Gruppo 13: Maurizio Minieri, Alessio Spina, Domenico Maione.

Relazione: **Progetto 3**

Descrizione del Problema:

Maggio, l'estate è alle porte. Matteo, promettente studente universitario, viene afflitto da un pensiero che non gli dà respiro, lo soffoca addirittura di più di come succedeva quando pensava all'esame di algoritmi e strutture dati, poi superato felicemente al primo appello: la prova costume. Per rimettersi in forma per questo asfissiante evento Matteo ha deciso di andare in università correndo, e pianifica il percorso che ritiene il migliore per lo scopo. Decide che la soluzione migliore è scegliere una strada che vada per una parte in salita e poi una seconda parte completamente in discesa, in modo tale da bruciare e sudare di più nella parte in salita, e poi prendere una leggera brezza nella parte in discesa correndo velocemente fino l'università. La corsa inizierà dalla sua abitazione e terminerà in università, e tutto il suo percorso è dettagliato su una mappa dove ha segnato le strade con **m** segmenti stradali (cioè una strada tra due intersezioni) e **n** intersezioni. Ogni segmento stradale ha una lunghezza positiva e ogni intersezione ha un valore che indica la sua elevazione. Inoltre non esistono due intersezioni che si trovano allo stesso livello.

1. Assumendo che ogni segmento stradale può essere classificato come segmento in salita oppure in discesa, sviluppare un algoritmo **efficiente** per trovare la strada più breve che soddisfi i vincoli descritti sopra.

2. (Opzionale*) Dare un algoritmo **efficiente** per risolvere il problema in cui qualche strada si trova allo stesso livello (cioè le intersezioni alla fine di un segmento stradale si trovano su una stessa elevazione) e perciò può essere preso in qualsiasi punto.

Strategie di Risoluzione e Strutture Dati Utilizzate:

Le **strutture dati** utilizzate per la realizzazione del primo quesito, sono:

- Il grafo è contenuto dalla struttura **struct TGraph**, essa a sua volta possiede, un intero che indica il numero di intersezioni esistenti e una lista di tipo **TList** che rappresenta il numero di archi che possiede ciascun nodo**. Tale struttura verrà utilizzata anche per salvare le strade che comporranno il percorso minimo dalla Casa a un nodo *n* sopraelevato rispetto all'Università. Successivamente, le strade che comporranno il percorso minimo completamente in discesa dall'intersezione *n* all'Università.**

```
struct TGraph
{
    List* adj;
    int n_intersezioni;
};
```

- La lista concatenata **TList,** specificherà una sequenza di **segmenti stradali** ad una relativa intersezione, essi vengono rappresentati: con un intero "**target"** che indica il vertice di destinazione dell'arco, con un intero "**lunghezza"** che indica la dimensione dell'arco.

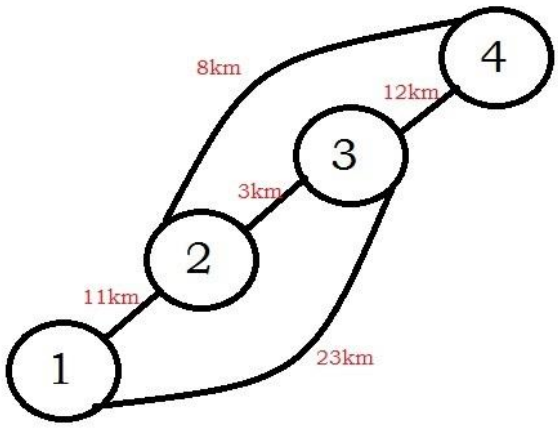
```
struct TList
{
    int target;
    int lunghezza;
    struct TList* next;
};
```

La **strategia** adottata per compiere il primo punto della traccia, prevede di creare un **grafo connesso**, con vertici in ordine crescente. Immaginando che ciascun vertice indichi un luogo, il proprio **indice** rappresenterà il grado di elevazione in cui si troverà il suddetto posto.

Le posizioni delle intersezioni relative alla **Casa** e **all'Università**, vengono sorteggiate secondo dei vincoli:

- L'intersezione della "Casa" viene posta nel nodo meno elevato.
- L'Università, quindi, viene sorteggiata in una posizione compresa tra "$Casa$" e $(V - 1)$ con V numero d'intersezioni.

Secondo quanto detto precedentemente, seguendo l'esempio mostrato dall'immagine adiacente, il **vertice 1** è situato ad una elevazione minore al **vertice 2,** cosi come il **vertice 4** si troverà più in alto del **vertice 3.**



Ammesso che Matteo abbia casa nel **vertice 1** e l'Università si trovi nel **vertice 3, dovrà attraversare in salita la strada di 11km che lo collocherà nel posto 2, successivamente dovrà risalire per la strada di 8km che lo condurrà nel vertice 4. Successivamente, una volta terminata la sua salita, incomincerà a scendere per la strada di 12km, fin quando non arriva al vertice 3, ovvero all'Università.**

In questo caso, il percorso, che da casa condurrà, Matteo, all'Università, è abbastanza banale e possibile, ma bisogna anche considerare i casi in cui la creazione randomica del grafo, non genererà delle soluzioni stradali che porteranno alla realizzazione del punto richiesto. Ciò si verificherà quando non sarà possibile raggiungere un'intersezione n posta più in alto rispetto all'Università.

