Contents

I Introduction

LLM 属于算法.

1. Computing

计算是什么? 通过图灵机定义.

2. Algorithm

Definition I.1. *An algorithm A is a procedure that takes an input I and produces an output O. (Must terminate)*

$$A(I) = O$$

A procedure can go on forever. 无法确认一个程序是否停机 (停机问题).

2.1 Russell's Paradox

罗素悖论 (自指)

A town has only one male barber. A man is shaved by the barber if he does not shave himself. Dose the barber shave himself?

Let $S(x)$ = set of people shaved by x, so $S(barber) = \{x|x \notin S(x)\}$.
If $barber \notin S(barber)$, then $barber \in S(barber)$.
If $barber \in S(barber)$, then $barber \notin S(barber)$.

Definition I.2. *A procedure P is an algorithm if P takes any input (binary encoding) always stops and output "yes" or "no".*

然后就是形式化的罗素悖论: Define an "algorithm" P_k

- Input: Any algorithm P (binary encoding)
- Output:
  No if $P(P)$ = Yes
  Yes if $P(P)$ = No

P_k is barber, P are other.

3. Undecidable Problems

不可判定问题. 就是一个判定问题, 被证明没有算法可以判定它.

3.1 Post Corresponding Problem

PCP 问题.
给一些多米诺.
目标: 寻找有限的多米诺, 让上下的字符串一致.

Example I.1. Give three tiles(dominoes)

Type A	Type B	Type C
1	10111	10
111	10	0

One solution is BAAC

- 10111.1.1.10
- 10.111.111.0

3.2 *Other Undecidable Problem*

- Wang tiles
- Hilbert's tenth problem

4. Algorithm Evaluation

目标:

- 高质量低花销
- 最坏情况分析 (上界)
- 平均情况分析 (给输入更多的约束)

4.1 *Running Time*

Definition I.3. *Running time:*

- *the number of "primitive/key/basic" operations or "steps".*
- *function of the input size*

Input size: number of itmes/bits

关键操作是指无法通过技巧消除的操作. e.g. 线性搜索算法中, 确认是否找到的比较操作就无法去除, 是关键操作.

4.2 *Example Sort*

构建 decision tree, 建模任意的 sort 算法.

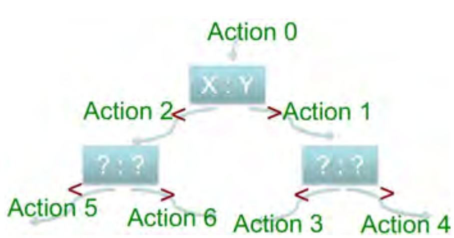


Figure I.1: decision tree

两次比较只能区分出四个排列 (permutation). 所以排序 n 个元素, 有

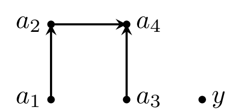
- $n!$ 个排列
- 下界, decision tree 的高度 $\lceil \log_2 n! \rceil$.

Table I.1: optimal sort

n	2	3	4	5	6	7	8	9
$n!$	2	6	24	120	720	5040	40320	362880
$\log_2(n!)$	1	2.58	4.58	6.91	9.49	12.30	15.30	18.47
$\lceil \log_2 n! \rceil$	1	3	5	7	10	13	16	19

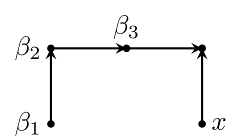
在 7 次比较之内排序 5 个元素 并不简单, 因为平常的比较式需要 8 次. 然后就是一个论文专门讲了这个算法. 第二步比较的应该是 a_4

1) compare $\max(a_1, a_2)$ and $\max(a_3, a_4)$

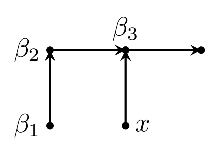


2) merge y into the chain

compare with a_2 then a_1



or compare with a_2 then a_4



3) merge x into the chain, compare with β_2 then β_1 or β_3

4.3 *Merge Insertion Sort*

主要思想: 将元素合并到有序链中.

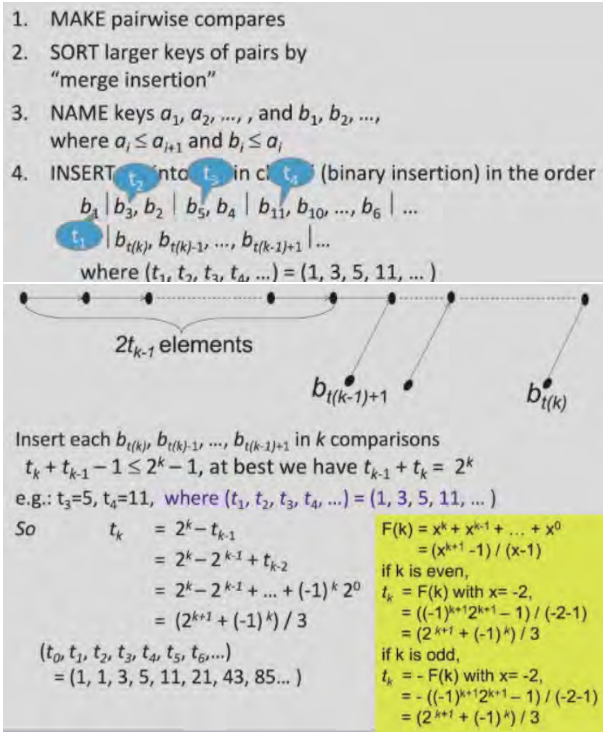


Figure I.2: Merge Insertion Sort

Time Complexity of Merge Insert

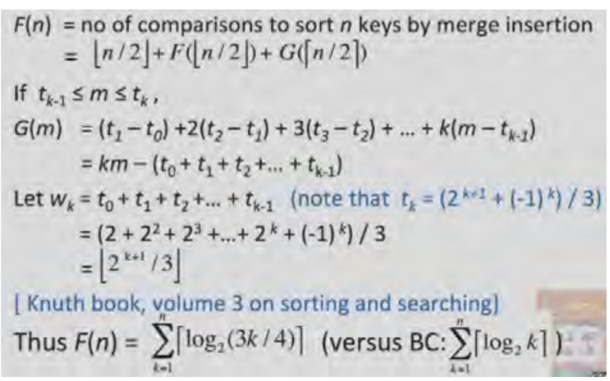


Figure I.3: Time Complexity

II Lower Bound

下界和问题有关. 算法复杂度不可能小于下界. 若复杂度达到下界, 算法就是最优的. 但即使算法是最优的, 也可能复杂度大于下界.

Techniques:

- decision tree
- Adversary(Oracle)

1. Adversary Argument(Oracle)

Definition II.1 (Adversary). *A strategy to create situation to make the algorithm to work hard (releasing the least information and changing scenario).*

为问题建立最坏的情况.

2. Decision Tree

2.1 硬币称重

以硬币称重问题为例, 就是天平可以给左倾, 右倾, 平衡三种选项.

- 给 8 个硬币, 找到一个轻的, 最少需要 $\lceil \log_3 8 \rceil = 2$

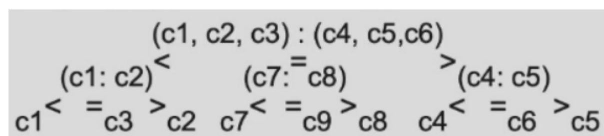


Figure II.1: 8 个硬币

- 给 12 个硬币, 找到一个重量不一样的, 最少需要 $\lceil \log_3 2 * 12 \rceil = 3$

  确保每次称重, 都让可能的结果减少到 1/3

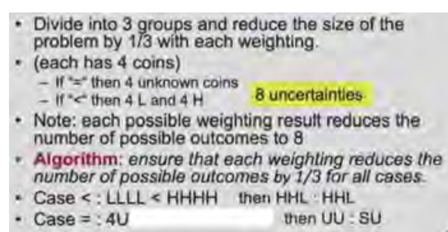


Figure II.2: 12 个硬币

S 表示 standard

2.2 Finding an element in a sorted list

给定有序序列, 找特定元素. 可以使用比较 >=<, 可能找不到. 下界: $\lceil \log_2(n+1) \rceil$.

