ΠΕΡΙΕΧΟΜΕΝΑ:

Γιώργος Μ.
Σμαραγδένιος Χορηγός Μαθήματος

Γιώργος Τασιούλης
Σμαραγδένιος Χορηγός Μαθήματος

Ξένα Σύνολα:

- Δύο σύνολα που δεν περιέχουν κοινά στοιχεία.

Παραδείγματα:

- Τα σύνολα {1,2,3} και {4,5,6} είναι ξένα μεταξύ τους.
- Τα σύνολα {1,2,3} και {3,5,6} δεν είναι ξένα μεταξύ τους.

Δομή Δεδομένων "Ξένα Σύνολα":

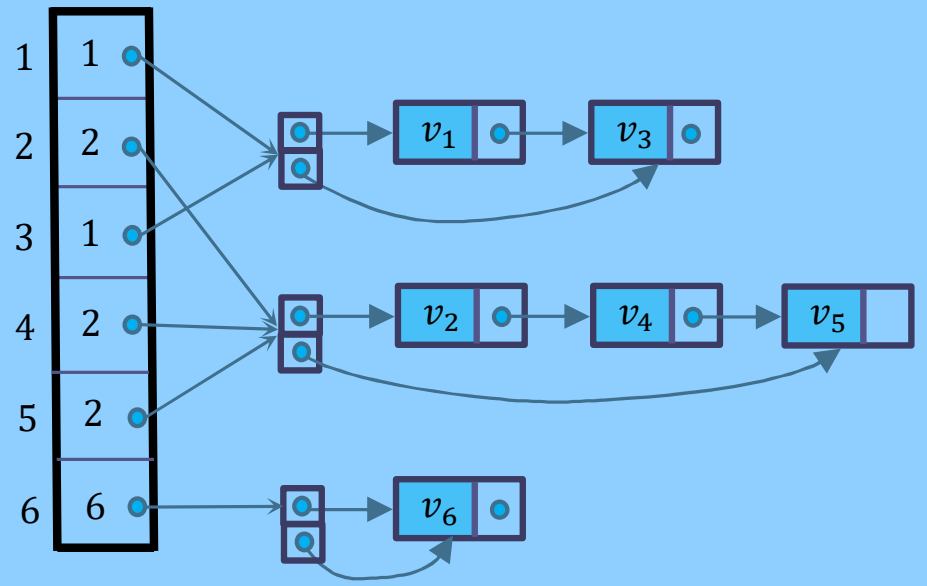
- Μπορεί να απεικονίσει αποδοτικά το χωρισμό δεδομένων σε ξένα σύνολα και να κάνει τις ακόλουθες βασικές πράξεις:
  Κάθε σύνολο προσδιορίζεται από έναν "**αντιπρόσωπο**" (που είναι ένα στοιχείο που περιέχεται σε αυτό)
  - **make_set(x):** Κατασκευάζει ένα καινούργιο σύνολο, που αντιπροσωπεύεται από το x
  - **union(x,y):** Ενοποιεί τα σύνολα που περιέχουν το x και το y
  - **find_set(x):** Επιστρέφει τον αντιπρόσωπο του συνόλου που περιέχει το x

Παράδειγμα επιθυμητής χρήσης της δομής:

1.	make_set(x) για x=1,2,…,6:	1.	{1} {2} {3} {4} {5} {6}
2.	union(1, 3)	2.	{1,3} {2} {4} {5} {6}
3.	union(2, 4)	3.	{1,3} {2, 4} {5} {6}
4.	union(2, 5)	4.	{1,3} {2, 4, 5} {6}
5.	union(3, 5)	5.	{1, 2, 3, 4, 5} {6}

Σημαντικό: Αντιπρόσωπος σε κάθε σύνολο είναι κάποιο τυχαίο στοιχείο του συνόλου.

