Experiment:4 Date:11/2/18

Aim:- Implementing Djkstra [Shortest path algorithm](https://moodle.spit.ac.in/mod/resource/view.php?id=9414) using greedy method.

Theory:-

Among all the algorithmic approaches, the simplest and straightforward approach is the Greedy method. In this approach, the decision is taken on the basis of current available information without worrying about the effect of the current decision in future.

Greedy algorithms build a solution part by part, choosing the next part in such a way, that it gives an immediate benefit. This approach never reconsiders the choices taken previously. This approach is mainly used to solve optimization problems. Greedy method is easy to implement and quite efficient in most of the cases. Hence, we can say that Greedy algorithm is an algorithmic paradigm based on heuristic that follows local optimal choice at each step with the hope of finding global optimal solution.

In many problems, it does not produce an optimal solution though it gives an approximate (near optimal) solution in a reasonable time.

Areas of Application:-

Greedy approach is used to solve many problems, such as

Finding the shortest path between two vertices using Dijkstra’s algorithm.

Finding the minimal spanning tree in a graph using Prim’s /Kruskal’s algorithm, etc.

Algorithm:-

Dijkstra algorithm is also called single source shortest path algorithm. It is based on greedy technique. The algorithm maintains a list visited[ ] of vertices, whose shortest distance from the source is already known.

If visited[1], equals 1, then the shortest distance of vertex i is already known. Initially, visited[i] is marked as, for source vertex.

At each step, we mark visited[v] as 1. Vertex v is a vertex at shortest distance from the source vertex. At each step of [the algorithm](https://www.thecrazyprogrammer.com/2013/10/recursive-permutation-in-c.html), shortest distance of each vertex is stored in an array distance[ ].

1. Create cost matrix C[ ][ ] from adjacency matrix adj[ ][ ]. C[i][j] is the cost of going from vertex i to vertex j. If there is no edge between vertices i and j then C[i][j] is infinity.

2. Array visited[ ] is initialized to zero.

               for(i=0;i<n;i++)

                              visited[i]=0;

3. If the vertex 0 is the source vertex then visited[0] is marked as 1.

4. Create the distance matrix, by storing the cost of vertices from vertex no. 0 to n-1 from the source vertex 0.

               for(i=1;i<n;i++)

                              distance[i]=cost[0][i];

Initially, distance of source vertex is taken as 0. i.e. distance[0]=0;

5. for(i=1;i<n;i++)

– Choose a vertex w, such that distance[w] is minimum and visited[w] is 0. Mark visited[w] as 1.

– Recalculate the shortest distance of remaining vertices from the source.

– Only, the vertices not marked as 1 in array visited[ ] should be considered for recalculation of distance. i.e. for each vertex v

               if(visited[v]==0)

  distance[v]=min(distance[v], distance[w]+cost[w][v])

**Time Complexity**

The program contains two nested loops each of which has a complexity of O(n). n is number of vertices. So the complexity of algorithm is O(n2).

**Program:-**

#include<stdio.h>

#define INFINITY 9999

#define MAX 10

void dijkstra(int G[MAX][MAX], int n, int startnode);

void dijkstra(int G[MAX][MAX], int n, int startnode) {

int cost[MAX][MAX], distance[MAX], pred[MAX];

int visited[MAX], count, mindistance, nextnode, i, j;

//pred[] stores the predecessor of each node

//count gives the number of nodes seen so far

//create the cost matrix

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

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

if (G[i][j] == 0)

cost[i][j] = INFINITY;

else

cost[i][j] = G[i][j];

//initialize pred[],distance[] and visited[]

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

distance[i] = cost[startnode][i];

pred[i] = startnode;

visited[i] = 0;

}

distance[startnode] = 0;

visited[startnode] = 1;

count = 1;

while (count < n - 1) {

mindistance = INFINITY;

//nextnode gives the node at minimum distance

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

if (distance[i] < mindistance&&!visited[i]) {

mindistance = distance[i];

nextnode = i;

}

//check if a better path exists through nextnode

visited[nextnode] = 1;

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

if (!visited[i])

if (mindistance + cost[nextnode][i] < distance[i]) {

distance[i] = mindistance + cost[nextnode][i];

pred[i] = nextnode;

}

count++;

}

//print the path and distance of each node

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

if (i != startnode) {

printf("\nDistance of node%d=%d", i, distance[i]);

printf("\nPath=%d", i);

j = i;

do {

j = pred[j];

printf("<-%d", j);

} while (j != startnode);

}

}

int main() {

int G[MAX][MAX], i, j, n, u;

printf("Enter no. of vertices:");

scanf("%d", &n);

printf("\nEnter the adjacency matrix:\n");

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

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

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

printf("\nEnter the starting node:");

scanf("%d", &u);

dijkstra(G, n, u);

return 0;

}

Output:-

Enter no. of vertices:8

Enter the adjacency matrix:

