main() {
    int inputs[] = {100, 500, 1000, 5000, 10000, 50000};
    int num = sizeof(inputs) / sizeof(inputs[0]);

    FILE *fp = fopen("sorting_times.csv", "w");
    if (!fp) {
        printf("Error\n");
        return 0;
    }
    fprintf(fp, "Input, Selection, Merge\n");

    for (int i = 0; i < num; i++) {
        int size = inputs[i];
        int *data = malloc(size * sizeof(int));
        int *data_copy = malloc(size * sizeof(int));
        for (int j = 0; j < size; j++) {
            data_copy[j] = data[j];
        }
        free(data);
        free(data_copy);
    }
    fclose(fp);
    return 0;
}
```

Output 1 - 5

Input Size	Selection Sort	Merge Sort
1000	0.000001	0.000002
5000	0.000005	0.000008
10000	0.000010	0.000014
50000	0.000050	0.000070
100000	0.000100	0.000150



30/5/24

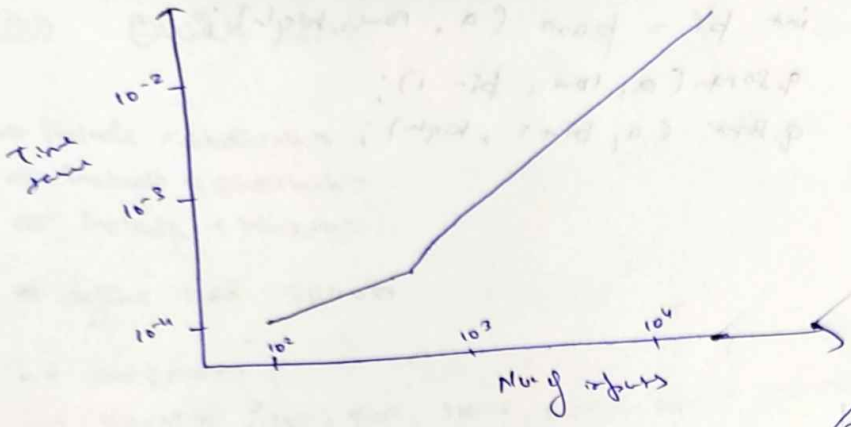
ADA Week-6 6/6/24

(1) Quick Sort

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

#define MAX 50000

int arr[MAX];
int sizes[] = {100, 500, 1000, 5000, 10000, 50000};
int num_sizes = sizeof(sizes) / sizeof(sizes[0]);

int part(int a[], int low, int high){
    int piv = a[high];
    int i = (low - 1);

    for(int j = low; j <= high - 1; j++){
        if(a[j] < piv){
            i++;
            int temp = a[i];
            a[i] = a[j];
            a[j] = temp;
        }
    }
    int temp = a[i+1];
    a[i+1] = a[high];
    a[high] = temp;
    return (i+1);
}
```

```
void qSort (int arr, int low, int high) {
    if (low < high) {
        int pi = part (a, low, high);
        qSort (a, low, pi-1);
        qSort (a, pi+1, high);
    }
}

int main() {
    FILE *fp = fopen ("quick.csv", "w");
    if (!fp) {
        perror ("unable to open");
        return EXIT_FAILURE;
    }
    fprintf (fp, "size, time");
    for (int i=0; i< sizes; i++) {
        int n = sizes[i];
        gen_rand_arr (arr, n);
        double time = meas_time (qSort, arr, n);
        fprintf (fp, "%d, %lf\n", n, time);
    }
    fclose (fp);
    return 0;
}
```

Output



6/6/24

ADA Week-7

(8) Brute Force Substring Matching

```
#include <stdio.h>
#include <string.h>

void match (char *text, char pattern) {
int n = strlen(text);
int m = strlen(pattern);

for (int i=0; i<=n-m; i++) {
    int j;
    for (j=0; j<m; j++) {
        if (text[i+j] != pattern[j]){
        break;
    }
    }
    if (j==m) {
        printf("Pattern found");
    }
}

int main() {
    char text[] = "Hello Bye";
    char pattern[] = "Hell";
    match(text, pattern);
    return 0;
}
```

