

Contents	IV Searching	9
I Introduction	1. Unbound Searching	9
1. Computing	1.1 Preliminary Algorithms	9
2. Algorithm	1.2 Higher Binary Search	9
2.1 Russell's Paradox	1.3 Ultimate Algorithm	9
3. Undecidable Problems	1.4 Sketch of Lower Bound	9
3.1 Post Corresponding Problem	2. Find X in a Sorted Matrix	9
3.2 Other Undecidable Problem	2.1 Solution	9
4. Algorithm Evaluation	2.2 Lower Bound	10
4.1 Running Time	V Reduction	10
4.2 Example Sort	1. Lower Bound by Reduction	10
在 7 次比较之内排序 5 个元素	2. Distinct Integers Sorting	10
4.3 Merge Insertion Sort	3. Proof of NP-completeness	11
Time Complexity	4. Integer Distinctness Problem	11
II Lower Bound	4.1 Closest Pair Problem	11
1. Adversary Argument(Oracle)	5. Data Compression	11
2. Decision Tree	5.1 Huffman code	11
2.1 硬币称重	5.2 Lower Bound for Huffman	11
2.2 Finding an element in a sorted list	6. 3SUM Problem	12
3. Finding the Maximum and Minimum	6.1 Collinearity	12
3.1 Divide and Conquer Approach	6.2 Segment Splitting Problem	12
Complexity	6.3 Motion Planning	12
4. Finding Second Maximum Element	6.4 M-3SUM and 3SUM are Equivalent	12
4.1 Adversary	VI String Matching	12
4.2 Lower Bound	1. Brute Force	12
5. Merging Sorted Lists	2. Knuth-Morris-Pratt(KMP) Algorithm	12
5.1 Adversary proof	3. The Boyer-Moore(BM) Algorithm	12
III Selection	3.1 Bad Character Rule	12
1. Lower Bound for Finding Median	3.2 Good Suffix Rules	13
1.1 Adversary	4. Tire	13
1.2 Median Algorithm	4.1 reTRIEval	13
1.3 Time Analysis	4.2 Find all strings start with a pattern	13
2. Finding the k th Largest/Smallest Elements	5. Suffix Tree	14
2.1 Lower Bound(Sketch)	5.1 Ukkonen's Algorithm	14
	5.2 Exact String Matching	15
	5.3 Generalized Suffix Tree	15
	6. Suffix Array	15

VII	Dynamic Programming	16
1.	Divide and Conquer	16
2.	Multistage Graph	16
2.1	DP Implementation	16
3.	Principle of Optimality	16
3.1	integer Decomposition	16
3.2	Number of Tilings	17
4.	How to Solve Problems by DP	17
4.1	Longest Common Subsequence	17
4.2	Palindrome Problem	17
4.3	Longest Increasing Subsequence	17
4.4	Tree Coloring	18
4.5	Traveling Salesman Problem	18
VIII	Matching	18
1.	Bipartite matching	18
1.1	Bipartite Matching and Max Flow	18
1.2	Alternating Path Approach	19
2.	Perfect Matching	19
2.1	Marriage Theorem	19
3.	Perfect Matching for Dense Graph	19
4.	Stable Marriage	19
4.1	Example Preference Lists	19
4.2	Gale-Shapley (GS) Algorithm	19
5.	Assignment Problem	20
A	Examination	20

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. Does the barber shave himself?

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

If $\text{barber} \notin S(\text{barber})$, then $\text{barber} \in S(\text{barber})$.

If $\text{barber} \in S(\text{barber})$, then $\text{barber} \notin S(\text{barber})$.

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 $P(P) = \text{Yes}$
Yes if $P(P) = \text{No}$

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.

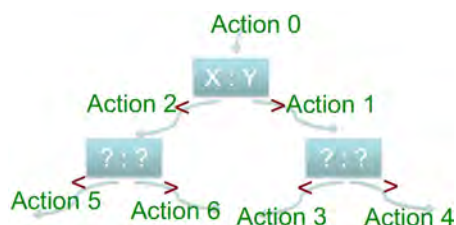


Figure I.1: decision tree

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

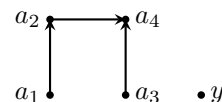
- $n!$ 个排列
- 下界, decision tree 的高度 $\lceil \log_2 n! \rceil$.