0 20 0 80 0 0 90 0

0 0 0 0 0 10 0 0

0 0 0 10 0 50 0 20

0 0 10 0 0 0 20 0

0 50 0 0 0 0 30 0

0 0 10 40 0 0 0 0

20 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0

Enter the starting node:0

Distance of node1=20

Path=1<-0

Distance of node2=40

Path=2<-5<-1<-0

Distance of node3=50

Path=3<-2<-5<-1<-0

Distance of node4=9999

Path=4<-0

Distance of node5=30

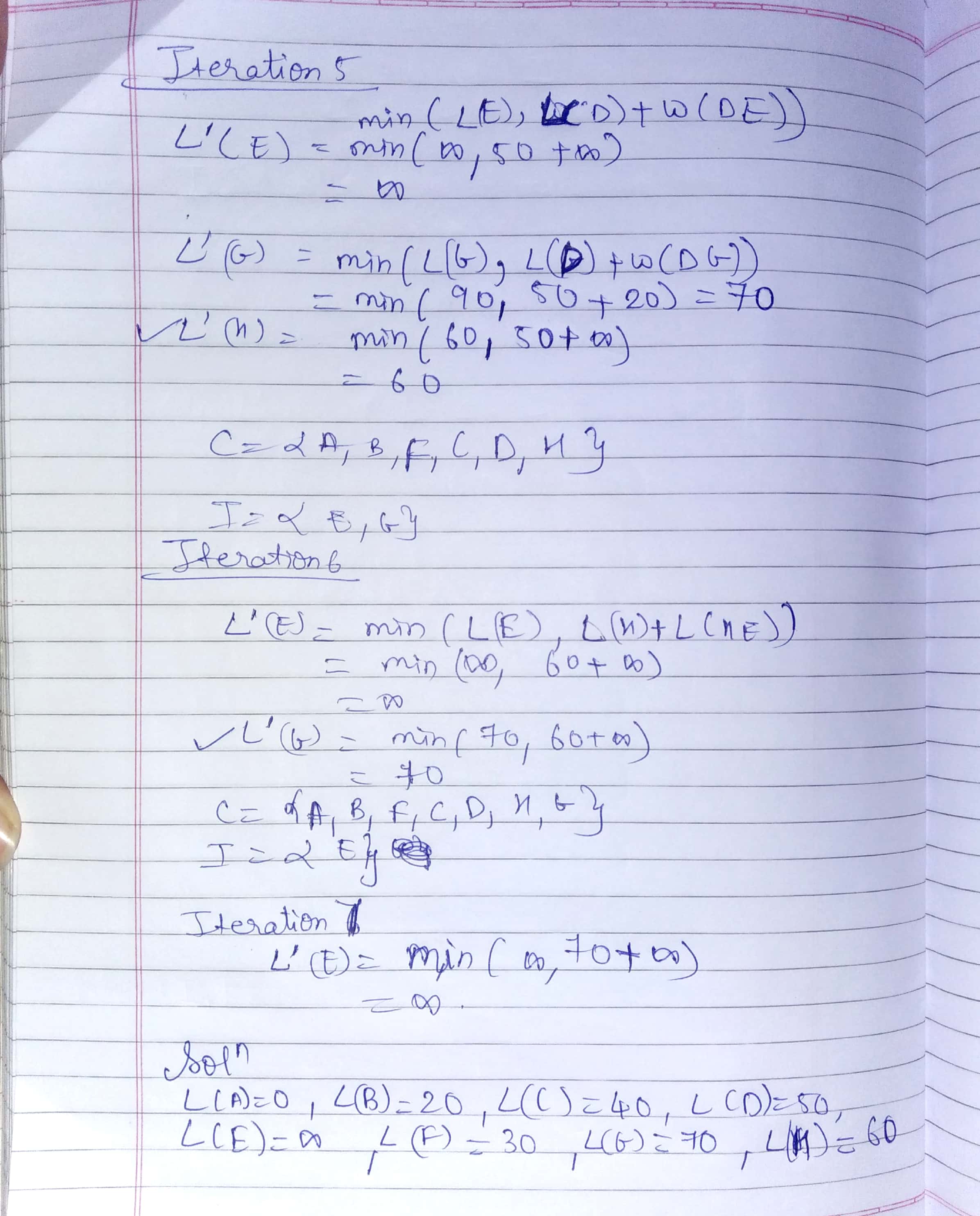
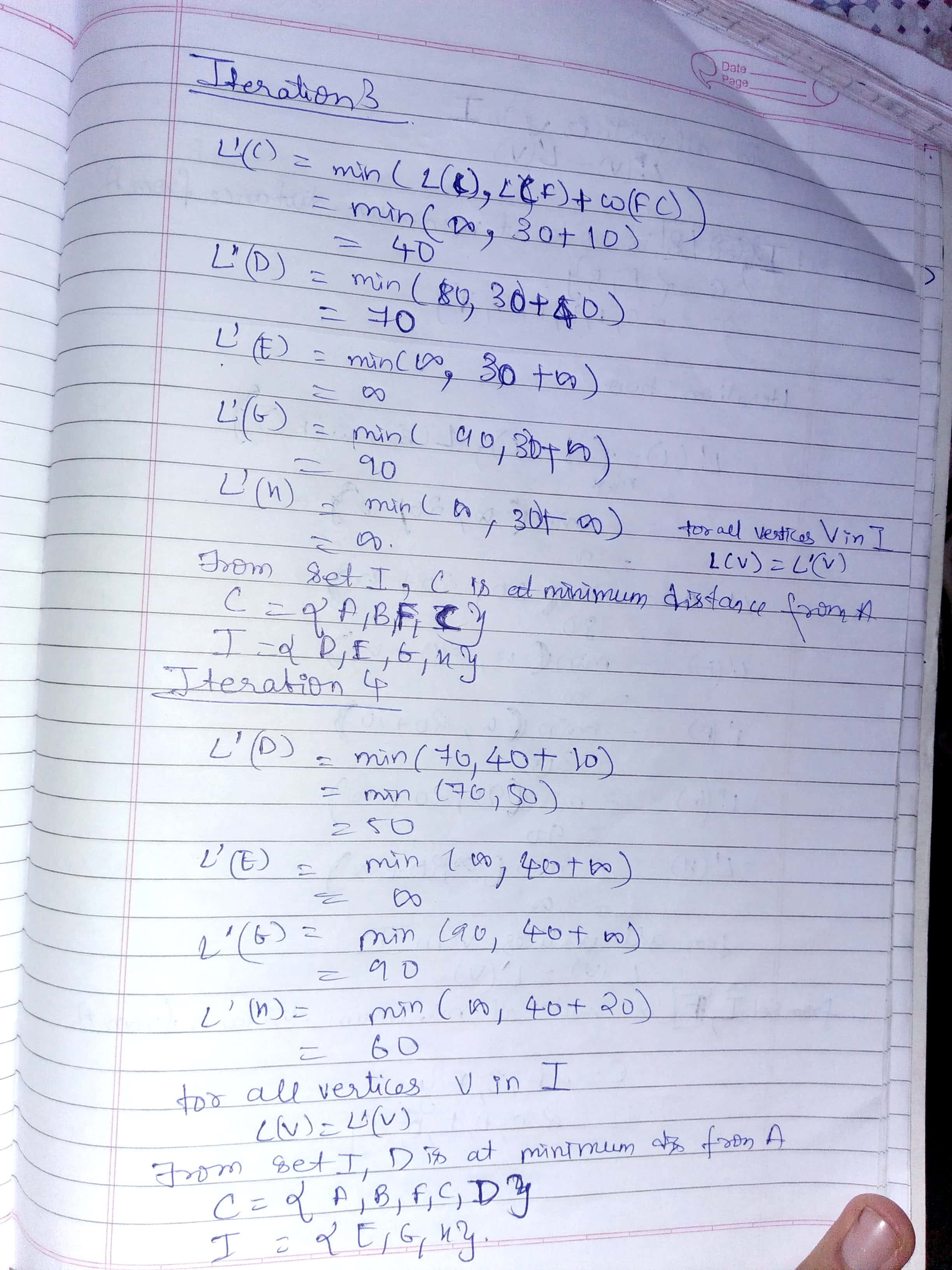
