Discrete Optimization
Graphs

Ngày 20 tháng 7 năm 2011

Lecture 6: Graphs

In this class we shall prove some simple, useful theorems about graphs. We start with very simple observaations.

Lecture 6: Graphs

In this class we shall prove some simple, useful theorems about graphs. We start with very simple obseravations.

Theorem (6-1)

Let $d(v_1), d(v_2), \ldots, d(v_n)$ be the degree sequence of the vertices of the graph $G(V, E)$. Then $\sum_{i=1}^{n} d(v_i) = 2|E|$.

In this class we shall prove some simple, useful theorems about graphs. We start with very simple observations.

Theorem (6-1)

Let $d(v_1), d(v_2), \ldots, d(v_n)$ be the degree sequence of the vertices of the graph $G(V, E)$. Then $\sum_{i=1}^{n} d(v_i) = 2|E|$.

Observation

Lecture 6: Graphs

In this class we shall prove some simple, useful theorems about graphs. We start with very simple observations.

Theorem (6-1)

Let $d(v_1), d(v_2), \ldots, d(v_n)$ be the degree sequence of the vertices of the graph $G(V, E)$. Then $\sum_{i=1}^{n} d(v_i) = 2|E|$.

Observation

- *If $d_G(v) \geq 2 \; \forall v \in V(G)$ then G contains cycles.*

Lecture 6: Graphs

In this class we shall prove some simple, useful theorems about graphs. We start with very simple obseravations.

Theorem (6-1)

Let $d(v_1), d(v_2), \ldots, d(v_n)$ be the degree sequence of the vertices of the graph $G(V, E)$. Then $\sum_{i=1}^{n} d(v_i) = 2|E|$.

Observation

- *If $d_G(v) \geq 2 \; \forall v \in V(G)$ then G contains cycles.*

Corolary (6-2)

A tree has vertices of degree 1. Such vertices are called **leaves**.

In this class we shall prove some simple, useful theorems about graphs. We start with very simple observations.

Theorem (6-1)

Let $d(v_1), d(v_2), \dots, d(v_n)$ be the degree sequence of the vertices of the graph $G(V, E)$. Then $\sum_{i=1}^{n} d(v_i) = 2|E|$.

Observation

- *If $d_G(v) \geq 2 \; \forall v \in V(G)$ then G contains cycles.*

Corolary (6-2)

A tree has vertices of degree 1. Such vertices are called **leaves**.

- *If G is a connected graph and $d_G(v_i) = 1$ then $G \setminus \{v_i\}$ remains connected.*

Lecture 6: Graphs

In this class we shall prove some simple, useful theorems about graphs. We start with very simple observations.

Theorem (6-1)

Let $d(v_1), d(v_2), \ldots, d(v_n)$ be the degree sequence of the vertices of the graph $G(V, E)$. Then $\sum_{i=1}^{n} d(v_i) = 2|E|$.

Observation

- *If $d_G(v) \geq 2 \; \forall v \in V(G)$ then G contains cycles.*

Corolary (6-2)

A tree has vertices of degree 1. Such vertices are called **leaves**.

- *If G is a connected graph and $d_G(v_i) = 1$ then $G \setminus \{v_i\}$ remains connected.*
- *If C_k is a cycle in G and $e \in E(G) \cap C_k$ then $G \setminus \{e\}$ remains connected.*

Mathematical representation of graphs

Two structures are commonly used to represent graphs:

- **Adjacency Matrix** (smetimes called incidence matrix)

Mathematical representation of graphs

Two structures are commonly used to represent graphs:

- **Adjacency Matrix** (smetimes called incidence matrix)
- **Adjacency list**

Mathematical representation of graphs

Two structures are commonly used to represent graphs:

- **Adjacency Matrix** (smetimes called incidence matrix)
- **Adjacency list**

(The adjacency matrix)

Let $G(V, E)$ be a labeled graph of order n. The adjacency matrix of G denoted by $A(G)$ is the $n \times n$ matrix defined by:

$$A_{i,j} = \begin{cases} 0 & \textit{if } i = j \text{ or } (i,j) \notin E(G) \\ 1 & \textit{if } (i,j) \in E(G) \end{cases}$$

Mathematical representation of graphs

Two structures are commonly used to represent graphs:

- **Adjacency Matrix** (smetimes called incidence matrix)
- **Adjacency list**

(The adjacency matrix)

Let $G(V, E)$ be a labeled graph of order n. The adjacency matrix of G denoted by $A(G)$ is the $n \times n$ matrix defined by:

$$A_{i,j} = \begin{cases} 0 & \textit{if } i = j \textit{ or } (i,j) \notin E(G) \\ 1 & \textit{if } (i,j) \in E(G) \end{cases}$$

Comment

This representation can also be used for digraphs or weighted graphs. For a simple graph, the matrix is symmetric. In weighted graphs $1'$s will be replaced by the weight of the edge (i, j).

