ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

Programmazione

Andrea Piroddi

Dipartimento di Informatica, Scienza e Ingegneria

Elaborato 9

Un aspetto da considerare prima di passare all'implementazione è dove posizionare la testa del serpente nella lista concatenata:

in testa o coda?

Elaborato 9

- ***snake create():*** ***crea la struttura che rappresenta un serpente di lunghezza 1 (la testa del un serpente), posizionato in una qualsiasi cella in una matrice di dimensione rows×cols.***

```
struct snake *snake_create(unsigned int rows, unsigned int cols) {
          struct snake *s = (struct snake *)malloc(sizeof(struct snake));

                    // assegnate il numero di righe
                    // assegnate il numero di colonne

                    // verificate che il puntatore sia diverso da NULL
{
                    //posizionate il corpo del serpente
                    // date dimensione 1 alla lunghezza del corpo del serpente
{                   // se il corpo è NULL liberate la memoria e assegnate ad s il valore NULL

    }
  }
                    // restituite s
}
```

Elaborato 9

- ***snake kill():*** ***distrugge la struttura serpente (i.e. libera la memoria allocata dinamicamente).***

```
void snake_kill(struct snake *s) {

                // fintanto che la lunghezza del serpente >0 rimuovete la coda del serpente
                // infine liberate la memoria
}
```

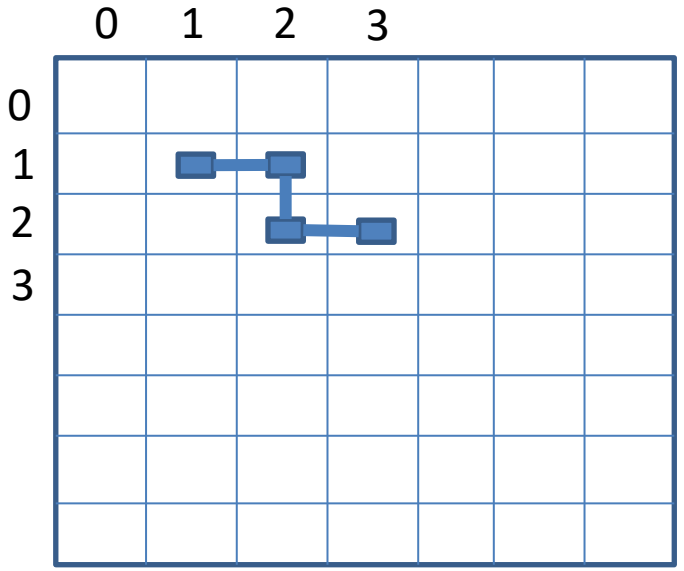
Elaborato 9

•***snake increase():*** ***aggiunge una nuova testa al serpente in una direzione specificata (up, down, right, left) rispetto alla testa attuale.***
Ad esempio, partendo dal set di coordinate

{(1,1), (1,2), (2,2), (2,3)}

dove assumiamo che la testa sia nella posizione (1,1), il nuovo set di coordinate in direzione down è equivalente al seguente set di coordinate

{(2,1),(1,1), (1,2), (2,2), (2,3)}




```
void snake_increase(struct snake *s, enum direction dir) {

}
```

Come sono calcolate le nuove posizioni rispetto alle direzioni **up, down, left, right**?

Ricordiamo che il mondo è toroidale di dimensione rows × cols: possiamo attraversare i bordi per sbucare dalla parte opposta.

- up: la posizione (i, j) diventa ((i − 1+rows) % rows, j)
- down: la posizione (i, j) diventa ((i + 1+rows) % rows, j)
- left: la posizione (i, j) diventa (i, (j − 1+cols) % cols)
- right: la posizione (i, j) diventa (i, (j + 1+cols) % cols)

Elaborato 9

- ***snake increase():*** ***aggiunge una nuova testa al serpente in una direzione specificata (up, down, right, left) rispetto alla testa attuale.***

