ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ
UNIVERSITY OF PATRAS

ΑΝΟΙΚΤΑ ακαδημαϊκά μαθήματα ΠΠ

Τεχνολογίες Υλοποίησης Αλγορίθμων

Χρήστος Ζαρολιάγκης

Καθηγητής
Τμήμα Μηχ/κων Η/Υ & Πληροφορικής
Πανεπιστήμιο Πατρών
email: zaro@ceid.upatras.gr

Ενότητα 8

Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο

ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ & ΘΡΗΣΚΕΥΜΑΤΩΝ, ΠΟΛΙΤΙΣΜΟΥ & ΑΘΛΗΤΙΣΜΟΥ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ

ΕΣΠΑ 2007-2013
πρόγραμμα για την ανάπτυξη
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ

Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

Άδειες Χρήσης

- Το παρόν εκπαιδευτικό υλικό υπόκειται σε άδειες χρήσης Creative Commons.
- Για εκπαιδευτικό υλικό, όπως εικόνες, που υπόκειται σε άλλου τύπου άδειας χρήσης, η άδεια χρήσης αναφέρεται ρητώς.

Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο **«Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Πατρών»** έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

Περιεχόμενα

Μέγιστη Ροή – Γενικά

- Κατευθυνόμενο γράφημα $G = (V, E)$ με συνάρτηση χωρητικότητας $cap : E \to \mathbb{R}_{\geq 0}$ και δύο διακεκριμένες κορυφές s και t
- $In(v) = \{e : target(e) = v\}$, $Out(v) = \{e : source(e) = v\}$, $\forall v \in V$
- (s, t)-*ροή* (ή απλά ροή): συνάρτηση $f : E \to \mathbb{R}_{\geq 0}$ για την οποία:

$$(1) \qquad 0 \leq f(e) \leq cap(e) \qquad \forall e \in E$$

$$(2) \quad excess(v) = \sum_{e \in Out(v)} f(e) - \sum_{e \in In(v)} f(e) = 0 \quad \forall v \in V - \{s, t\}$$

- *Τιμή ροής* $|f| = excess(t)$
- **Μέγιστη Ροή**: ροή f που μεγιστοποιεί την $|f|$

Μέγιστη Ροή – Γενικά

- *(s,t)-αποκοπή* (ή απλά αποκοπή): σύνολο S κορυφών με $s \in S$ και $t \in T = V - S$
  Έστω $(S,T) = \{e : source(e) \in S, target(e) \in T\}$

- *Χωρητικότητα αποκοπής*:

$$cap(S) = \sum_{e \in (S,T)} cap(e)$$

- *Κορεσμένη αποκοπή* : αποκοπή S για την οποία
  $f(e) = cap(e), \forall e \in (S,T)$ και $f(e) = 0, \forall e \in (T,S)$

- Έστω f οποιαδήποτε ροή και έστω S οποιαδήποτε αποκοπή. Τότε, $|f| \leq cap(S)$

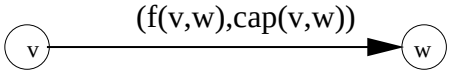
- **Ελάχιστη αποκοπή**: αποκοπή S που ελαχιστοποιεί την $cap(S)$

