Algorithms (I)

Introduction

Guoqiang Li

School of Software, Shanghai Jiao Tong University

Instructor and Teaching Assistants

- Guoqiang LI
    - Homepage: http://basics.sjtu.edu.cn/∼liguoqiang
    - Course page: http://basics.sjtu.edu.cn/∼liguoqiang/teaching/Galgo17/index.htm
    - Email: li.g@outlook.com
    - Office: Rm. 1212, Building of Software
    - Phone: 3420-4167
- TA:
    - Qizhe YANG: yangqizhe1994 (AT) gmail (DOT) com
    - Yuwei WANG: wangyuwei95 (AT) qq (DOT) com
- Office hour: Tue. 14:00-17:00 @ SEIEE 3-327 & Software Building 3203

Notification

- Students who take this lecture are assumed to have a solid background of algorithms.
- Principle of Algorithms.
- Students are NOT expected to give a presentation in this lecture.

Algorithm Design

- Basic methodologies:
  - Algorithms on Lists, Trees and Graphs
  - Divide and Conquer
    - Master Theorem
  - Recursion
- Advanced topics:
  - Dynamic Programming
  - Greedy Algorithm
  - Linear Programming
  - Approximation Algorithm
  - Randomized Algorithm
  - Computational Geometry
  - …

Algorithm Analysis

- Big-O Notation
- Advanced Methodology:
  - Probability Analysis
  - Amortized Analysis
  - Competition Analysis

Standard Algorithms

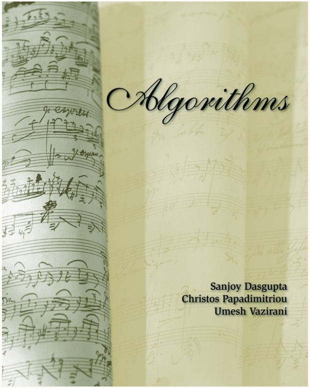
- Sorting
- Searching & Hashing
- Strongly connected components
- Finding shortest paths in graphs
- Minimum spanning trees in graphs
- Matchings in bipartite graphs
- Maximum flows in networks

Data Structure

- Link lists
- Trees, graphs
- Kripke structure, automata
- Priority queue
- Disjoint set

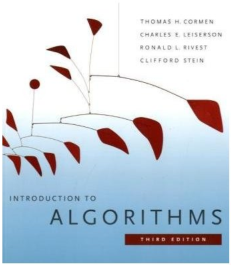
Reference Book

- Algorithms
    - Sanjoy Dasgupta
    - San Diego Christos Papadimitriou
    - Umesh Vazirani
    - McGraw-Hill, 2007.



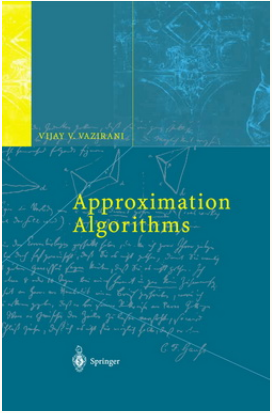
Reference Book

- Introduction to Algorithms
    - Thomas H. Cormen
    - Charles E. Leiserson
    - Ronald L. Rivest
    - Clifford Stein
    - The MIT Press (3rd edition), 2009.



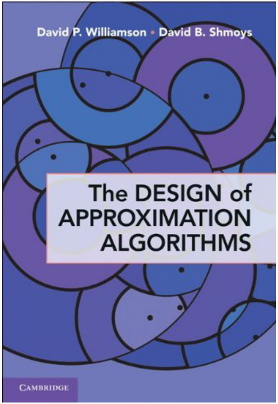
Reference Book

- Approximation Algorithms
  - Vijay V. Vazirani
  - Springer-Verlag, 2004



Reference Book

- The Design of Approximation Algorithms
  - David P. Williamson
  - David B. Shmoys
  - Cambridge University Press, 2011.



Scoring Policy

- 10% Attendance.
- 20% Homework.
  - Four assignments.
  - Each one is 5pts.
  - Work out individually.
  - Each assignment will be evaluated by A, B, C, D, F (Excellent(5), Good(5), Fair(4), Delay(3), Fail(0))
- 70% Final exam.

Any Questions?

