

Graph Decomposition

— DFS&BFS, Cycle, DAG, SCC, and Biconnectivity

hengxin0912@gmail.com

May 19, 2016

Graph Decomposition

- 1 Overview
- 2 DFS and BFS
- 3 Cycle
- 4 DAG

Contents of Tutorials

1. Graph Traversal Decomposition
2. MST (Greedy Algorithm) & Path
3. DP: Dynamic Programming

Graph Decomposition

Graph decomposition vs. Graph traversal

Graph Decomposition

- 1 Overview
- 2 DFS and BFS**
- 3 Cycle
- 4 DAG

Classifying edges

Definition (Classifying edges)

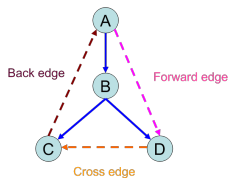
Given a dfs/bfs traversal:

- ▶ Tree edge
- ▶ Back edge
- ▶ Forward edge
- ▶ Cross edge

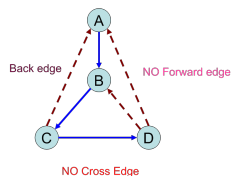
Remarks:

- ▶ Applicable to both DFS and BFS
- ▶ With respect to DFS/BFS trees

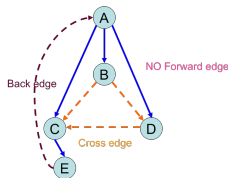
Classifying edges [Problem: 3.4.1]



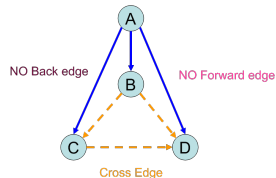
(a) DFS on directed graph.



(b) DFS on undirected graph.



(c) BFS on directed graph.



(d) BFS on undirected graph.

Classifying edges

DFS tree and BFS tree coincide [Problem: 3.4.30]

$G = (V, E), v \in V$. DFS tree T = BFS tree T' .

- ▶ G is an undirected graph $\Rightarrow G = T$.
- ▶ G is a digraph $\Rightarrow^? G = T$.

Solution.

- ▶ T : tree + back; T' : tree + cross
- ▶ T : tree + back + forward + cross; T' : tree + back + cross

Distance constraints for BFS

Distance constraints for BFS [Problem: 3.4.4]

BFS on digraph:

TE: $d[v] = d[u] + 1$

BE: $0 \leq d[v] \leq d[u]$

CE: $d[v] \leq d[u] + 1$

BFS on undirected graph:

TE: $d[v] = d[u] + 1$

CE: $d[v] = d[u] \vee d[v] = d[u] + 1$

Solution to “CE in BFS on *undirected* graph”.

- ▶ $d[v] = d[u], d[v] = d[u] + 1$
- ▶ $d[v] < d[u], d[v] > d[u] + 1$

Remark.

- ▶ BFS tree defines a *shortest-path* from its root to every other node.
- ▶ Layers in BFS on *undirected* graph \Rightarrow bipartite testing [Problem: 3.4.26]

Graph Decomposition

- 1 Overview
- 2 DFS and BFS
- 3 Cycle**
- 4 DAG

Cycle detection

Table: Cycle detection [Problem: 3.4.21]

	Digraph	Undirected graph
DFS	back edge \iff cycle	back edge \iff cycle
BFS	back edge \Rightarrow cycle cycle \nRightarrow back edge	cross edge \iff cycle

Remark.

- ▶ cycle in undirected graphs
- ▶ cycle in digraphs
 - ▶ DAG
 - ▶ SCC

Orientation of undirected graph

Orientation of undirected graph [Problem: 3.4.13]

Undirected (connected) graph G , edge oriented s.t. $\forall v, \text{in}[v] \geq 1$.

Solution.

orientation $\iff \exists$ cycle; DFS

Bipartite graph

Bipartite graph [Problem: 3.4.26; 3.4.32]

To test bipartiteness of an undirected graph.

Solution.

BFS + Coloring:

- ▶ pick any s , $c[s] = 0$
- ▶ $u \leftarrow \text{Dequeue}(Q)$
- ▶ $\forall(u, v)$:
 - ▶ tree edge
 - ▶ cross edge: **check**

Proof.