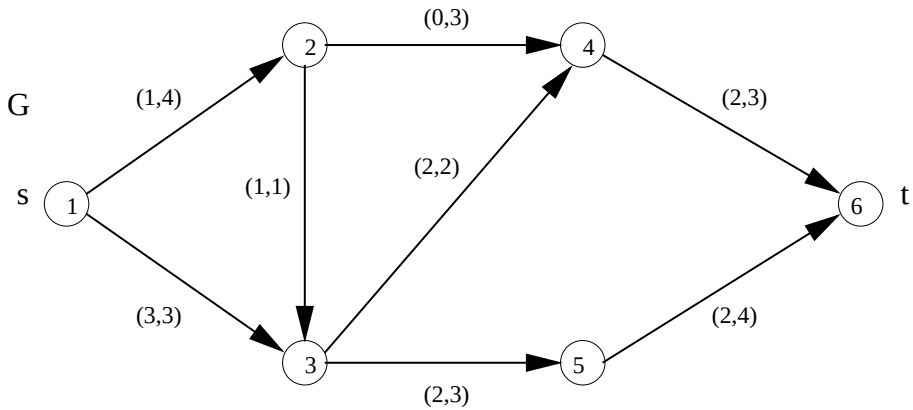
Μέγιστη Ροή – Γενικά

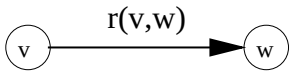
- *Υπόθεση:* $\forall e \in E \Longrightarrow e' \in E$
- *Υπολειπόμενη χωρητικότητα* ακμής $e = (u, v)$:
  $r(e) = cap(e) - f(e) + f(e')$, όπου $e' = (v, u)$
- *Υπολειπόμενο δίκτυο* $G_f = (V, E_f)$, όπου $E_f = \{e \in E : r(e) > 0\}$

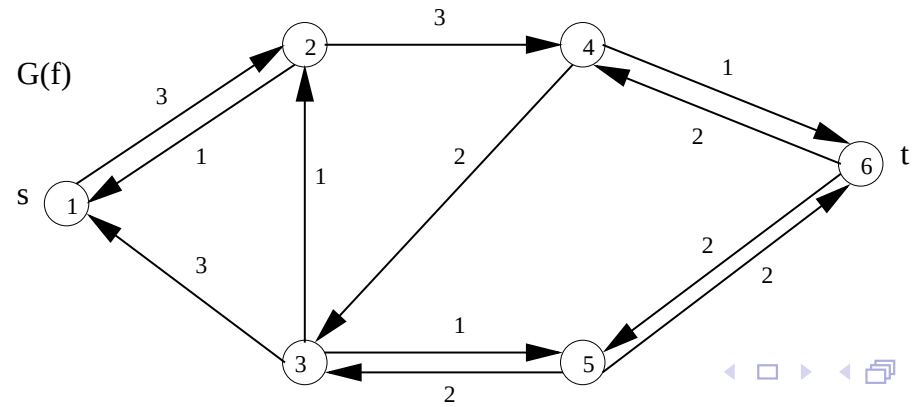
Μέγιστη Ροή – Γενικά

- *Υπολειπόμενη χωρητικότητα* (u, v): $r(u, v) = cap(u, v) - f(u, v) + f(v, u)$
- *Υπολειπόμενο δίκτυο* $G_f = (V, E_f)$, όπου $E_f = \{e \in E : r(e) > 0\}$









Μέγιστη Ροή – Γενικά

- Υπολειπόμενη χωρητικότητα – μια συμπληρωματική εικόνα
  Έστω $e \in G$
  - Αν $f(e) < cap(e)$, τότε $e \in G(f)$ με $r(e) = cap(e) - f(e)$
  - Αν $f(e) > 0$, τότε $e' \in G(f)$ με $r(e') = f(e)$

- Έστω f μια ροή και έστω S το σύνολο των κορυφών που είναι προσπελάσιμες από την s στο $G(f)$
  (α) Αν $t \in S$, τότε η f δεν είναι μέγιστη
  (β) Αν $t \notin S$, τότε η f είναι μέγιστη και το S αποτελεί μια κορεσμένη αποκοπή

- **Θεώρημα Μέγιστης-Ροής Ελάχιστης-Αποκοπής:** Η τιμή της μέγιστης ροής ισούται με την χωρητικότητα της ελάχιστης αποκοπής

Μέγιστη Ροή – Ελεγκτής Ορθότητας

- `bool CHECK_MAX_FLOW_T(const graph& G, node s, node t`
  `const edge_array<NT>& cap,`
  `const edge_array<NT>& f)`

  ελέγχει αν η `f` είναι μια (s, t)-ροή και επιστρέφει `true` αν όντως είναι, αλλιώς `false`

- Ιδέα ελεγκτή:
  - Έλεγχος της συνθήκης χωρητικότητας κάθε ακμής
  - Υπολογισμός πλεονάσματος (excess) κάθε κορυφής: όλες οι κορυφές, εκτός από τις s και t, πρέπει να έχουν πλεόνασμα μηδέν
  - Υπολογισμός των προσπελάσιμων από την s κορυφών στο $G(f)$ με χρήση ΑΠΠ: η t δεν πρέπει να είναι προσπελάσιμη

