Olimpiadi di Informatica 2011
Giornate preparatorie

Dipartimento di Informatica
Università di Torino

marzo 2011

15 – Grafi: visite e algoritmi.
(versione 04/04/11)

Visite di un grafo e algoritmi sui grafi.

Visita di un grafo.

Si possono usare gli stessi algoritmi introdotti per gli alberi. Poiché però nei grafi, diversamente dagli alberi, un nodo può essere raggiungibile attraverso più archi, bisogna marcare i nodi che vengono visitati, in modo che se si raggiunge un nodo già visitato non lo si visiti di nuovo.

Nella visita generica di un grafo si fa uso di un insieme F, o frangia, che contiene almeno tutti i nodi non ancora visitati ma adiacenti a nodi visitati.

Tale insieme F è l'analogo della coda e della pila usate dagli algoritmi iterativi di visita di alberi rispettivamente per ampiezza e per profondità. Anche qui, poiché un nodo in un grafo può essere in generale adiacente a nodi diversi, quando si inserisce un nodo in F conviene controllare che non sia già presente (perché inserito precedentemente).

A seconda della struttura-dati impiegata per F, si ottengono, come nel caso degli alberi, i diversi tipi di visita:

- coda: visita in ampiezza;
  - coda con priorità: visita in ampiezza per cammini minimi e minimo albero ricoprente
- pila: visita in profondità.

Specializzando le visite per ampiezza e per profondità si ottengono i più importanti algoritmi sui grafi.

Algoritmi sui grafi (sommario)

- visita generica;
  - visita in ampiezza (iterativa con uso di *coda*);

    su grafi pesati:
    - cammini minimi da un nodo: algoritmo di Dijkstra (uso di *coda con priorità*);
    - minimo albero di copertura: algoritmo di Prim (uso di *coda con priorità*);
    - minimo albero di copertura: algoritmo di Kruskal (uso di *struttura union-find*);
  - visita in profondità (iterativa con uso di *pila*, o ricorsiva);
    - ordinamento topologico;
    - componenti fortemente connesse.

Visita di un grafo generica: visita di sottografo connesso a partire da un nodo.

- È un algoritmo di visita *visit(G, s)* che, dati un grafo *G* e un nodo di partenza *s*, visita di tutti i nodi raggiungibili da *s*, cioè di tutti i nodi del sottografo connesso contenente *s*.
- Costruisce implicitamente l'albero di visita, cioè l'albero di radice *s* in cui il genitore di un nodo *v* è l'altro estremo *u* dell'arco (u, v) percorrendo il quale si è arrivati a *v*.
- La costruzione dell'albero di visita può essere resa esplicita nell'algoritmo stesso, ad es. memorizzando per ogni nodo il suo genitore, o memorizzando l'insieme di archi percorsi.
- Per modularità, esprimiamo l'algoritmo per mezzo di due pseudo-procedure, in modo da operare prima della visita la marcatura iniziale dei nodi come *non-visitati*.

Visita di un grafo generica: visita di sottografo connesso a partire da un nodo.

```
visitConnected(G, s) {
  marca tutti i nodi di G come non visitati;
  crea la struttura F vuota;
  visit(G, s);
}

visit(G, s) {
  inserisci s in F;
  while(F non è vuota) {
     estrai da F un nodo u;
     visita u;  marca u come visitato;
     for each (nodo v adiacente a u)
       if (v è non visitato && v non è in F ) inserisci v in F;
   }
}
```


Visita di un grafo: versione con costruzione dell'albero di visita.

```
visitConnected(G, s) {  … ; crea l'albero T vuoto; visit(G, s); }

visit(G, s) {
  inserisci s in F; metti (null, s) in T;
  while(F non è vuota) {
     estrai da F un nodo u;
     visita u;  marca u come visitato;
      for each (nodo v adiacente a u)
        if (v è non visitato && v non è in F ) {
           inserisci v in F;
           metti in T l'arco (u, v);
        }
      }
   }
}
```

Nota

```
visit(G, s) {
  inserisci s in F;
  while(F non è vuota) {
     estrai da F un nodo u;
     visita u;  marca u come visitato;
     for each (nodo v adiacente a u)
        if (v è non visitato && v non è in F ) inserisci v in F;
   }
}
```

Nota: il controllo "*v non è in F*" è necessario per evitare di inserire in F un elemento già presente perché adiacente anche ad un nodo precedentemente visitato.

Per eseguire tale test in tempo costante conviene marcare i nodi come "appartenenti a F" all'istante dell'inserimento in F.

