Αναδρομή

#11

Τμήμα Πληροφορικής και Τηλεπικοινωνιών
Πανεπιστήμιο Ιωαννίνων (Άρτα)
Γκόγκος Χρήστος

Διάσπαση (αποσύνθεση) προβλήματος

- Η διάσπαση προβλήματος (problem decomposition) είναι μια από τις βασικές τεχνικές επίλυσης προβλημάτων στον προγραμματισμό. Προκειμένου ένα μεγάλο πρόβλημα να γίνει ευκολότερα διαχειρίσιμο διασπάται σε μικρότερα και ευκολότερα προβλήματα. Η λύση του μεγάλου προβλήματος προκύπτει συνδυάζοντας τις λύσεις των μικρότερων προβλημάτων.
- Παράδειγμα 1:
  - Πρόβλημα: Ταξινόμησε την ακολουθία `A[1..N]`
  - Διάσπασε σε:
    - **Υποπρόβλημα 1**: Ταξινόμησε την υποακολουθία `A[2..N]` (μικρότερο πρόβλημα)
    - **Υποπρόβλημα 2**: Τοποθέτησε το `A[1]` στη σωστή θέση στην ταξινομημένη υποακολουθία `A[2..N]` (ευκολότερο πρόβλημα)

Διάσπαση προβλήματος

- Παράδειγμα 2:
  - Πρόβλημα (μέγεθος = N): Υπολογισμός του $\sum_{i=1}^{N} i^3$
  - Διάσπαση σε:
    - Υποπρόβλημα (μεγέθος = N-1): Υπολογισμός του X $= \sum_{i=1}^{N-1} i^3$
    - Η λύση είναι `X + N * N * N`
- Παράδειγμα 3:
  - Πρόβλημα: Εύρεση του αθροίσματος `A[1..N]`
  - Διάσπαση σε:
    - X = άθροισμα των `A[2..N]`
    - Η λύση είναι `X + A[1]`

Διάσπαση προβλήματος και αναδρομή

- Όταν ένα μεγάλο πρόβλημα μπορεί να λυθεί επιλύοντας μικρότερα προβλήματα της ίδια μορφής τότε η αναδρομή αποτελεί ένα φυσικό τρόπο για την αντιμετώπιση του προβλήματος.
- Η συγγραφή αναδρομικών συναρτήσεων μπορεί να γίνει ως μια διαδικασία 4 βημάτων (*κατανόηση αναδρομής, πρωτότυπο αναδρομικής συνάρτησης, βασική περίπτωση, αναδρομική περίπτωση*) που θα εφαρμοστεί στη συνέχεια για το ακόλουθο παράδειγμα:
  - Πρόβλημα: Εύρεση του αθροίσματος `A[1..N]`
  - Διάσπαση σε:
    - `X` = άθροισμα των `A[2..N]`
    - Η λύση είναι `X + A[1]`

Συγγραφή αναδρομικών συναρτήσεων (1/4)

- **Βήμα 1**: κατανόηση του πως ένα πρόβλημα μπορεί να διασπαστεί σε ένα σύνολο από μικρότερα προβλήματα της **ίδιας μορφής** καθώς και πως οι λύσεις των μικρότερων προβλημάτων μπορούν να συνδυαστούν για να σχηματίσουν τη λύση του μεγαλύτερου προβλήματος.
- Παράδειγμα:
  - Πρόβλημα: Εύρεση του αθροίσματος `A[1..N]`
  - Γενίκευση του προβλήματος στην εύρεση του αθροίσματος του `A[beg..end]`
  - Διάσπαση σε:
    - X = άθροισμα των A[beg+1..end]
    - Η λύση είναι `X + A[beg]`

Συγγραφή αναδρομικών συναρτήσεων (2/4)

- **Βήμα 2**: διατύπωση της λύσης του προβλήματος ως μια συνάρτηση που θα δέχεται τις κατάλληλες παραμέτρους. Η βασική ιδέα είναι ότι θα πρέπει τόσο το αρχικό πρόβλημα όσο και τα μικρότερα υποπροβλήματα να χρησιμοποιούν την ίδια συνάρτηση.
- Διατύπωση του προβλήματος ως τη συνάρτηση:
  - `sum(A, beg, end)` είναι το άθροισμα του `A[beg..end]` (το αρχικό πρόβλημα)
  - `sum(A, beg+1, end)` είναι το άθροισμα του `A[beg+1..end]` (υποπρόβλημα)
- Η συνάρτηση μπορεί να έχει την ακόλουθη μορφή (πρωτότυπο αναδρομικής συνάρτησης):

```
int sum(int A[], int beg, int end);
```

Συγγραφή αναδρομικών συναρτήσεων (3/4)