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

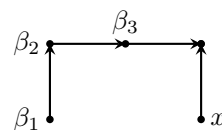
在 7 次比较之内排序 5 个元素 并不简单, 因为平常的比较式需要 8 次. 然后就是一个论文专门讲了这个算法. 第二步比较的应该是 a_4

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

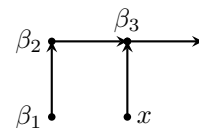


2) merge y into the chain

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.

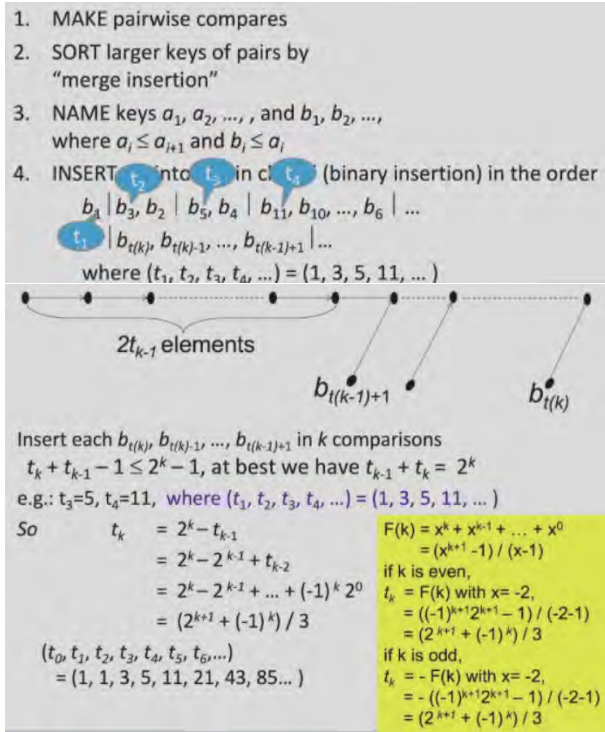


Figure I.2: Merge Insertion Sort

Time Complexity of Merge Insert

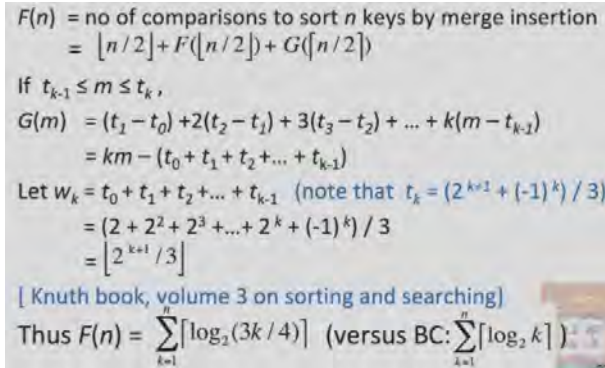


Figure I.3: Time Complexity

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 $\lceil \log_3 8 \rceil = 2$

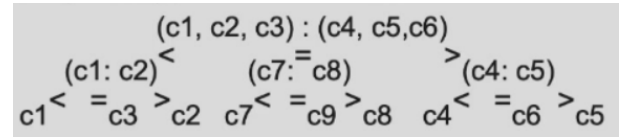


Figure II.1: 8 个硬币

- 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

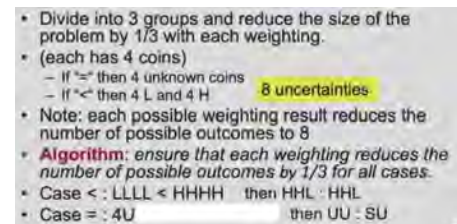


Figure II.2: 12 个硬币

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: $\lceil \log_2(n+1) \rceil$.