Adcacency matrix representation of a digraph

0:	0	0	1	1	1	1	0	1	1	1	0	1	0	1	0	1	0	0	0	1
1:	0	0	1	1	0	0	1	0	0	0	1	0	1	0	1	1	0	1	0	0
2:	1	1	0	0	0	1	0	0	0	1	1	0	1	1	0	0	1	1	0	1
3:	1	1	0	0	1	0	1	1	0	1	0	0	0	1	0	1	0	1	0	1
4:	1	0	0	1	0	1	0	0	0	0	1	0	0	0	0	1	1	1	0	0
5:	1	0	1	0	1	0	0	1	0	0	0	1	1	0	1	1	0	0	1	0
6:	0	1	0	1	0	0	0	1	0	1	1	0	0	1	1	1	0	0	0	1
7:	1	0	0	1	0	1	1	0	1	1	0	0	1	0	0	0	0	0	0	0
8:	1	0	0	0	0	0	0	1	0	0	0	1	1	0	0	0	1	0	0	1
9:	1	0	1	1	0	0	1	1	0	0	1	0	1	0	0	1	0	0	1	1
10:	0	1	1	0	1	0	1	0	0	1	0	0	0	1	1	0	1	0	0	0
11:	1	0	0	0	0	1	0	0	1	0	0	0	1	0	1	0	1	1	1	0
12:	0	1	1	0	0	1	0	1	1	1	0	1	0	0	1	0	1	0	1	0
13:	1	0	1	1	0	0	1	0	0	0	1	0	0	0	0	0	1	0	0	1
14:	0	1	0	0	0	1	1	0	0	0	1	1	1	0	0	1	1	1	1	0
15:	1	1	0	1	1	1	1	0	0	1	0	0	0	0	1	0	0	1	0	0
16:	0	0	1	0	1	0	0	0	1	0	1	1	1	1	1	0	0	0	0	1
17:	0	1	1	1	1	0	0	0	0	0	0	1	0	0	1	1	0	0	0	1
18:	0	0	0	0	0	1	0	0	0	1	0	1	1	0	1	0	0	0	0	0
19:	1	0	1	1	0	0	1	0	1	1	0	0	0	1	0	0	1	1	0	0

A simple graph of order 20

Adcacency matrix representation of a digraph

0:	0	0	0	1	1	1	1	1	1	0	1	0	1	1	0	0	0	0	1	1
1:	1	0	0	1	0	0	0	0	1	0	1	0	1	1	0	1	0	1	1	0
2:	0	1	0	0	0	1	1	1	1	1	1	0	0	1	0	0	1	0	1	1
3:	0	0	0	0	0	1	0	0	1	1	1	1	1	1	1	1	0	1	1	1
4:	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	1	1	0	1	1
5:	0	0	0	1	1	0	0	0	1	0	1	0	0	1	1	1	1	0	0	0
6:	1	1	0	1	1	1	0	0	1	0	0	0	1	0	0	1	0	0	0	1
7:	0	0	0	1	1	1	0	0	0	1	1	0	0	0	0	1	0	1	1	1
8:	1	1	1	1	1	1	1	1	0	1	0	0	1	0	1	0	0	0	0	0
9:	0	1	0	1	1	0	1	0	0	0	0	0	0	1	1	0	0	0	1	0
10:	1	1	0	0	1	0	0	1	1	1	0	1	0	1	0	1	1	1	1	0
11:	0	1	1	0	0	0	0	1	1	0	1	0	0	1	0	1	0	1	0	1
12:	0	0	0	0	1	1	0	0	0	1	0	0	0	0	0	0	1	1	0	1
13:	0	0	0	0	1	0	0	1	1	1	1	1	1	0	1	1	1	0	1	1
14:	1	0	0	0	0	0	1	0	0	1	1	1	0	0	0	1	0	0	0	0
15:	0	0	0	0	0	1	0	0	0	0	0	1	0	1	1	0	1	0	1	1
16:	0	1	0	1	1	1	1	0	1	0	0	0	0	0	0	1	0	0	0	1
17:	0	0	0	1	1	1	1	0	1	1	1	1	0	1	0	1	1	0	0	0
18:	0	0	0	0	1	0	1	0	0	1	1	1	0	1	1	0	1	0	0	0
19:	0	0	1	0	0	1	1	1	1	1	0	1	0	1	0	1	1	1	0	0