Ad esempio, partendo dal set di coordinate

{(1,1), (1,2), (2,2), (2,3)}

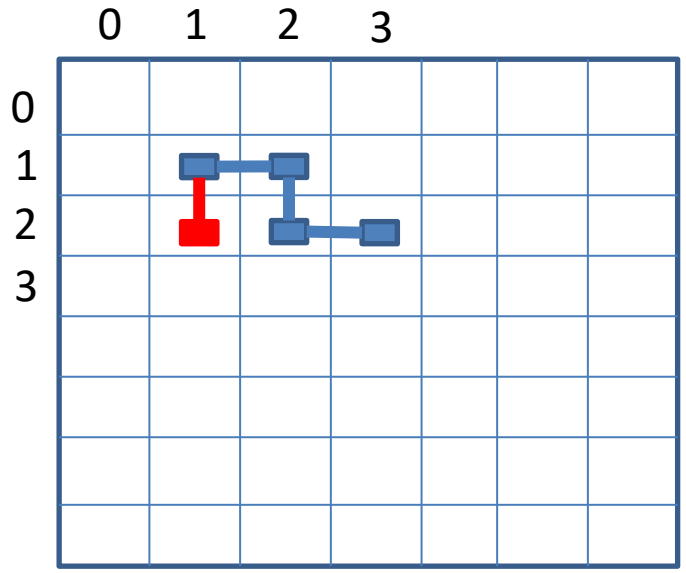
dove assumiamo che la testa sia nella posizione (1,1), il nuovo set di coordinate in direzione down è equivalente al seguente set di coordinate

{(2,1),(1,1), (1,2), (2,2), (2,3)}

```
void snake_increase(struct snake *s, enum direction dir) {
        //definite i e j a cosa corrispondono
        // definite un puntatore temporaneo

        switch(dir) { // a seconda della direzione aggiornate il valore della variabile
i o j
                …
                …
                …
        }

        // al puntatore temporaneo assegnate la nuova posizione della testa del serpente
        // aggiornate il puntatore reale
        // aggiornate la lunghezza del serpente
}
```



Elaborato 9

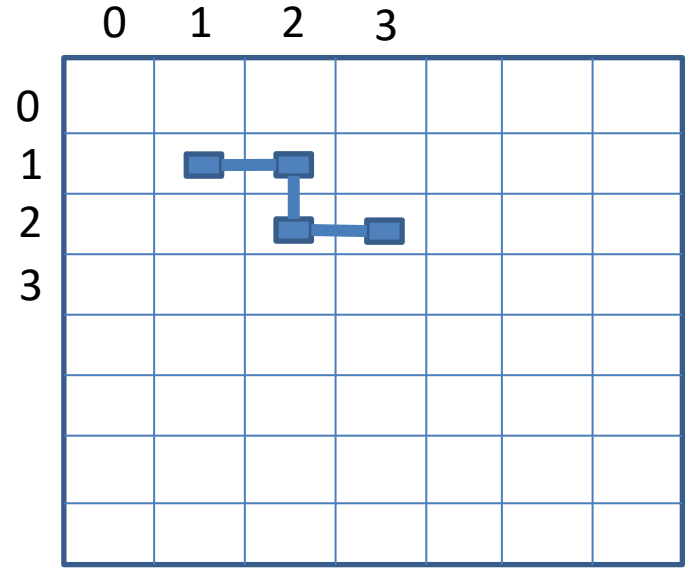
- ***snake decrease():*** ***rimuove un determinato numero di parti del serpente, a partire dalla coda.***

Ad esempio, la rimozione di una singola posizione nel set di coordinate

{(1,1), (1,2), (2,2), (2,3)}

produce la seguente lista

{(1,1), (1,2), (2,2)}




```
void snake_decrease(struct snake *s, unsigned int decrease_len) {

}
```

Elaborato 9

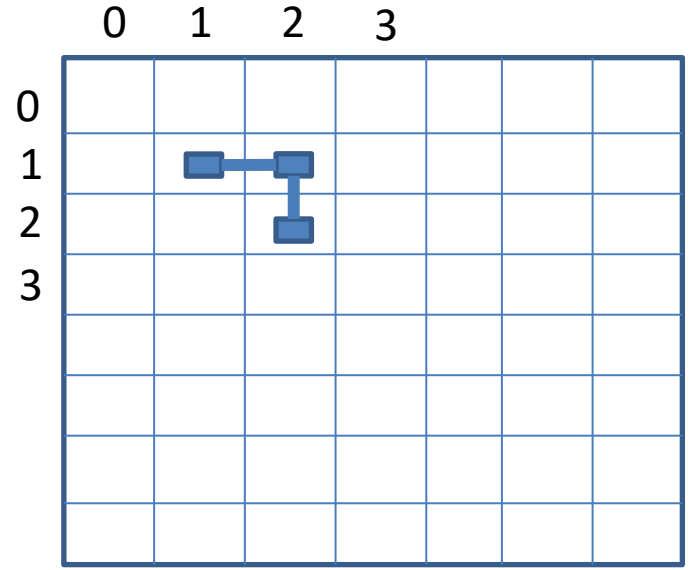
- ***snake decrease():*** ***rimuove un determinato numero di parti del serpente, a partire dalla coda.***