Output

Pattern found successfully

(8) Johnson - Trotter Algorithm

```
#include <stdio.h>

void fn(int *p, int n) {
    for(int i=0; i<n; i++)
        printf("%d ", p[i]);
    printf("\n");
}

int gm(int *p, int *d, int n) {
    int m=-1, mi=-1;
    for(int i=0; i<n; i++) {
        if(d[i] && i>0 && p[i] > m) {
            m = p[i];
            mi = i;
        }
    }
    return mi;
}

void swp(int *a, int *b) {
    int t = *a;
    *a = *b;
    *b = t;
```

```
void rd (int *p, int *d, int n, int m){
    for (int i=0; i<n; i++)
        if (b[i] > m)
            d[i] = !d[i];
}

void jit (int n){
    int p[n], d[n];
    for (int i=0; i<n; i++)
        p[i] = i+1, d[i] = 1;
    pr(p,n);
    while(1){
        int mi = gm(p,d,n);
        if (mi == -1)
            break;
        int m = p[mi];
        if (d[mi]){
            sw(p[mi], p[mi-1]);
            sw(p[mi], d[mi-1]);
        }
        else{
            sw(p[mi], p[mi+1]);
            sw(d[mi], d[mi+1]);
        }
        rd(p,d,n,m);
        pr(p,n);
    }
}
```

```
in main()
    jt(4);
    return 0;
}
```

Output:-

1 2 3 4
1 2 4 3
1 4 2 3
4 1 2 3
4 1 3 2
1 4 3 2
1 3 4 2
3 1 4 2
3 1 2 4
1 3 2 4
1 2 3 4
2 1 3 4
2 1 4 3
2 4 1 3
4 2 1 3
4 2 3 1
2 4 3 1
2 3 4 1
3 2 1 4
2 3 1 4
2 1 3 4
1 2 3 4

(20) Leetcode – 1985 :- Kth Largest Element

```
int cmp (const void *a, const void *b) {
    const char *str1 = *(const char **) a;
    const char *str2 = *(const char **) b;
    if (strlen(str1) == strlen(str2)) {
        return strcmp(str1, str2);
    }
    return strlen(str1) - strlen(str2);
}

char * kthLargestNumber(char ** nums, int numsSize, int k){
    qsort(nums, numsSize, sizeof(char*), cmp);
    return nums[numsSize - k];
}
```

13/6/24

Test Case :-

(1) Input : nums = ["3", "6", "7", "10"]
k = 4
Output = "3"

(2) Input : nums = ["2", "21", "12", "1"]
k = 3
Output = "2"

(3) Input : nums = ["0", "0"]
k = 2
Output = "0"

ADA - Week 8

(a) Heap Sort Technique using 'c'.

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

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

void hp(int a[], int n, int i) {
    int lrg = i;
    int l = 2*i+1;
    int r = 2*i+1;

    if (l<n && a[l] > a[lrg])
        lrg = l
    }
}

void hs(int a[], int n) {
    for (int i = n/2 - 1; i>=0; i--) {
        hp(a, n, i);
    }
    for (int i = n-1; i>0; i--) {
        swap(&a[0], &a[i]);
        hp(a, i, 0);
    }
}
```

```
int main() {
    int a[] = {12, 11, 13, 5, 6, 7};
    int n = sizeof(a) / sizeof(a[0]);

    clock_t st = clock();
    hs(a, n);
    clock_t en = clock();
    double tt = ((double)(en - st));

    printf("Sorted array");

    return 0;
}
```

Output

Sorted array:
5 6 7 11 12 13
Time Taken : 0.000321 seconds

(c) Shortest Path using Floyd's Algorithm