A digraph of order 20

Adcacency matrix representation of a graph

0:	0	80	0	0	0	0	0	70	0	47	78	95	0	0	0	0	0	0	34	0	94	24
1:	80	0	28	69	96	22	31	0	78	52	0	0	0	0	0	0	73	0	96	43	70	75
2:	0	28	0	46	58	0	0	0	57	0	0	91	0	13	26	0	0	89	61	0	0	0
3:	0	69	46	0	0	0	0	0	0	33	0	0	0	98	0	0	0	71	0	67	88	98
4:	0	96	58	0	0	0	11	12	0	69	0	0	80	82	0	0	86	0	0	0	0	99
5:	0	22	0	0	0	0	0	0	0	22	0	79	0	0	0	0	0	50	0	36	57	0
6:	0	31	0	0	11	0	0	0	0	98	0	0	17	0	0	50	0	74	0	11	97	0
7:	70	0	0	0	12	0	0	0	0	31	0	0	50	0	43	20	91	0	31	0	0	0
8:	0	78	57	0	0	0	0	0	0	0	0	51	0	63	70	10	86	0	0	0	0	10
9:	47	52	0	33	69	22	98	31	0	0	68	79	0	0	0	91	0	0	40	53	0	0
10:	78	0	0	0	0	0	0	0	0	68	0	0	0	0	56	0	78	0	0	0	0	36
11:	95	0	91	0	0	79	0	0	51	79	0	0	67	0	0	0	77	0	52	88	11	0
12:	0	0	0	0	80	0	17	50	0	0	0	67	0	0	0	97	0	47	0	0	0	0
13:	0	0	13	98	82	0	0	0	63	0	0	0	0	0	0	73	0	0	0	76	94	0
14:	0	0	26	0	0	0	0	43	70	0	56	0	0	0	0	0	77	18	23	0	0	0
15:	0	0	0	0	0	0	50	20	10	91	0	0	97	73	0	0	0	0	0	37	0	0
16:	0	73	0	0	86	0	0	91	86	0	78	77	0	0	77	0	0	0	0	0	0	21
17:	0	0	89	71	0	50	74	0	0	0	0	0	47	0	18	0	0	0	0	0	0	0
18:	34	96	61	0	0	0	0	31	0	40	0	52	0	0	23	0	0	0	0	0	0	0
19:	0	43	0	67	0	36	11	0	0	53	0	88	0	76	0	37	0	0	0	0	0	0
20:	94	70	0	88	0	57	97	0	0	0	0	11	0	94	0	0	0	0	0	0	0	0
21:	24	75	0	98	99	0	0	0	10	0	36	0	0	0	0	0	21	0	0	0	0	0

A weighted graph of order 22

The adjacency list

As the name suggests, it is a list of neighbors of every vertex. The following example should make it clear:

0: 7, 3, 15	*1: 2, 13, 14*
2: 1, 10, 13	*3: 7, 0, 15*
4: 8, 9, 12, 14	*5: 8, 9, 11, 13*
6: 8, 10, 11, 13	*7: 0, 3, 11, 8*
8: 4, 5, 6, 7	*9: 4, 5, 10*
10: 2, 6, 9	*11: 5, 6, 7, 12*
12: 4, 11	*13: 1, 2, 5, 6*
14: 1, 4, 15	*15: 14, 4, 3*

A labeled graph of order 16

Question

Graph Traversals

Question

- *What is the distance from vertex i to vertex j in a graph G?*

Graph Traversals

Question

- *What is the distance from vertex i to vertex j in a graph G?*
- *Is the graph connected?*

Graph Traversals

Question

- *What is the distance from vertex i to vertex j in a graph G?*
- *Is the graph connected?*
- *Does it have a cut-vertex?*

Graph Traversals

Question

- *What is the distance from vertex i to vertex j in a graph G?*
- *Is the graph connected?*
- *Does it have a cut-vertex?*

These are some common questions in many applications.

Graph Traversals

Question

- *What is the distance from vertex i to vertex j in a graph G?*
- *Is the graph connected?*
- *Does it have a cut-vertex?*

These are some common questions in many applications.

Answer

Graph traversals are tools that will help us answer these and many other questions.

There are two fundamental graph traversals:

Graph Traversals

Question

- *What is the distance from vertex i to vertex j in a graph G?*
- *Is the graph connected?*
- *Does it have a cut-vertex?*

These are some common questions in many applications.

Answer

Graph traversals are tools that will help us answer these and many other questions.