Check cross edge (u, v) :

- ▶ $(\exists) d[v] = d[u] \Rightarrow$ the same layer \Rightarrow odd cycle ([Problem: 3.4.17]; **EX**) \Rightarrow non-bipartite
- ▶ $(\forall) d[v] = d[u] + 1 \Rightarrow$ different layers \Rightarrow different colors



Graph Decomposition

- 1 Overview
- 2 DFS and BFS
- 3 Cycle
- 4 DAG**

DAG

no back edge \iff DAG $\iff \exists$ topo. ordering

Topo. sorting

DFS on digraph, $u \rightarrow v$:

- ▶ back edge: $f(u) < f(v)$
- ▶ others: $f(u) > f(v)$

$$u \rightarrow v \Rightarrow u \prec v$$

$$u \rightarrow v \Rightarrow f(u) > f(v)$$

Topo. sorting: sort vertices in *decreasing* order of their *finish* times.

Digraph as DAG

Digraph as DAG [Problem: 3.4.6]

Theorem

Every digraph is a dag of its SCCs.

Remark.

- ▶ SCC algorithm
- ▶ SCC: reachability/connectivity equivalence class
- ▶ two tiered structure of digraphs

Kahn's toposort algorithm

Kahn's toposort algorithm (1962) [Problem: 3.4.19]

- ▶ queue for source vertices ($\text{in}[v] = 0$)
- ▶ repeat: dequeue v , delete it, output it

Solution.

Lemma

Every DAG has at least one source and at least one sink vertex.

Remark

DFS on DAG:

- ▶ $\arg \max_v f(v) \Rightarrow$ source (used in SCC algorithm)
- ▶ $\arg \min_v f(v) \Rightarrow$ sink

Hamiltonian path in DAG

Hamilton path in DAG [Problem: 3.4.16]

- ▶ DAG G
- ▶ path visiting each vertex once

Solution.

- ▶ general digraph: NP-hard
- ▶ dag: \exists HP $\iff \exists!$ topo. ordering
 - ▶ \Leftarrow : By contradiction. $\exists u \sim v : u \nrightarrow v$; swap

Algorithm:

- ▶ toposort, check edges
- ▶ the Kahn toposort algorithm

Semi-connected DAG

Definition

Semi-connected digraph $\forall u, v : u \rightsquigarrow v \vee v \rightsquigarrow u$

Semi-connected DAG [Problem: 3.4.21 (c) + (d)]

To test whether a DAG is semi-connected.

Solution.

dag: $\exists \text{ HP} \iff \exists! \text{ topo. ordering} \iff \text{semi-connected}$

Proof.

- ▶ \Leftarrow : by contradiction; total order ($\forall u, v : u \prec v \vee v \prec u$)
- ▶ \Rightarrow : $\exists \text{HP}$



Minimum cost reachable

Minimum cost reachable [Problem: 3.4.22]

Compute $\text{cost}[u] = \min\{\text{cost}[v] \mid u \rightsquigarrow v\}$.

Solution.

- ▶ dag: reverse topo. ordering
 - ▶ backtracking: $\text{cost}[u] = \min_{u \rightarrow v} \{\text{cost}[v]\}$
- ▶ digraph: dag of scc

Line up

Line up [Problem: 3.4.29]

- ▶ i hates j : $i \prec j$
- ▶ i hates j : $\#i < \#j$

Solution.

i hates j : $i \rightarrow j$;

- ▶ DAG?
- ▶ longest path; critical path

One-to-all reachability

One-to-all reachability [Problem: 3.4.28]

- ▶ given $v : v \rightsquigarrow^? \forall u$
- ▶ $\exists? v : v \rightsquigarrow \forall u$

Solution.

- ▶ DFS/BFS
- ▶ SCC; $\exists!$ source vertex $v \iff v \rightsquigarrow \forall u$

Proof.

- ▶ \Rightarrow : By contradiction. $\exists u : v \nrightarrow u \wedge \text{in}[u] > 0 \Rightarrow \exists u' \rightarrow u \wedge v \nrightarrow u'$. Cycle.
- ▶ \Leftarrow : (1) source; (2) $\exists!$