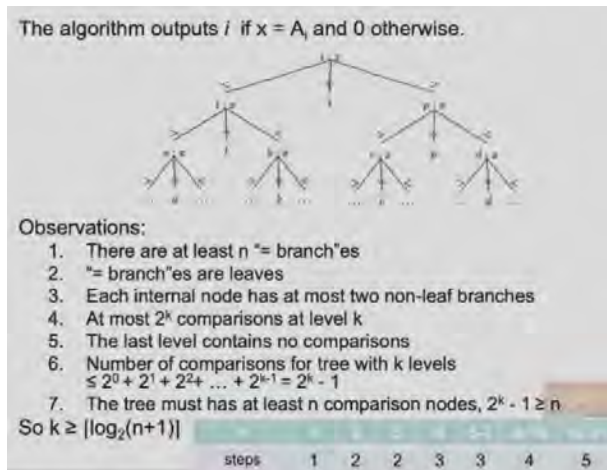


Figure II.3: 下界推导

3. Finding the Maximum and Minimum of N-Elements

3.1 Divide and Conquer Approach

```

Procedure MAXMIN(S): /*two outputs (max, min)*/
case  $S = \{a\}$  : return ( $a, a$ )
case  $S = \{a, b\}$  : return (MAX( $a, b$ ), MIN( $a, b$ ))
Else begin
    divide  $S$  into  $S_1$  and  $S_2$  /*equal-size subsets */
    ( $\text{max1}, \text{min1}$ )  $\leftarrow$  MAXMIN( $S_1$ );
    ( $\text{max2}, \text{min2}$ )  $\leftarrow$  MAXMIN( $S_2$ );
    return (MAX( $\text{max1}, \text{max2}$ ), MIN( $\text{min1}, \text{min2}$ ))

```

Figure II.4: Divide and Conquer Approach

Proof by Adversary (give out the least amount of information or take more comparisons)

- Max wins all
- Min loses all
- All other elements win and lose

Win (W) or lose (L) is one unit of information.

- Max is W (no lose) and Min is L (no win)
- All other elements are W and L

So at least $2n - 2$ units of information are needed

Each element can be

- W – at least one win, never lost
- L – at least one lost, never won
- WL – at least one win and one lost
- N – no compare

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
WL 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.

- At most $n/2$ comparisons which gives 2 units of information (when both are N)
 n units of information will be given out
- The remaining $(2n - 2 - n)$ units of information will take at least $(2n - 2 - n)$ comparisons.
 (as each comparison give at most 1 unit of information)
- Thus,
 $n/2 + (2n - 2 - n) = 3n/2 - 2$ comparisons are needed.

Figure II.7: Analysis

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)

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.

• Adversary: "bigger" winner (maximum) keeps winning

• Adversary answers $x > y$ if x has more losers (self too)

Assigning a weight $w(x)$ to each key (number of losers to x)

Initially all $w(x)=1$

Adversary rules:

Case	Adversary reply	Updating of weights
$w(x) > w(y)$	$x > y$	$w(x) := w(x) + w(y); w(y) := 0$
$w(x) = w(y) > 0$	$x > y$	$w(x) := w(x) + w(y); w(y) := 0$
$w(y) > w(x)$	$y > x$	$w(y) := w(x) + w(y); w(x) := 0$
$w(x) = w(y) = 0$	Consistent reply	No change

Figure II.8: Adversary strategy

5.1 Adversary proof

- Assume Algorithm A runs in $(2n - 2)$ comparisons and correctly merges 2 sorted lists X and Y of size n .
- Adversary: (# comparisons is max when X and Y interleave)
- Let $X_i = 2i - 1$ and $Y_i = 2i$ for $i = 1$ to n .
- Sorted list: $X_1 Y_1 X_2 Y_2 \dots X_n Y_n$
- Answering a query $Y_i < X_j$ as "YES" when $i < j$ and "NO" as $i \geq j$.
- At least 1 element X_i did not compare to both Y_{i-1} and Y_i (note that X_i with Y_1 only) if only $(2n - 2)$ comparisons.
- WLOG, X_i did not compare to Y_i .
- Let $X_i = 2i$ and $Y_i = 2i - 1$ (i.e., $X_1 Y_1 X_2 Y_2 \dots Y_{i-1} Y_i X_i X_{i+1} \dots X_n Y_n$)
- Orders of X and Y remain the same, but the resulting order should be different.
- Algorithm A gives the same sorted output which is wrong.
- Any correct algorithm that merges two sorted lists of size n takes at least $(2n - 1)$ comparisons.