```
#include <stdio.h>
#define INF 99999
#define V 4

void print(int d[][V]) {
    printf("Shortest distance : \n");
    for (int i = 0; i < V; i++) {
        if (d[i][j] == INF)
```

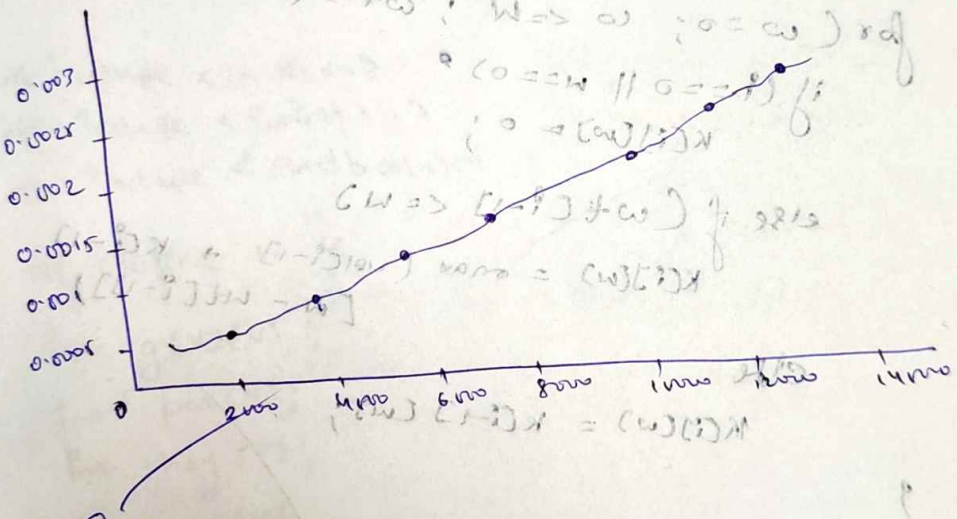
```
        printf("%7d", d[i][j]);
    }
    printf("\n");
}

void floyd(int g[V][V]) {
    int d[V][V], i, j, k;
    for (i=0; i<V; i++) {
        for (j=0; j<V; j++) {
            d[i][j] = g[i][j];
        }
    }
    printf(d);
}

int main() {
    int g[V][V] = { {0, 5, INF, 10},
                    {INF, 0, 3, INF},
                    {INF, INF, 0, 1},
                    {INF, INF, INF, 0} };
    floyd(g);
    return 0;
}
```

Output :-

Shortest Distance :

0	5	8	9
INF	0	3	4
INF	INF	0	1
INF	INF	INF	0

Heap Sort Graph



10/6/24

(Q) Implement Knapsack Problem using Dynamic Programming.

```
#Include <stdio.h>
int max (int a, int b) {
    return (a>b)? a:b;
}
int Knap (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]]);
            else
                K[i][w] = K[i-1][w];
        }
    }
    return K[n][W];
}
int main () {
    int val[] = {60, 100, 120};
    int wt[] = {10, 20, 30};
```

```
int W = 50
int n = sizeof(val) / sizeof(val[0]);
printf("Max value = %d", knap(W, wt, val, n));
return 0;
}
```

Output :-

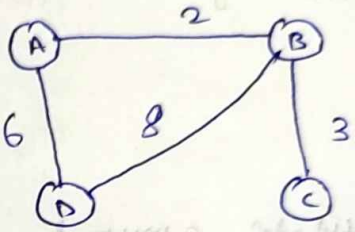
Max value in Knapsack = 220

(Q) Implement prim's Algorithm to find the MST.

```
#include <stdio.h>
#include <limits.h>
#include <stdbool.h>

#define V 4

int g[V][V];
int par[V];
int key[V];
bool mst[V];

int minKey() {
    int min = INT_MAX, min_idx;

    for (int v=0; v<V; v++)
        if (!mst[v] && key[v] < min)
            min = key[v], min_idx = v;

    return min_idx;
}
```

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

void primMST() {
    for (int i = 0; i < V; i++)
        key[i] = INT_MAX, mst[i] = false;

    key[0] = 0;
    par[0] = -1;

    for (int cnt = 0; cnt < V-1; cnt++) {
        int u = minKey();
        mst[u] = true;

        for (int v = 0; v < V; v++)
            if (graph[u][v] && !mst[v])
                par[v] = u, key[v] = graph[u][v];
    }
    printMST();
}

int main() {
    primMST();
}
```

```
graph[V][V] = { { 0, 2, 0, 6 },
                { 2, 0, 3, 8 },
                { 0, 3, 0, 0 },
                { 6, 8, 0, 0 } }

primMST();
return 0;
}
```