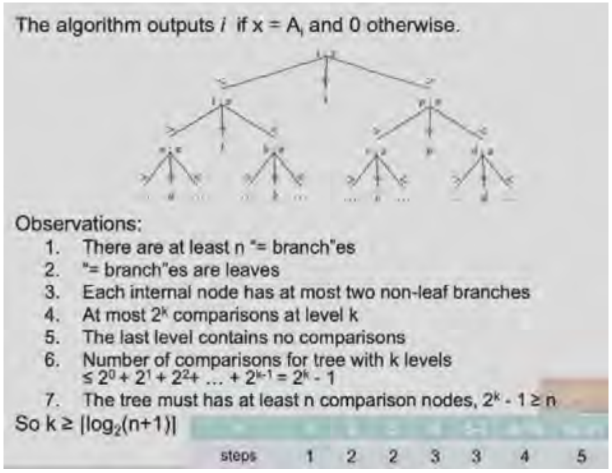


Figure II.3: 下界推导

3. Finding the Maximum and Minimum of N-Elements

3.1 Divide and Conquer Approach

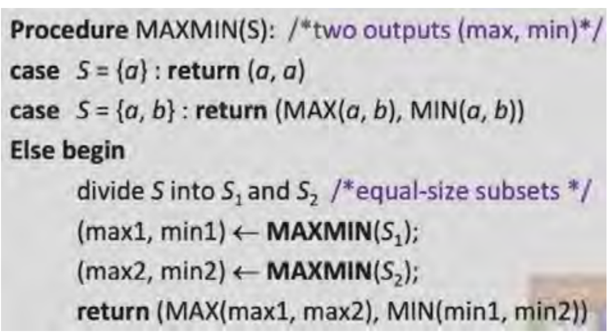


Figure II.4: Divide and Conquer Approach

Complexity 用公式推导

$$
\begin{aligned}
T(n) &= 2T(n/2) + 2 \\
&= 3n/2 - 2
\end{aligned}
$$

where $|S| = n = 2^m$

Proof by Adversary. (给予最少的信息/最多的比较)

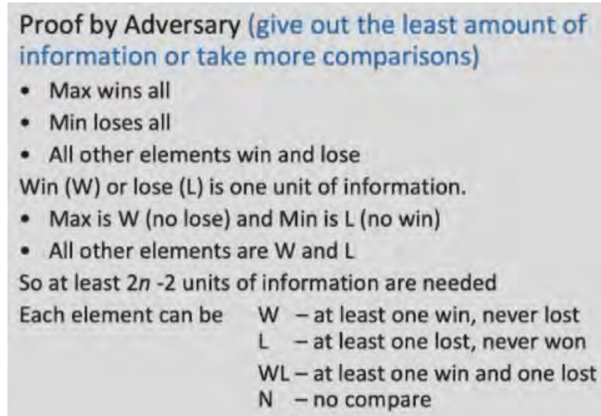
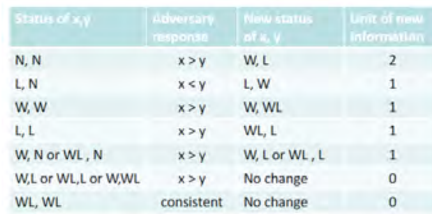


Figure II.5: Proof by Adversary



Status of x,y	Adversary response	New status of x, y	Unit of new information
N, N	x > y	W, L	2
L, N	x < y	L, W	1
W, W	x > y	W, WL	1
L, L	x > y	WL, L	1
W, N or WL , N	x > y	W, L or WL , L	1
W,L or WL,L or W,WL	x > y	No change	0
WL, WL	consistent	No change	0

Figure II.6: Adversary strategy

最后一行是随机给结果, 不给信息.

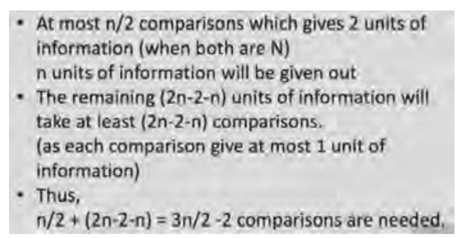


Figure II.7: Analysis

Q.E.D.

example: 略了. 就是可以依据策略改结果.

4. Finding Second Maximum Element

必须找到最大值才能找到次大值.

4.1 Adversary

最大值一直 win, 让其难以找到次大值.

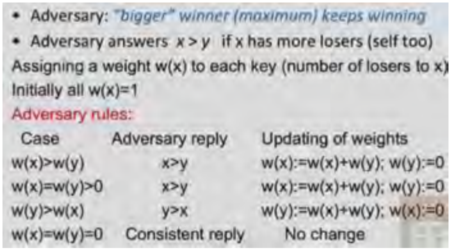


Figure II.8: Adversary strategy

4.2 *Lower Bound*

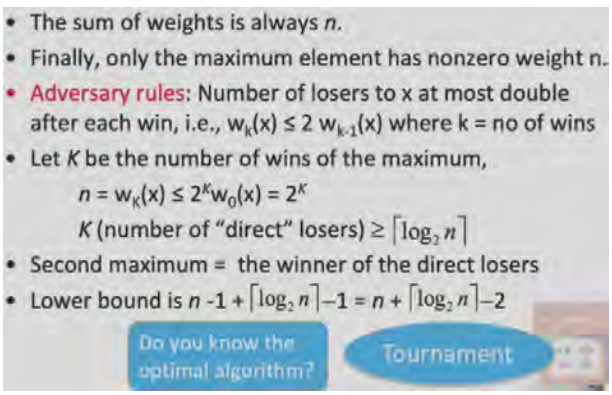


Figure II.9: Lower Bound

5. Merging Sorted Lists

Problem: Merge two sorted lists

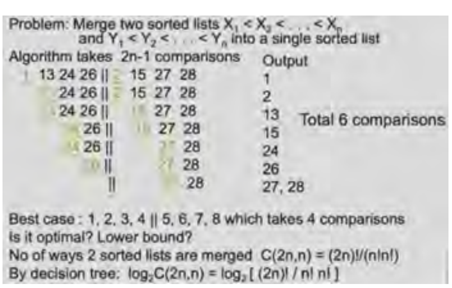


Figure II.10: Merging Sorted Lists

5.1 *Adversary proof*

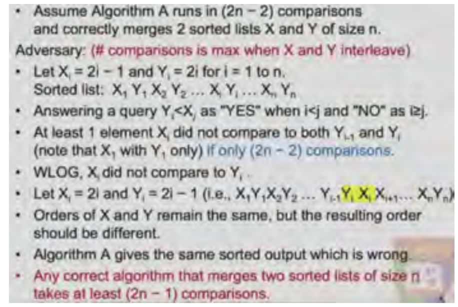


Figure II.11: Adversary proof

III Selection

1. Lower Bound for Finding Median

Trival lower bound: $n-1$

but there are crucial comparison and non-crucial comparison.

