Theory of Computation
Chapter 1

Guan-Shieng Huang

Feb. 24, 2003

Slide 1

Text Book

Computational Complexity, C. H. Papadimitriou, Addison-Wesley, 1994.

Slide 2

Scope

- Chapter 1: Problems and algorithms
- Chapter 2: Turing machines
- Chapter 3: Computability
- Chapter 4: Boolean logic
- Chapter 7: Relation between complexity classes
- Chapter 8: Reductions and completeness

Slide 3

- Chapter 9: NP-complete problems
- Chapter 10: coNP and function problems
- Chapter 11: Randomized computation
- Chapter 13: Approximability

Slide 4

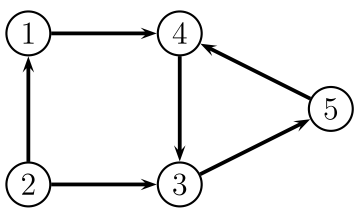
Outline of Chap. 1

- Graph Reachability
- Maximum Flow
- Matching
- Traveling Salesman Problem

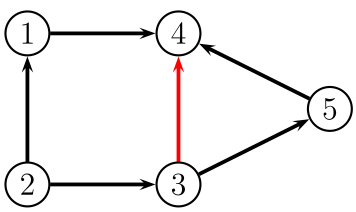
Slide 5

Graph Reachability

Problem 1 *Given a directed graph* $G = (V, E)$*, where* $V = \{1, 2, \ldots, n\}$*, ask whether there is a path from node* 1 *to node* n*.*



$1 \to 4 \to 3 \to 5$



Algorithm

Slide 6

1. Let $S = \{1\}$.
2. If S is empty, go to 5; otherwise, remove one node, say t, in S.
3. For each edge $(t, u) \in E$, if u is not marked, mark u and add u to S.
4. Go to 2.
5. If node n is marked, answer "yes"; otherwise answer "no."

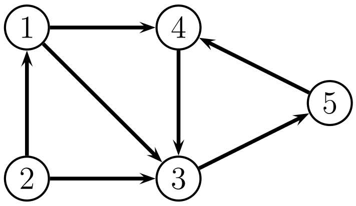
Problem 1.4.2

Slide 7

1. Show by induction on i that, if v is the ith node added by the search algorithm to the set S, then there is a path from node 1 to v.
2. Show by induction on l that if node v is reachable from node 1 via a path with l edges, then the search algorithm will add v to set S.

Slide 8

Example



S	t
$\{1\}$	1
$\{3, 4\}$	3
$\{5, 4\}$	4
$\{5\}$	5
$\{\}$	

Slide 9

Complexity

1. Observe that each node can stay in S at most once.
2. Each edge is used at most once.
3. There are at most n^2 edges.
4. The time complexity is at most n^2.
5. Space complexity is n.

Slide 10

Remark

1. How to implement Step 2 in the algorithm?
   stack $\Rightarrow$ DFS, queue $\Rightarrow$ BFS
2. How to implement Step 3? Random access memory.
3. What is the computational model?
4. Big-O.

Slide 11

Big-O

Let f and g be functions from N to N. We write $f(n) = O(g(n))$ if there are positive integers c and n_0 such that, for all $n \geq n_0$, $f(n) \leq c \cdot g(n)$.

1. $f(n) = O(g(n))$ means intuitively that f grows as g or slower.
2. We write $f(n) = \Omega(g(n))$ if $g(n) = O(f(n))$.
3. We write $f(n) = \Theta(g(n))$ if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.

Slide 12

Examples

1. $n = O(n^2)$
2. $n^{1.5} = O(n^2)$
3. If $p(n)$ is a polynomial of degree d, then $p(n) = \Theta(n^d)$.
4. If $c > 1$ is a positive integer and $p(n)$ any polynomial, then $p(n) = O(c^n)$.
5. $\lg n = O(n)$, or $(\lg n)^k = O(n)$.

Slide 13

Determining Big-O

1. $f(n) = O(g(n))$ if
$$\lim_{n\to\infty} \frac{f(n)}{g(n)} \le c$$
for some constant c.
$f(n) = O(g(n))$ if
$$\limsup_{n\to\infty} \frac{f(n)}{g(n)} \le c$$
for some constant c.
2. If $f(n) = O(g(n))$ and $g(n) = O(h(n))$, then $f(n) = O(h(n))$.