Output

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

4/7/24

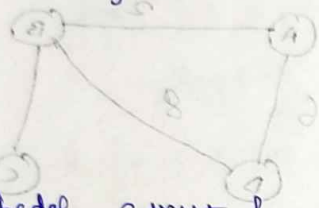
ADA Week - 10

11/4/24

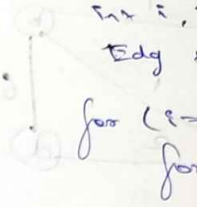
50

(@) Implement Kruskal's Algorithm using "C".

```
#include <stdio.h>
#define max 1000
#define inf 999

int par[max], v = 4, E;
int adj[max][max] = { {0, 1, 3, inf},
                      {1, 0, 1, 6},
                      {3, 1, 0, 2},
                      {inf, 6, 2, 0} };
```



```
typedef struct {
        int u, v, w;
} Edg;

Edg edg[max];

int find(int i) {
    while (par[i] != i)
        i = par[i];
    return i;
}

void uni(int i, int j) {
    int a = find(i);
    int b = find(j);
    par[a] = b;
}
```

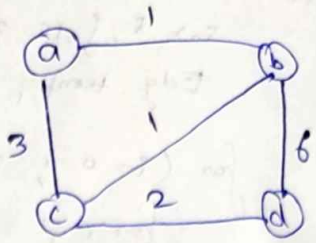


```
void kruskal
int i,
Edg
for (i=
for
E =
for (
for
prin
}
int
}
```

```
void kruskal() {
    int i, j, k = 0, cst = 0;
    Edg temp;

    for (i=0; i<V; i++)
        for (j = i+1; j<V; j++)
            if (adj[i][j] != inf && adj[i][j] != 0) {
                edg[k].u = i;
                edg[k].v = j;
                edg[k].w = adj[i][j];
                k++;
            }

    E = k;

    for (i=0; i<V; i++)
        par[i] = i;

    for
    printf("MST Edges : \n");
    printf("Min cost = %d \n", cst);
}

int main() {
    kruskal();
    return 0;
}
```

Output

0	1	3	∞
1	0	1	6
3	1	0	2
∞	6	2	0



MST Edges

1 - 2 : 1

2 - 3 : 2

0 - 1 : [redacted]

Min. Cost = 4

(Q) Implement Dijstra's Algorithm using 'C'.

```
#include <stdio.h>

#define V 5
#define INF 999

int dist[V], vis[V];
int graph[V][V] = { {0, 10, 20, 0, 0},
                    {10, 0, 0, 50, 10},
                    {20, 0, 0, 20, 33},
                    {0, 50, 20, 0, 2},
                    {0, 10, 33, 2, 0} };
```

```
int minDist() {
    int min = inf, min_idx, v;
    for (v=0; v<V; v++)
        if (!visited[v] && dist[v] < min) min = dist[v], min_idx = v;
    return min_idx;
}

void dijkstra(int src) {
    int i, cnt, u, v;
    for (i=0; i<V; i++)
        dist[i] = inf, visited[i] = 0;
    dist[src] = 0;
    for (cnt=0; cnt<V-1; cnt++) {
        u = minDist();
        visited[u] = 1;
    }
    printf("Vertex Dist from src\n");
    for (i=0; i<V; i++)
        printf("%d \t %d\n", i, dist[i]);
}

int main() {
    dijkstra(0);
    return 0;
}
```

Output

Vertex	Dist from Src
0	0
1	10
2	20
3	42
4	20