There are two fundamental graph traversals:

- *BFS breadth-first-search.*

Graph Traversals

Question

- *What is the distance from vertex i to vertex j in a graph G?*
- *Is the graph connected?*
- *Does it have a cut-vertex?*

These are some common questions in many applications.

Answer

Graph traversals are tools that will help us answer these and many other questions.

There are two fundamental graph traversals:

- *BFS breadth-first-search.*
- *DFS depth-first-search.*

Graph Traversals

Question

- *What is the distance from vertex i to vertex j in a graph G?*
- *Is the graph connected?*
- *Does it have a cut-vertex?*

These are some common questions in many applications.

Answer

Graph traversals are tools that will help us answer these and many other questions.

There are two fundamental graph traversals:

- *BFS breadth-first-search.*
- *DFS depth-first-search.*

Both turn out to be useful in many applications.

BFS of a connected graph G starts with a vertex $v \in V(G)$ and produces a spanning tree T such that a shortest path from $v \to w$ in G is the path from $v \to w$ in T.

(BFS Algorithm)

BFS of a connected graph G starts with a vertex $v \in V(G)$ and produces a spanning tree T such that a shortest path from $v \to w$ in G is the path from $v \to w$ in T.

(BFS Algorithm)

- *Start with an empty tree T.*

BFS

BFS of a connected graph G starts with a vertex $v \in V(G)$ and produces a spanning tree T such that a shortest path from $v \to w$ in G is the path from $v \to w$ in T.

(BFS Algorithm)

- *Start with an empty tree T.*
- *Add v to T.*

BFS of a connected graph G starts with a vertex $v \in V(G)$ and produces a spanning tree T such that a shortest path from $v \to w$ in G is the path from $v \to w$ in T.

(BFS Algorithm)

- *Start with an empty tree T.*
- *Add v to T.*
- *Scan and add all vertices $v_1, \ldots, v_{d_G(v)}$ that are connected by an edge in G to v and all these edges. Delete v from G.*

BFS of a connected graph G starts with a vertex $v \in V(G)$ and produces a spanning tree T such that a shortest path from $v \to w$ in G is the path from $v \to w$ in T.

(BFS Algorithm)

- *Start with an empty tree T.*
- *Add v to T.*
- *Scan and add all vertices $v_1, \ldots, v_{d_G(v)}$ that are connected by an edge in G to v and all these edges. Delete v from G.*
- *For each vertex v_i of T add all neighbors of v_i that are not yet in T and the edges connecting them to v_i and remove v_i from G.*

BFS of a connected graph G starts with a vertex $v \in V(G)$ and produces a spanning tree T such that a shortest path from $v \to w$ in G is the path from $v \to w$ in T.

(BFS Algorithm)

- *Start with an empty tree T.*
- *Add v to T.*
- *Scan and add all vertices $v_1, \ldots, v_{d_G(v)}$ that are connected by an edge in G to v and all these edges. Delete v from G.*
- *For each vertex v_i of T add all neighbors of v_i that are not yet in T and the edges connecting them to v_i and remove v_i from G.*
- *Stop when G is empty.*

Example

Comment

BFS can be used to detect whether G is connected. It can also be applied to Digraphs.

Example

Comment

BFS can be used to detect whether G is connected. It can also be applied to Digraphs.

Recall: a knight's move on a chess board is one square in one direction (horizontal or vertical) followed by two squares in the perpendicular direction.
In the following 8×8 chessboard some of the squares are white and some are black. Two squares are marked by S and F
*Your goal is to find the smallest number of knight moves starting at **S** and ending at **F** . A knight is not allowed to use a black box.*

Example

Comment

BFS can be used to detect whether G is connected. It can also be applied to Digraphs.

Recall: a knight's move on a chess board is one square in one direction (horizontal or vertical) followed by two squares in the perpendicular direction.
In the following 8×8 chessboard some of the squares are white and some are black. Two squares are marked by S and F
Your goal is to find the smallest number of knight moves starting at **S** *and ending at* **F** *. A knight is not allowed to use a black box.*

Question

How would you solve this puzzle?

Example: knigths move on a chessboard.

Remark

The graph will be built conceptually. We can actually build the BFS tree on the chessboard.

Example: knigths move on a chessboard.

Remark

The graph will be built conceptually. We can actually build the BFS tree on the chessboard.

	A	B	C	D	E	F	G	H
1		■	■			S	■	
2	■	■		■	■			
3			■			■	■	
4				■	■	■		
5		■	■				F	■
6			■			■		
7	■	■					■	■
8	■	■						

Bảng: The knights shortest path