Figure II.11: Adversary proof

4.2 Lower Bound

- The sum of weights is always n .
 - Finally, only the maximum element has nonzero weight n .
 - Adversary rules:** Number of losers to x at most double after each win, i.e., $w_k(x) \leq 2 w_{k-1}(x)$ where k = no of wins
 - Let K be the number of wins of the maximum,

$$n = w_K(x) \leq 2^K w_0(x) = 2^K$$

$$K \text{ (number of "direct" losers)} \geq \lceil \log_2 n \rceil$$
 - Second maximum = the winner of the direct losers
 - Lower bound is $n - 1 + \lceil \log_2 n \rceil - 1 = n + \lceil \log_2 n \rceil - 2$
- Do you know the optimal algorithm? Tournament

Figure II.9: Lower Bound

5. Merging Sorted Lists

Problem: Merge two sorted lists

Problem: Merge two sorted lists $X_1 < X_2 < \dots < X_n$ and $Y_1 < Y_2 < \dots < Y_n$ into a single sorted list

Algorithm takes $2n-1$ comparisons

13 24 26 15 27 28	Output	
24 26 15 27 28	1	
24 26 15 27 28	2	
24 26 15 27 28	13	Total 6 comparisons
26 15 27 28	15	
26 15 27 28	24	
15 27 28	26	
15 27 28	27, 28	

Best case : 1, 2, 3, 4 || 5, 6, 7, 8 which takes 4 comparisons

Is it optimal? Lower bound?

No of ways 2 sorted lists are merged $C(2n, n) = (2n)! / (n!n!)$

By decision tree: $\log_2 C(2n, n) = \log_2 [(2n)! / (n!n!)]$

Figure II.10: Merging Sorted Lists

III Selection

1. Lower Bound for Finding Median

Trivial lower bound: $n - 1$

but there are crucial comparison and non-crucial comparison.

Adversary: give non-crucial comparison

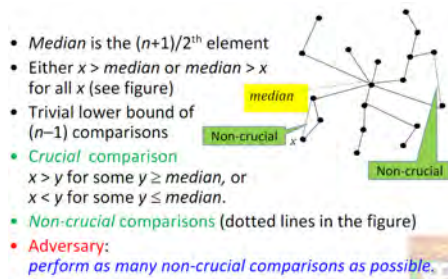


Figure III.1: Finding Median

1.1 Adversary

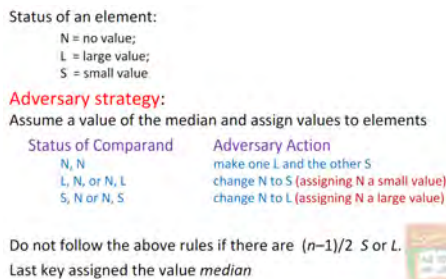


Figure III.2: Adversary

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

Example III.1. Median of 5 elements

- sorting: 7
- lower bound: 6

Assume median = 50

Comp	rand	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}	x_{16}	x_{17}	x_{18}	x_{19}	x_{20}
		N	-	N	-	N	-	N	-	N	-	N	-	N	-	N	-	N	-	N	-
x_1, x_2	L	60	S	20																	
x_3, x_4																					
x_5, x_6																					
x_7, x_8																					
x_9, x_{10}																					
x_{11}, x_{12}																					
x_{13}, x_{14}																					
x_{15}, x_{16}																					
x_{17}, x_{18}																					
x_{19}, x_{20}																					

There are already $(n-1)/2 = 3$ S. 50

At least $(n-1)/2$ L or S can be assigned freely,
thus at least $(n-1)/2$ non-crucial comparisons.
Thus the lower bound is $(n-1) + (n-1)/2 = 3(n-1)/2$ comparisons
crucial non-crucial

Figure III.3: Example for Median Finding

1.2 Median Algorithm

$O(n \log n)$

By sorting – $O(n \log n)$ time
Assume all keys are distinct.



If $|S_1| < |S_2|$, median is in S_2 , but **not** the median of S_2
i.e., the $(n/2 - |S_1|)^{\text{th}}$ element in S_2
So the general selection problem
Select (S, k) - find the k^{th} element of n elements.

Figure III.4: Median Algorithm

- Divide S into sets of 5 each, find median of each set **time complexity** = $6(n/5)$
- Find median of the medians, say m^* , recursively (using Select) **time complexity** = $W(n/5)$
where $W(n)$ = no. of comparisons by Select on n elements
- Partition S by m^* : S_1 contains elements $< m^*$
 S_2 contains elements $> m^*$
- if $|S_1| + 1 = k$ then output m^*
else if $k \leq |S_1|$ then Select (S_1, k)
else Select ($S_2, k - |S_1| - 1$)

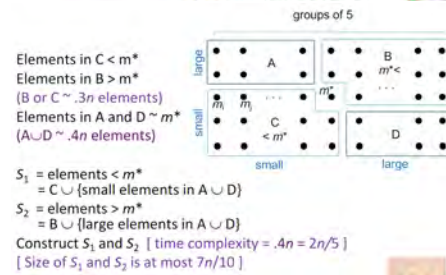


Figure III.5: Select(S, k)