- **Βήμα 3**: Ορίζεται η βασική περίπτωση (base case) της αναδρομής. Συνήθως πρόκειται για τις εύκολες περιπτώσεις στις οποίες το μέγεθος του προβλήματος είναι 0 ή 1.
- `int sum(int A[], int beg, int end);`
  - Η βασική περίπτωση μπορεί να είναι το άθροισμα όταν δεν υπάρχει κάποιο στοιχείο στον πίνακα που να περιέχεται ανάμεσα στο δείκτη beg και στο δείκτη end.

```
if (beg>end) return 0;
```

Συγγραφή αναδρομικών συναρτήσεων (4/4)

- **Βήμα 4**: Ορίζεται η αναδρομική περίπτωση, δηλαδή ο συνδυασμός των λύσεων των μικρότερων προβλημάτων που σχηματίζει τη λύση στο αρχικό πρόβλημα.
  - Διάσπαση σε:
    - `X` = άθροισμα των `A[beg+1..end]`
    - Η λύση είναι `X + A[beg]`
  - Αναδρομική περίπτωση:

```
X=sum(A, beg+1, end);
return X + A[beg];
```

ή απλούστερα:

```
return A[beg] + sum(A, beg_1, end);
```

Τελική μορφή αναδρομικής συνάρτησης

- Η αναδρομική συνάρτηση στο σύνολό της είναι η ακόλουθη:

```
int sum(int A[], int beg, int end)
{
    if (beg > end)
        return 0;
    return A[beg] + sum(A, beg + 1, end);
}
```

https://github.com/chgogos/oop/blob/master/various/COP3330/lect11/sample1.cpp

Ιχνηλάτηση (tracing) της αναδρομικής συνάρτησης

- Έστω A = {1,2,3,4,5};
- sum(A, 0, 4)
  - A[0] + sum(A, 1, 4)
    - A[1] + sum(A, 2, 4)
      - A[2] + sum (A, 3, 4)
        - A[3] + sum(A, 4, 4)
          - A[4] + sum(A, 5, 4) ← η sum(A, 5, 4) επιστρέφει 0
          - A[4] + 0 = 5 ← η sum(A, 4, 4) επιστρέφει 5
        - A[3] + 5 = 9 ← η sum(A, 3, 4) επιστρέφει 9
      - A[2] + 9 = 12 ← η sum(A, 2, 4) επιστρέφει 12
    - A[1]+12 = 14 ← η sum(A, 1, 4) επιστρέφει 14
  - A[0] + 14 = 15 ← η sum(A, 0, 4) επιστρέφει 15
- Σε κάθε αναδρομικό βήμα, το πρόγραμμα είναι ένα βήμα πλησιέστερα στη βασική περίπτωση, όταν φθάσει στη βασική περίπτωση, αρχίζουν να σχηματίζονται οι λύσεις σταδιακά για τα μεγαλύτερα προβλήματα

Παράδειγμα αναδρομής 2

- Ταξινόμηση της ακολουθίας `A[beg..end]`
- Διάσπαση σε:
  - Υποπρόβλημα 1: Ταξινόμησε την ακολουθία `A[beg+1..end]`
  - Υποπρόβλημα 2: Εισήγαγε το A[beg] στην κατάλληλη θέση της ακολουθίας `A[beg+1..end]`
- Πρωτότυπο συνάρτησης:
  - `void sort(A, beg, end);` // ταξινόμησε την ακολουθία A από beg μέχρι end
  - Όταν η ακολουθία δεν έχει στοιχεία είναι ταξινομημένη (`beg>end`)
  - Όταν η ακολουθία έχει μόνο ένα στοιχείο είναι ταξινομημένη (`beg==end`)

Παράδειγμα αναδρομής 2

- Βασική περίπτωση:
  - `if (beg>=end) return;`
- Αναδρομική περίπτωση, η ακολουθία έχει περισσότερα από ένα στοιχεία (`beg<end`):
  - Υποπρόβλημα 1: Ταξινόμησε την ακολουθία `A[beg+1..end]`

```
sort(A, beg + 1, end);
```

  - Υποπρόβλημα 2: Εισήγαγε το `A[beg]` στην ταξινομημένη ακολουθία `A[beg+1..end]`

```
tmp = A[beg];
for (i = beg + 1; i <= end; i++) {
  if (tmp > A[i])
    A[i - 1] = A[i];
  else
    break;
}
A[i - 1] = tmp;
```

Τελική μορφή αναδρομικής συνάρτησης

- Η αναδρομική συνάρτηση στο σύνολό της είναι η ακόλουθη:

```
void sort(int A[], int beg, int end)
{
    if (beg >= end)
        return;
    sort(A, beg + 1, end);
    int tmp, i;
    tmp = A[beg];
    for (i = beg + 1; i <= end; i++)
    {
        if (tmp > A[i])
            A[i - 1] = A[i];
        else
            break;
    }
    A[i - 1] = tmp;
}
```

https://github.com/chgogos/oop/blob/master/various/COP3330/lect11/sample2.cpp

Αναδρομική σκέψη

- Εντοπισμός της βασικής περίπτωσης (base case) για την οποία συνήθως είναι προφανής η αντιμετώπισή της.
- Ερώτηση: Αν μπορούμε να επιλύσουμε ένα πρόβλημα μεγέθους `N-1`, πως μπορούμε να χρησιμοποιήσουμε αυτή τη λύση για να λύσουμε ένα πρόβλημα μεγέθους `N`;
  - Αυτή είναι η αναδρομική περίπτωση (recursive case)

