04_graphtransexer.pdf

Αση 1) Καταγραφή λιστών γειτνίασης από το γεν. δένδρο της ΔιΒ.

Παρατηρήσεις:
- Για την καταγραφή λιστών, πρέπει να γνωρίζω τις ακέραιες γειτνίασης φορείς των κόμβων.

Π.Χ. 

1 → 2
2 → 1

Σε τι δομή θα μπορούσε να αποθηκευτεί αυτή η πληροφορία; Πίνακας ραφεγ-

Τι αλγόριθμο θα χρησιμοποιήσω; ΔιΒ

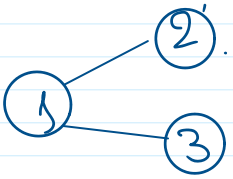
Διάσχιση κατά Βάθος

```
int main () {
  for v∈V {
    visited[v] ← false;
    parent[v] ← 0;
  }
  for v∈V {
    if not visited[v] then {
        explore(v, visited, parent);
    }
  }
}
```

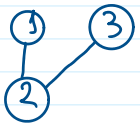
```
void explore (int v, bool visited[], int parent[]) {
  visited[v] ← true;
  for u ∈ N(v) {
    if not visited[u] then {
      parent[u] ← v;
      explore(u, visited, parent);
    }
  }
}
```

Τρέξε τον αλγόριθμο για τον γράφο της αση 1)
Αντί για γράμματα βάλε αριθμούς ως κόμβους.

parent[]	1	2	3
	0	1	1



Παρατηρήσεις:
- Για την αναπαράσταση των λιστών θα χρησιμοποιούμε το διάνυσμα συνόλων $N_T[]$

Π.Χ:


$N_T[] \Rightarrow$

1	→ {2}
2	→ {1,3}
3	→ {2}

$N_T[2] = \{1,3\}$

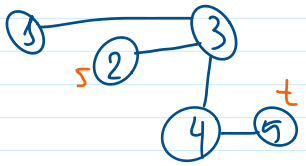
```
void sp_tree_lists (int parent[], set N_T[]) {
  for v∈V {
    N_T[] ← ∅;
  }
  for v∈V {
    if parent[v] ≠ 0 then {
      N_T[v] ← N_T[v] ∪ {parent[v]};
      N_T[parent[v]] ← N[parent[v]] ∪ {v};
    }
  }
```

Ασκ 2) Αλγόριθμος Εύρεσης Μονοπατιού από s σε t

Παρατηρήσεις:

- Θα χρησιμοποιήσω ΔκΒ
- Η διάσχιση θα σταματά όταν φτάσω στον κόμβο t. (Χρειάζομαι μεταβλητή bool found. για να ξέρω εάν πρέπει να σταματήσω)
- Χρειάζομαι μία δομή πίνακας για την αποθήκευση του μονοπατιού.

Π.χ:



path[] ⇒ | 2 | 3 | 4 | 5 |
s t

int length, για το μήκος του path

- Η διάσχιση θα ξεκινά από την κορυφή s. Επομένως: int main(int s, int t)

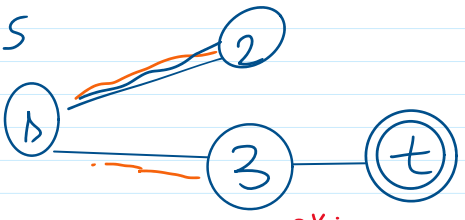
```
for v∈V {
    visited[v] ← false;
}
found ← false;
length ← 0;
```

→ find_path(s, t, ...)

```
for u∈V {
    if not visited[u] {
        find_path(....)
    }
}
```

Παραλλαγή ΔκΒ

```
void find_path ( int V, int t, bool visited[], int path[],int length, bool found) {
    visited[v] ← true;
    length++;
    path[length] ← v;
    if v=t then {
        found ← true;
    }
    if not visited[v] and not found then {
        for u∈N(v) {
            if not visited[u] and not found then {
                find_path(u,t,visited,path,length,found);
            }
            if found then ←
                break;
        }
    }
    if not found then {
        path[length] ← 0;
        length --;
    }
}
```



π.χ:

find_path → v=1, u=∅
find_path → v=1, u=2
u=3
find_path → v=3, u=t
find_path → v=t, u=∅

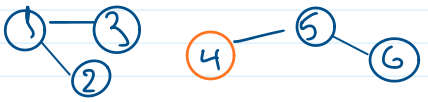
← Μαζεύω το λάθος

Ασκ 4) Αλγόριθμος Εύρεσης Γραφικών Συνιστωσών

Παρατηρήσεις:

- Στην διάσχιση κατά βάθος όταν ολοκληρωθούν οι αναδρομικές κλήσεις της explore(), ο έλεγχος του προγράμματος επιστρέφει στον κύριο βρόχο for $v \in V$ της main().
- Στην επόμενη επανάληψη του κύριου βρόχου, επιλέγεται ένας νέος κόμβος προς εξερεύνηση ο οποίος δεν γειτονεύει με κανέναν από τους προηγούμενους

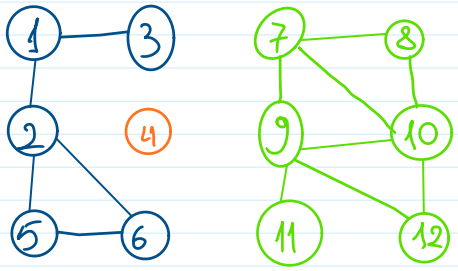
π.χ:



Οπότε εκείνος ο κόμβος ανήκει σε άλλη γραφική συνιστώσα.

```
main () {
    for v∈V {
        visited[v]←false;
        comp[v] ←0;
    }
    comp_num←0;
    for v∈V {
        if not visited[v] then {
            comp_num ++;
            explore1(v, visited, comp, comp_num);
        }
    }
}
```

```
void explore1(v, visited[], comp[], comp_num) {
    visited[v]← true;
    comp[v]← comp_num;
    for u∈N(v) {
        if not visited[u] then {
            explore1(u, visited, comp, comp_num)
        }
    }
}
```



Συνιστώσα 1

Συνιστώσα 2

Συνιστώσα 3

comp[]

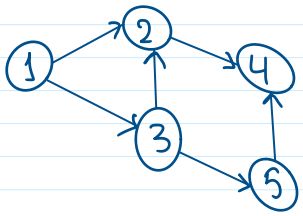
1	2	3	4	5	6	7	8	9	10	11	12
1	1	1	2	1	1	3	3	3	3	3	3

Κατευθυνόμενα Γραφήματα (Εισαγωγή)

— Τι αλλάζει; Για μια κορυφή v το σύνολο $N(v)$ διαιρείται σε

— $N^+(v)$ π.χ: $N^+(3) = \{2, 5\}$

— $N^-(v)$ π.χ: $N^-(3) = \{1\}$



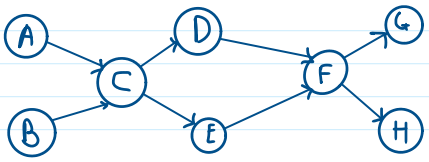
— Στην συνάρτηση explore() ο παραπάνω βρόχος πως τροποποιείται;

for $u \in N(v)$ ⇒ for $u \in N^+(v)$

Άσκ 8) Τοπολογική Ταξινόμηση (Αλγόριθμος σελ. 53)

visited[]

A	B	C	D	E	F	G	H
1	1	1	1	1	1	1	1



Q = A
Q =
Q = C
Q =
Q = D
Q =
Q = F
Q =
Q = G

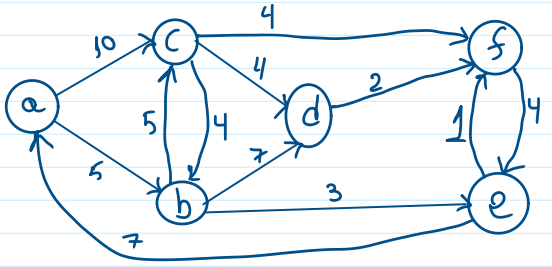
Q = F G
Q = G
Q = HG
Q = FHG
Q = DFHG
Q = CDFHG
Q = DFHG
Q = EDFHG
Q = CEDFHG

Q = ACEDFHG
Q = BACEDFHG

Dijkstra

Παρατήρηση: Ο Dijkstra εφαρμόζεται σε γραφήματα κατευθυνόμενα με βάρη - μόνιμη

- Βρίσκει το συντομότερο μονοπάτι από μία κορυφή (source) προς οποιοδήποτε κορυφή



source = a

steps	Explored	Unexplored
0		a^{∞} b^{∞} c^{∞} d^{∞} e^{∞} f^{∞}
1	a^{0}	b^{5a} c^{10a} d^{∞} e^{∞} f^{∞}
2	a^{0} b^{5a}	c^{10a} d^{12b} e^{8b} f^{∞}
3	a^{0} b^{5a} e^{8b}	c^{10a} d^{12b} f^{9e}
4	a^{0} b^{5a} e^{8b} f^{9e}	c^{10a} d^{12b}
5	a^{0} b^{5a} e^{8b} f^{9e} c^{10a}	d^{12b}
6	a^{0} b^{5a} e^{8b} f^{9e} c^{10a} d^{12b}	

from a

b: $a \to b = 5$

c: $a \to c = 10$

d: $a \to b \to d = 12$

e: $a \to b \to e = 8$

f: $a \to b \to e \to f = 9$