Μέγιστη Ροή – Ελεγκτής Ορθότητας

```
template <class NT>
bool CHECK_MAX_FLOW_T(const graph& G, node s, node t,
                      const edge_array<NT>& cap,
                      const edge_array<NT>& f)
{
  node v; edge e;
  string loc = "CHECK_MAX_FLOW_T: ";
  bool res = true;

  forall_edges(e,G)
    res = res && leda_assert(f[e] >= 0 && f[e] <= cap[e],
                             loc + "illegal flow value");

  node_array<NT> excess(G,0);
  forall_edges(e,G)
  { node v = G.source(e); node w = G.target(e);
    excess[v] -= f[e]; excess[w] += f[e];
  }
  forall_nodes(v,G)
    res = res && leda_assert(v == s || v == t ||
    excess[v] == 0,loc + "node with non-zero excess");

// Compute nodes reachable from s using BFS
}
```

Μέγιστη Ροή – Ελεγκτής Ορθότητας

```
// Compute nodes reachable from s using BFS

  node_array<bool> reached(G,false);
  queue<node> Q;

  Q.append(s); reached[s] = true;
  while ( !Q.empty() )
  { node v = Q.pop();
    forall_out_edges(e,v)
    { node w = G.target(e);
      if ( f[e] < cap[e] && !reached[w] )
      { reached[w] = true; Q.append(w); }
    }
    forall_in_edges(e,v)
    { node w = G.source(e);
      if ( f[e] > 0 && !reached[w] )
      { reached[w] = true; Q.append(w); }
    }
  }
  res = res && leda_assert(!reached[t],
                "t is reachable in G_f");
  return res;
```

Μέγιστη Ροή – Αλγόριθμος Προροής-Προώθησης

- *Προροή* : συνάρτηση $f : E \to \mathbb{R}_{\geq 0}$ για την οποία:

$$(1) \quad 0 \leq f(e) \leq cap(e) \quad \forall e \in E$$

$$(2) \quad excess(v) \geq 0 \quad \forall v \in V^+ = V - \{s, t\}$$

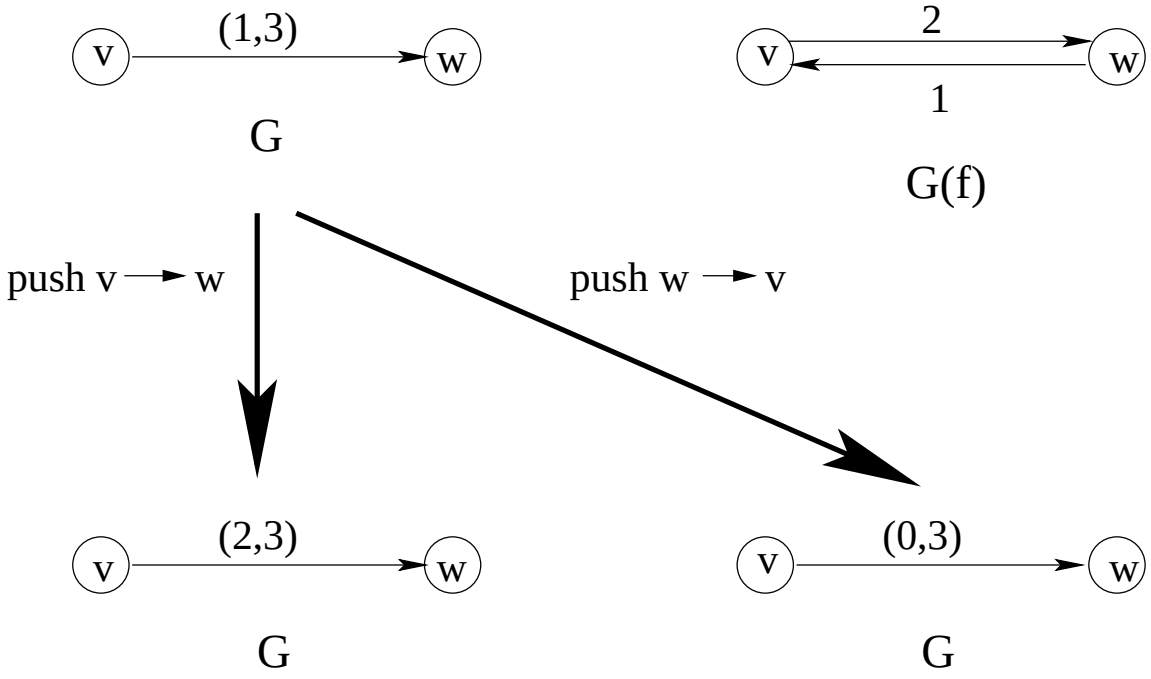
- *Ενεργή κορυφή* : $v \in V^+$ και $excess(v) > 0$