  - Βήματα αν η απάντηση είναι ΝΑΙ:
    - Εύρεση του κατάλληλου πρωτοτύπου συνάρτησης
    - Βασική περίπτωση
    - Αναδρομική περίπτωση

Αναδρομή και μαθηματική επαγωγή

- Η μαθηματική επαγωγή είναι χρήσιμο εργαλείο για την απόδειξη θεωρημάτων
  - Πρώτα αποδεικνύεται η βασική περίπτωση (N=1)
    - Επίδειξη ότι το θεώρημα ισχύει για κάποιες μικρές τιμές.
  - Στη συνέχεια, διατύπωση της επαγωγικής υπόθεσης
    - Υπόθεση ότι το θεώρημα είναι αληθές για όλες τις περιπτώσεις μέχρι κάποιο όριο (N=k)
  - Απόδειξη ότι το θεώρημα ισχύει για την επόμενη τιμή (N=k+1)
  - Παράδειγμα: $\sum_{i=1}^{N} i = \frac{\mathrm{N(N+1)}}{2}$
- Αναδρομή
  - Βασική περίπτωση: γνωρίζουμε πως να επιλύσουμε το πρόβλημα για τη βασική περίπτωση (N=0 ή 1).
  - Αναδρομική περίπτωση: Υποθέτοντας ότι μπορούμε να λύσουμε το πρόβλημα για N=k-1, προσπαθούμε να λύσουμε το πρόβλημα για N=k.
- Η αναδρομή είναι βασικά η εφαρμογή της επαγωγής στην επίλυση προβλημάτων.

Αντιγραφή λεκτικών αναδρομικά με τη συνάρτηση `mystrcpy`

- `void mystrcpy(char *dst, char *src);`
  - Αντιγραφή του λεκτικού `src` στο λεκτικό `dst`.

  - Βασική περίπτωση: `if (*src == '\0') *dst = *src;`
  - Αναδρομική περίπτωση: Αν γνωρίζουμε πως να αντιγράψουμε ένα λεκτικό με έναν λιγότερο χαρακτήρα σε σχέση με το λεκτικό που πραγματικά θέλουμε να αντιγράψουμε, πώς μπορούμε να χρησιμοποιήσουμε αυτή τη δυνατότητα για να αντιγράψουμε το πλήρες λεκτικό;
    - Αντιγραφή ενός χαρακτήρα `*dst = *src`
    - Αντιγραφή του υπόλοιπου λεκτικού ← αναδρομικά με την mystrcpy

Αντιγραφή λεκτικών αναδρομικά

```
void mystrcpy(char *dst, const char *src)
{
    if (*src == '\0')
    {
        *dst = *src;
        return;
    }
    *dst = *src;
    mystrcpy(dst + 1, src + 1);
}
```

https://github.com/chgogos/oop/blob/master/various/COP3330/lect11/sample3.cpp

Υπολογισμός μήκους λεκτικών αναδρομικά με τη συνάρτηση `mystrlen`

- `void mystrlen(char* str);`
- Αν γνωρίζουμε πως να μετρήσουμε το μήκος ενός λεκτικού με έναν λιγότερο χαρακτήρα, τότε αρκεί στην τιμή αυτή να προσθέσουμε 1 για να λάβουμε το μήκος του λεκτικού.

```
int mystrlen(const char *str)
{
    if (*str == '\0')
    {
        return 0;
    }
    return 1 + mystrlen(str + 1);
}
```

https://github.com/chgogos/oop/blob/master/various/COP3330/lect11/sample4.cpp

Ερωτήσεις σύνοψης

- Τι είναι η βασική περίπτωση (base case) στην αναδρομή;
- Τι είναι η αναδρομική περίπτωση (recursive case) στην αναδρομή;
- Πως μπορούμε αναδρομικά να βρούμε μια λύση για το εάν μια λέξη είναι παλινδρομική ή όχι (π.χ. ΣΟΦΟΣ, ΑΝΝΑ); Συμπληρώστε τον κώδικα:
  - `bool ispalindrome(string word, int front, int back){…}`
- Πως μπορούμε αναδρομικά να αναζητήσουμε ένα κλειδί σε μια ακολουθία αριθμών; Συμπληρώστε τον κώδικα:
  - `bool search(int A[], int beg, int end, int key) {…}`
- Γράψτε αναδρομική συνάρτηση υπολογισμού του παραγοντικού ενός θετικού ακεραίου αριθμού (n!=1*2*3*…*n).

Αναφορές

- http://www.cs.fsu.edu/~xyuan/cop3330/