- **Μία υλοποίηση με συνδεδεμένες λίστες**
  - Κάθε σύνολο αναπαρίσταται με μια συνδεδεμένη λίστα στην οποία έχουμε δύο δείκτες, έναν για το πρώτο και έναν για το τελευταίο στοιχείο.
  - Ένας πίνακας "repr" που για κάθε στοιχείο απεικονίζει τον αντιπρόσωπο του. Ένας πίνακας "list" που για κάθε στοιχείο απεικονίζει λίστα στην οποία ανήκει
  - Αντιπρόσωπος είναι το 1ο στοιχείο της κάθε λίστας

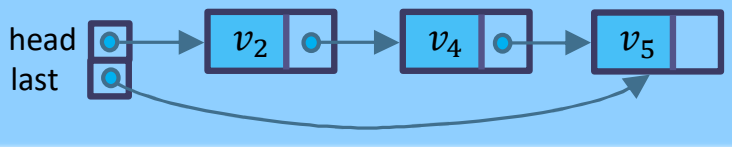


Πολυπλοκότητα:

- Οι make_set και η find_set είναι O(1)
- Η union είναι O(n) και επιδέχεται βελτίωσης.

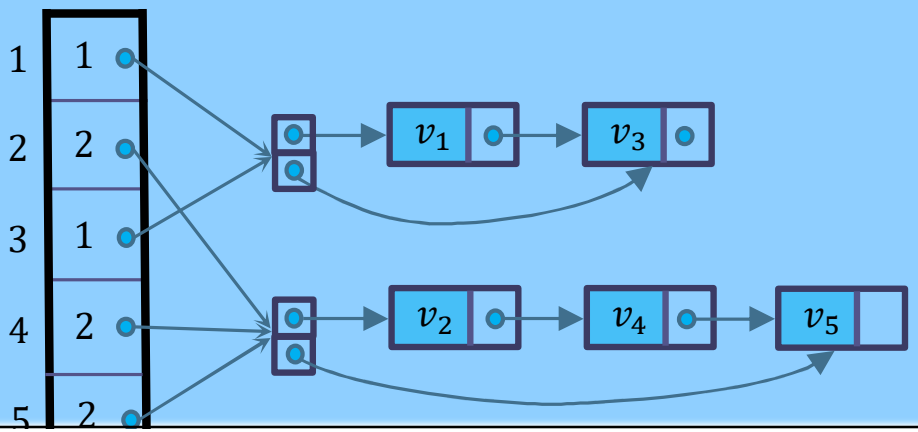
Εκτεταμένη Λίστα:

- Επεκτείνουμε την απλά συνδεδεμένη λίστα, έτσι ώστε να περιέχει δείκτη και στην αρχή και στο τέλος της λίστας:




```
struct extended_list
{
   LIST_PTR head;
   LIST_PTR last;
};
typedef struct extended_list EXTENDED_LIST;
```

- Και τα ξένα σύνολα ορίζονται από δύο πίνακες: τον repr (για τον αντιπρόσωπο του συνόλου που ανήκει) και τον list (που έχει δείκτη στην εκτεταμένη λίστα στην οποία ανήκει)




```
struct disjoint
{
  elem *repr;              /* pinakas antiproswpwn */
  EXTENDED_LIST *list;     /* pinakas listwn */
  int N;                   /* Plithos stoixeiwn */
};
typedef struct disjoint DISJOINT;
```

Πράξεις επί των Ξένων Συνόλων:

- Εκτός από τις τρεις βασικές πράξεις που πρέπει να υποστηρίζει η δομή:
  - DS_make_set
  - DS_union
  - DS_find_set
- Ορίζουμε επίσης τις πράξεις:
  - DS_init (Αρχικοποίηση της δομής)
  - DS_destroy (Καταστροφή της δομής)

```
/* Basikes Prakseis disjoint sets */
void DS_init(DISJOINT *d, int N);
void DS_make_set(DISJOINT *d, elem x);
int DS_union(DISJOINT *d, elem x, elem y);
elem DS_find_set(DISJOINT d, elem x);
void DS_destroy(DISJOINT *d);
/* Deutereouses Prakseis disjoint sets */
void DS_print(DISJOINT d);
```

DS_Init:

- Αρχικοποιεί τη δομή, καλώντας τη make_set για κάθε στοιχείο.

```
void DS_init(DISJOINT *d, int N)
{
   int i;

   d->N = N;
   d->repr = (elem *)malloc(sizeof(elem)*N);
   if (!d->repr)
   {
      printf("Error allocating memory!");
      exit(0);
   }
   d->list = (EXTENDED_LIST *)malloc(sizeof(EXTENDED_LIST)*N);
   if (!d->list)
   {
      printf("Error allocating memory!");
      exit(0);
   }
   for (i=0; i<N; i++)
   {
      d->list[i].head=NULL;
      d->list[i].last=NULL;
   }
   for (i=0; i<N; i++)
   {
      DS_make_set(d, i);
      d->repr[i]=i;
   }
};
```

DS_make_set:

- Θέτει τη λίστα του στοιχείου i να είναι ίση με μία λίστα που περιέχει μόνο αυτό το στοιχείο.

```
void DS_make_set(DISJOINT *d, elem x)
{
   LL_insert_start(&((d->list[x]).head),x);
   d->list[x].last=d->list[x].head;
}
```

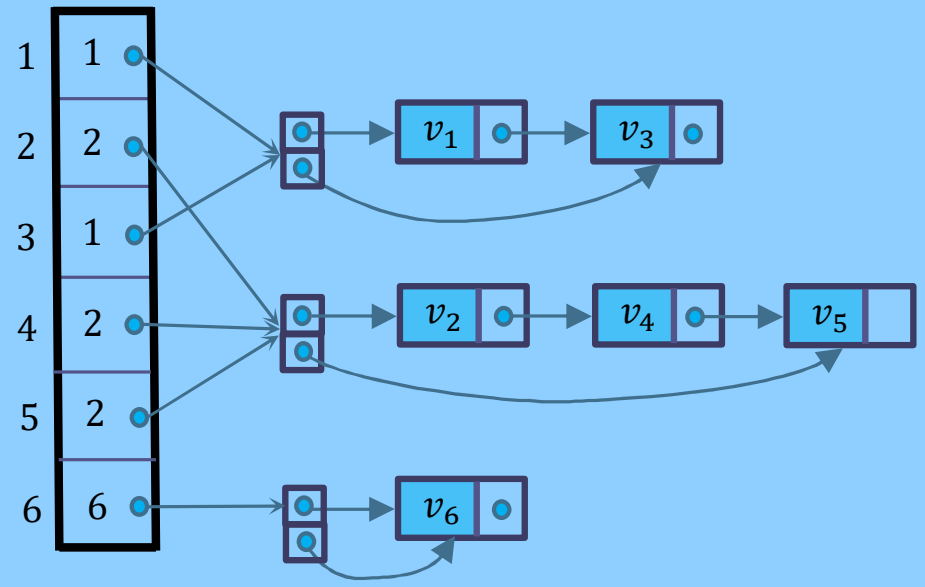
DS_find_set:

- Επιστρέφει τον αντιπρόσωπο του στοιχείου που δέχεται ως όρισμα

```
elem DS_find_set(DISJOINT d, elem x)
{
   return d.repr[x];
}
```

DS_Union:

- Παίρνει ως ορίσματα δύο στοιχεία και συνενώνει τις λίστες στις οποίες ανήκουν



Στα παραπάνω σύνολα, έστω ότι θέλουμε να ενοποιήσουμε τα 3,4:

- Θα ενοποιήσουμε τις δύο λίστες στην πρώτη.
- Θέτουμε στην πρώτη λίστα: το next του τελευταίου κόμβου, να δείχνει στο πρώτο στοιχείο της δεύτερης λίστας.
- Θέτουμε στην πρώτη λίστα: το last να είναι ίσο με το last της δεύτερης λίστας.
- Διορθώνουμε όλα τα στοιχεία της δεύτερης λίστας, ώστε να έχουν αντιπρόσωπο στοιχείο της 1ης λίστας.