- *Προώθηση* : έστω v ενεργή κορυφή, $e = (v, w) \in G(f)$, και $\delta \leq \min\{excess(v), r(e)\}$
  Προώθηση δ μονάδων ροής κατά μήκος της $e \Longrightarrow$

$$\begin{cases} \text{αυξάνει την } f(e) \text{ κατά } \delta, & \text{αν } e \in E \\ \text{μειώνει την } f(e') \text{ κατά } \delta, & \text{αν } e' \in E \end{cases}$$

- Αν $\delta = r(e)$, τότε η προώθηση καλείται *κορεσμένη*

Μέγιστη Ροή – Αλγόριθμος Προροής-Προώθησης



Προώθηση 1 μονάδας ροής κατά μήκος της (v, w) (ή της (w, v))

Μέγιστη Ροή – Αλγόριθμος Προροής-Προώθησης

- *Ποιες προωθήσεις πρέπει να εκτελεστούν ;*

- Τοποθέτηση των κορυφών σε επίπεδα αρχίζοντας από την t και εκτέλεση προωθήσεων που μεταφέρουν πλεόνασμα από μεγαλύτερο σε μικρότερο επίπεδο

Μέγιστη Ροή – Αλγόριθμος Προροής-Προώθησης

- $d(v)$: επίπεδο κορυφής v
- Η ακμή $e = (v, w) \in G(f)$ καλείται *κατάλληλη* αν $d(w) < d(v)$
- **Προώθηση** κατά μήκος της $e = (v, w) \in G(f)$ εκτελείται αν **η v είναι ενεργή και η e κατάλληλη**

Μέγιστη Ροή – Αλγόριθμος Προροής-Προώθησης

- *Τι γίνεται όμως αν κάποια κορυφή v είναι ενεργή και $\nexists$ κατάλληλη ακμή προσκείμενη στην v;*

- Η v ανεβαίνει επίπεδο, δηλ. το $d(v)$ αυξάνεται κατά 1

Μέγιστη Ροή – Γενικευμένος Αλγόριθμος Προροής-Προώθησης

/* αρχικοποίηση */
Θέσε $f(e) = cap(e)$ για όλες τις ακμές με source$(e) = s$;
Θέσε $f(e) = 0$ για όλες τις υπόλοιπες ακμές ;
Θέσε $d(s) = n$ και $d(v) = 0$ για όλες τις κορυφές ;

/* κύριος βρόχος */
while $\exists$ ενεργή κορυφή
{ έστω v μια ενεργή κορυφή;
if $\exists$ κατάλληλη ακμή $e = (v, w) \in G_f$ **then**
{ προώθησε $\delta \leq \min\{excess(v), r(e)\}$ μονάδες ροής στην e;
else
{ αύξησε το $d(v)$; }
}

