**-Popis myšlienky:**

Algoritmus zoradí svahy podľa strmosti a postupne ich bude pridávať a testovať či sa môže dostať z bodu s do bodu f.

**-Popis dátových štruktúr:**

Inty v,e,s,f so vstupom, pole h s vstupom, 2d pole vstup so vstupom(začiatok svahu, koniec svahu, vypočítaná strmosť), pole do kt. sa budú pridávať svahy, vytvorenie pola bol v ktorom budú hodnoty či už som bol na stanici lanovky pod indexom, queue s pozíciami staníc lanoviek kde som už bol ale dalej som z nich nepostupoval.

**-Popis algoritmu:**

Algoritmus najskôr načíta vstup vypočíta strmosť svahov a potom zoradí svahy podľa strmosti od najmenej strmého. Postupne pridáva svahy podľa strmosti(keď má viac svahov rovnakú strmosť tak ich pridá všetky a až potom skúša hľadať cestu z f do s) používa bread first search aby zistil či existuje cesta z s do f. A keď áno tak vie že našiel cestu s najmenšou strmosťou vypíše ju a skončí.

**-Zdôvodnenie správnosti:**

Algoritmus vždy nájde správnu odpoveď preto že skúša všetky možnosti od najmenšieho a nemôže nič vynechať a pomýliť sa.

**-Odhad časovej zložitosti:**

Časová maximálna zložitosť je závislá od súčtu v,e,e\*log(e) a e\*(e+v).

E\*log(e) je zložitosť zoradenia a e\*(e+v) je zložitosť hľadania kedby boli všetky strmosti rôzne a bolo by potrebné mať všetky svahy aby sa dalo dostať do ciela.

**-Odhad pamäťovej zložitosti:**

Pamäťová zložitosť je lineárna a závislá od súčtu v + e.

**-Lepšia úprava algoritmu:**

Rýchlejší algoritmus by počítal iba či sa dá dostať do f s polovicou svahov a potom podľa toho či áno alebo nie tak by počítal s ¼ alebo ¾ a takto by postupne zmenšoval kroky o ktoré by sa posúval a zabralo by mu to oveľa menej operácií. Toto riešenie som však neimplementoval a píšem ho sem iba ako môj nápad.

**Môj kód:**

from collections import deque

v,e,s,f=map(int,input().split(" "))#vstup

h=[0]\*v #vytvorenie pola na vstup

for i in range(v):

    h[i]=int(input()) #zapisovanie vstupu do pola

vstup=[[0 for i in range(3)] for j in range(e)] #vytvorenie pola na vstup

for i in range(e):

    vstup[i][0],vstup[i][1],vstup[i][2]=map(float,input().split(" "))#zapisovanie vstupu

    vstup[i][2]=float(abs(h[int(vstup[i][0])]-h[int(vstup[i][1])]))/vstup[i][2]#počítanie strmosti

vstup = sorted(vstup, key=lambda x: x[2])#zoradenie pola podla strmosti

pole = [[] for i in range(v+1)]#vytvorenie pola do kt. sa budú pridávať svahy

for i in range(0,e):#testovanie

    pole[int(vstup[i][0])].append(int(vstup[i][1])) #pridanie cesty do pola

    pole[int(vstup[i][1])].append(int(vstup[i][0])) #pridanie otocenej cesty do pola

    if i<e-1:#keď toto nieje posledny element

        if vstup[i][2]==vstup[i+1][2]:#ked je hodnota strmosti 2 vstupov rovnaka tak to nepocita 2x

            continue

    pq = deque() #vytvorenie priority queue

    pq.append(s) #pridanie zaciatku do priority queue

    bol = [False for i in range(v)] #vytvorenie pola s hodnotami kde som uz bol

    bol[s] = True #nastavenie zaciatku na true

    while len(pq)>0:#keď je v deque nejaký prvok

        z = pq.popleft() #vymazanie elementu zlava a ulozenie do z

        for el in pole[z]: #testovanie ciest od z

            if (el == f):#testovanie či sme v cieli

                print("{:.2f}".format(vstup[i][2])) #vypísanie odpovede

                exit() #koniec programu

            if bol[el]==False:#ked sme v novom bode tak sa nastaví na true a prida sa do pq

                pq.append(el)

                bol[el] = True