Typical Algorithms

Sorting

Sorting

Input: A sequence of n numbers $\langle a_1, a_2, \ldots, a_n \rangle$.
Output: A permutation (reordering) $\langle a'_1, a'_2, \ldots, a'_n \rangle$ of the input sequence such that $a'_1 \leq a'_2 \leq \ldots \leq a'_n$

Various Sorts

- Insert Sort
- Bubble Sort
- Heap Sort
- Quick Sort
- Merge Sort

Merge Sort

The Algorithm

```
MERGESORT(a[1...n])
An array of numbers a[1...n];
if n > 1 then
    return(MERGE(MERGESORT(a[1...⌊n/2⌋]),
    MERGESORT(a[⌊n/2⌋...,n])));
    else return(a);
end
```

```
MERGE(x[1...k], y[1...l])
if k = 0 then return y[1...l];
if l = 0 then return x[1...k];
if x[1] ≤ y[1] then
    return( x[1]∘MERGE(x[2...k], y[1...l]));
    else return( y[1]∘MERGE(x[1...k], y[2...l]));
end
```

An Iterative Version

```
ITERTIVE-MERGESORT(a[1...n])
An array of numbers a[1...n];
Q = [  ]empty queue;
for i = 1 to n do
|   Inject(Q, [a]);
end
while |Q| > 1 do
|   Inject(Q,MERGE(Eject(Q),Eject(Q)));
end
return(Eject(Q));
```

The Time Analysis

- The recurrence relation:

$$T(n) = 2T(n/2) + O(n)$$

- By Master Theorem:

$$T(n) = O(n \log n)$$

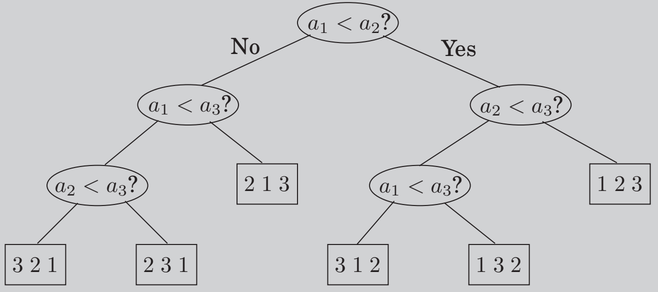
Master Theorem

If $T(n) = aT(\lceil n/b \rceil) + O(n^d)$ for some constants $a > 0$, $b > 1$ and $d \geq 0$, then

$$T(n) = \begin{cases} O(n^d) & \text{if } d > \log_b a \\ O(n^d \log n) & \text{if } d = \log_b a \\ O(n^{\log_b a}) & \text{if } d < \log_b a \end{cases}$$

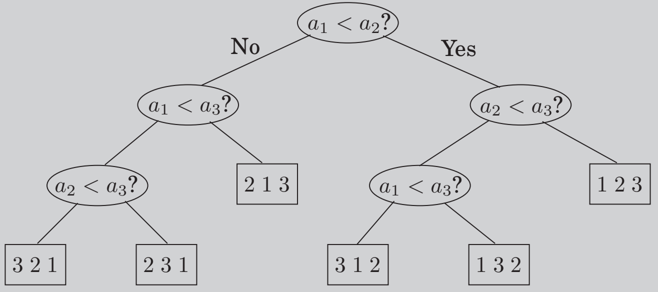
Can we do better?

Sorting



- A sorting algorithm can be depicted as a decision tree.
- The depth of the tree - the number of comparisons on the longest path from root to leaf, is the worst-case time complexity of the algorithm.
- Assume n elements. Each of its leaves is labeled by a permutation of $\{1, 2, \ldots, n\}$.

Sorting



- Every permutation must appear as the label of a leaf.
- This is a binary tree with $n!$ leaves.
- So ,the depth of the tree - and the complexity of the algorithm - must be at least

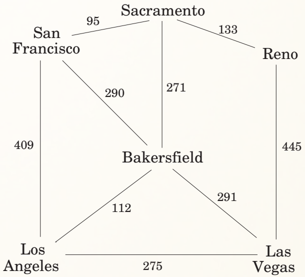
$$\log(n!) \approx \log(\sqrt{\pi(2n+1/3)} \cdot n^n \cdot e^{-n}) = \Omega(n \log n)$$

Shortest Path in Graph

Lengths on Edges

- BFS treats all edges as having the same length.
- It is rarely true in applications where shortest paths are to be found.
- Every edge $e \in E$ with a length l_e.
- If $e = (u, v)$, we will sometimes also write

$$l(u, v) \quad \text{or} \quad l_{uv}$$



Dijkstra's Algorithm

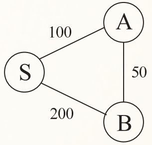
An Adaption of Breadth-First Search

- BFS finds shortest paths in any graph whose edges have unit length.
- Q: Can we adapt it to a more general graph $G = (V, E)$ whose edge lengths l_e are positive integers?
- A simple trick: For any edge $e = (u, v)$ of E, replace it by l_e edges of length 1, by adding $l_e - 1$ dummy nodes between u and v. It might take time
$$O(|V| + \sum_{e \in E} l_e)$$
- It is bad in case we have edges with high length.