Adversary: give non-crucial comparison

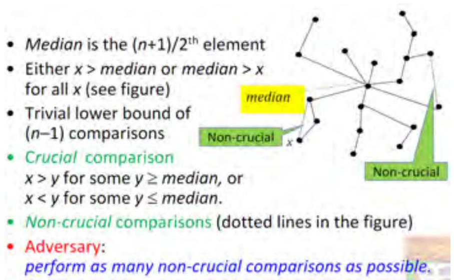


Figure III.1: Finding Median

1.1 Adversary

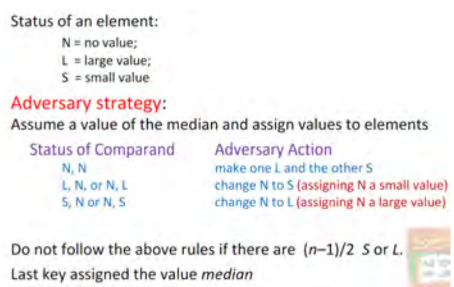


Figure III.2: Adversary

lower bound $=$ crucial $+$ non-crucial $=3(n-1)/2$

Example III.1. Median of 5 elements

- sorting: 7
- lower bound: 6

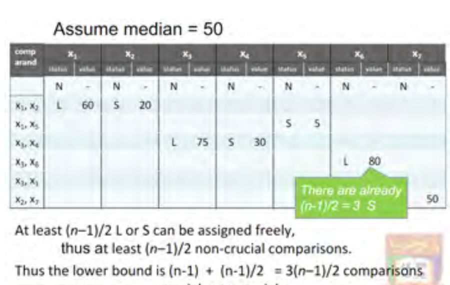


Figure III.3: Example for Median Finding

1.2 Median Algorithm

$O(n\log n)$

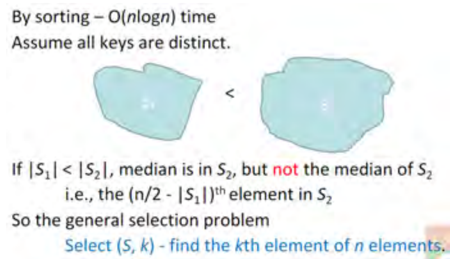


Figure III.4: Median Algorithm

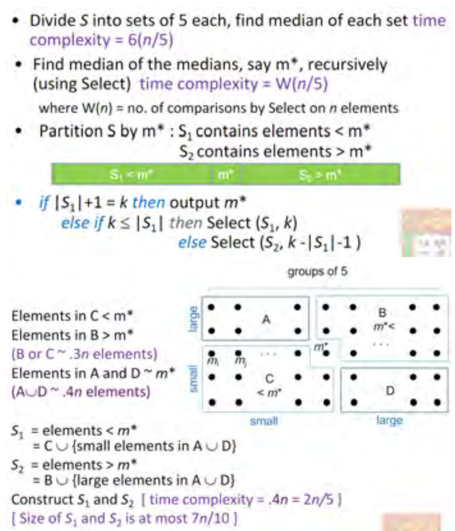


Figure III.5: Select(S,k)

1.3 Time Analysis

归纳法证明, 线性算法

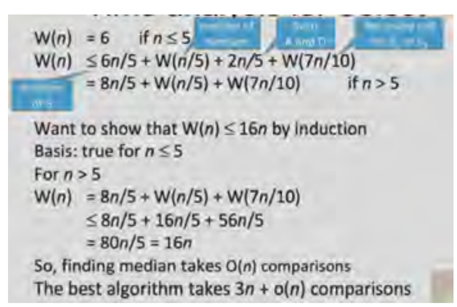


Figure III.6: Time Analysis

2. Finding the kth Largest/Smallest Elements

k-selection problem

sorting : $O(n\log n)$

use Max heap

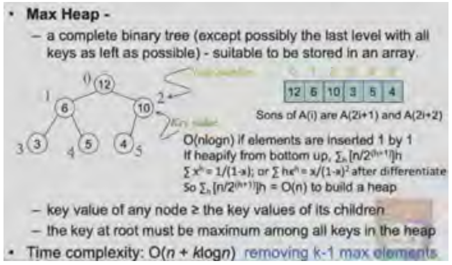


Figure III.7: Max heap

And we can also find max use mim heap

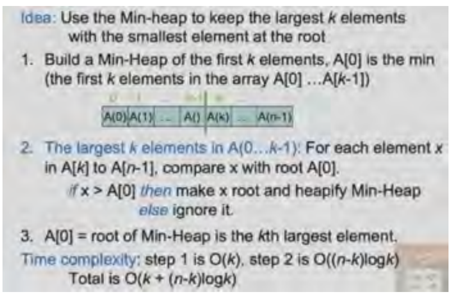


Figure III.8: mim heap

2.1 Lower Bound(Sketch)

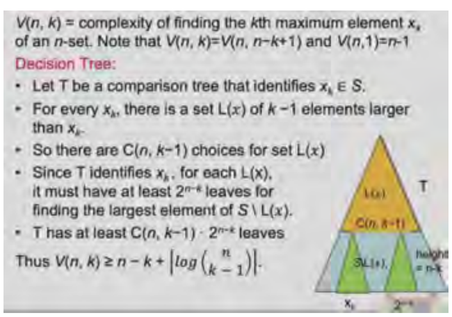


Figure III.9: Lower Bound

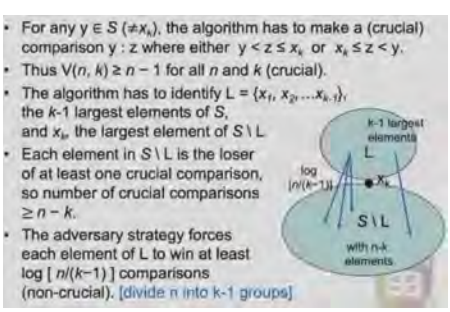


Figure III.10: Adversary selection

Theorem III.1.

$$V(n,k) \geq n - k + (k-1)\left\lceil \log \frac{n}{k-1} \right\rceil$$

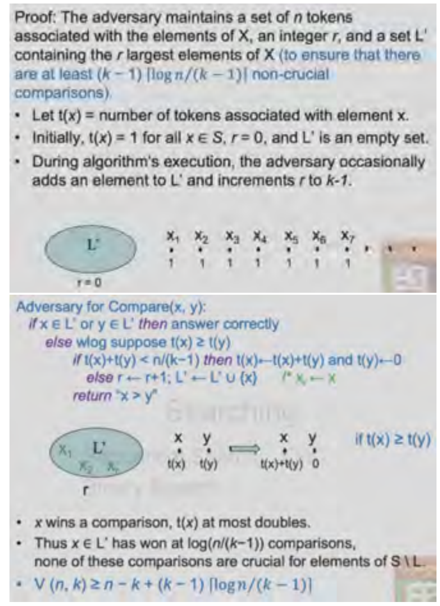


Figure III.11: theorem proof

IV Searching

1. Unbound Searching

give a sorted list and the function

$$F(i) = \begin{cases} 0 & \text{if } 0 < i < n \\ 1 & \text{if } i \geq n \end{cases}$$

Only we can do is to query $F(k)$. How to find n?

可以使用二分, 但我们不知道最大的边界在哪.

Let $S_A(n) = m$ if algorithm A uses m queries to find n.

1.1 *Preliminary Algorithms*

- Unary search B_0
- Binary search B_1

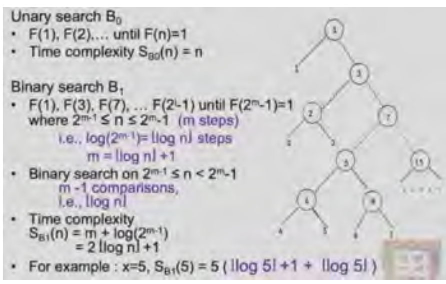


Figure IV.1: Preliminary Algorithms

1.2 *Higher Binary Search*

- Double binary search B_2
- Triple binary search B_3
- k binary search B_k

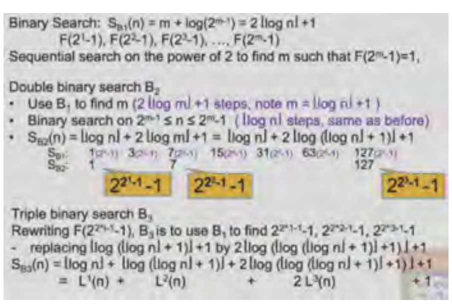


Figure IV.2: Higher Binary Search

1.3 *Ultimate Algorithm*

determine k for B_k to find n.