1.3 Time Analysis

归纳法证明, 线性算法

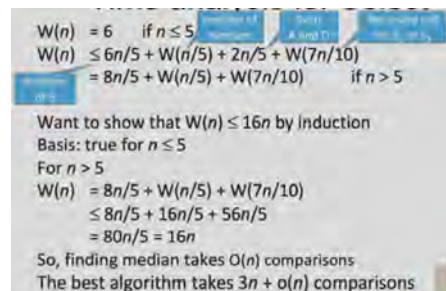


Figure III.6: Time Analysis

2. Finding the k^{th} Largest/Smallest Elements

k -selection problem

sorting: $O(n \log n)$

use Max heap

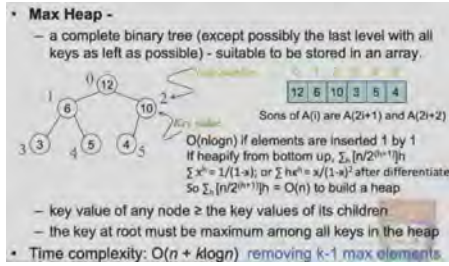


Figure III.7: Max heap

And we can also find max use mim heap

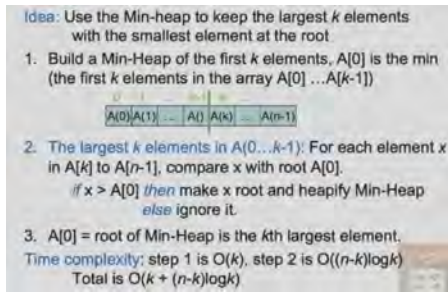


Figure III.8: mim heap

Theorem III.1.

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

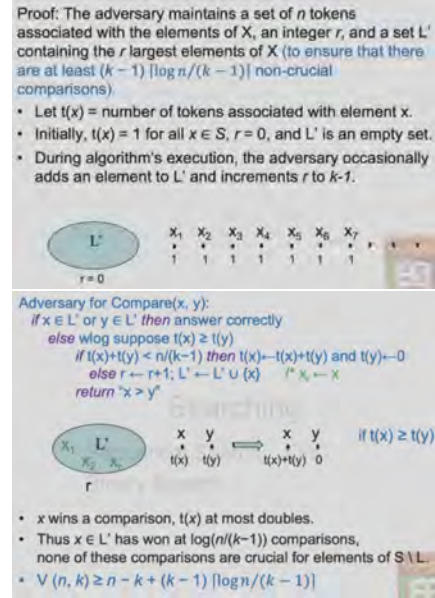


Figure III.11: theorem proof

2.1 Lower Bound(Sketch)

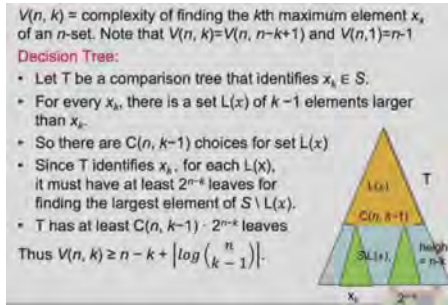


Figure III.9: Lower Bound

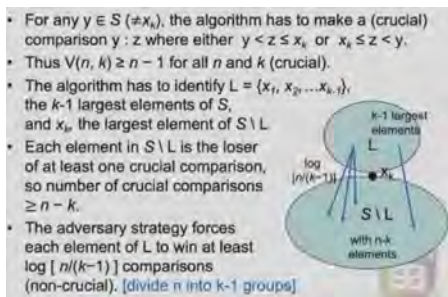


Figure III.10: Adversary selection

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

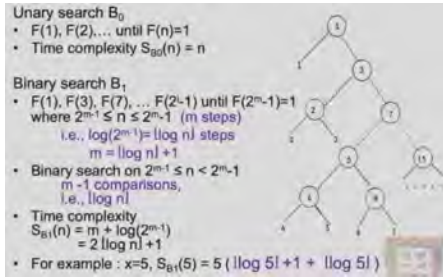


Figure IV.1: Preliminary Algorithms

1.2 Higher Binary Search

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

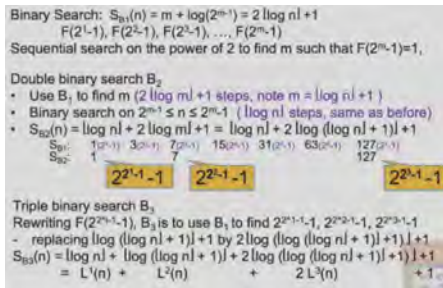


Figure IV.2: Higher Binary Search

1.3 Ultimate Algorithm

determine k for B_k to find n .