Slide 14

Problem 1.4.10

Let $f(n)$ and $g(n)$ be any two of the following functions . Determine whether (i) $f(n) = O(g(n))$; (ii) $f(n) = \Omega(g(n))$; $f(n) = \Theta(g(n))$: (a) n^2; (b) n^3; (c) $n^2 \log n$; (d) 2^n; (e) n^n; (f) $n^{\log n}$; (g) 2^{2^n}; (h) $2^{2^{n+1}}$; (j) n^2 if n is odd, 2^n otherwise.

It is easy to see that (a) $\prec$ (c) $\prec$ (b) $\prec$ (d) $\prec$ (e).
Also, (f) $\prec$ (e), and (g) $\prec$ (h).

Slide 15

Polyniomial-Time Algorithm

- polynomial time $\Rightarrow$ practical, efficient
  exponential time $\Rightarrow$ impractical, inefficient (<u>intractable</u>)
- n^{80} algorithm v.s. $2^{\frac{n}{100}}$ algorithm
- worst case v.s. average case
  The exponential worst-case performance of an algorithm may due to a statistically insignificant fraction of the input.

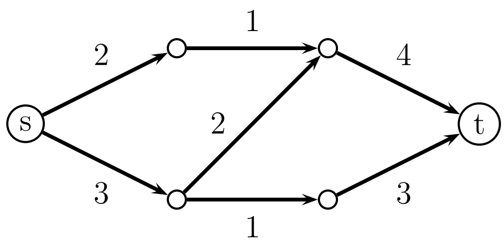
Maximum Flow

A <u>network</u> $N = (V, E, s, t, c)$ is a graph (V, E) with two specified nodes s (the source) and t (the sink) such that

- the source has no incoming edges and the sink has no outgoing edges;
- for each edge (i, j), we are given a capacity $c(i, j)$, a positive integer.

Slide 16

A Network



Slide 17

Slide 18

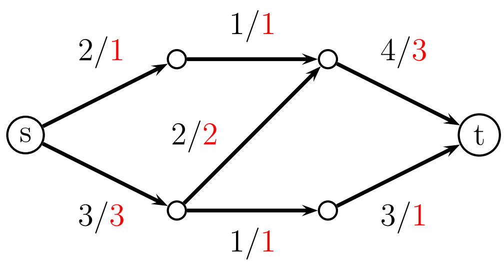
Flow

1. an assignment of a nonnegative integer value $f(i,j) \le c(i,j)$ to each edge (i,j);
2. for each node, other than s and t, the sum of the fs of the incoming edges is equal to the sum of the outgoing edges; (流進=流出)
3. the value of a flow is the sum of the flows in the edges leaving s (or, equivalently, the sum coming to t).

Slide 19

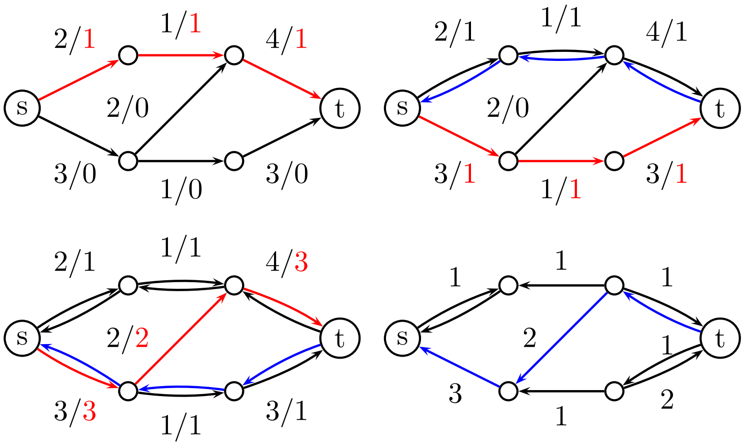
Maximum Flow Problem

Given a network, find a flow of the largest possible value.



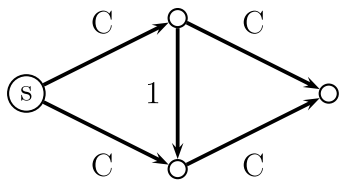
Slide 20

Algorithm



Slide 21

Another Example



Require $O(n^3C)$ time in the worst case!