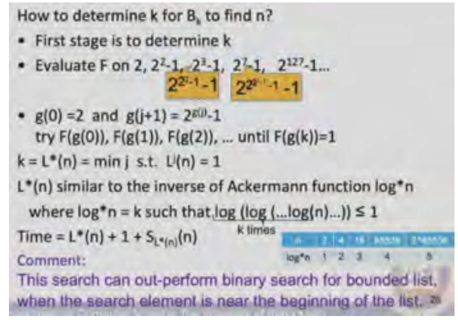


Figure IV.3: Ultimate Algorithm

1.4 *Sketch of Lower Bound*

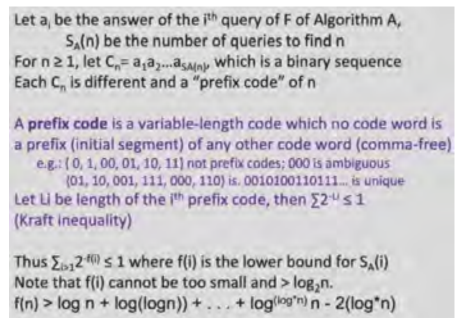


Figure IV.4: Sketch of Lower Bound

2. Find X in a Sotred Matrix

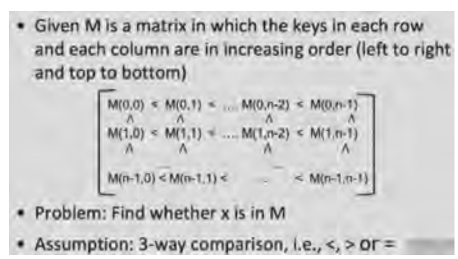


Figure IV.5: Find X in a Sotred Matrix

2.1 *Solution*

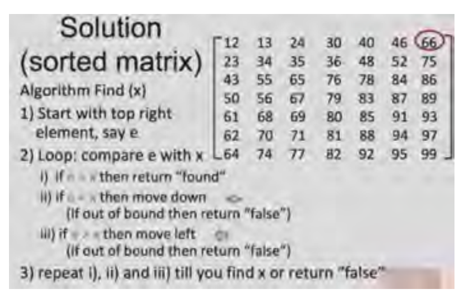


Figure IV.6: Solution

Complexity: $2n-1$ comparisons

2.2 Lower Bound

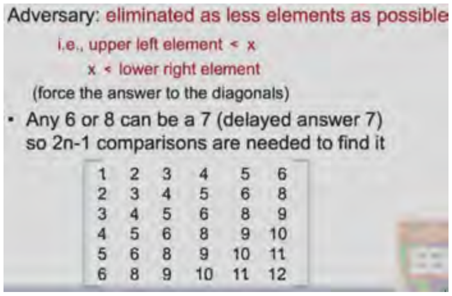


Figure IV.7: Adversary

V Reduction

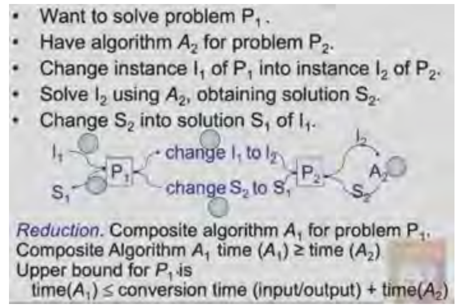


Figure V.1: Problem reduction P_1 to P_2.($P_1 \propto P_2$)

1. Lower Bound by Reduction

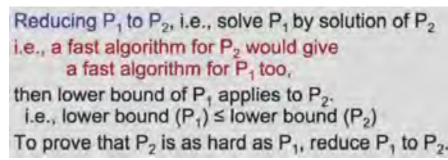


Figure V.2: Lower Bound by Reduction

2. Distinct Integers Sorting

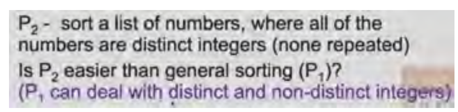


Figure V.3: Distinct Integers Sorting

Theorem V.1. *Distinct integer sorting is not easier*

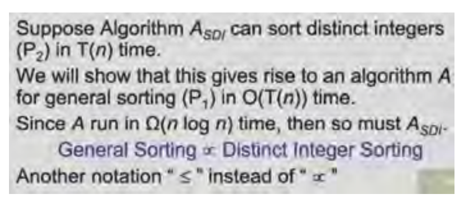


Figure V.4: proof

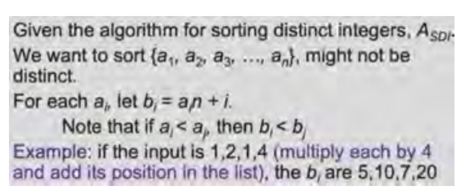


Figure V.5: How to build A from A_{SDI}

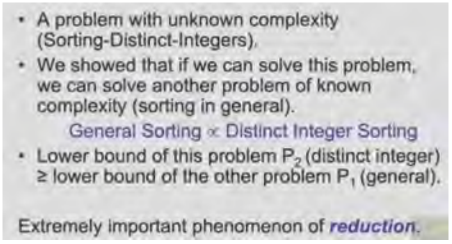


Figure V.6: phenomenon reduction

3. Proof of NP-completeness

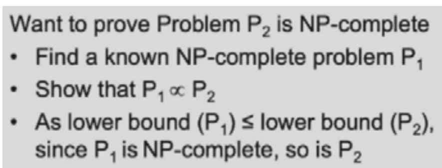


Figure V.7: Proof of NP-completeness

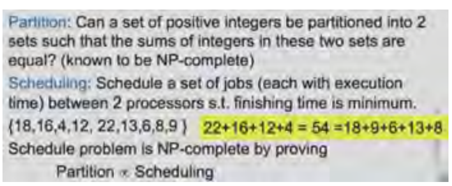


Figure V.8: Example

4. Integer Distinctness Problem

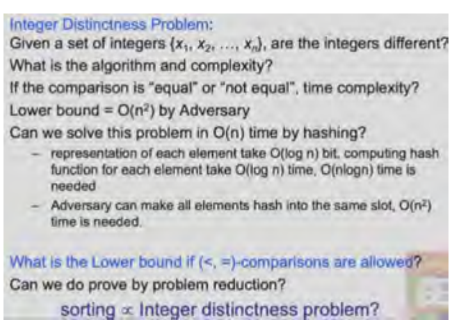


Figure V.9: Integer Distinctness Problem

计算 hash 也有复杂度, 一般是个大于 $\log n$ 的常数.