Μέγιστη Ροή – Αλγόριθμος Προροής-Προώθησης

- Μη κορεσμένες προωθήσεις μπορεί να δημιουργήσουν πρόβλημα
  $\Downarrow$
- **Κανόνας 1 – μέγιστη προώθηση:** προώθηση $\delta = \min\{excess(v), r(e)\}$
  $\Rightarrow$ Κάθε μη-κορεσμένη προώθηση μιας ακμής (v, w) αφήνει την v ανενεργή
- **Κανόνας 2 – επιμονή:** Όταν επιλέγεται μια ενεργή κορυφή v, εκτελούνται προωθήσεις από την v μέχρις ότου
  - είτε η v γίνει ανενεργή
  - είτε δεν υπάρχουν προσκείμενες κατάλληλες ακμές, οπότε πρέπει να αυξηθεί το $d(v)$
- *Πως επιλέγονται οι ενεργές κορυφές ;*

Επιλογή ενεργών κορυφών

- Αυθαίρετη $\Rightarrow$ αριθμός μη-κορεσμένων προωθήσεων $O(n^2 m)$
- FIFO $\Rightarrow$ αριθμός μη-κορεσμένων προωθήσεων $O(n^3)$
- Υψηλότερου επιπέδου $\Rightarrow$ αριθμός μη-κορεσμένων προωθήσεων $O(n^2\sqrt{m})$

Μέγιστη Ροή – Υλοποίηση Αλγορίθμου Προροής-Προώθησης

- ∃ δύο παράμετροι αρχετύπων: ο αριθμητικός τύπος `NT` και η υλοποίηση του συνόλου των ενεργών κορυφών `U`
- `// max_flow_basic`

```
template<class NT, class SET>
NT MAX_FLOW_BASIC_T(const graph& G, node s, node t,
                    const edge_array<NT>& cap,
                    edge_array<NT>& flow,
                    SET& U, int& num_pushes,
                    int& num_edge_inspections,
                    int& num_relabels)
{
 if (s == t) error_handler(1,"MAXFLOW: source == sink");

 // MF_BASIC: initialization

 // MF_BASIC: main loop

 #ifndef LEDA_CHECKING_OFF
   assert(CHECK_MAX_FLOW_T(G,s,t,cap,flow));
 #endif

 return excess[t];
}
```

Αρχικοποίηση & και Δομές Δεδομένων

```
// MF_BASIC: initialization

// initialize flow and excess, and saturate edges out of s
flow.init(G,0);
if (G.outdeg(s) == 0) return 0;

int n = G.number_of_nodes();
int max_level = 2*n - 1;
int m = G.number_of_edges();

node_array<NT>  excess(G,0);

// saturate all edges leaving s
edge e;
forall_out_edges(e,s)
{ NT c = cap[e];
  if (c == 0) continue;
  node v = target(e);
  flow[e] = c;
  excess[s] -= c;
  excess[v] += c;
}
```

Αρχικοποίηση & και Δομές Δεδομένων

```
// MF_BASIC: initialize dist and U

node_array<int> dist(G,0); dist[s] = n;
node v;
forall_nodes(v,G)
 if ( excess[v] > 0 ) U.insert(v,dist[v]);

// MF_BASIC: initialize counters

num_relabels = num_pushes = num_edge_inspections = 0;
```

Υλοποίηση του Συνόλου `U` των Ενεργών Κορυφών – Λειτουργίες

	Συναρτήσεις κατασκευής και κατάργησης
`node U.del():`	διαγραφή κορυφής από το `U` και επιστροφή της (επιστρέφει `nil` αν το `U` είναι κενό)
`U.insert(node v, int d):`	εισάγει την `v` με επίπεδο `d`. Αυτή η εκδοχή χρησιμοποιείται στην φάση της αρχικοποίησης και όταν μια κορυφή επανεισάγεται στο `U` μετά από αύξηση του επιπέδου της
`U.insert0(node v, int d):`	εισάγει την `v` με επίπεδο `d`. Αυτή η εκδοχή χρησιμοποιείται όταν μια κορυφή γίνεται ενεργή από μια ώθηση προς αυτή την κορυφή
`bool U.empty():`	επιστρέφει `true` όταν το `U` είναι κενό
`U.clear():`	διαγράφει όλα τα στοιχεία του `U`