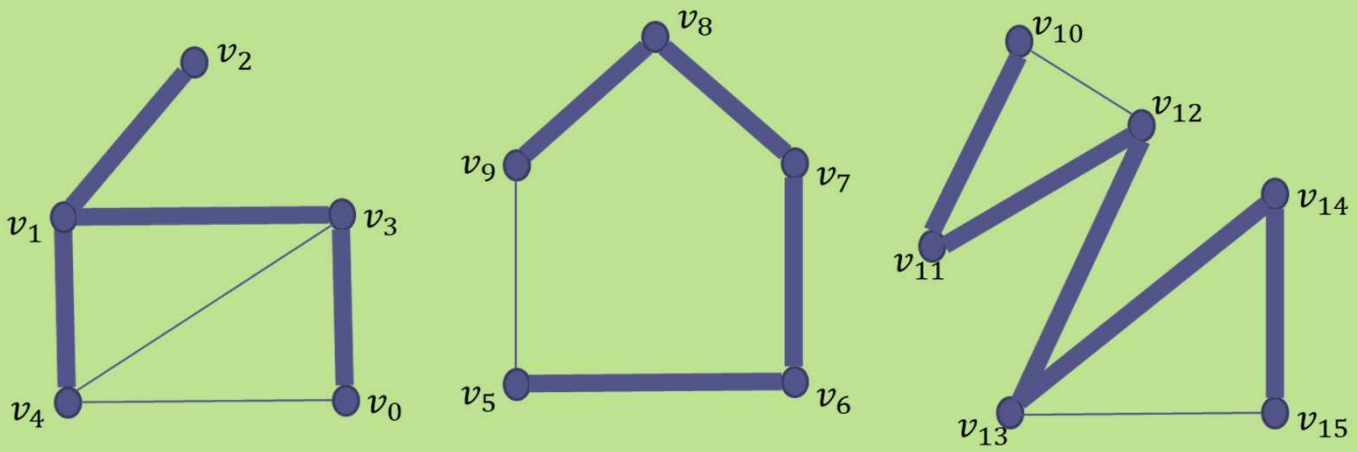
```
int DS_union(DISJOINT *d, elem x, elem y)
{
   LIST_PTR current;
   if (d->repr[x]==d->repr[y])
   {
      printf("Error: Both elements in the same set");
      return FALSE;
   }
   d->list[x].last->next = d->list[y].head;
   d->list[x].last = d->list[y].last;
   current = d->list[x].head;
   while(current!=NULL)
   {
      d->list[current->data]=d->list[x];
      current=current->next;
   }
   current = d->list[y].head;
   while(current!=NULL)
   {
      d->repr[current->data]=x;
      d->list[current->data]=d->list[x];
      current=current->next;
   }
   return TRUE;
}
```

Άσκηση 1: Συνεκτικές Συνιστώσες

Κατασκευάστε ένα πρόγραμμα που εντοπίζει τις κορυφές των συνεκτικών συνιστωσών ενός μη συνδεόμενου γραφήματος.

Παράδειγμα:

- Για τον ακόλουθο γράφο, το πρόγραμμα θα πρέπει να τυπώνει τις συνεκτικές συνιστώσες από τις οποίες αποτελείται: { 0, 1, 2, 3, 4 } { 5, 6, 7, 8, 9 } { 10, 11, 12, 13, 14, 15 }



