Visite di grafi

Gustave Klimt *Albero della vita* 1909

Visite di grafi

- Le visite dei grafi possono essere realizzate partendo dalla realizzazione delle visite degli alberi
- Dato un grafo G=(V,E), gli algoritmi per la visita di un grafo partono da un vertice $v \in V$ e visitano tutti i vertici che sono raggiungibili da v attraverso gli archi E
- Visto che un grafo può contenere dei cicli, sia la visita in ampiezza (BFS) sia la visita in profondità (DFS) usano un array booleano di appoggio *raggiunto[V]* tale per cui, dato $v \in V$, raggiunto[v] = TRUE sse il nodo è già stato scoperto

Visita BFS iterativa

Algoritmo visitaBFSIterativa(grafo G=(V, E), vertice v)

1) raggiunto[V]=FALSE
2) coda C
3) enqueue(C,v)
4) raggiunto[v]=TRUE
5) while (C!=NULL)
   a) u=dequeue(C)
   b) visita il vertice u
   c) for each vertex w of Adj[u]
      If ! raggiunto[w]
         a) raggiunto[w]=TRUE
         b) enqueue(C,w)

Modulo codaBFS

Abbiamo bisogno di un modulo che implementi la coda usata nella BFS.

Possiamo partire dal modulo codaBFS creato per gli alberi e:

- modificare il tipo dell'informazione contenuta nell'elemento

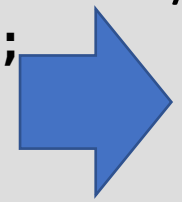
```
struct elemBFS
{  node* inf;
   elemBFS* pun ;
};
```

```
struct elemBFS
{  int inf;
   elemBFS* pun ;
};
```

Identificativo del nodo

- modificare le primitive che accedono/usano il contenuto informativo:

```
codaBFS enqueue(codaBFS, node*);
node* dequeue(codaBFS&);
node* first(codaBFS);
static elemBFS* new_elem(node*);
```



```
codaBFS enqueue(codaBFS, int);
int dequeue(codaBFS&);
int first(codaBFS);
static elemBFS* new_elem(int);
```

SOLUZIONE Vedi directory *coda-bfs*

Il grafo è connesso?

- Consideriamo grafi non orientati
- L'algoritmo di visita BFS può essere usato per verificare se il grafo è connesso
- Un grafo è connesso se al termine della visita raggiunto[v] = TRUE, per ogni v $\in V$

ESERCIZIO Si assuma che i grafi acquisiti da file siano sempre non orientati. Estendere il progetto sui grafi:

- Aggiungendo il modulo codaBFS
- Aggiungendo al file graph_sol.cc la funzione booleana **`connected`** che restituisce TRUE se il grafo è connesso, FALSE altrimenti

SOLUZIONE Vedi sorgente *graph_connected_sol.cc*

Componenti connesse

- Sia G=(V,E) un grafo non orientato
- Una componente connessa (**connected component**) è un sottografo G' di G connesso (ovvero ogni coppia di nodi di G' è connessa) e massimale di G
- Le visite dei grafi sono utili per individuare le componenti connesse di un grafo non orientato:
  1. Eseguo la visita del grafo
  2. L'elenco dei nodi v che sono stati raggiunti dalla visita costituiscono una componente connessa
  3. Se non ho raggiunto tutti i nodi torno al passo 1 partendo da uno dei nodi che non sono stati raggiunti dalla visita

Progettazione del codice

- Rivedere la funzione **`connected(graph g, node v)`** affinché restituisca il vettore dei nodi raggiungibili a partire da v
- Aggiungere al file *graph_connected_sol.cc* la procedura **`connected_component`** che:
  - Mantenga traccia dei nodi raggiunti fino a quel attraverso il vettore *raggiunto-globale[V]* inizializzato a FALSE
  - RIchiami **`connected`**, per ogni v $\in V$, tale per cui *raggiunto-globale[v]=FALSE*
  - Usi il risultato di **`connected`** per
    - stampare l'elenco dei nodi che fanno parte della stessa componente connessa
    - Aggiornare il vettore *raggiunto-globale*

Albero BFS e albero di copertura

- In una visita BFS gli archi che conducono a vertici ancora non visitati formano un albero detto **albero BFS** la cui struttura dipende dall'ordine di visita
- Per poter costruire l'albero BFS durante la visita possiamo registrare il padre di ogni nodo nel *vettore dei padri*
  - L'uso del vettore dei padri non rappresenta un limite in questo caso perché la dimensione del grafo è nota e quindi è possibile allocare dinamicamente il vettore dei padri all'inizio della visita
- Se l'albero BFS include tutti i vertici di G allora l'albero BFS ottenuto è un **albero di copertura** o **spanning tree**

Albero BFS

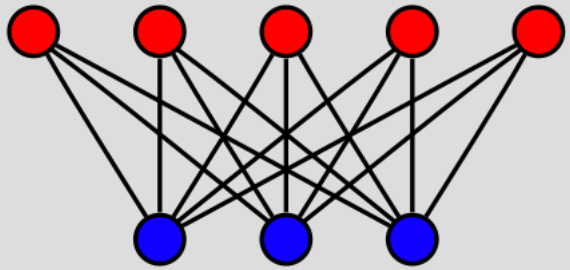
Algoritmo visitaBFSIterativa(grafo G=(V, E), vertice v)

1) raggiunto[V]=FALSE
2) padre[V]=-1
3) coda C
4) raggiunto[v]=TRUE
5) enqueue(C,v)
6) while (C!=NULL)
   a) u=dequeue(C)
   b) for each vertex w of Adj[u]
      If ! raggiunto[w]
         a) raggiunto[w]=TRUE
         b) padre[w]=u
         c) enqueue(C,w)

ESERCIZIO Estendere il file *graph_connected_sol.cc* affinché consenta di avviare la visita del grafo a partire da un nodo specificato dall'utente e stampi l'albero BFS di vista solo se è uno spanning tree

Esercizio non risolto

Un **grafo bipartito** è un grafo tale che l'insieme dei suoi vertici si può partizionare in due sottoinsiemi tali che ogni vertice di una di queste due parti è collegato solo a vertici dell'altra.



Si assuma di avere un grafo non orientato e connesso, estendere il file graph_connected_sol.cc con la funzione booleana bipartito che restituisce TRUE se il grafo è biparito, FALSE altrimenti