Corso di Programmazione

II Accertamento del 29 Giugno 2020 / A

cognome e nome

Riporta in modo chiaro negli appositi spazi le soluzioni degli esercizi, oppure precise indicazioni se alcune soluzioni si trovano in un foglio separato. Scrivi inoltre il tuo nome nell'intestazione e su ciascun ulteriore foglio che intendi consegnare.

Selezione degli esercizi proposti

attraverso la piattaforma Moodle

2. Programmazione dinamica

Considera la seguente procedura funzionale (metodo statico), basata su una ricorsione ad albero:

```
public static long rec( int x, int y, int z ) {  // 1 <= x, y <= z

  if ( (x == 1) || (y == z) ) {
    return 1;
  } else {
    return rec( x-1, y, z ) + x * rec( x, y+1, z );
  }
}
```

2.1. Supponi che nel corso dell'esecuzione di un programma che utilizza `rec` venga valutata l'espressione:

```
rec( 8, 5, 12 )
```

Questa valutazione si svilupperà attraverso successive invocazioni ricorsive di `rec(`*x*`,`*y*`,`*z*`)` per diversi valori degli argomenti *x*, *y* e *z*. Quali sono il valore più piccolo e il valore più grande che assumerà ciascuno degli argomenti nelle ricorsioni che discendono da `rec(8,5,12)` ?

- Valore più piccolo di *x* : ..... e valore più grande di *x* : ..... ;
- Valore più piccolo di *y* : ..... e valore più grande di *y* : ..... ;
- Valore più piccolo di *z* : ..... e valore più grande di *z* : ..... .

2.2. Applica una tecnica di programmazione dinamica *bottom-up* (iterativa) per realizzare una versione più efficiente della procedura `rec`.

3. Ricorsione e iterazione

Il seguente programma, basato sulla ricorsione, risolve il rompicapo della *torre di Hanoi* a partire da una qualsiasi disposizione valida dei dischi in corrispondenza alle tre asticelle, e restituisce la sequenza di mosse codificata da una stringa. Per tenere traccia dello stato del gioco viene utilizzato un oggetto di tipo `Towers`, che inizialmente rappresenta la configurazione da cui si intende partire e successivamente, attraverso il metodo `moves`, permette di acquisire la stringa che rappresenta le mosse effettuate. (Ai fini di questo esercizio non è comunque necessario conoscere le specifiche del protocollo di `Towers`).

```
public static String hanoi( Towers hts, int d ) {  // hts: stato iniziale gioco
                                                   // d:   posizione finale torre
  hanoiRec( hts.height(), d, hts );
  return hts.moves();
}

private static void hanoiRec( int n, int d, Towers hts ) {

  if ( n > 0 ) {
    if ( hts.site(n) == d ) {
      hanoiRec( n-1, d, hts );
    } else {
      int t = hts.transit( n, d );
      hanoiRec( n-1, t, hts );
      hanoiRec(  -n, d, hts );
      hanoiRec( n-1, d, hts );
    }
  } else if ( n < 0 ) {
    hts.move( -n, d );
  }
}
```

3.1. Completa la definizione del metodo `hanoiIter` che trasforma la ricorsione in iterazione applicando uno stack.

```
public static String hanoiIter( Towers hts, int d ) {
  int n = hts.height();
  Stack<int[]> stk = new Stack<int[]>();
  stk.push( new int[]{ n, d } );
  while ( ..... ) {
    int[] f = ..... ;
    n = ..... ;
    d = ..... ;
    if ( n > 0 ) {
      if ( hts.site(n) == d ) {
        stk.push( new int[]{ ..... } );
      } else {
        int t = hts.transit( n, d );
        stk.push( ..... );
        ..... ;
        ..... ;
      }
    } else if ( n < 0 ) {
      ..... ;
    }
  }
  return hts.moves();
}
```

3.2. In base al codice di `hanoiRec`, il parametro *n* può anche assumere valori negativi. Quale potrebbe esserne la funzione, a tuo giudizio? Spiega brevemente la tua interpretazione.