Visita completa di un grafo

- La visita completa di un grafo si effettua invocando successivamente l'algoritmo visit(G, s) sui nodi che risultano non ancora visitati, cioè sui nodi non raggiunti nelle invocazioni precedenti.

```
visitAll(G) {
  marca tutti i nodi di G come non visitati (o come bianchi);
  crea la struttura F vuota; crea la foresta T vuota;
  for each (nodo u del grafo G)
     if (u è non visitato) visit(G, u);
}
```

- L'algoritmo costruisce (implicitamente o esplicitamente) la foresta di visita, costituita da tutti gli alberi di visita dei sottografi connessi.

Complessità (della visita completa)

Siano n = numero dei nodi, m = numero degli archi.

Analisi:

- inizializzazione delle marcature dei nodi: $O(n)$;
- inserimenti in F ed estrazioni da F: $O(n)$
  (perché ogni nodo viene inserito ed estratto una volta sola, e si assume che inserimento ed estrazione siano $O(1)$);
- tempo totale di scansione delle liste di adiacenza: $O(m)$
  perché ogni lista di adiacenza di un nodo viene scandita una volta sola, e la somma dei gradi dei nodi di un grafo non orientato è $2m$, la somma dei gradi uscenti dei nodi di un grafo orientato è m.

Complessivamente:

$$T(n, m) = O(n + m)$$

F è una coda: visita in ampiezza.

```
visitAllInAmpiezza(G) {
  marca tutti i nodi di G come non visitati (o bianchi);
   F = CodaVuota(); T = ForestaVuota();
   for each (nodo u del grafo G)
     if (u è non visitato) visitaInAmpiezza(G, u);
}
```

F è una pila (stack): visita in profondità.

```
visitAllInProfondita(G) {
  marca tutti i nodi di G come non visitati (o bianchi);
   F = StackVuoto(); T = ForestaVuota();
   for each (nodo u del grafo G)
     if (u è non visitato) visitaInProfondita(G, u);
}
```

Visita in ampiezza; albero rappresentato dall' array dei padri.

```
visitaInAmpiezza (il grafo G a partire dal nodo s) {
  inizia_visita(s);
  marca s come visitato; padre[s] = null; accoda s in F;
  while(F non è vuota) {
     u = estrai il primo da F;
     for each (nodo v adiacente a u) {
        if (not v visitato) {
          inizia_visita(v); marca v come visitato;
          padre[v] = u;
          accoda v in F;
        }
     }termina_visita(u);
  }
}
```


Visita in profondità; albero rappresentato dall' array dei padri.

```
visitaInProfondita (il grafo G a partire dal nodo s) {
  inizia_visita(s);
  marca s come visitato; padre[s] = null; push s in F;
  while(F non è vuota) {
     u = pop da F;
     for each (nodo v adiacente a u) {
        if (not v visitato) {
          inizia_visita(v); marca v come visitato;
          padre[v] = u;
          push v in F;
        }
     }termina_visita(u);
  }
}
```

Visita in profondità ricorsiva.

Come negli alberi, la visita in profondità si può realizzare molto semplicemente in modo ricorsivo. Occorre però, anche qui, marcare i nodi che vengono visitati.

```
visitaInProfondità (il grafo G a partire dal nodo s) {
  marca tutti i nodi di G come non visitati; T = albero vuoto;
  visitaRic(s, T);
}

visitaRic (il grafo G dal nodo u) {
  inizia_visita(u); marca u come visitato;
  for each (nodo v adiacente a u)
     if (v è non visitato) {
       aggiungi l'arco (u, v) a T;
       visitaRic(v, T);
      }
  termina_visita(u);
}
```


Visita in profondità ricorsiva.

- Si noti che, come sempre, lo stack esplicito della versione iterativa dell'algoritmo diventa, nella versione ricorsiva, lo stack "automatico" con cui è implementata la ricorsione.
- L'istante di inizio visita, che nella versione iterativa è l'istante in cui il nodo viene spinto sullo stack, nella versione ricorsiva è naturalmente l'istante in cui il frame relativo a quel nodo viene allocato sullo stack, cioè è l'istante iniziale della chiamata ricorsiva.
- L'istante di fine visita, che nella versione iterativa è l'istante in cui il nodo viene tolto dallo stack, nella versione ricorsiva è naturalmente l'istante in cui il frame relativo a quel nodo viene deallocato, cioè è l'istante finale della chiamata ricorsiva.

Complessità

Tutti gli algoritmi di semplice visita considerati nelle slides precedenti non sono che specializzazioni dell'algoritmo di visita generico dove F è una coda oppure una pila.

Per pila e coda l'assunzione che inserimento ed estrazione avvengano in tempo costante è valida, quindi la complessità è sempre $O(n + m)$.

Nota. Negli algoritmi successivi, ancora basati su visite in ampiezza o in profondità, ma con uso per F di strutture-dati più sofisticate, l'assunzione di cui sopra potrà non essere più valida, e la complessità potrà quindi essere diversa (vedi ad esempio gli algoritmi di Dijkstra e di Prim).