

# Batterie

- Siete a bordo di un'auto elettrica su un'autostrada. Entrate in autostrada al km 0 con la batteria carica e dovete uscire al km  $L$ .
- Prima che la vostra batteria si esaurisca (dopo  $r$  km), dovete fermarvi in un'area di servizio e sostituirla con una batteria di ricambio.
- Sia  $D[1 \dots n]$  un vettore di interi, dove  $D[i]$  è la distanza dell'area di servizio  $i$ -esima dall'inizio dell'autostrada.
- Scrivete un algoritmo che prenda in input  $D$ ,  $n$ ,  $L$  e  $r$  e restituisca il numero minimo di fermate necessarie per completare il viaggio. Discutere correttezza e complessità.

# Batterie

- Siete a bordo di un'auto elettrica su un'autostrada. Entrate in autostrada al km 0 con la batteria carica e dovete uscire al km  $L$ .
- Prima che la vostra batteria si esaurisca (dopo  $r$  km), dovete fermarvi in un'area di servizio e sostituirla con una batteria di ricambio.
- Siano  $D[1 \dots n]$  e  $C[1 \dots n]$  due vettori di interi, dove  $D[i]$  è la distanza dell'area di servizio  $i$ -esima dall'inizio dell'autostrada, e  $C[i]$  è il costo di una nuova batteria nell'area  $i$ .
- Il costo totale del viaggio è dato dalla somma dei costi delle batterie sostituite per arrivare al km  $L$ .
- Scrivete un algoritmo che prenda in input  $D$  e  $C$ ,  $n$ ,  $L$ ,  $r$  e restituisca il costo totale minimo. Discutere correttezza e complessità.

# Supersequenza comune minimale

- Una stringa  $P$  è una supersequenza di una stringa  $T$  se  $T$  è una sottosequenza di  $P$ .
- Scrivere un algoritmo che restituisca la lunghezza della *supersequenza comune minimale* di due stringhe  $P$ ,  $T$ , ovvero la più piccola supersequenza di entrambe le stringhe.
- Discutere correttezza e complessità dell'algoritmo proposto.
- Esempio: L'unica supersequenza comune minimale di  $AB$  e  $BC$  è  $ABC$ , e la sua lunghezza è pari a 3.
- Esempio: Esistono due supersequenze comuni minimali di  $DAB$  e  $DCB$ , ovvero  $DACB$  e  $DCAB$ , e la loro lunghezza è pari a 4.

## $k$ -cammini (Compito 25/7/18)

Scrivere un algoritmo che prende in input un grafo  $G = (V, E)$ , un nodo  $s \in V$  e un intero  $k$  e stampa tutti i cammini contenuti nel grafo tali che:

- partano da  $s$ ,
- abbiano lunghezza esattamente  $k$ ,
- siano semplici (senza nodi ripetuti).

Discutere informalmente la correttezza della soluzione proposta e calcolare la complessità computazionale.

Spoiler alert!

## Batterie (Esercizio 1.3 di 14-greedy.pdf)

---

SET minStops(int[]  $D$ , int  $n$ , int  $L$ , int  $r$ )

---

int  $deadline = r$

SET  $stops = \text{Set}()$

for  $i = 2$  to  $n$  do

    if  $D[i] \geq deadline$  then  
         $stops.insert(i - 1)$   
         $deadline = D[i - 1] + r$

if  $deadline < L$  then

$stops.insert(n)$

return  $stops$

---

## Batterie (Esercizio 1.3 di 14-greedy.pdf)

- Assumiamo che l'autostrada sia percorribile, ovvero l'autonomia  $r$  sia sufficiente per andare:
  - da una stazione alla successiva
  - dal km 0 alla prima stazione
  - dall'ultima stazione al km  $L$
- Per minimizzare il numero di fermate, procediamo in modo greedy: cerchiamo la prima stazione che non può essere raggiunta con autonomia  $r$  e inseriamo la stazione precedente.
- Dobbiamo dimostrare:
  - Sottostruttura ottima
  - Scelta greedy
  - Correttezza nei casi limite e nei valori estremi
- La complessità dell'algoritmo è  $O(n)$ .

## Batterie (Esercizio 1.3 di 14-greedy.pdf)

### Casi limite

- La prima stazione deve essere raggiungibile con autonomia  $r$ , quindi partiamo dalla seconda
- Se l'ultima stazione a cui abbiamo fatto rifornimento non ci permette di raggiungere  $L$ , aggiungiamo la stazione precedente;
- Se esiste una stazione sola, ci sono due casi:
  - se  $r \geq L$ , la carica ci permette di uscire dall'autostrada; ma allora  $deadline \geq L$  e l'algoritmo restituisce l'insieme vuoto
  - se  $r < L$ , l'algoritmo restituisce l'unica stazione presente



## Batterie (Esercizio 1.3 di 14-greedy.pdf)

### Scelta greedy

- Assumiamo che esista una soluzione  $S$  che non includa la stazione  $k$ , l'ultima stazione tale che  $D[k] \leq r$ , a partire dall'inizio (km 0).
- sia  $i$  la prima stazione utilizzata in  $S$ , con  $i < k$ ;
- se sostituisco  $i$  con  $k$ , ottengo una soluzione  $S' = S - \{i\} \cup \{k\}$  che è comunque una soluzione ottima (ha la stessa dimensione) e rispetta tutti i vincoli ( $k$  è raggiungibile dall'inizio e può raggiungere la stazione successiva in  $S$ , perchè questa era raggiungibile da  $i$  che era più indietro).