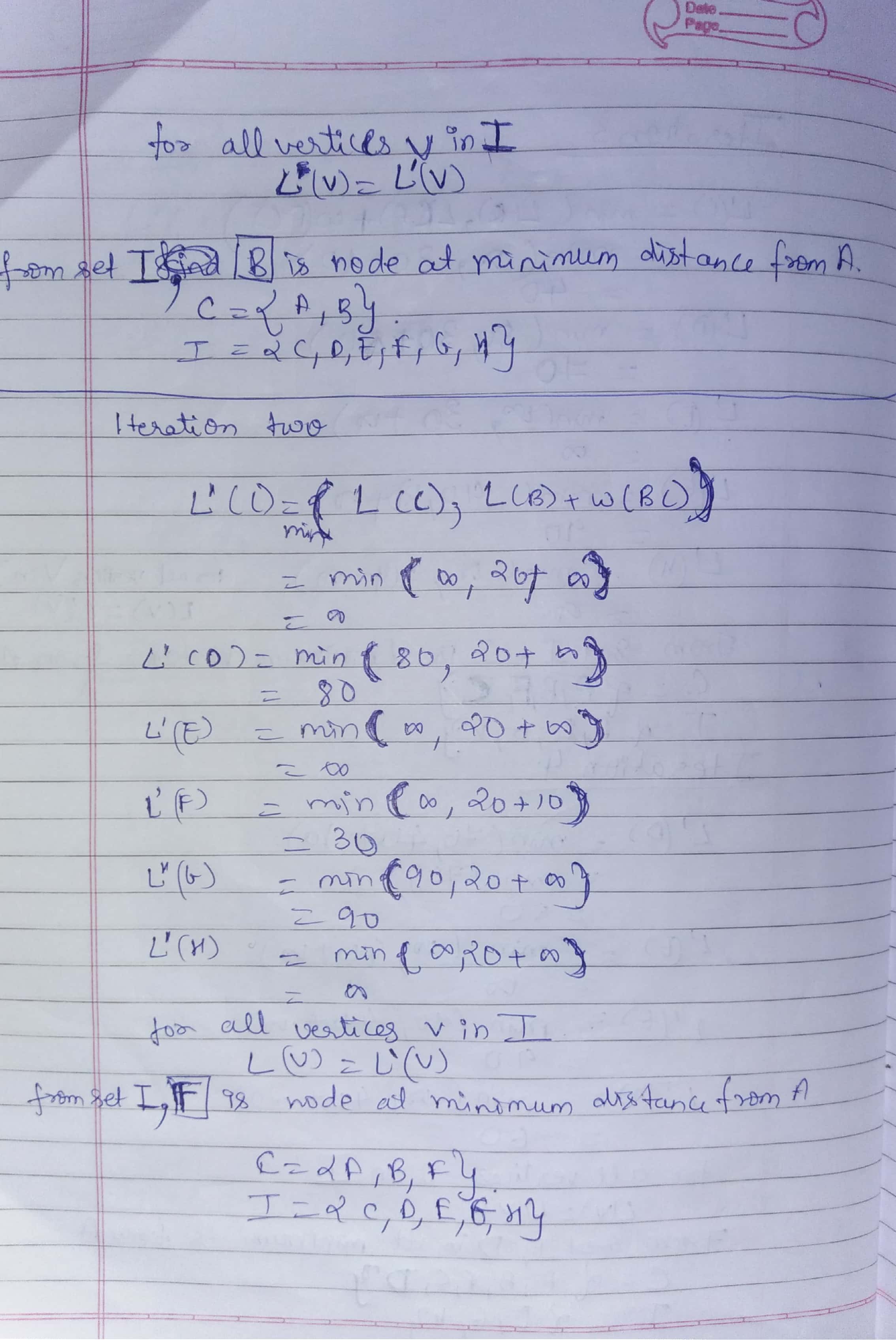
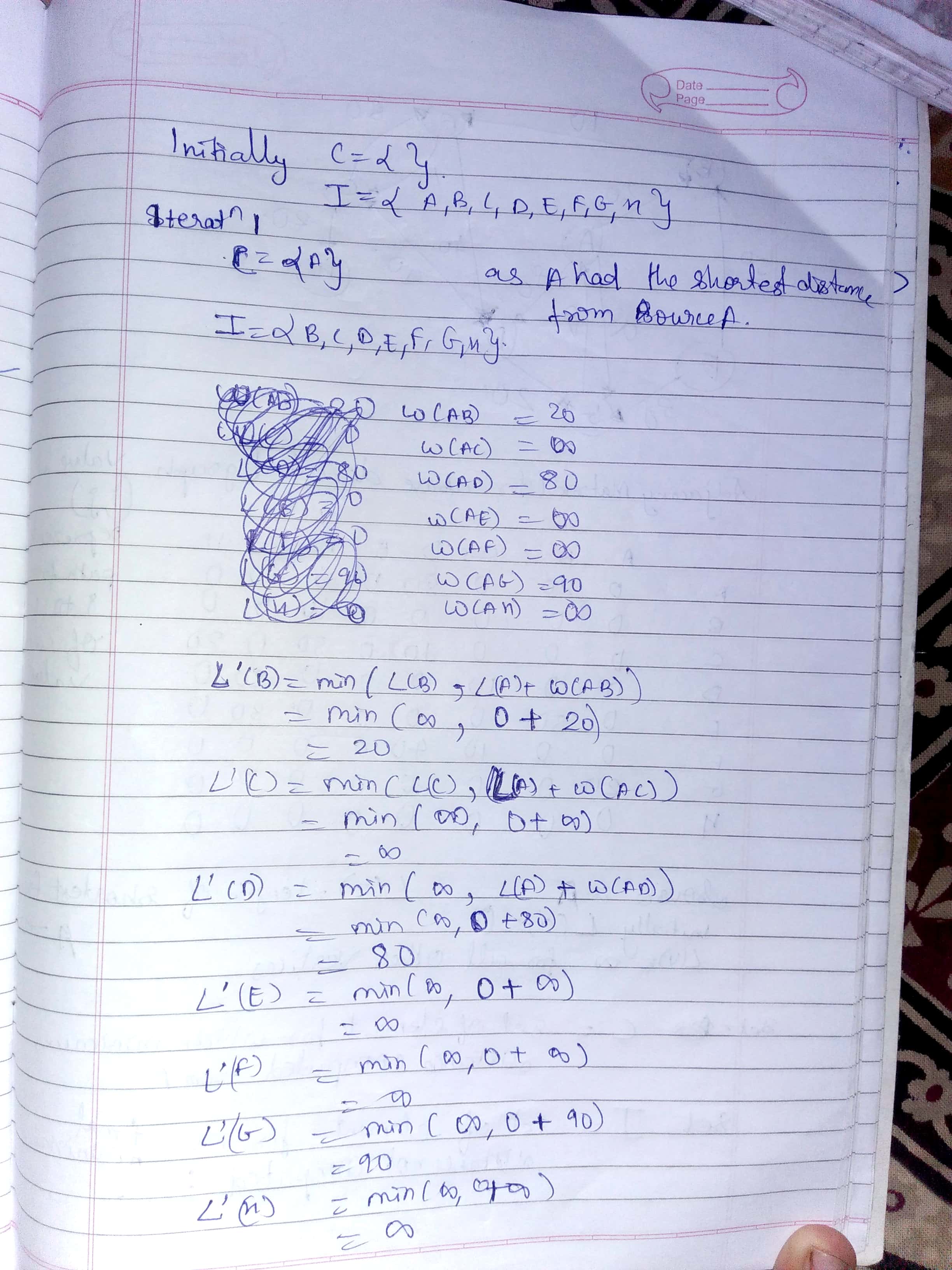
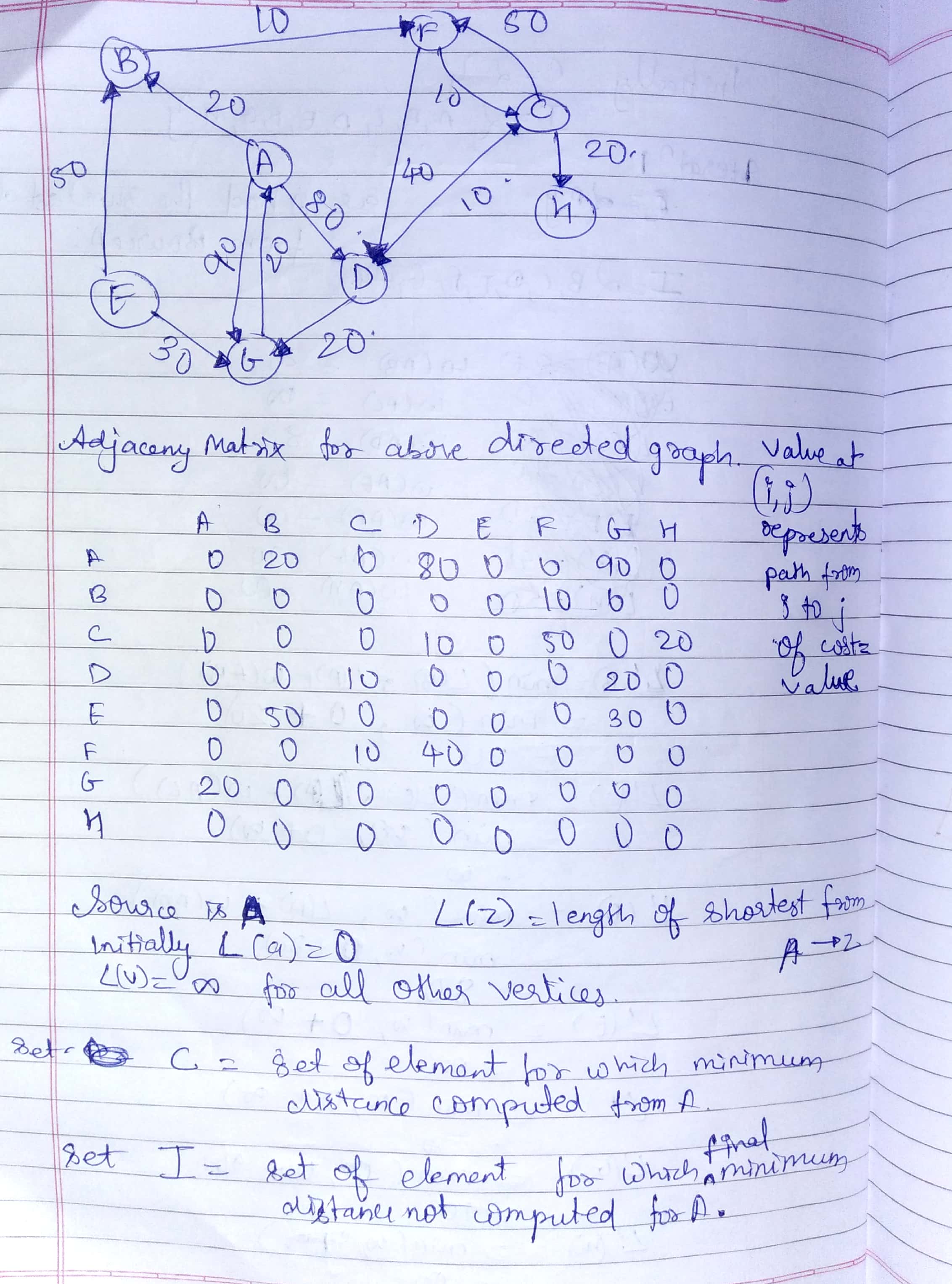
Path=5<-1<-0

