SSSP

● Graded

Student
ANSHUMAN SINGH

Total Points
50 / 50 pts

Autograder Score
50.0 / 50.0

Passed Tests
Test 1 (10/10)
Test 2 (20/20)
Test 3 (20/20)

Autograder Results

Test 1 (10/10)

Test 2 (20/20)

Test 3 (20/20)

Submitted Files

hw.c

Download

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


struct Node {
    int vrtx;
    int wt;
    struct Node* nxt;
};

struct Graph {
    int vrtx;
    struct Node** adj_list;
};

struct Node* createNode(int v, int wt) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->vrtx = v;
    newNode->wt = wt;
    newNode->nxt = NULL;
    return newNode;
}

struct Graph* createGraph(int ver) {
    struct Graph* G = (struct Graph*)malloc(sizeof(struct Graph));
    G->vrtx = ver;
    G->adj_list = (struct Node**)malloc(ver * sizeof(struct Node*));
    for (int i = 0; i < ver; i++)
        G->adj_list[i] = NULL;
    return G;
}

void addEdge(struct Graph* G, int src, int dest, int wt) {
    struct Node* newNode = createNode(dest, wt);
    newNode->nxt = G->adj_list[src];
    G->adj_list[src] = newNode;
}


typedef struct {
    int *arr;
    int *dist;
    int size;
    int capacity;
} MinHeap;

void initHeap(MinHeap *heap, int capacity) {
    heap->arr = (int *)malloc(capacity *sizeof(int));
```

```
    heap->dist = (int *)malloc(capacity *sizeof(int));
    heap->size = 0;
    heap->capacity = capacity;
}

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

void minHeapify(MinHeap *heap, int i) {
    int k = i;
    int left = 2 * i + 1;
    int right = 2 * i + 2;
    if (left < heap->size && heap->dist[heap->arr[left]] < heap->dist[heap->arr[k]])
        k = left;
    if (right < heap->size && heap->dist[heap->arr[right]] < heap->dist[heap->arr[k]])
        k = right;
    if (k != i) {
        swap(&heap->arr[i], &heap->arr[k]);
        minHeapify(heap, k);
    }
}

void buildHeap(MinHeap *heap, int n, int *dist) {
    heap->size = n;
    for (int i=0;i<n;i++) {
        heap->arr[i] = i;
        heap->dist[i] = dist[i];
    }
    for (int i=(heap->size/2)-1;i>=0;i--)
        minHeapify(heap, i);
}

void decreaseKey(MinHeap *heap, int node, int newValue) {
    int i;
    for (i=0;i<heap->size;i++) {
        if (heap->arr[i] == node)
            break;
    }
    if (i>=heap->size||heap->dist[node]<=newValue)
        return;
    heap->dist[node] = newValue;
    while (i > 0 && heap->dist[heap->arr[(i - 1) / 2]] > heap->dist[heap->arr[i]]) {
        swap(&heap->arr[i], &heap->arr[(i - 1) / 2]);
        i = (i - 1) / 2;
    }
}

int extractMin(MinHeap *heap) {
    if (heap->size <= 0) {
```

```
        return -1;
    }
    int min = heap->arr[0];
    heap->arr[0] = heap->arr[heap->size - 1];
    heap->size--;
    minHeapify(heap, 0);
    return min;
}

void SSSP(struct Graph* G, int src) {
    int n = G->vrtx;
    int *dist = (int *)malloc(n * sizeof(int));
    int *parent = (int *)malloc(n * sizeof(int));
    MinHeap minHeap;
    initHeap(&minHeap, n);
    for (int v = 0; v < n; v++) {
        dist[v] = INT_MAX;
        parent[v] = -1;
    }
    dist[src] = 0;
    buildHeap(&minHeap, n, dist);
    while (minHeap.size > 0) {
        int u = extractMin(&minHeap);
        struct Node* temp = G->adj_list[u];
        while (temp) {
            int v = temp->vrtx;
            int wt = temp->wt;
            if (dist[u] != INT_MAX && dist[u] + wt < dist[v]) {
                dist[v] = dist[u] + wt;
                parent[v] = u;
                decreaseKey(&minHeap, v, dist[v]);
            }
            temp = temp->nxt;
        }
    }
    int *ans= (int*)malloc(n*sizeof(int));
    for (int i=0;i<n;i++) {
        ans[i]=dist[i];
    }
    for(int i=0;i<n;i++){
        for(int j=i+1;j<n;j++){
            if(ans[i]>ans[j]){
                int temp=ans[i];
                ans[i]=ans[j];
                ans[j]=temp;
            }
        }
    }
    for(int i=0;i<n;i++){
        for(int j=0;j<n;j++){
            if(dist[j]==ans[i]){
                printf("%d %d ",j, ans[i]);
```

```
            break;
        }
      }
    }
    free(dist);
    free(parent);
    free(minHeap.arr);
    free(minHeap.dist);
}

int main() {
    int n;
    scanf("%d",&n);
    struct Graph* G = createGraph(n);
      for(int i=0;i<n;i++){
       int ver,wt;
       scanf("%d",&ver);
       if(ver==-1)
       continue;
       else{
          scanf("%d",&wt);
          addEdge(G, i, ver, wt);
          i=i-1;
       }
      }
     int src;
     scanf("%d",&src);
     SSSP(G, src);
    return 0;
}

```