4. Definizione di classi Java

Un'istanza della classe `Towers` rappresenta lo stato del rompicapo della *torre di Hanoi* e permette di modellare l'evoluzione del gioco, attraverso successive disposizioni valide di *n* dischi, tenendo traccia delle mosse effettuate. I dischi sono numerati da 1 a *n* in ordine crescente di diametro; le tre asticciole previste dal rompicapo sono identificate da 1, 2 e 3. Il protocollo della classe è specificato come segue:

- `public Towers( int n )`
  costruttore, il cui parametro *n* indica il numero di dischi, ovvero l'altezza della torre da ricostruire.
- `public void put( int disk, int rod )`
  metodo per stabilire la posizione iniziale del disco *disk* in corrispondenza all'asticella *rod*, prima di giocare.
- `public void move( int disk, int dst )`
  metodo che consente di modificare lo stato del gioco effettuando una mossa che sposta il disco *disk* dall'asticella in cui si trova all'asticella di destinazione *dst*.
- `public int height()`
  metodo che restituisce l'altezza della torre da ricostruire (o equivalentemente il numero di dischi del gioco).
- `public int site( int disk )`
  metodo che restituisce il numero che identifica l'asticella in corrispondenza alla quale è collocato il disco *disk*.
- `public int transit( int disk, int dst )`
  metodo che restituisce il numero che identifica l'asticella *di transito* per il disco *disk* con destinazione *dst*: l'asticella di transito è quella che rimane escludendo l'asticella in cui è collocato *disk* e l'asticella *dst*.
- `public String moves()`
  metodo per acquisire la stringa che codifica la sequenza di mosse effettuate.

Nella stringa restituita da `moves`, una mossa dall'asticella *src* all'asticella *dst* è codificata da cinque caratteri: uno spazio bianco, seguito dalla cifra che denota *src*, seguita dalla coppia di caratteri `"->"` e dalla la cifra che denota *dst*. Inoltre, le codifiche di mosse successive sono semplicemente giustapposte. Per esempio, le 7 mosse che risolvono il rompicapo per tre dischi, inizialmente tutti collocati in corrispondenza all'asticella 1, ricostruendo la torre in corrispondenza all'asticella 2 sono rappresentate dalla stringa:

```
" 1->2 1->3 2->3 1->2 3->1 3->2 1->2"
```

La classe `Towers` può essere impostata in Java come segue:

```
public class Towers {

  // Variabili di istanza ...

  // Costruttore ...

  // Metodi ...

  public int transit( int disk, int dst ) {
    return ( 6 - site(disk) - dst );
  }

}  // class Towers
```

4.1. Introduci una rappresentazione interna adatta al fine di realizzare le funzionalità specificate dal protocollo, definendo con precisione le variabili di istanza e il costruttore della classe `Towers`.

4.2. Una semplice definizione del metodo `transit` è riportata sopra. Definisci gli altri cinque metodi del protocollo di `Towers`, in accordo alle specifiche fornite.

Corso di Programmazione

II Accertamento del 29 Giugno 2020 / B

cognome e nome

Riporta in modo chiaro negli appositi spazi le soluzioni degli esercizi, oppure precise indicazioni se alcune soluzioni si trovano in un foglio separato. Scrivi inoltre il tuo nome nell'intestazione e su ciascun ulteriore foglio che intendi consegnare.

Selezione degli esercizi proposti

attraverso la piattaforma Moodle

2. Programmazione dinamica

Considera la seguente procedura funzionale (metodo statico), basata su una ricorsione ad albero:

```
public static long rec( int t, int u, int v ) {  // 1 <= u, v <= t

  if ( (u > 1) && (v < t) ) {
    return rec( t, u-1, v ) + u * rec( t, u, v+1 );
  } else {
    return 1;
  }
}
```

2.1. Supponi che nel corso dell'esecuzione di un programma che utilizza `rec` venga valutata l'espressione:

```
rec( 14, 10, 7 )
```

Questa valutazione si svilupperà attraverso successive invocazioni ricorsive di `rec(`*t*`,`*u*`,`*v*`)` per diversi valori degli argomenti *t*, *u* e *v*. Quali sono il valore più piccolo e il valore più grande che assumerà ciascuno degli argomenti nelle ricorsioni che discendono da `rec(14,10,7)` ?

- Valore più piccolo di *t* : ..... e valore più grande di *t* : ..... ;
- Valore più piccolo di *u* : ..... e valore più grande di *u* : ..... ;
- Valore più piccolo di *v* : ..... e valore più grande di *v* : ..... .

2.2. Applica una tecnica di programmazione dinamica *bottom-up* (iterativa) per realizzare una versione più efficiente della procedura `rec`.