Υλοποίηση FIFO – διατηρεί το U σαν ουρά

```
// FIFO implementation of SET

#include <LEDA/list.h>

class fifo_set{

  list<node> L;

public:

  fifo_set(){}
  node del() { if (!L.empty()) return L.pop();
                               else return nil; }

  void insert(node v, int d)  { L.append(v); }
  void insert0(node v, int d) { L.append(v); }

  bool empty() { return L.empty(); }
  void clear() { L.clear(); }
  ~fifo_set(){}
};
```

Υλοποίηση τροποποιημένης FIFO (MFIFO) – διατηρεί το U σαν λίστα

- Πάντοτε επιλέγει την πρώτη κορυφή της λίστας. Κορυφές που επανεισάγονται μετά από αύξηση επιπέδου τοποθετούνται στην αρχή της λίστας, ενώ οι κορυφές που γίνονται ενεργές λόγω μιας ώθησης τοποθετούνται στο τέλος της λίστας

```
// MFIFO implementation of SET
#include <LEDA/list.h>

class mfifo_set{
  list<node> L;

public:
  mfifo_set(){}
  node del() { if ( !L.empty() ) return L.pop();
                                 else return nil; }

  void insert(node v, int d) { L.push(v); }
  void insert0(node v, int d){ L.append(v); }

  bool empty() { return L.empty(); }
  void clear() { L.clear(); }
  ~mfifo_set(){}
};
```

Υλοποίηση Υψηλότερου Επιπέδου

- Διατηρεί ένα διάνυσμα `A` γραμμικών λιστών με αριθμοδείκτες στο διάστημα [0..max_level], όπου max_level είναι μια παράμετρος της συνάρτησης κατασκευής
- Η λίστα `A[d]` περιέχει όλες τις κορυφές `v` που εισήχθηκαν από `insert(v,d)` ή από `insert0(v,d)`
- Διατηρείται μια μεταβλητή `max`, έτσι ώστε η `A[d]` να είναι κενή αν $d >$ `max`
- Στην `insert0` λαμβάνεται υπόψη το γεγονός ότι η εισαγωγή κορυφών αφορά επίπεδα μικρότερα του max_level

Μέγιστη Ροή – Υλοποίηση Αλγορίθμου Προρροής-Προώθησης

```
// Highest level implementation of SET

#include <LEDA/list.h>
#include <LEDA/array.h>

class hl_set{

  int max, max_lev;
  array<list<node> > A;

public:

  hl_set(int max_level):A(max_level+1)
  { max = -1; max_lev = max_level;}

  node del()
  { while (max >= 0 && A[max].empty()) max--;
    if (max >= 0) return A[max].pop(); else return nil;
  }

  void insert(node v, int d)
  { A[d].push(v);
    if (d > max) max = d;
  }
```

Μέγιστη Ροή – Υλοποίηση Αλγορίθμου Προροής-Προώθησης

```
  void insert0(node v, int d) { A[d].append(v); }

  bool empty()
  { while (max >= 0 && A[max].empty()) max--;
    return ( max < 0 );
  }

  ~hl_set(){}

  void clear()
  { for (int i = 0; i <= max_lev; i++) A[i].clear();
    max = -1;
  }
};
```

Κύριος Βρόχος

- Επιλέγεται μια κορυφή v από το U
  - Αν $\nexists v$, τότε η εκτέλεση του βρόχου σταματάει
  - Αν η v είναι ταυτόσημη της t, τότε η εκτέλεση του βρόχου προχωρά στην επόμενη επανάληψη
- Προσπάθεια προώθησης του πλεονάσματος της v στις γειτονικές κορυφές της στο G_f
  - Εξέταση των ακμών του G_f που αντιστοιχούν σε εξερχόμενες ακμές της v στο G
  - Εξέταση των ακμών του G_f που αντιστοιχούν σε εισερχόμενες ακμές της v στο G
- Αν η v παραμένει ενεργή μετά τον κορεσμό όλων των γειτονικών ακμών της, τότε της αυξάνουμε το επίπεδο και την εισάγουμε ξανά στο U