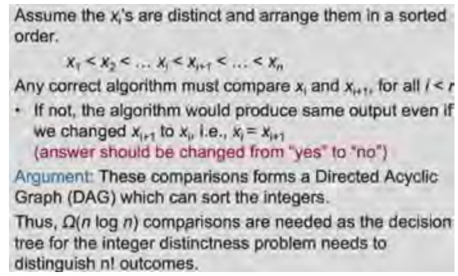


Figure V.10: Integer Distinctness Problem

4.1 Closest Pair Problem

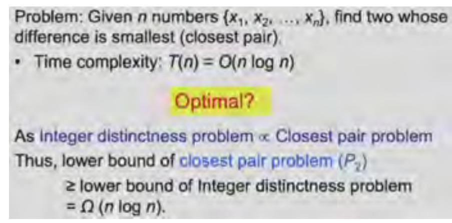


Figure V.11: Closest Pair Problem

5. Data Compression

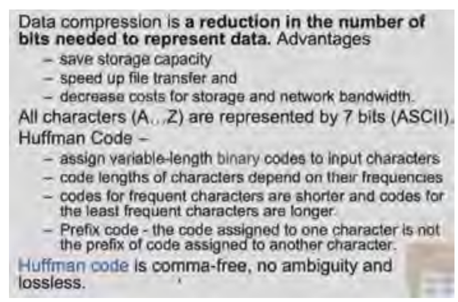


Figure V.12: Data Compression

5.1 Huffman code

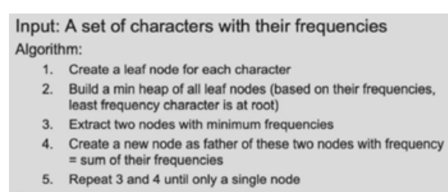


Figure V.13: Huffman code

Complexity: $O(n \log n)$

5.2 Lower Bound for Huffman

convert x_i with $y_i = 2^{x_i}$

6. 3SUM Problem

$a + b + c = 0$?
example and sketch proof
目前最佳的算法, 但下界没有被证明

6.1 Collinearity

3SUM $\propto$ General Collinearity
Another proof

6.2 Segment Splitting Problem

3SUM $\propto$ Collinearity $\propto$ Segment Splitting
gadgets

6.3 Motion Planning

3SUM $\propto$ Collinearity $\propto$ Segment Splitting $\propto$ Motion Planning