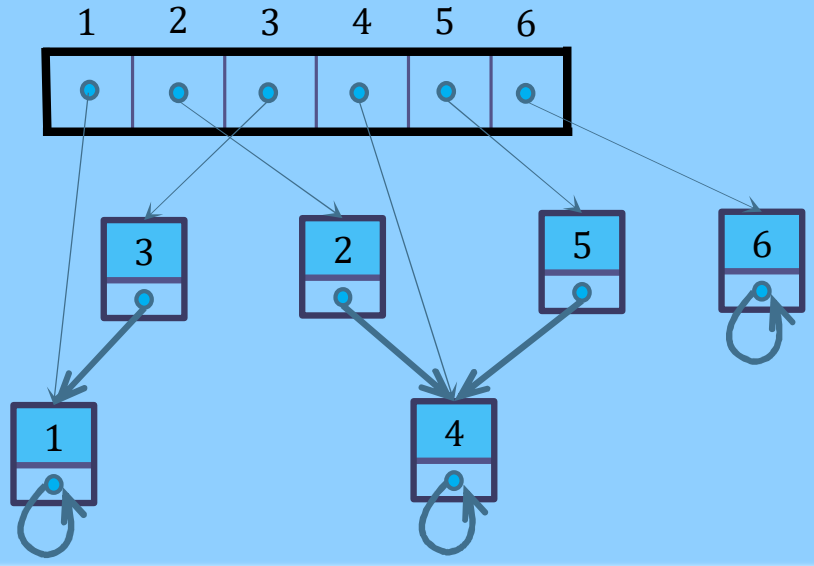
Άσκηση 2: Χειρότερη Περίπτωση

- Αρχικοποιήστε μια δομή ξένων συνόλων με n στοιχεία.
- Έπειτα σκεφθείτε (και υλοποιήστε) μια σειρά από n-1 unions, ώστενα μεγιστοποιείται το πλήθος των διορθώσεων των αντιπροσώπων και η συνολική πολυπλοκότητα να είναι $O(n^2)$

Αντεστραμμένα Δένδρα:

Στην 2η και αποδοτικότερη υλοποίηση των ξένων συνόλων

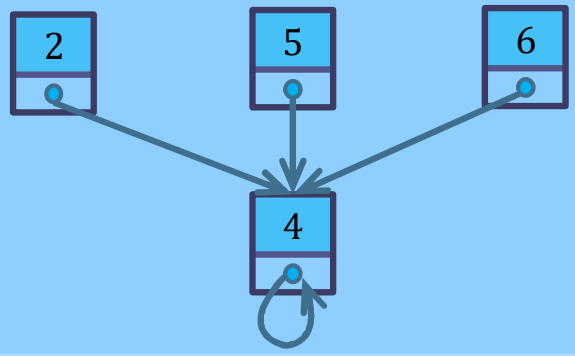
- Χρησιμοποιείται μια απεικόνιση μέσω αντεστραμμένων δένδρων
- Κάθε κόμβος περιέχει το αναγνωριστικό του και έναν δείκτη προς τον γονέα του.
- Ενώ η ρίζα κάθε δένδρου αντιστοιχεί στον αντιπρόσωπο του συνόλου.



Σημειώσεις:

- Η παραπάνω διευθέτηση αντιστοιχεί στα ξένα σύνολα {1, 3} {2, 4, 5} και {6} με αντίστοιχους αντιπρόσωπους τα στοιχεία 1, 4 και 6.
- Στη ρίζα κάθε δένδρου θα κρατάμε επίσης το ύψος του δένδρου, ώστε να κάνουμε πιο αποδοτικές ενώσεις δένδρων.

- **Υλοποίηση:**
  - Η make_set κατασκευάζει ένα μονοσύνολο και θέτει το ύψος του κόμβου ίσο με το 0.
  - Η find_set ακολουθεί το δείκτη του γονέα μέχρι τη ρίζα και επιστρέφει τον κόμβο.
  - Η union βάζει τον γονέα του συνόλου με το μικρότερο ύψος να δείχνει στη ρίζα του συνόλου με το μεγαλύτερο ύψος. Π.χ. η ένωση των {2, 4, 5} και {6} γίνεται ως εξής:



Παρατήρηση:

- Η υλοποίηση ως έχει μπορεί να οδηγήσει σε εκφυλισμένες μορφές (π.χ. ένα δένδρο που έχει μετατραπεί σε αλυσίδα), χωρίς να έχουμε βελτίωση σε σχέση με την προηγούμενη υλοποίηση.
- Ωστόσο, με κατάλληλη τροποποίηση της find_set (ώστε να θέτει όλους τους κόμβους που συναντά ως παιδιά του κόμβου-αντιπροσώπου) βελτιώνει την πολυπλόκότητα (για να ενώσουμε τους n κόμβους)

ΜΑΘΗΜΑ 11: Ξένα Σύνολα 2. Αντεστραμμένα Δένδρα (2. Υλοποίηση σε C) Δομές Δεδομένων σε C psounis

Δηλώσεις:

- Ορίζουμε τον κόμβο να περιέχει τα στοιχεία (data, height και parent):

```
typedef int elem;

struct disjoint_node
{
   elem data;
   int height;
   struct disjoint_node *parent;
};

typedef struct disjoint_node DISJOINT_NODE;
typedef struct disjoint_node *DISJOINT_PTR;

struct disjoint
{
  DISJOINT_PTR *array;     /* pinakas deiktwn */
  int N;                   /* Plithos stoixeiwn */
};

typedef struct disjoint DISJOINT;
```

Πράξεις επί των Ξένων Συνόλων:

- Εκτός από τις τρεις βασικές πράξεις που πρέπει να υποστηρίζει η δομή:
  - DS_make_set
  - DS_union
  - DS_find_set
- Ορίζουμε επίσης τις πράξεις:
  - DS_init (Αρχικοποίηση της δομής)
  - DS_destroy (Καταστροφή της δομής)

DS_Init:

- Αρχικοποιεί τη δομή, καλώντας τη make_set για κάθε στοιχείο.

```
void DS_init(DISJOINT *d, int N)
{
   int i;
   d->N = N;
   d->array = (DISJOINT_PTR *)malloc(sizeof(DISJOINT_PTR)*N);
   if (!d->array)
   {
      printf("Error allocating memory!");
      exit(0);
   }
   for (i=0; i<N; i++)
      DS_make_set(d, i);
}
```

DS_make_set:

- Κατασκευάζει τον κόμβο του στοιχείου i (δεν περιέχεται κάποιο άλλο στοιχείο στο δένδρο)

```
void DS_make_set(DISJOINT *d, elem x)
{
   d->array[x] = (DISJOINT_NODE *)malloc(sizeof(DISJOINT_NODE));
   if (!d->array[x])
   {
      printf("Error allocating memory!");
      exit(0);
   }
   d->array[x]->data = x;
   d->array[x]->height = 0;
   d->array[x]->parent = d->array[x];
}
```

DS_union:

- Παίρνει ως ορίσματα δύο στοιχεία και συνενώνει τις λίστες στις οποίες ανήκουν
- Γίνεται η σύγκριση των υψών και το δένδρο μικρότερου ύψους ενώνεται στη ρίζα του δένδρου μεγαλύτερου ύψους
- Εκτός και αν αυτά είναι ίσα, οπότε αφού τα ενώσουμε πρέπει να διορθώσουμε το ύψος του συνολικού δένδρου.

```
int DS_union(DISJOINT *d, elem x, elem y)
{
   int p1, p2;

   p1 = DS_find_set(*d, x);
   p2 = DS_find_set(*d, y);

   if (p1==p2)
   {
      printf("Elements %d and %d are in the same set(%d)", x, y, p1);
      return FALSE;
   }

   if (d->array[p1]->height < d->array[p2]->height)
      d->array[p1]->parent = d->array[p2];
   else if (d->array[p1]->height > d->array[p2]->height)
      d->array[p2]->parent = d->array[p1];
   else
   {
      d->array[p2]->parent = d->array[p1];
      (d->array[p1]->height)++;
   }

   return TRUE;
}
```

DS_find_set:

- Επιστρέφει τον αντιπρόσωπο του στοιχείου που δέχεται ως όρισμα

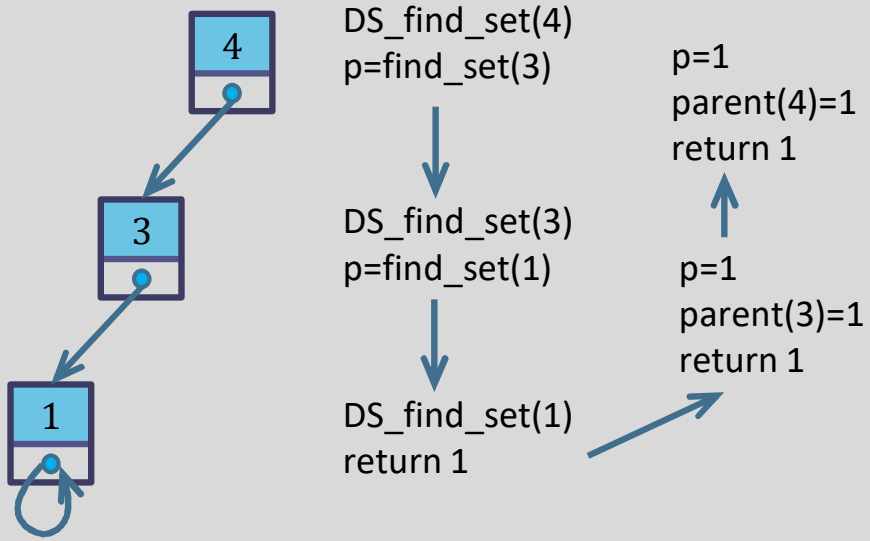
```
elem DS_find_set(DISJOINT d, elem x)
{
   if (d.array[x]->parent == d.array[x])
      return x;
   else
      return DS_find_set(d, d.array[x]->parent->data);
}
```