Μέγιστη Ροή – Υλοποίηση Αλγορίθμου Προρροής-Προώθησης

```
// MF_BASIC: main loop

for(;;)
{
  node v = U.del();
  if (v == nil) break;
  if (v == t) continue;

  NT  ev = excess[v]; // excess of v
  int dv = dist[v];   // level of v
  edge e;

  // MF_BASIC: push across edges out of v

  if ( ev > 0 )
  { // MF_BASIC: push across edges into v }

  excess[v] = ev;

  if (ev > 0)
  { dist[v]++;
    num_relabels++;
    U.insert(v,dist[v]);
  }
}
```

Προώθηση πλεονάσματος μέσω εξερχόμενων ακμών

- Προώθηση πλεονάσματος μέσω κατάλληλων ακμών.
  Μια ακμή $e \in G(f)$ είναι
  - είτε και ακμή του G, οπότε $flow[e] < cap[e]$
  - είτε είναι η αντίθετη μιας ακμής του G, οπότε $flow[e'] > 0$

  $\Downarrow$

  Εξέταση όλων των εξερχόμενων και εισερχόμενων ακμών της v
- Για κάθε εξερχόμενη ακμή e μιας κορυφής v, προωθούμε $\min\{excess[v], cap[e] - flow[e]\}$
  Αν το πλεόνασμα της v γίνει μηδέν, τότε ο βρόχος `for` τερματίζει

Μέγιστη Ροή – Υλοποίηση Αλγορίθμου Προρροής-Προώθησης

```
// MF_BASIC: push across edges out of v

for (e = G.first_adj_edge(v); e; e = G.adj_succ(e))
{ num_edge_inspections++;
  NT& fe = flow[e];
  NT  rc = cap[e] - fe;
  if (rc == 0) continue;
  node w = target(e);
  int dw = dist[w];
  if ( dw < dv ) // equivalent to ( dw == dv - 1 )
  { num_pushes++;
    NT& ew = excess[w];
    if (ew == 0) U.insert0(w,dw);
    if (ev <= rc)
    { ew += ev; fe += ev;
      ev = 0;   // stop: excess[v] exhausted
      break;
    }
    else
    { ew += rc; fe += rc;
      ev -= rc;
    }
  }
}
```

Μέγιστη Ροή – Υλοποίηση Αλγορίθμου Προρροής-Προώθησης

```
// MF_BASIC: push across edges into v

for (e = G.first_in_edge(v); e; e = G.in_succ(e))
{ num_edge_inspections++;
  NT& fe = flow[e];
  if (fe == 0) continue;
  node w = source(e);
  int dw = dist[w];
  if ( dw < dv ) // equivalent to ( dw == dv - 1 )
  { num_pushes++;
    NT& ew = excess[w];
    if (ew == 0) U.insert0(w,dw);
    if (ev <= fe)
    { fe -= ev; ew += ev;
      ev = 0;    // stop: excess[v] exhausted
      break;
    }
    else
    { ew += fe; ev -= fe;
      fe = 0;
    }
  }
}
```

Αριθμητικοί Τύποι και Ορθότητα Αλγορίθμων

- Οι αποδείξεις ορθότητας των αλγορίθμων βασίζονται στους κανόνες της αριθμητικής
- Η ορθότητα των υλοποιημένων αλγορίθμων εξακολουθεί να ισχύει αν η υλοποίηση των αριθμητικών τύπων σέβεται τους κανόνες της αριθμητικής
- Όμως, ...
  - αριθμητική με τύπους `int` μπορεί να προκαλέσει υπερχείλιση, ή περίεργα αποτελέσματα λόγω αναδίπλωσης
  - αριθμητική με τύπους `double` μπορεί να προκαλέσει σφάλματα στρογγυλοποίησης

Πώς εξασφαλίζουμε ορθότητα ;

1. Αναλύουμε τις αριθμητικές απαιτήσεις του αλγορίθμου
   Στόχος: εύρεση μιας τιμής f, για την οποία ισχύει το εξής:

   Αν η μέγιστη απόλυτη ακέραια τιμή εισόδου φράσσεται από την C, τότε όλοι οι αριθμοί που χρησιμοποιούνται από τον αλγόριθμο είναι ακέραιοι και φράσσονται από την ποσότητα $f \cdot C \Longrightarrow$ f-φραγμένος αλγόριθμος