Distance of node6=70

Path=6<-3<-2<-5<-1<-0

Distance of node7=60

Path=7<-2<-5<-1<-0



Conclusion:-

Dijkstra’s algorithm helps to find shortest path form a given source using greedy approach, but it doesn’t work for negative edge length.

**Aim :** Implement Huffman Coding using greedy strategy.

**Theory:**

Algorithm for [Huffman Coding](https://moodle.spit.ac.in/mod/resource/view.php?id=9458) with example

1. Scan text to be compressed and tally occurrence of all characters.

2. Sort or prioritize characters based on number of occurrences in text.

3. Build Huffman code tree based on prioritized list.

4. Perform a traversal of tree to determine all code words.

5. Scan text again and create new file using the Huffman codes.

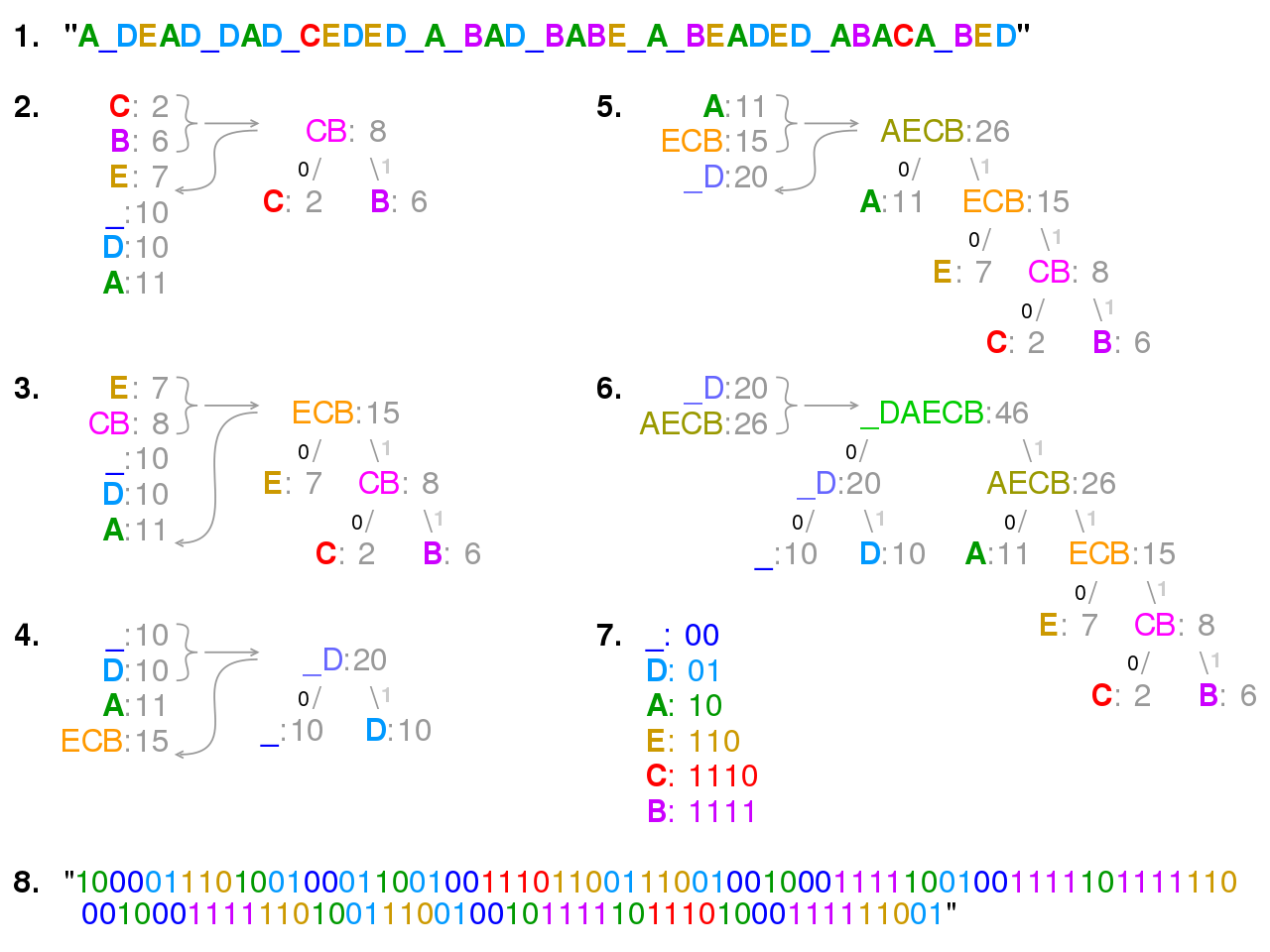
Importance of Huffman coding:- If you assign less number or bits or **shorter code** words for **most frequently** used symbols you will be saving a lot of storage space.

Huffman encoding gives prefix codes.

**Construction of Huffman tree:**

A greedy approach to construct Huffman tree for n characters is as follows:

places n characters in n sub-trees. Starts by combining the two least weight nodes into a tree which is assigned the sum of the two leaf node weights as the weight for its root node. Do this until you get a single tree.



Visualisation of the use of Huffman coding to encode the message "A\_DEAD\_DAD\_CEDED\_A\_BAD\_BABE\_A\_BEADED\_ABACA\_BED". In steps 2 to 6, the letters are sorted by increasing frequency, and the least frequent two at each step are combined and reinserted into the list, and a partial tree is constructed. The final tree in step 6 is traversed to generate the dictionary in step 7. Step 8 uses it to encode the message.