## Batterie (Esercizio 1.8 di 13-pd.pdf)

È possibile definire ricorsivamente il problema come segue. Assumiamo che esistano due stazioni fittizie  $0, n + 1$ , con  $D[0] = 0$  e  $D[n + 1] = L$  e  $C[0] = C[n + 1] = 0$ . Non è quindi necessario passare  $L$ .

Definiamo una tabella  $DP[0 \dots n + 1]$ , tale che  $DP[i]$  rappresenti il costo minimo da pagare nel caso si acquisti la batteria alla stazione  $i$ -esima. La nostra soluzione si trova in  $DP[0]$ .  $DP[i]$  può essere definito in maniera ricorsiva nel modo seguente:

$$DP[i] = \begin{cases} 0 & \text{se } i = n + 1 \\ \min_{j: j > i \wedge D[j] \leq D[i] + r} \{DP[j]\} + C[i] & \text{altrimenti} \end{cases}$$

## Batterie (Esercizio 1.8 di 13-pd.pdf)

---

```
int minCostStops(int[] D, int[] C, int n, int r)
```

---

```
int[] DP = new int[0...n + 1]
```

```
DP[n + 1] = 0
```

```
for i = n downto 0 do
```

```
    int j = i + 1
```

```
    int DP[i] =  $+\infty$ 
```

```
    while j ≤ n + 1 and D[j] ≤ D[i] + r do
```

```
        DP[i] = min(DP[i], DP[j])
```

```
        j = j + 1
```

```
    DP[i] = DP[i] + C[i]
```

```
return DP[0]
```

---

## Batterie (Esercizio 1.8 di 13-pd.pdf)

Inviduare l'indice  $j$  per cui  $D[j] \leq D[i] + r$  può richiedere  $O(n)$ ; quindi il costo pessimo di tale algoritmo è  $O(n^2)$ .

## Supersequenza comune minimale (Compito 10/01/2017)

La lunghezza della più lunga supersequenza comune può essere calcolata tramite la seguente espressione ricorsiva:

$$DP[i][j] = \begin{cases} i & i \geq 0 \wedge j = 0 \\ j & i = 0 \wedge j \geq 0 \\ DP[i-1][j-1] + 1 & i > 0 \wedge j > 0 \wedge p_i = t_j \\ \min\{DP[i-1][j], DP[i][j-1]\} + 1 & i > 0 \wedge j > 0 \wedge p_i \neq t_j \end{cases}$$

- Nel caso una delle stringhe sia vuota, tutti i caratteri dell'altra stringa devono essere presenti nella supersequenza, da cui i due casi base.
- Se gli ultimi caratteri delle due stringhe sono uguali, si considera il sottoproblema in cui viene rimosso l'ultimo carattere di entrambi e si aggiunge +1 per contare questo carattere.
- Nel caso siano diversi, si prenderanno i due sottoproblemi in cui si rimuove l'ultimo carattere di una delle stringhe e si aggiunge +1 per contare questo carattere, prendendo il valore più piccolo fra i due.

## Supersequenza comune minimale (Compito 10/01/2017)

---

```
int scs(ITEM[] P, ITEM[] T, int n, int m)  
  
int[][] D = new int[0...n][0...m]  
for i = 0 to n do  
     $DP[i][0] = i$   
  
for j = 0 to m do  
     $DP[0][j] = j$   
  
for i = 1 to n do  
    for j = 1 to m do  
        if  $P[i] == T[j]$  then  
             $DP[i][j] = DP[i - 1][j - 1] + 1$   
        else  
             $DP[i][j] = \min(DP[i - 1][j], DP[i][j - 1]) + 1$   
  
return  $DP[n][m]$ 
```

---

## $k$ -cammini

---

```
visit(GRAPH  $G$ , int  $k$ , int  $s$ )
```

---

```
boolean[] visited = new boolean[1 ...  $G.n$ ] = { false } % Init to false  
int[] path = new int[1 ...  $k + 1$ ]  
visitRec( $G$ ,  $k$ ,  $s$ , 1, path, visited)
```

---

---

```
visitRec(GRAPH  $G$ , int  $k$ , NODE  $u$ , int  $i$ , int[] path, boolean[] visited)
```

---

```
path[ $i$ ] =  $u$   
if  $i == k + 1$  then  
    | print path  
else  
    | visited[ $u$ ] = true  
    | foreach  $v \in G.\text{adj}(u)$  do  
    | | if not visited[ $v$ ] then  
    | | | visitRec( $G$ ,  $k$ ,  $i + 1$ ,  $v$ , path, visited)  
    | visited[ $u$ ] = false
```

---

## $k$ -cammini

- Struttura di una visita in profondità, con queste differenze:
  - Contatore per limitare profondità
  - Meccanismo di backtracking:  $visited[u] = \mathbf{false}$  quando si è completata la visita del nodo  $u$
- Caso pessimo: grafo completo
- Numero di cammini:
$$(n-1) \cdot (n-2) \cdot \dots \cdot (n-k) = \frac{(n-1)!}{(n-k-1)!} = O(n^k)$$
- Stampa cammino:  $\Theta(k)$
- Complessità:  $O(n^k \cdot k)$