Παρατήρηση:

- Η παραπάνω υλοποίηση μπορεί να βελτιωθεί, θέτοντας αναδρομικά ως γονέα όλων των κόμβων που εξετάσαμε τον γονέα που βρήκαμε στο τέλος των αναδρομικών κλήσεων ως εξής:

```
elem DS_find_set(DISJOINT d, elem x)
{
   int p;
   if (d.array[x]->parent == d.array[x])
      return x;
   else
      {
        p = DS_find_set(d, d.array[x]->parent->data);
        d.array[x]->parent=d.array[p];
        return p;
      }
}
```

Παράδειγμα (της παραπάνω αναδρομικής συνάρτησης)



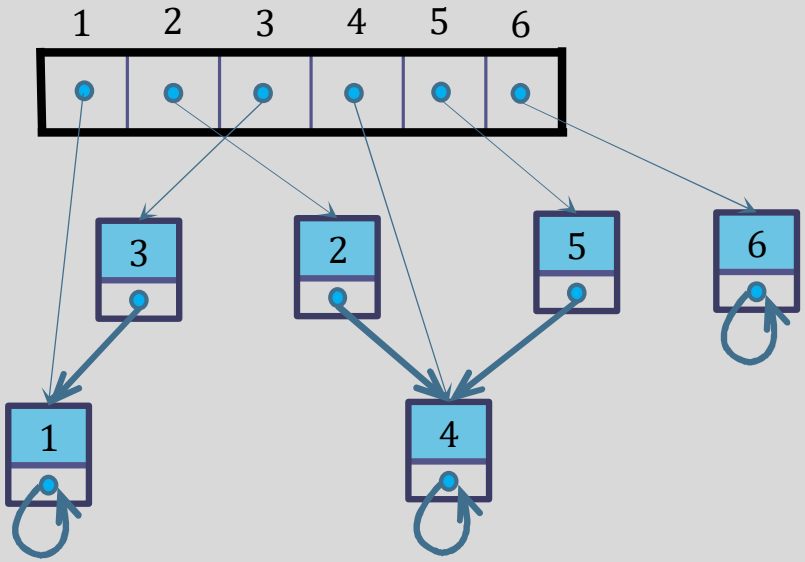
Παρατήρηση:

- Με τον τρόπο αυτό όλες οι πράξεις απαιτούν σχεδόν σταθερό χρόνο και η δομή δεδομένων είναι εξαιρετικά αποδοτική O(a(n)) όπου a(n)<=4 για λογικά n, οπότε για m πράξεις επί της δομής είναι O(ma(n))=O(mlogn).

Άσκηση 3: Εκτύπωση Συνόλων

Τροποποιήστε τη δομή έτσι ώστε:

- Να τυπώνει τα σύνολα με πληροφορία για τα αντεστραμμένα δένδρα με καλύτερη μορφοποίηση (χωρίς να επηρεάζει τη δομή).
- Π.χ. για το ακόλουθο στιγμιότυπο:



- Να τυπώνει για κάθε σύνολο τα στοιχεία και μέσα σε παρένθεση τον γονέα τους στο αντεστραμμένο δένδρο:

{1(1), 3(1)} {2(4) 4(4) 5(4)} {6(6)}