Priority queue using linked list and binary tree, is used in program below.

**Program:-**

**//here priority queue implemented using linked list.**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include<string.h>**

**#define MAX\_LIMIT 50**

**//this struct used to build huffman tree.**

**typedef struct element {**

**char c;**

**int count;**

**struct element \*next, \*left, \*right;**

**} el;**

**//in below structure flexible array member char str[]**

**//is used for which memory allocated at time of object creation.**

**typedef struct code{**

**char c;**

**struct code \*next;**

**char str[];**

**}code;**

**//create function returning new element**

**el\* create()**

**{**

**el\* node = (el\*) malloc(sizeof (el));**

**node->next = NULL;**

**node->left = NULL;**

**node->right = NULL;**

**node->count = 1;**

**return node;**

**}**

**//pop function returning top element of linked list**

**el\* pop(el \*\*AddressOfHead) {**

**el \*temp = \*AddressOfHead;**

**if (temp) {**

**\*AddressOfHead = temp->next;**

**}**

**return temp;**

**}**

**//inserting newly created node at proper**

**//position according to frequency**

**el\* insert(el\* head, el\* node)**

**{**

**el\* temp=head;**

**el\* follow=NULL;**

**int freq=node->count;**

**if(head==NULL)**

**return node;**

**while(temp!=NULL&& freq>=(temp->count))**

**{**

**follow=temp;**

**temp=temp->next;**

**}**

**if(follow==NULL)**

**{**

**node->next=temp;**

**head=node;**

**}**

**else**

**{**

**follow->next=node;**

**node->next=temp;**

**}**

**return head;**

**}**

**//this builds the linked list of**

**//characters given in input String**

**el\* makeFreq(char str[]) {**

**int n = strlen(str);**

**el\* head=create();**

**head->c = str[0];**

**int i;**

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

**el\* temp = head;**

**el\* follow = head;**

**while (temp != NULL) {**

**if (temp->c == str[i]) {**

**temp->count = ++(temp->count);**

**break;**

**}**

**follow = temp;**

**temp = temp->next;**

**}**

**if (temp == NULL) {**

**el\* node=create();**

**node->c = str[i];**

**follow->next = node;**

**}**

**}**

**return head;**

**}**

**/\* function to swap count of two nodes a and b\*/**

**void swap( el\* a,el\* b)**

**{**

**int temp = a->count;**

**char temp2=a->c;**

**a->c=b->c;**

**b->c=temp2;**

**a->count = b->count;**

**b->count = temp;**

**}**

**//sorting by frequency level 1.**

**void bubbleSort(el\* start)**

**{**

**int swapped, i;**

**el \*ptr1;**

**el \*lptr = NULL;**

**/\* Checking for empty list \*/**

**if (start == NULL)**

**return;**

**do**

**{**

**swapped = 0;**

**ptr1 = start;**

**while (ptr1->next != lptr)**

**{**

**if (ptr1->count > ptr1->next->count)**

**{**

**swap(ptr1, ptr1->next);**

**swapped = 1;**

**}**

**ptr1 = ptr1->next;**

**}**

**lptr = ptr1;**

**}**

**while (swapped);**

**}**

**//helper function for sorting by acsii value.(level 2)**

**void bubbleSortChar(el\* start,el\* end)**

**{**

**int swapped;**

**el \*ptr1;**

**el \*lptr = end;**

**do**

**{**

**swapped = 0;**

**ptr1 = start;**

**while (ptr1->next != lptr)**

**{**

**if (ptr1->c > ptr1->next->c)**

**{**

**swap(ptr1, ptr1->next);**

**swapped = 1;**

**}**

**ptr1 = ptr1->next;**

**}**

**lptr = ptr1;**

**}**

**while (swapped);**

**}**

**//sorting by ascii value.(level 2)**

**void sort(el\* head)**

**{**

**if(head==NULL)**

**return;**

**el\* start=head;**

**while(start!=NULL)**

**{**

**el\* end=start;**

**int a=end->count;**

**while(end->next!=NULL&& end->next->count==a)**

**{**

**end=end->next;**

**}**

**if(end!=start)**

**bubbleSortChar(start,end->next);**

**start=end->next;**

**}**

**}**

**//printing frequency of each char from linked list and returning total no of nodes made.**

**int printList(el\* head)**

**{**

**printf("char :: frequency \n");**

**el\* temp = head;**

**int count = 0;**

**while (temp != NULL) {**

**count++;**

**printf(" %c::%d \n", temp->c, temp->count);**

**temp = temp->next;**

**}**

**return count;**

**}**

**//building huffman tree from linked list made**

**//first time.,and modifying it and doing until single node is left.**

**el\* makeHuffTree(el\* head)**

**{**

**el\* start=head;**

**while(start->next!=NULL)**

**{**

**el\* newNode=create();**

**newNode->c='$';**

**newNode->left=pop(&start);**

**newNode->right=pop(&start);**

**newNode->count=(newNode->left->count)+(newNode->right->count);**

**start=insert(start,newNode);**

**//printf("After removing 2 nodes\n");**

**//printList(start);**

**}**

**return start;**

**}**

**//printing code in getcode function from int array.**

**void printArr(int arrr[], int n)**

**{**

**int i;**

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