Dettagli Implementativi:

Funzioni Principali relative alla gestione del Grafo.

- Graph randomGraphConnesso(int n , int m)
  - Input: Riceve in input: n e m che rappresenteranno il numero di intersezioni e il numero di segmenti stradali
  - Spiegazione: La funzione genera un Grafo connesso, formando randomicamente m archi. Il minimo numero di segmenti stradali possibile è $n - 1$.

- void addEdge(Graph G, int intersezione, int target, int lunghezza)
  - Input: Riceve in input: un grafo, un vertice mittente, un vertice destinatario, e la lunghezza dell'arco.
  - Spiegazione: Aggiunge un arco bidirezionale (strada) di lunghezza inserita, tra il vertice mittente e il destinatario. Tenendo conto che, i due vertici presi in input siano diversi, e che quindi non esistano archi che partono e arrivano nella stessa intersezione.

- void addEdgeforG2(Graph G, int intersezione, int target, int lunghezza)
  - Input: Riceve in input: un grafo, un vertice mittente, un vertice destinatario, e la lunghezza dell'arco.
  - Spiegazione: Aggiunge un arco di lunghezza inserita, tra il vertice mittente e il destinatario. Tale funzione servirà per tenere traccia delle strade che comporranno il percorso minimo che eseguirà Matteo, prima in salita e successivamente in discesa, per arrivare a destinazione.

Funzioni Principali relative alla risoluzione del punto 1.

- void Calcola_percorso_min_salita(Graph G, int CASA, int UNI)
  - Input: Riceve in input: un grafo, il vertice in cui c'è la CASA e il vertice in cui c'è UNI.
  - Spiegazione: Calcola il percorso minimo tra l'intersezione "Casa" e un intersezione posta più in alto rispetto a quest'ultima.

- void Calcola_percorso_min_discesa(Graph G, int partenza, int UNI)
  - Input: Riceve in input: un grafo, il vertice in cui c'è la partenza (ovvero l'intersezione posta più in alto dell'Università, determinata dalla funzione **Calcola_percorso_min_salita**) e il vertice in cui c'è l'Università.
  - Spiegazione: Calcola il percorso minimo in completamente in discesa tra l'intersezione "Casa" e un l' "Uni".

- void STAMPA_PERCORSO(Graph G, int min, int partenza)
  - Input: Riceve in input: un grafo, il vertice di destinazione e il vertice relativo alla partenza.
  - Spiegazione: Stampa tutte le indicazioni stradali che comporranno il percorso seguito da Matteo per giungere a destinazione.

Definizione Complessità:

In risposta al punto 1, per trovare il percorso minimo tra la Casa e l'Università, vengono eseguite tali funzioni:

void Calcola_percorso_min_salita(Graph G, int CASA, int UNI)

1. Scorre l'array dist[] inizializzando i suoi valori a INT_MAX : $O(V)$
2. Scorre l'intero Grafo per determinare il percorso minimo: $O(|V| + |E|)$

$La\ complessità$ è $O(V) + O(|V| + |E|) = O(|V| + |E|)$.

void Calcola_percorso_min_discesa(Graph G, int partenza, int UNI)

1. Scorre l'array dist[] inizializzando i suoi valori a INT_MAX : $O(V)$
2. Scorre l'intero Grafo per determinare il percorso minimo: $O(|V| + |E|)$, con V s'intende il numero di intersezioni tra il nodo trovato da Calcola_mercorso_min_salita e l'Università.

$La\ complessità$ è $O(V) + O(|V| + |E|) = O(|V| + |E|)$.

Da ciò si deduce che: $La\ complessità\ Totale$ è $O(|V| + |E|) + O(|V| + |E|) = O(|V| + |E|)$.

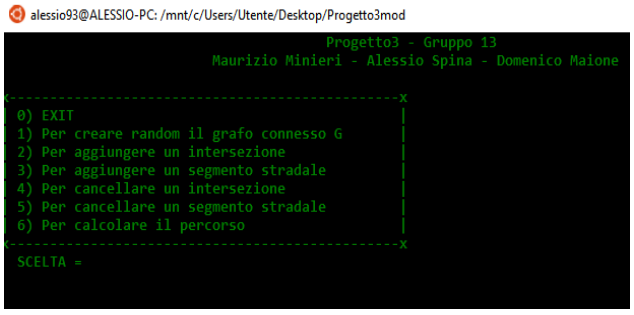
Manuale D'uso:

Il progetto conterrà i seguenti file:

- **Funzioni.c** e **Funzioni.h** – Funzioni riguardarti la realizzazione del punto 1;
- **Graph.c** e **Graph.h –** Funzioni riguardanti la gestione dei grafi;
- **List.c** e **List.h –** Funzioni riguardanti la gestione delle liste concatenate.

On Linux:

Comando di Esecuzione: gcc main.c && ./a.out



On Windows:

Esecuzione: lanciare il file main.exe