Ad esempio, la rimozione di una singola posizione nel set di coordinate

{(1,1), (1,2), (2,2), (2,3)}

produce la seguente lista

{(1,1), (1,2), (2,2)}




```
void snake_decrease(struct snake *s, unsigned int decrease_len) {

        // fintanto che il puntatore è NON NULL e la lunghezza del serpente è >0

        //rimuove il pezzo di coda (remove_tail)
}
```

Elaborato 9

- ***snake move():*** ***sposta il serpente in una direzione (up, down, right, left). Tale funzione produce un nuovo set di posizioni per l'oggetto serpente equivalente al set di coordinate ottenuto con la seguente procedura:***
  1. ***aggiungiamo una nuova testa nella direzione dir rispetto alla vecchia testa,***
  2. ***rimuoviamo la posizione in coda***

```
void snake_move(struct snake *s, enum direction dir) {

        // utilizzando l'increase aggiungiamo una nuova testa
       // rimuoviamo la coda

}
```

Elaborato 9

- ***snake reverse():*** ***inverte il serpente (i.e. la coda diventa la testa e viceversa).***

Ad esempio, una lista con posizioni

{(1,1), (1,2), (2,2), (2,3)}

dopo una chiamata alla snake reverse() diventa

{(2,3), (2,2), (1,2), (1,1)}

```
void snake_reverse(struct snake *s) {

        // dopo aver verificato che il corpo del serpente non sia NULL
{
                // dopo aver definito dei puntatori di appoggio

                // con un ciclo while invertite il serpente
{
                                        }
        // restituite il nuovo corpo del serpente
        }
}
```

Elaborato 9

- ***snake_head():*** *restituisce le coordinate della testa del serpente (ossia la posizione attuale)*

```
struct position snake_head(struct snake *s) {

        // si ricordi che pos è un campo della struttura body

}
```

Elaborato 9

- ***snake_body():*** ***Restituisce la (posizione della) parte i-esima del corpo del serpente. La posizione 0 equivale alla testa del serpente. Se il serpente è più corto di i, la posizione restituita non è definita. In altre parole ritorna la posizione attuale di una parte del corpo del serpente, dove si assume che la testa sia in posizione 0.***

```
struct position snake_body(struct snake *s, unsigned int i) {
        struct body *b = s->body;

         // facciamo un ciclo per assegnare il valore next a b

        // restituiamo la posizione
}
```

Elaborato 9

- ***snake_knotted():*** ***Restituisce 1 se il serpente attraversa se stesso, 0 altrimenti. In altre parole verifica che il serpente non si sia annodato. Per verificare se il serpente è annodato è sufficiente verificare che le coordinate della testa non siano ripetute nella lista.***

```
int snake_knotted(struct snake *s) {

        //utilizzando il puntatore search, dopo aver verificato che sia !=NULL
}
```

Elaborato 9

- ***snake save():*** *salva le coordinate del serpente in un file. Il formato di salvataggio è libero.*

```
void snake_save(struct snake *s, char *filename) {
        FILE *out = fopen(filename,"w");

        //se i due puntatori sono non NULL
{
                // utilizziamo un puntatore temporaneo
                // fintanto che il puntatore è non NULL
{
                        // scriviamo sullo Stream la posizione del serpente
                        // aggiorniamo il puntatore temporaneo con il valore successivo
                                }
        }
        // chiudiamo lo Stream
}
```

Elaborato 9

- ***snake load():*** *carica le coordinate del serpente da un file.*

```
struct snake *snake_load(char *filename) {
        struct snake *s = (struct snake *)calloc(1,sizeof(struct snake));
        FILE *in = fopen(filename,"r");

        // se il puntatore è non NULL
{
                // fintanto che la scansione dello Stream è diversa da EOF recuperiamo le posizioni i e j
{
                        //aggiorniamo la posizione del serpente
                        //aggiorniamo la lunghezza del serpente
                }
        }
        // ritorniamo il puntatore
}
```