3. Ricorsione e iterazione

Il seguente programma, basato sulla ricorsione, risolve il rompicapo della *torre di Hanoi* a partire da una qualsiasi disposizione valida dei dischi in corrispondenza alle tre asticelle, e restituisce la sequenza di mosse codificata da una stringa. Per tenere traccia dello stato del gioco viene utilizzato un oggetto di tipo `Towers`, che inizialmente rappresenta la configurazione da cui si intende partire e successivamente, attraverso il metodo `moves`, permette di acquisire la stringa che rappresenta le mosse effettuate. (Ai fini di questo esercizio non è comunque necessario conoscere le specifiche del protocollo di `Towers`).

```
public static String hanoi( Towers hts, int t ) {  // hts: stato iniziale gioco
                                                   // t:   posizione finale torre
  hanoiRec( t, hts.height(), hts );
  return hts.moves();
}

private static void hanoiRec( int t, int n, Towers hts ) {

  if ( n != 0 ) {
    if ( n < 0 ) {
      hts.move( -n, t );
    } else if ( hts.site(n) == t ) {
      hanoiRec( t, n-1, hts );
    } else {
      int x = hts.transit( n, t );
      hanoiRec( x, n-1, hts );
      hanoiRec( t,  -n, hts );
      hanoiRec( t, n-1, hts );
  }}
}
```

3.1. Completa la definizione del metodo `hanoiIter` che trasforma la ricorsione in iterazione applicando uno stack.

```
public static String hanoiIter( Towers hts, int t ) {
  int n = hts.height();
  Stack<int[]> stk = new Stack<int[]>();
  stk.push( new int[]{ t, n } );
  while ( ..... ) {
    int[] f = ..... ;
    t = ..... ;
    n = ..... ;
    if ( n != 0 ) {
      if ( n < 0 ) {
        ..... ;
      } else if ( hts.site(n) == t ) {
        stk.push( new int[]{ ..... } );
      } else {
        int x = hts.transit( n, t );
        stk.push( ..... );
        ..... ;
        ..... ;
    }}
  }
  return hts.moves();
}
```

3.2. In base al codice di `hanoiRec`, il parametro *n* può anche assumere valori negativi. Quale potrebbe esserne la funzione, a tuo giudizio? Spiega brevemente la tua interpretazione.

4. Definizione di classi Java

Un'istanza della classe `Towers` rappresenta lo stato del rompicapo della *torre di Hanoi* e permette di modellare l'evoluzione del gioco, attraverso successive disposizioni valide di *n* dischi, tenendo traccia delle mosse effettuate. I dischi sono numerati da 1 a *n* in ordine crescente di diametro; le tre asticciole previste dal rompicapo sono identificate da 1, 2 e 3. Il protocollo della classe è specificato come segue:

- `public Towers( int n )`
  costruttore, il cui parametro *n* indica il numero di dischi, ovvero l'altezza della torre da ricostruire.
- `public void add( int rod )`
  metodo per stabilire la posizione iniziale di ciascun disco, prima di giocare, procedendo in ordine di diametro decrescente; il disco più grande fra i rimanenti viene collocato in corrispondenza all'asticella *rod*.
- `public void move( int disk, int trg )`
  metodo che consente di modificare lo stato del gioco effettuando una mossa che sposta il disco *disk* dall'asticella in cui si trova all'asticella di destinazione *trg*.
- `public int height()`
  metodo che restituisce l'altezza della torre da ricostruire (o equivalentemente il numero di dischi del gioco).
- `public int site( int disk )`
  metodo che restituisce il numero che identifica l'asticella in corrispondenza alla quale è collocato il disco *disk*.
- `public int transit( int disk, int trg )`
  metodo che restituisce il numero che identifica l'asticella *di transito* per il disco *disk* con destinazione *trg*: l'asticella di transito è quella che rimane escludendo l'asticella in cui è collocato *disk* e l'asticella *trg*.
- `public String moves()`
  metodo per acquisire la stringa che codifica la sequenza di mosse effettuate.

Nella stringa restituita da `moves`, una mossa che sposta il disco *disk* in corrispondenza all'asticella *trg* è codificata da quattro elementi: uno spazio bianco, seguito dal numero che denota *disk*, seguito dal carattere `":"` e dalla la cifra che denota *trg*. Inoltre, le codifiche di mosse successive sono semplicemente giustapposte. Per esempio, le 7 mosse che risolvono il rompicapo per tre dischi, inizialmente tutti collocati in corrispondenza all'asticella 1, ricostruendo la torre in corrispondenza all'asticella 2 sono rappresentate dalla stringa:

```
" 1:2 2:3 1:3 3:2 1:1 2:2 1:2"
```

La classe `Towers` può essere impostata in Java come segue:

```
public class Towers {

  // Variabili di istanza ...

  // Costruttore ...


  // Metodi ...


  public int transit( int disk, int trg ) {
    return ( 6 - site(disk) - trg );
  }


}  // class Towers
```

4.1. Introduci una rappresentazione interna adatta al fine di realizzare le funzionalità specificate dal protocollo, definendo con precisione le variabili di istanza e il costruttore della classe `Towers`.

4.2. Una semplice definizione del metodo `transit` è riportata sopra. Definisci gli altri cinque metodi del protocollo di `Towers`, in accordo alle specifiche fornite.