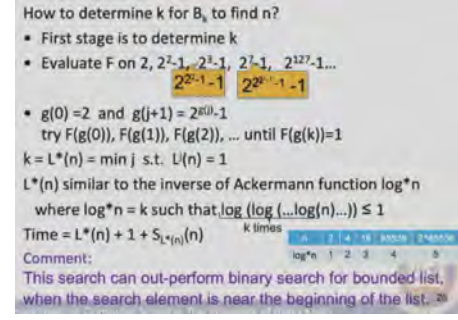


Figure IV.3: Ultimate Algorithm

1.4 Sketch of Lower Bound

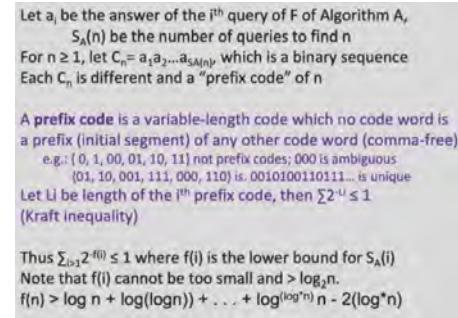


Figure IV.4: Sketch of Lower Bound

2. Find X in a Sorted Matrix

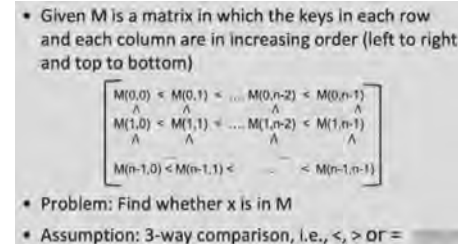


Figure IV.5: Find X in a Sorted Matrix

2.1 Solution

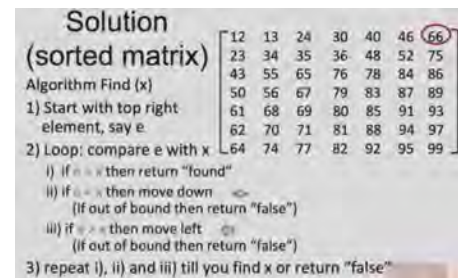


Figure IV.6: Solution

Complexity: $2n - 1$ comparisons

2.2 Lower Bound

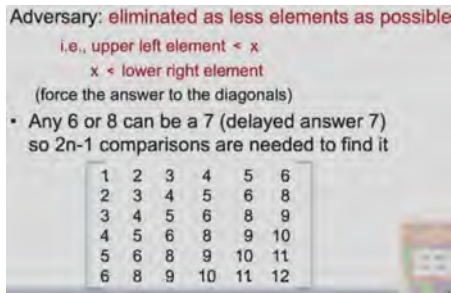


Figure IV.7: Adversary

V Reduction

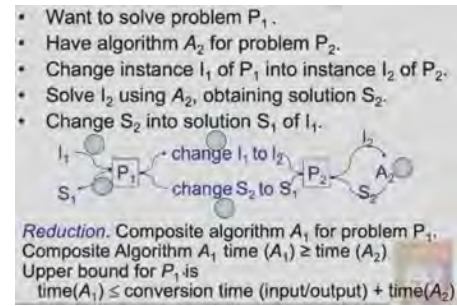


Figure V.1: Problem reduction P_1 to P_2 . ($P_1 \propto P_2$)

1. Lower Bound by Reduction

Reducing P_1 to P_2 , i.e., solve P_1 by solution of P_2
i.e., a fast algorithm for P_2 would give
a fast algorithm for P_1 too,
then lower bound of P_1 applies to P_2 .
i.e., lower bound $(P_1) \leq \text{lower bound}(P_2)$
To prove that P_2 is as hard as P_1 , reduce P_1 to P_2 .