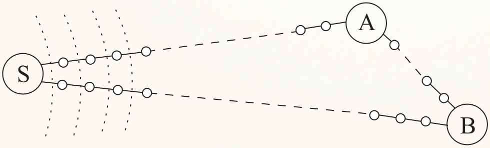
Alarm Clocks

- Set an alarm clock for node s at time 0.
- Repeat until there are no more alarms:
- Say the next alarm goes off at time T, for node u. Then:
  - The distance from s to u is T.
  - For each neighbor v of u in G:
    - If there is no alarm yet for v, set one for time $T + l(u, v)$.
    - If v's alarm is set for later than $T + l(u, v)$, then reset it to this earlier time.

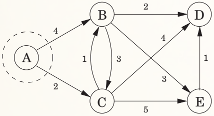
G:



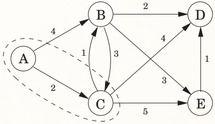
G':



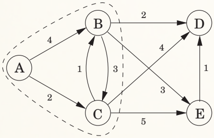
An Example



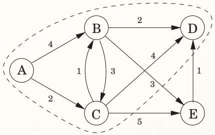
A: 0	D: ∞
B: 4	E: ∞
C: 2	



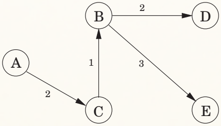
A: 0	D: 6
B: 3	E: 7
C: 2	



A: 0	D: 5
B: 3	E: 6
C: 2	



A: 0	D: 5
B: 3	E: 6
C: 2	



Priority Queue

- Priority queue is a data structure usually implemented by heap.
  - Insert: Add a new element to the set.
  - Decrease-key: Accommodate the decrease in key value of a particular element.
  - Delete-min: Return the element with the smallest key, and remove it from the set.
  - Make-queue: Build a priority queue out of the given elements, with the given key values. (In many implementations, this is significantly faster than inserting the elements one by one.)
- The first two let us set alarms, and the third tells us which alarm is next to go off.

Dijkstra's Shortest-Path Algorithm

```
DIJKSTRA(G, l, s)
input  : Graph G = (V, E), directed or undirected; positive edge length {l_e | e ∈ E};
         Vertex s ∈ V
output: For all vertices u reachable from s, dist(u) is the set to the distance from s to u

for all u ∈ V do
    dist(u) = ∞;
    prev(u) = nil;
end
dist(s) = 0;
H =makequeue(V) \\ using dist-values as keys;
while H is not empty do
    u=deletemin(H);
    for all edge (u, v) ∈ E do
        if dist(v) > dist(u) + l(u, v) then
            dist(v) = dist(u) + l(u, v);   prev(v) = u;
            decreasekey(H,v);
        end
    end
end
```

Running Time

- Since `makequeue` takes at most as long as $|V|$ insert operations, we get a total of $|V|$ `deletemin` and $|V| + |E|$ `insert/decreasekey` operations.
- The time needed for these varies by implementation; for instance, a binary heap gives an overall running time of

$$O((|V| + |E|) \log |V|)$$

Which Heap is Best

Implementation	deletemin	insert/decreasekey	$\|V\| \times \texttt{deletemin} + (\|V\| + \|E\|) \times \texttt{ insert}$
Array	$O(\|V\|)$	$O(1)$	$O(\|V\|^2)$
Binary heap	$O(\log \|V\|)$	$O(\log \|V\|)$	$O((\|V\| + \|E\|) \log \|V\|)$
d-ary heap	$O(\frac{d \log \|V\|}{\log d})$	$O(\frac{\log \|V\|}{\log d})$	$O(\frac{(d\|V\|+\|E\|) \log \|V\|}{\log d})$
Fibonacci heap	$O(\log \|V\|)$	$O(1)$ (amortized)	$O(\|V\| \log \|V\| + \|E\|)$

Which heap is Best

- A naive array implementation gives a respectable time complexity of $O(|V|^2)$, whereas with a binary heap we get $O((|V| + |E|) \log |V|)$. Which is preferable?
- This depends on whether the graph is sparse or dense.
  - $|E|$ is less than $|V|^2$. If it is $\Omega(|V|^2)$, then clearly the array implementation is the faster.
  - On the other hand, the binary heap becomes preferable as soon as $|E|$ dips below $|V|^2 / \log |V|$.
  - The d-ary heap is a generalization of the binary heap and leads to a running time that is a function of d. The optimal choice is $d \approx |E|/|V|$;

Edit Distance

The problem

- When a spell checker encounters a possible misspelling, it looks in its dictionary for other words that are close by.
    - Q: What is the appropriate notion of closeness in this case?
- A natural measure of the distance between two strings is the extent to which they can be aligned, or matched up.
- Technically, an alignment is simply a way of writing the strings one above the other.