   Π.χ. για αλγόριθμους συντομότερων διαδρομών $f = n$, ενώ για αλγόριθμους ροών $f = \text{out-degree}(s)$

Πώς εξασφαλίζουμε ορθότητα ;

2. *Τύπος* `int`*:* ελέγχουμε αν όλες οι τιμές w της εισόδου ικανοποιούν $f \cdot w \leq \text{MAXINT}$
2. *Τύπος* `double`*:* μετασχηματίζουμε κατάλληλα τις αριθμητικές τιμές εισόδου

$$w \longrightarrow sign(w) \cdot \lfloor |w| \cdot S \rfloor / S$$

όπου $S = 2^s$ είναι η παράμετρος κλιμάκωσης (κατάλληλη δύναμη του 2)

$\Downarrow$

Η αριθμητική κινητής υποδιαστολής δεν θα δημιουργεί σφάλματα στρογγυλοποίησης όταν τα ενδιάμεσα αποτελέσματα είναι της μορφής $z \cdot 2^{-s}$, όπου z είναι ακέραιος

Πως επιλέγεται η τιμή του s ;

- Έστω C η μεγαλύτερη απόλυτη τιμή εισόδου
- Το βήμα 1 πρέπει να ισχύει και για τις μετασχηματισμένες ακέραιες τιμές $sign(w) \cdot \lfloor |w| \cdot S \rfloor$
  Άρα, για έναν f-φραγμένο αλγόριθμο

$$f \cdot \lfloor C \cdot S \rfloor < 2^{53}$$

  αφού η αριθμητική κινητής υποδιαστολής για τύπο `double` μπορεί να αναπαραστήσει όλους τους ακεραίους στο διάστημα $[-(2^{53}-1)..2^{53}-1]$
- Η παραπάνω ανισότητα ικανοποιείται αν

$$f \cdot C \cdot S < 2^{53} \iff s < 53 - \log(f \cdot C)$$

Ποια είναι η σχέση μεταξύ του αποτελέσματος με μετασχηματισμένες τιμές και εκείνου με τις αυθεντικές τιμές εισόδου ;

- $\nexists$ γενικός κανόνας· συνήθως το αποτέλεσμα με μετασχηματισμένες τιμές είναι μια καλή προσέγγιση του πραγματικού αποτελέσματος
- Έστω ότι το αποτέλεσμα είναι το άθροισμα $\leq L$ τιμών
  Η διαφορά της πραγματικής τιμή w από την μετασχηματισμένη w' είναι

$$w - w' = w - \lfloor w \cdot S \rfloor / S = (w \cdot S - \lfloor w \cdot S \rfloor)/S \leq 1/S$$

  Επομένως, το άθροισμα L μετασχηματισμένων τιμών διαφέρει από το άθροισμα L πραγματικών τιμών το πολύ κατά L/S
- Αν επιλεγει το S ως η μέγιστη δύναμη του 2 για την οποία $S < 2^{53}/(f \cdot C)$, τότε $S \geq 2^{52}/(f \cdot C)$ και άρα

$$\frac{L}{S} \leq \frac{L \cdot f \cdot C}{2^{52}}$$

  που είναι το μέγιστο απόλυτο λάθος

Τέλος Ενότητας

Σημείωμα Ιστορικού Εκδόσεων Έργου

Το παρόν έργο αποτελεί την έκδοση **1.0**.

Σημείωμα Ιστορικού Εκδόσεων Έργου

Σημείωμα Ιστορικού Εκδόσεων Έργου

Διατήρηση Σημειωμάτων

Οποιαδήποτε αναπαραγωγή ή διασκευή του υλικού θα πρέπει να συμπεριλαμβάνει :

- το Σημείωμα Αναφοράς
- το Σημείωμα Αδειοδότησης
- τη δήλωση Διατήρησης Σημειωμάτων
- το Σημείωμα Χρήσης Έργων Τρίτων (εφόσον υπάρχει) μαζί με τους συνοδευόμενους υπερσυνδέσμους