Prova finale di algoritmi e strutture dati

Obiettivi didattici e realizzazione

- Obiettivi
  - Applicazione pratica delle tecniche apprese nel modulo di algoritmi e strutture dati del corso di algoritmi e principi dell'informatica
  - Implementazione di una soluzione ad un problema prestando attenzione ad aspetti concreti di efficienza del codice
- Realizzazione
  - Linguaggio C (C11, VLA ammessi)
  - Nessuna libreria esterna al di là della libreria standard C
  - No multithreading
  - Dati in ingresso ricevuti via stdin, risultati da fornire via stdout

Criteri di valutazione

- Correttezza ed efficienza della soluzione proposta sono valutate con batterie di test automatizzate
- Verranno forniti input/output d'esempio per poter collaudare la soluzione in locale
  - Non sottoponete soluzioni senza aver verificato che funzionino localmente
  - Verrà fornito anche uno strumento di generazione automatica di casi di test (input/output), per facilitarvi il testing in locale
- Il sistema di verifica calcola il tempo macchina e la memoria utilizzati
- La valutazione è immediatamente calcolata (e subito visibile), mediante 6 batterie di test (task, nel lessico del verificatore):
  - Ogni batteria ha una valutazione associata tra queste: {18,21,24,27,30,30 e lode}
  - Per ottenere una valutazione X è necessario superare *tutte le batterie con valutazione associata <= X*

CercaPercorso

- Autostrada: lista di stazioni di servizio
  - Ogni stazione è identificata dalla sua distanza (numero intero) dall'inizio dell'autostrada
  - Ogni stazione è dotata di un insieme di veicoli elettrici con autonomia data da un intero positivo
- Obiettivo:
  - Data una coppia di stazioni, identificare il percorso per arrivare dalla prima alla seconda nel più basso numero di tappe
  - Ad ogni tappa effettuata, è necessario cambiare veicolo, usandone uno tra quelli disponibili nella stazione di servizio

Comandi e risposte attese

- `aggiungi-stazione` *`distanza numero-auto auto-1 ... auto-n`*
  - Aggiunge una stazione all'autostrada, identificata da *distanza* e avente *numero-auto* veicoli. Le autonomie di ogni veicolo sono elencate dopo il numero di veicoli. Se esiste già una stazione alla distanza data, non viene effettuata l'aggiunta.
  - Output atteso: `aggiunta` / `non aggiunta`
- `demolisci-stazione` *`distanza`*
  - Rimuove la stazione di servizio presente alla distanza indicata, se è presente una stazione alla distanza indicata, non fa nulla altrimenti, salvo stampare "non demolita"
  - Output atteso: `demolita` / `non demolita`

Comandi e risposte attese

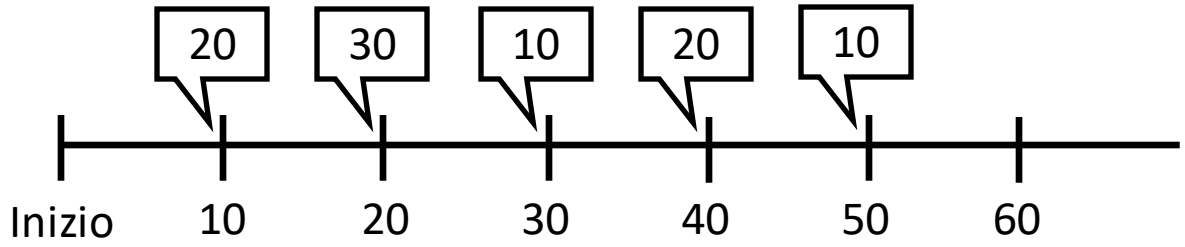
- `aggiungi-auto` *`distanza-stazione autonomia-auto`*
  - Aggiunge un'auto con l'autonomia specificata alla stazione a distanza data, posto che la stazione esista. N.B. è possibile avere più di un'automobile con la stessa autonomia.
  - Output atteso: `aggiunta` / `non aggiunta`
- `rottama-auto` *`distanza-stazione autonomia-auto`*
  - Rimuove un'auto con l'autonomia indicata dalla stazione la cui distanza è indicata. Se la stazione non esiste, o non esiste un'automobile con l'autonomia indicata nel parco veicoli della stazione stessa, non fa nulla se non stampare "non rottamata"
  - Output atteso: `rottamata` / `non rottamata`

Comandi e risposte attese

- `pianifica-percorso` *`stazione-partenza stazione-arrivo`*
  - Pianifica il percorso con il numero minore di tappe tra la stazione di partenza e quella di arrivo, posto che ne esista uno. Le stazioni sono identificate nel comando dalla loro distanza
  - Output atteso:
    - Se il percorso esiste: sequenza delle stazioni che compongono il percorso, in ordine di percorrenza, incluse la stazione di partenza e quella di arrivo. Le stazioni sono stampate come numeri interi, separati da spazi, su un solo rigo
    - Se il percorso non esiste: `nessun percorso`
  - Attenzione: l'autostrada ha due sensi di percorrenza!

Percorsi ambigui

- Tra due stazioni ci possono essere più percorsi lunghi uguali
- L'implementazione deve scegliere il percorso che nella sua parte finale predilige sempre le stazioni più vicine all'inizio dell'autostrada
  - Cioè le stazioni con il numero più basso possibile
  - Questa regola non viene influenzata dal senso di percorrenza
- Esempio:



Il percorso corretto tra 10 e 60 è 10→20→40→60

Non è corretto 10→30→40→60 perché 30 > 20

Non è corretto 10→20→50→60 perché 50 > 40

Esempio di ingresso e risposte attese

Testo in ingresso (stdin)	Risposta attesa	Commento
aggiungi-stazione 20 3 5 10 15	aggiunta	Aggiunta staz. km 20
aggiungi-stazione 4 3 1 2 3	aggiunta	Aggiunta staz. km 4
aggiungi-stazione 30 0	aggiunta	Aggiunta staz. km 30
demolisci-stazione 3	non demolita	Non esiste staz. km 3
demolisci-stazione 4	demolita	Demolita staz. km 4
pianifica-percorso 30 20	nessun percorso	Non esiste percorso da 30
aggiungi-auto 30 40	aggiunta	Aggiunta auto aut. 40
aggiungi-stazione 50 3 20 25 7	aggiunta	Aggiunta staz. km 50
rottama-auto 20 8	non rottamata	Non esiste auto aut. 8
rottama-auto 9999 5	non rottamata	Non esiste staz. km 9999
rottama-auto 50 7	rottamata	Rottamata auto aut. 7
pianifica-percorso 20 30	20 30	Usa auto con aut. 15
pianifica-percorso 20 50	20 30 50	Usa auto con aut. 15 e 40
pianifica-percorso 50 30	50 30	Usa auto con aut. 25
pianifica-percorso 50 20	50 30 20	Usa auto con aut. 25 e 40
aggiungi-auto 50 30	aggiunta	Aggiunta auto aut. 30
pianifica-percorso 50 20	50 20	Usa auto con aut. 30 e 40