S	—	N	O	W	Y		—	S	N	O	W	—	Y
S	U	N	N	—	Y		S	U	N	—	—	N	Y

Cost: 3   Cost: 5

The problem

- The cost of an alignment is the number of columns in which the letters differ.
- And the edit distance between two strings is the cost of their best possible alignment.
- Edit distance is so named because it can also be thought of as the minimum number of edits
    - insertions, deletions, and substitutions of characters needed to transform the first string into the second.

```
S — N O W Y        — S N O W — Y
S U N N — Y        S U N — — N Y
   Cost: 3              Cost: 5
```

A Dynamic Programming Solution

- When solving a problem by dynamic programming, the most crucial question is, What are the subproblems?
- Our goal is to find the edit distance between two strings
$$x[1,\ldots,m] \text{ and } y[1,\ldots,n] \qquad .$$
- For every i,j with $1 \leq i \leq m$ and $1 \leq j \leq n$, let
    - $E(i,j)$: the edit distance between some prefix of the first string, $x[1,\ldots,i]$, and some prefix of the second, $y[1,\ldots,j]$.
- $E(i,j) =$
$\min\{1+E(i-1,j), 1+E(i,j-1), \mathtt{diff}(i,j)+E(i-1,j-1)\}$, where $\mathtt{diff}(i,j)$ is defined to be 0 if $x[i]=y[j]$ and 1 otherwise.

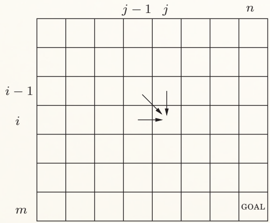
An Example

Edit distance between EXPONENTIAL and POLYNOMIAL, subproblem $E(4,3)$ corresponds to the prefixes EXPO and POL. The rightmost column of their best alignment must be one of the following:

$$\begin{array}{c} \texttt{O} \\ \texttt{-} \end{array} \quad \text{or} \quad \begin{array}{c} \texttt{-} \\ \texttt{L} \end{array} \quad \text{or} \quad \begin{array}{c} \texttt{O} \\ \texttt{L} \end{array}$$

Thus, $E(4,3) = \min\{1 + E(3,3), 1 + E(4,2); 1 + E(3,2)\}$.

(a)



(b)

		P	O	L	Y	N	O	M	I	A	L
	0	1	2	3	4	5	6	7	8	9	10
E	1	1	2	3	4	5	6	7	8	9	10
X	2	2	2	3	4	5	6	7	8	9	10
P	3	2	3	3	4	5	6	7	8	9	10
O	4	3	2	3	4	5	5	6	7	8	9
N	5	4	3	3	4	4	5	6	7	8	9
E	6	5	4	4	4	5	5	6	7	8	9
N	7	6	5	5	5	4	5	6	7	8	9
T	8	7	6	6	6	5	5	6	7	8	9
I	9	8	7	7	7	6	6	6	6	7	8
A	10	9	8	8	8	7	7	7	7	6	7
L	11	10	9	8	9	8	8	8	8	7	6

The Algorithm

for $i = 0$ *to* m **do**
    $E(i, 0) = i;$
end
for $j = 1$ *to* n **do**
    $E(0, j) = j;$
end
for $i = 1$ *to* m **do**
    for $j = 1$ *to* m **do**
        $E(i, j) =$
        $\min\{1 + E(i-1, j), 1 + E(i, j-1), \texttt{diff}(i, j) + E(i-1, j-1)\};$
    end
end
`return`$(E(m, n))$`;`

The over running time is $O(m \cdot n)$.

Then, What is An Algorithm?

What Is An Algorithm

An algorithm is a procedure that consists of

- a finite set of instructions which,
- given an input from some set of possible inputs,
- enables us to obtain an output through a systematic execution of the instructions
- that terminates in a finite number of steps.

Not Enough, Maybe

What Is An Algorithm

- In these problems we are searching for a solution (path, tree, matching, etc.) from among an exponential population of possibilities.
- All these problems could in principle be solved in exponential time by checking through all candidate solutions, one by one.
- The quest for algorithms is about finding clever ways to bypass this process of exhaustive search, using clues from the input in order to dramatically narrow down the search space.

Is That ALL?

Lecture Agenda

- NP Problem
- Coping with NP Completeness
- Linear Programming
- Approximation Algorithms

Referred Materials

- [DPV07] *Algorithms*
- [CLRS09] *Introduction to Algorithms*
- [Vaz04] *Approximation Algorithms*
- [WS11] *The Design of Approximation Algorithms*

- Content of this lecture comes from section 2.3, 4.4, 4.5 and 6.3 in [DPV07].
- Suggest to read Chapter 15 of [CLRS09] and Chapter 6 in [DPV07].