6.4 M-3SUM and 3SUM are Equivalent

VI String Matching

1. Brute Force

2. Knuth-Morris-Pratt(KMP) Algorithm

3. The Boyer-Moore(BM) Algorithm

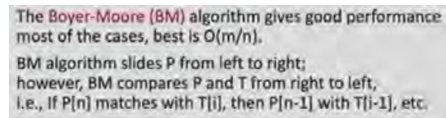


Figure VI.1: BM Algorithm

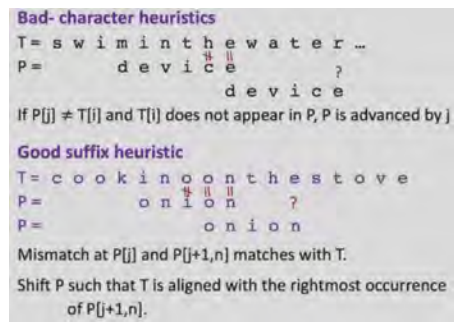


Figure VI.2: Example

3.1 Bad Character Rule

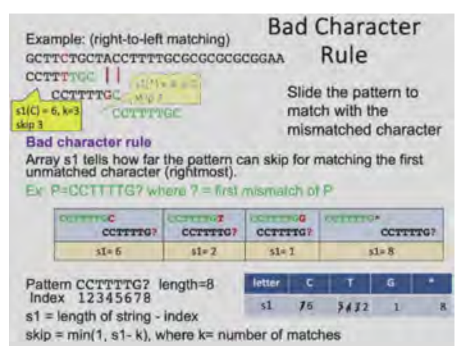


Figure VI.3: Bad Character Rule

3.2 Good Suffix Rules

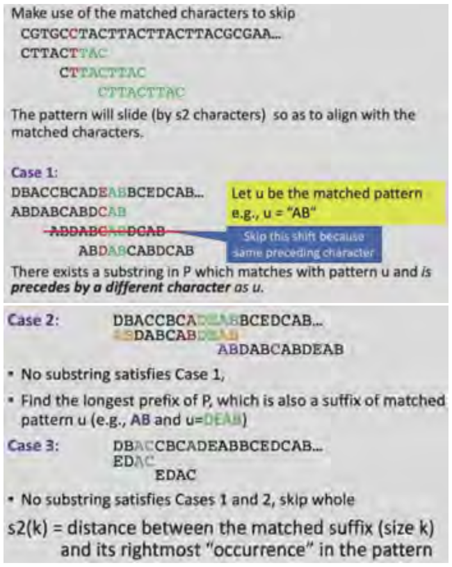


Figure VI.4: Good Suffix Rules

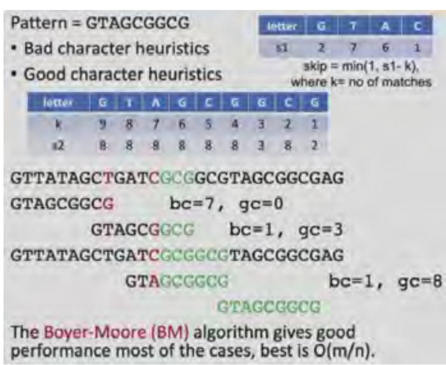


Figure VI.5: Example

4. Tire

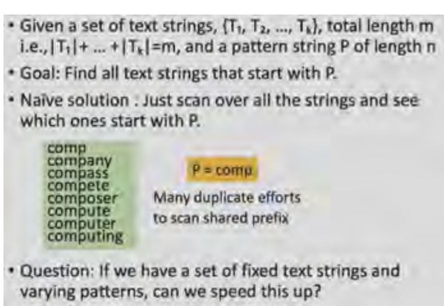


Figure VI.6: Autocomplete Problem

4.1 reTRIEval

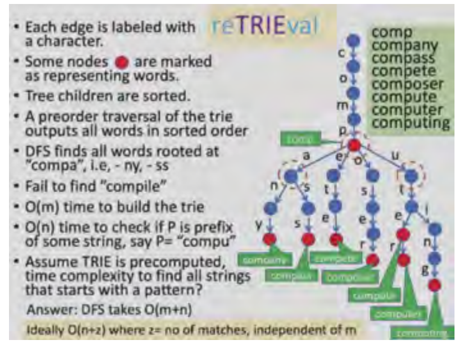


Figure VI.7: reTRIEval

4.2 Find all strings start with a pattern

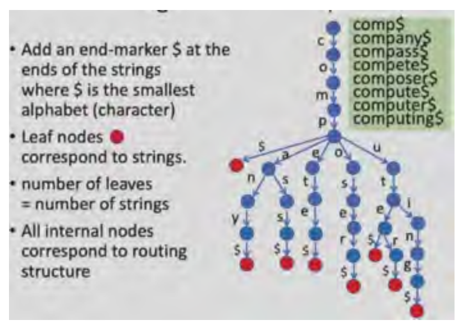


Figure VI.8: Find all strings start with a pattern

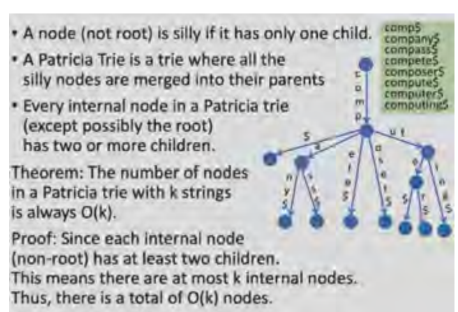


Figure VI.9: Patricia Tire

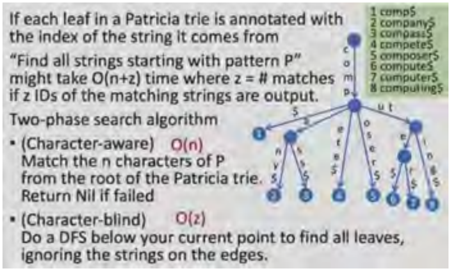


Figure VI.10: Indexed Patricia Tire

5. Suffix Tree

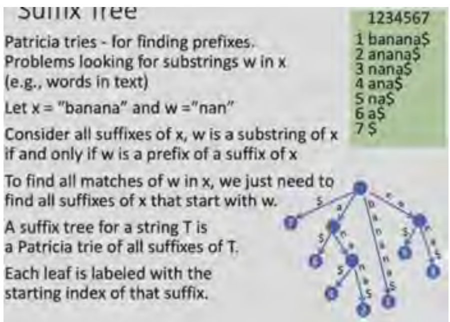


Figure VI.11: Suffix Tree

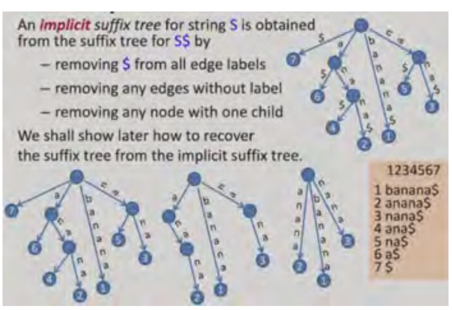


Figure VI.12: Implicit Suffix Tree

5.1 Ukkonen's Algorithm

Building Suffix Tree in Linear Time

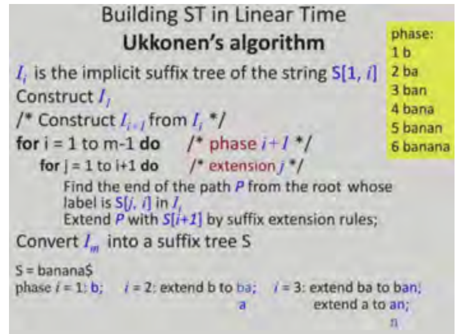


Figure VI.13: Ukkonen's Algorithm

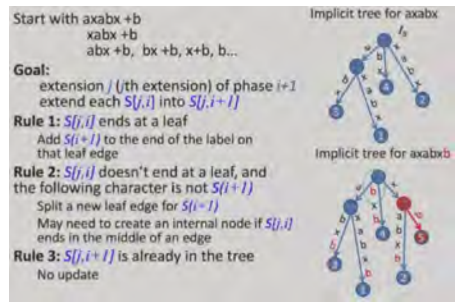


Figure VI.14: Extension Rules

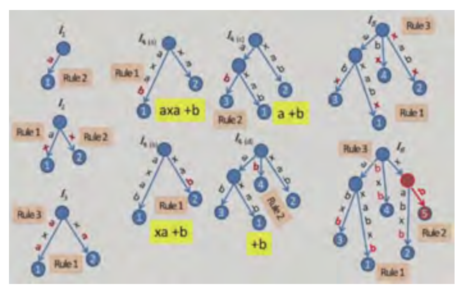


Figure VI.15: Naive Example for axabxb

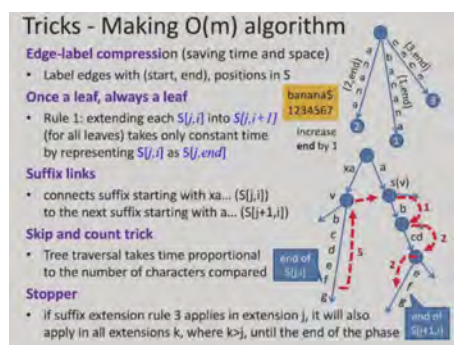


Figure VI.16: Tricks

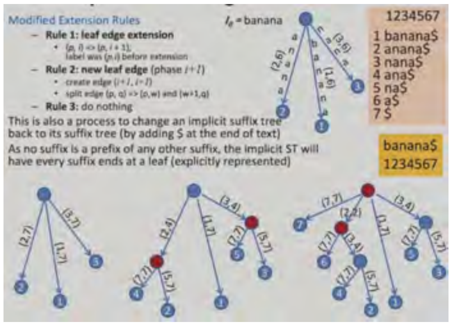