**printf("%d", arrr[i]);**

**}**

**printf("\n");**

**}**

**//function to allocate string of code for each character in code structure.**

**char\* getString(int arr[],int n)**

**{**

**char \*str;**

**int size = n+1; /\*one extra for /0\*/**

**str = (char \*)malloc(sizeof(char)\*size);**

**int i=0;**

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

**{ if(arr[i]==0)**

**\*(str+i) = '0';**

**else**

**\*(str+i) = '1';**

**}**

**\*(str+i)='\0';**

**return str;**

**}**

**//This is the only global variable used in program.**

**code \*firstNode=NULL;**

**//prints the code by recursion and also stores in**

**//string code for respective character which is used for decoding string.**

**void getCode(el \*root, int arr[], int top)**

**{**

**if (root->left==NULL&&root->right==NULL) {**

**printf("%c: ", root->c);**

**printArr(arr, top);**

**code \*s;**

**s = (code\*) malloc( sizeof(\*s) + sizeof(char)\*(top+1) );**

**strcpy(s->str,getString(arr,top));**

**s->c =root->c;**

**s->next =firstNode;**

**firstNode=s;**

**return;**

**}**

**if (root->left) {**

**arr[top] = 0;**

**getCode(root->left, arr, top + 1);**

**}**

**if (root->right) {**

**arr[top] = 1;**

**getCode(root->right, arr, top + 1);**

**}**

**}**

**//main driver function.**

**int main() {**

**char string[MAX\_LIMIT];**

**printf("enter sentence \n");**

**fgets(string, MAX\_LIMIT, stdin);**

**printf("%s\n", string);**

**el\* head = makeFreq(string);**

**int count=printList(head);**

**bubbleSort(head);**

**printf("After sorting by frequency\n");**

**printList(head);**

**printf("Now Sorting by Alphabetic order(i.e ASCII value)(level 2 sorting)\n");**

**sort(head);**

**printList(head);**

**el\* result = makeHuffTree(head);**

**printf("Code for characters are.\n");**

**int arr[10];**

**getCode(result,arr,0);**

**code \*temp=firstNode;**

**while(temp!=NULL)**

**{**

**printf("code for %c is %s\n",temp->c,temp->str);**

**temp=temp->next;**

**}**

**int length=strlen(string);**

**printf("huffman code for %s ",string);**

**int i=0;**

**for(;i<length;i++)**

**{**

**code \*temp=firstNode;**

**char ch=string[i];**

**while(temp!=NULL&&temp->c!=ch)**

**temp=temp->next;**

**if(temp!=NULL)**

**printf("%s",temp->str);**

**}**

**return 0;**

**}**

**Output:-**

enter sentence

SE computer is a very good class.

SE computer is a very good class.

char :: frequency

S::1

E::1

::6

c::2

o::3

m::1

p::1

u::1

t::1

e::2

r::2

i::1

s::3

a::2

v::1

y::1

g::1

d::1

l::1

.::1

After sorting by frequency

char :: frequency

S::1

E::1

m::1

p::1

u::1

t::1

i::1

v::1

y::1

g::1

d::1

l::1

.::1

c::2

e::2

r::2

a::2

o::3

s::3

::6

Now Sorting by Alphabetic order(i.e ASCII value)(level 2 sorting)

char :: frequency

.::1

E::1

S::1

d::1

g::1

i::1

l::1

m::1

p::1

t::1

u::1

v::1

y::1

a::2

c::2

e::2

r::2

o::3

s::3

::6

Code for characters are.

s: 000

y: 0010

a: 0011

c: 0100

e: 0101

r: 0110

.: 01110

E: 01111

S: 10000

d: 10001

g: 10010

i: 10011

l: 10100

m: 10101

p: 10110

t: 10111

u: 11000

v: 11001

o: 1101

: 111

code for is 111

code for o is 1101

code for v is 11001

code for u is 11000

code for t is 10111

code for p is 10110

code for m is 10101

code for l is 10100

code for i is 10011

code for g is 10010

code for d is 10001

code for S is 10000

code for E is 01111

code for . is 01110

code for r is 0110

code for e is 0101

code for c is 0100

code for a is 0011

code for y is 0010

code for s is 000

huffman code for SE computer is a very good class.

100000111111101001101101011011011000101110101011011110011000111001111111001010101100010111100101101110110001111010010100001100000001110

RUN SUCCESSFUL (total time: 19s)

**How much cost saved Calculation:-**

**(1-(135/(33\*8)))\*100=48.8636%;**

**Conclusion:-**

Thus this type of encoding of data helps in saving lot of space, but error occurred in transmission can lead to much difficulty in decoding of message, as this method uses prefix codes.