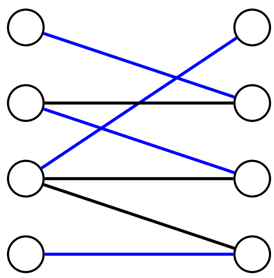
Improvements

The shortest-path heuristic can reduce the time to $O(n^5)$.

Slide 22

Bipartite Matching

A <u>bipartite graph</u> is a triple $B = (U, V, E)$ where $U = \{u_1, \ldots, u_n\}$, $V = \{v_1, \ldots, v_n\}$, and $E \subseteq U \times V$.



Slide 23

Slide 24

Matching

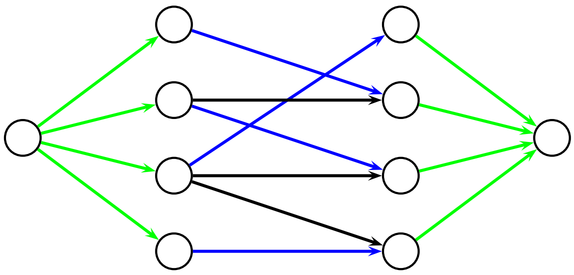
A matching of a bipartite graph B is a set $M \subseteq E$ s.t.

1. $|M| = n$;
2. for any two edges $(u, v), (u', v') \in M$, $u \neq u'$ and $v \neq v'$.

Slide 25

Algorithm

Reduce MATCHING to MAX FLOW.



Traveling Salesman Problem

Given n cities $1, 2, \ldots, n$ and a nonnegative integer distance $d_{i,j}$ between any two cities i and j, find the shortest tour $\sum_{i=1}^{n} d_{\pi(i),\pi(j)}$ where π is a permutation on $\{1, \ldots, n\}$.

Slide 26

Example

$$\begin{pmatrix} 10 & 5 & 1 & 11 \\ 8 & 3 & 4 & 5 \\ 6 & 16 & 4 & 5 \\ 20 & 2 & 8 & 2 \end{pmatrix}$$

Slide 27

Remark

- TSP is NP-hard (its decision version is in fact NP-complete).
- There is no known polynomial-time algorithm for solving TSP.
- If a problem is NP-complete, most computer scientists believe that there is no polynomial-time algorithm for it.

Slide 28

Summary

We have discussed

1. Graph Reachability
2. Big-O notation
3. Maximum Flow
4. Bipartite Matching
5. Traveling Salesman Problem

Slide 29

Slide 30

- MAX FLOW $\Longrightarrow$ REACHABILITY.
- MATCHING $\Longrightarrow$ MAX FLOW.

They are all polynomial-time solvable.

However, we don't know whether there exists a polynomial-time algorithm that can solve TSP.

Slide 31

Reduction

- Reduction is a classical technique, which transforms an unknown problem to an existing one.
- It usually implies to transform a harder problem into an easier one.
- However, in complexity theory, we use it in the perverse way.
- When A reduces to B, we say that B can not be easier that A. (越reduce越難)

Slide 32

Big-O

Big-O captures the asymptotic behavior for comparison of two positive functions. However, why we ignore the constant coefficient?

Slide 33

Problem 1.4.4

(a) A directed graph is *acyclic* if it has no cycles. Show that any acyclic graph has a source (a node with no incoming edges).

(b) Show that a graph with n nodes is acyclic if and only if its nodes can be numbered 1 to n so that all edges go from lower to higher numbers (use the property in (a) above repeatedly).

(c) Describe a polynomial-time algorithm that decides whether a graph is acyclic.

Problem 1.4.5

(a) Show that a graph is bipartite (that is, its nodes can be partitioned into two sets, not necessarily of equal cardinality, with edges going only from one to the other) if and only if it has no odd-length cycles.

(b) Describe a polynomial algorithm for testing whether a graph is bipartite.

Slide 34

Problem 1.4.9

Show that for any polynomial $p(n)$ and any constant $c > 0$ there is an integer n_0 such that, for all $n \geq n_0$, $2^{cn} > p(n)$. Calculate this n_0 when (a) $p(n) = n^2$ and $c = 1$; (b) when $p(n) = 100n^{100}$ and $c = \frac{1}{100}$.

Slide 35