Figure VI.17: Full Example for banana

5.2 *Exact String Matching*

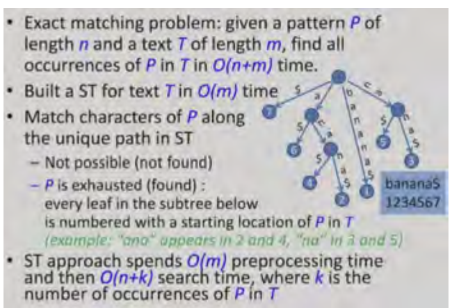


Figure VI.18: Exact String Matching

5.3 *Generalized Suffix Tree*

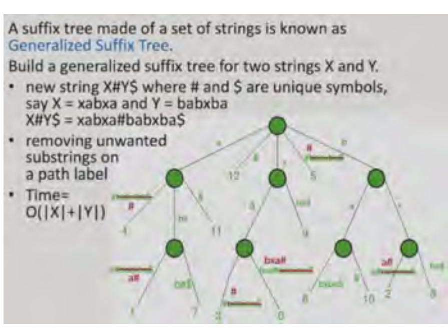


Figure VI.19: Generalized Suffix Tree

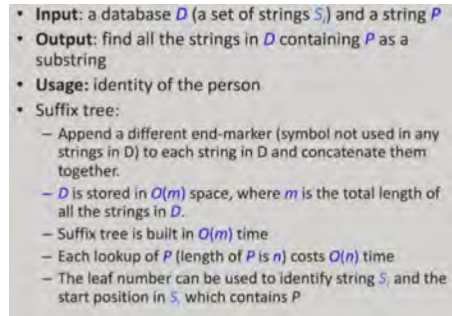


Figure VI.20: Substring Problem for a set of Strings

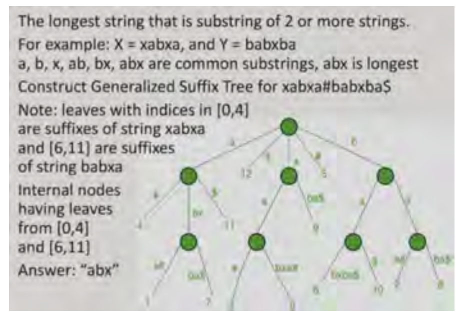


Figure VI.21: Longest Common Substring Problem

6. Suffix Array

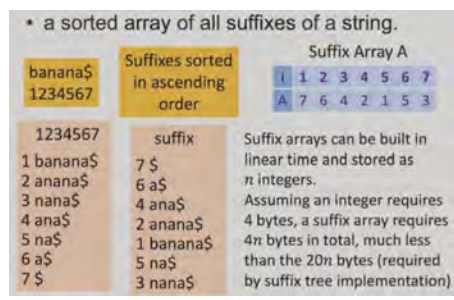


Figure VI.22: Suffix Array

VII Dynamic Programming

1. Divide and Conquer

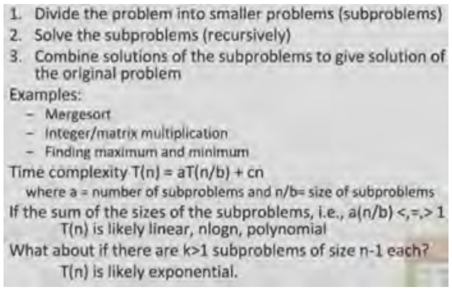


Figure VII.1: Divide and Conquer

2. Multistage Graph

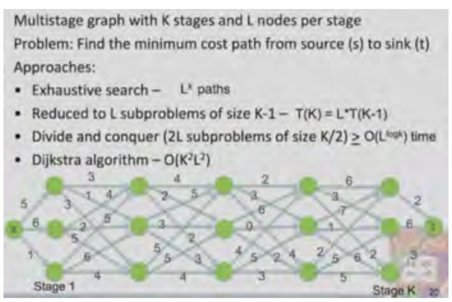


Figure VII.2: Shortest Path Problem

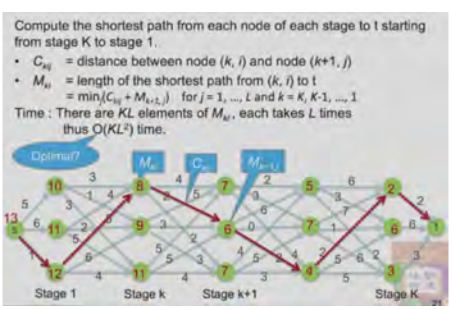


Figure VII.3: Multistage Graph - DP

Optimal

2.1 DP Implementation

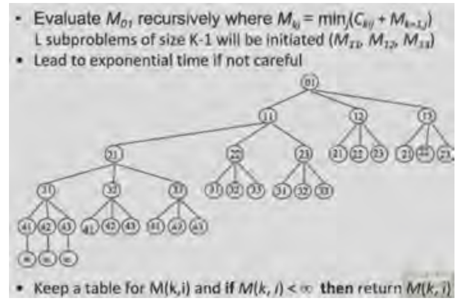


Figure VII.4: DP Implementation

3. Principle of Optimality

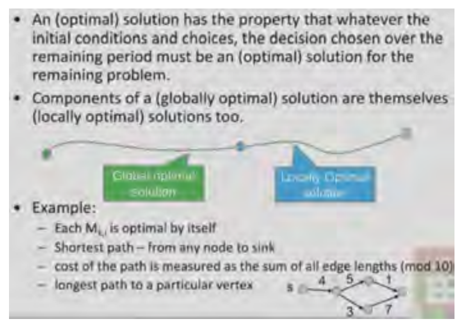


Figure VII.5: Principle of Optimality

3.1 integer Decomposition

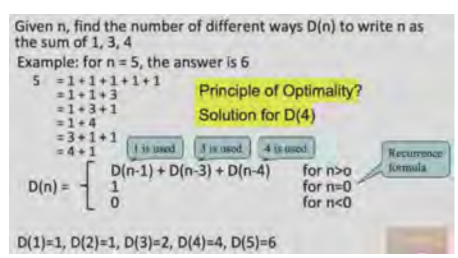


Figure VII.6: integer Decomposition

3.2 Number of Tilings

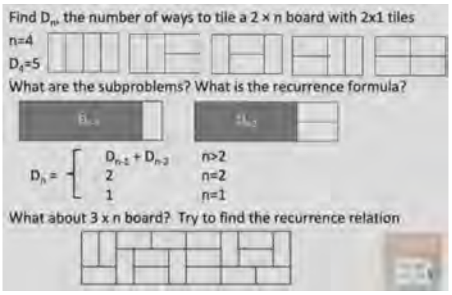


Figure VII.7: Number of Tilings

4. How to Solve Problems by DP

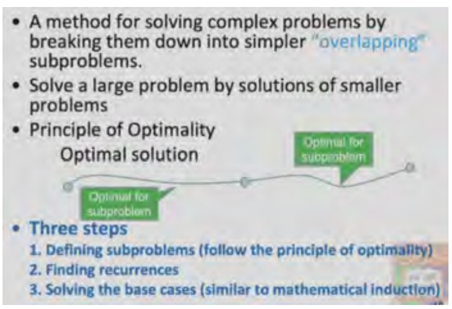


Figure VII.8: How to Solve Problems by DP

4.1 Longest Common Subsequence

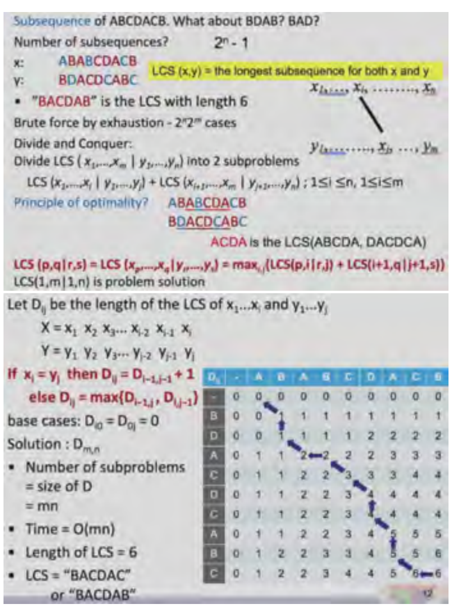


Figure VII.9: Longest Common Subsequence

4.2 Palindrome Problem

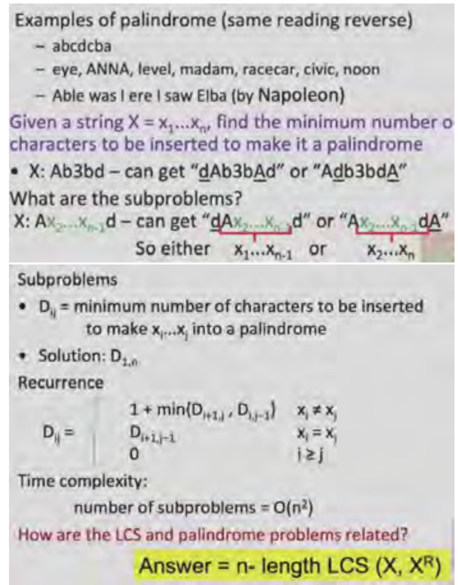


Figure VII.10: Palindrome Problem

4.3 Longest Increasing Subsequence

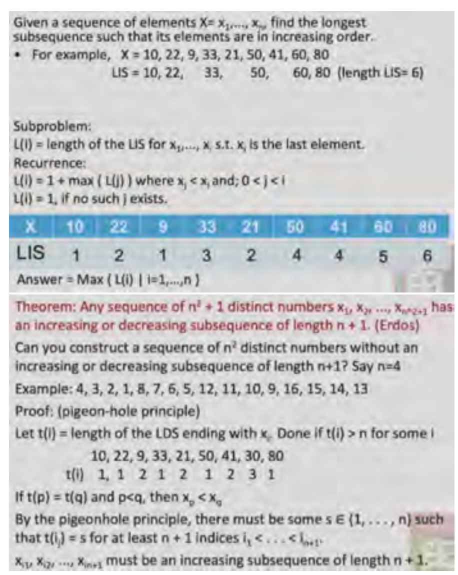


Figure VII.11: Longest Increasing Subsequence

4.4 *Tree Coloring*

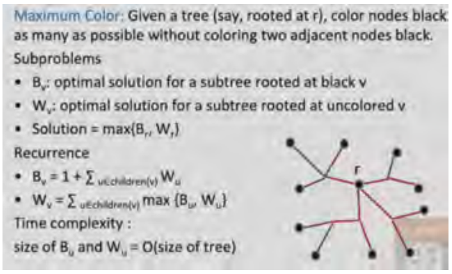


Figure VII.12: Tree Coloring

4.5 *Traveling Salesman Problem*

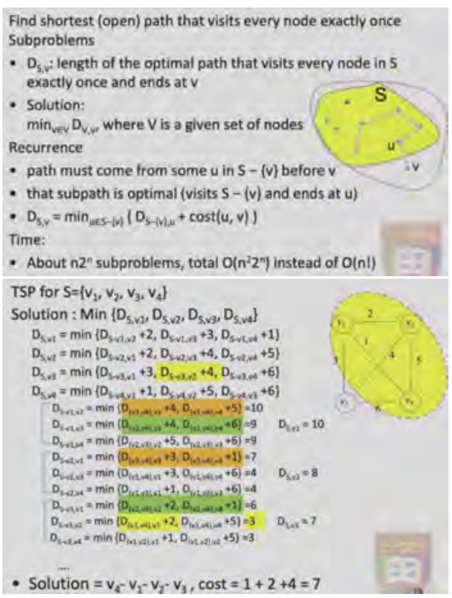


Figure VII.13: Traveling Salesman Problem

VIII Matching

1. Bipartite matching

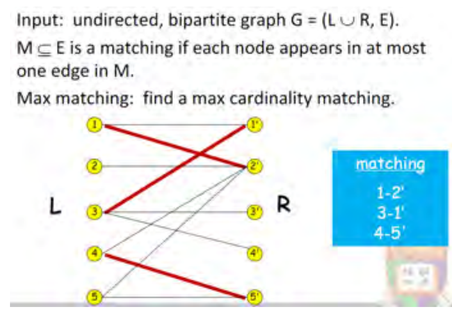


Figure VIII.1: Bipartite matching

1.1 *Bipartite Matching and Max Flow*

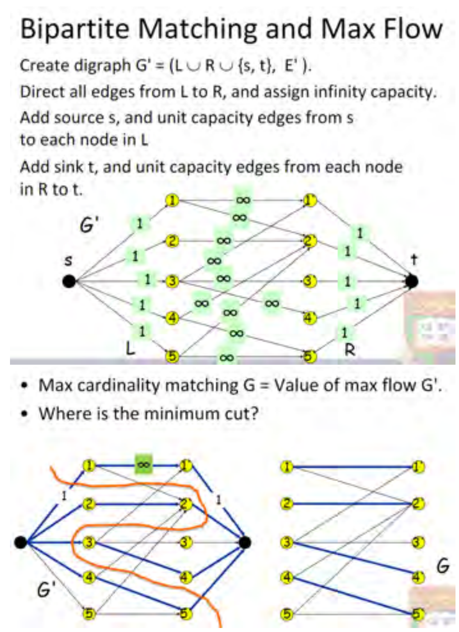


Figure VIII.2: Bipartite matching and Max Flow

1.2 *Alternating Path Approach*

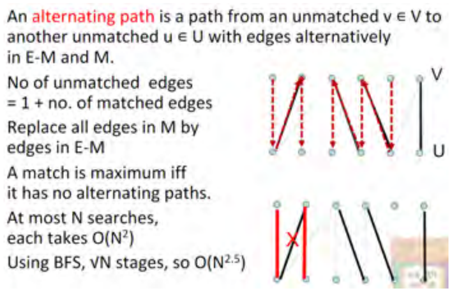


Figure VIII.3: Alternating Path Approach

2. Perfect Matching

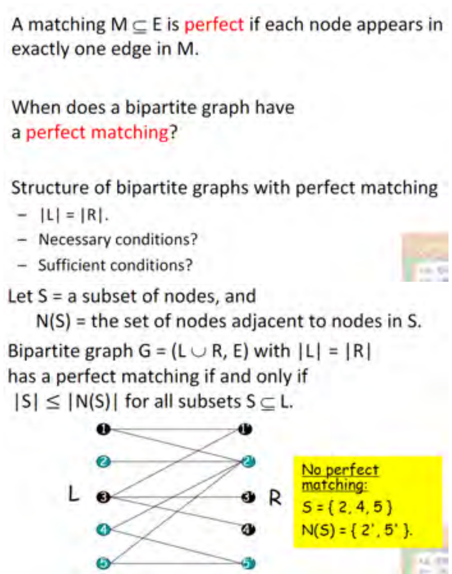


Figure VIII.4: Perfect Matching

2.1 *Marriage Theorem*

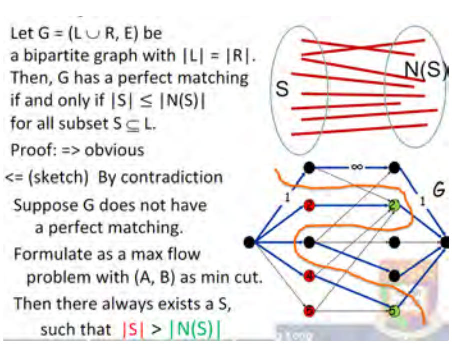


Figure VIII.5: Marriage Theorem

3. Perfect Matching for Dense Graph

4. Stable Marriage

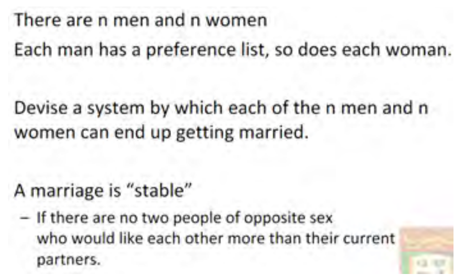


Figure VIII.6: Stable Marriage

4.1 *Example Preference Lists*

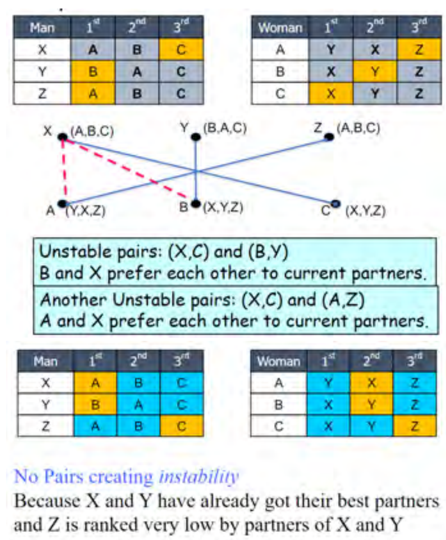


Figure VIII.7: Example Preference Lists

4.2 *Gale-Shapley (GS) Algorithm*

Men Propose (Women dispose)

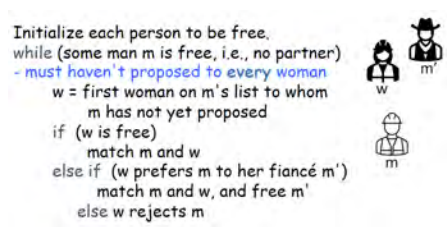


Figure VIII.8: Gale-Shapley (GS) Algorithm

后面是对这个算法的分析, 摸了.
不对, 我感觉这个肯定要考, 看看.

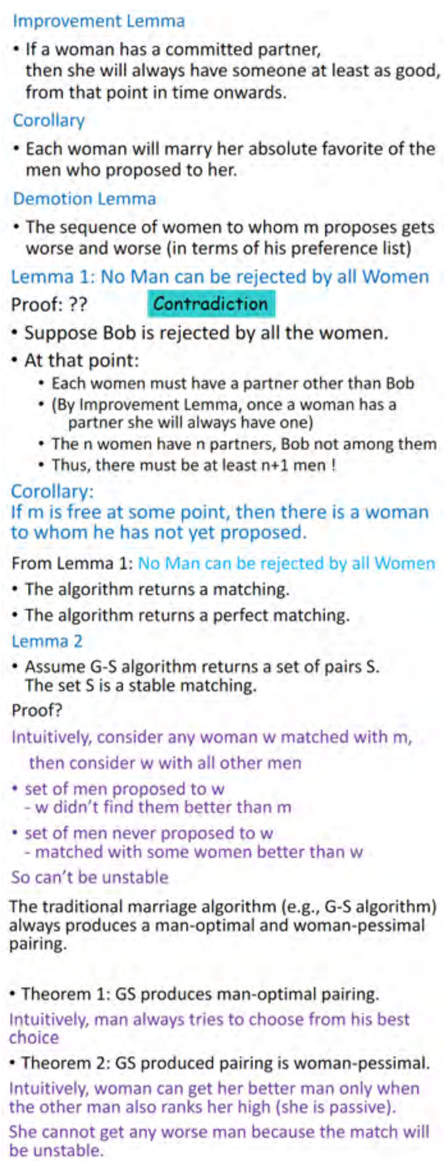


Figure VIII.9: Facts

5. Assignment Problem

A Examination

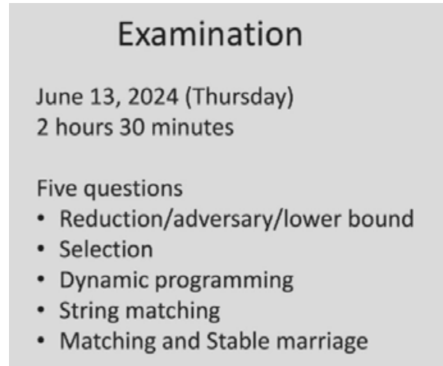


Figure A.1: exam