Figure V.2: Lower Bound by Reduction

2. Distinct Integers Sorting

P_2 - sort a list of numbers, where all of the
numbers are distinct integers (none repeated)
Is P_2 easier than general sorting (P_1)?
(P_1 can deal with distinct and non-distinct integers)

Figure V.3: Distinct Integers Sorting

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

Suppose Algorithm A_{SDI} can sort distinct integers
(P_2) in $T(n)$ time.
We will show that this gives rise to an algorithm A
for general sorting (P_1) in $O(T(n))$ time.
Since A run in $\Omega(n \log n)$ time, then so must A_{SDI} .
General Sorting \propto Distinct Integer Sorting
Another notation " \leq " instead of " \propto "

Figure V.4: proof

Given the algorithm for sorting distinct integers, A_{SDI} .
We want to sort $\{a_1, a_2, a_3, \dots, a_n\}$, might not be
distinct.
For each a_i , let $b_i = a_i + i$.
Note that if $a_i < a_j$, then $b_i < b_j$.
Example: if the input is 1,2,1,4 (multiply each by 4
and add its position in the list), the b_i are 5,10,7,20

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

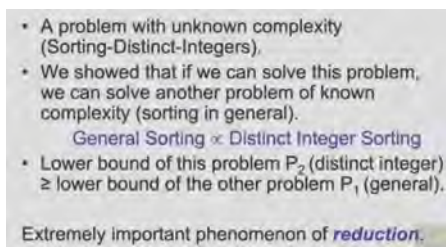


Figure V.6: phenomenon reduction

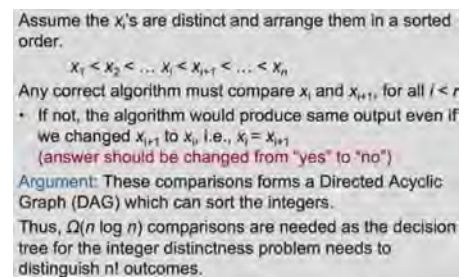


Figure V.10: Integer Distinctness Problem

3. Proof of NP-completeness

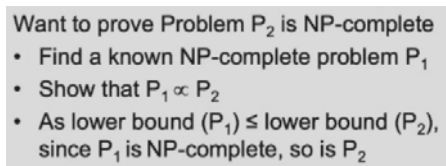


Figure V.7: Proof of NP-completeness

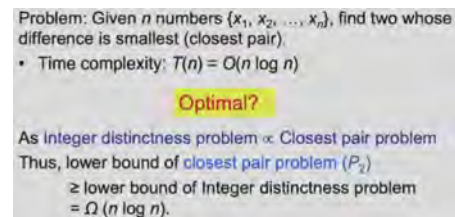


Figure V.11: Closest Pair Problem

5. Data Compression

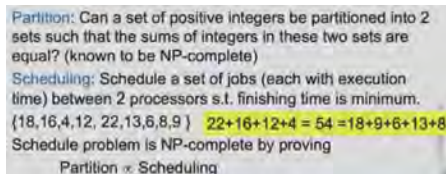


Figure V.8: Example

4. Integer Distinctness Problem

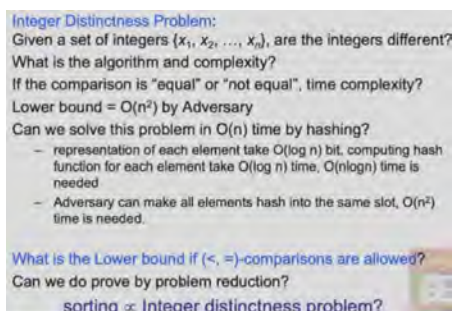


Figure V.9: Integer Distinctness Problem

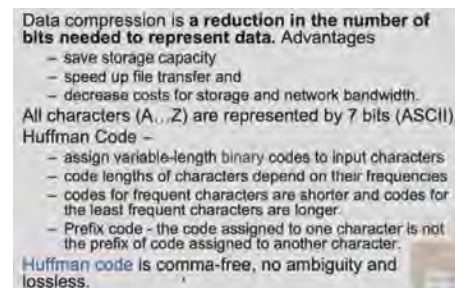


Figure V.12: Data Compression

5.1 Huffman code

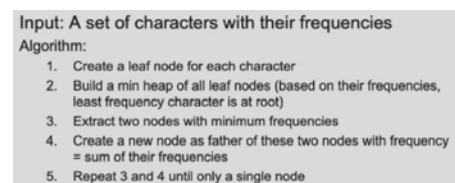


Figure V.13: Huffman code

Complexity: $O(n \log n)$

5.2 Lower Bound for Huffman

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

计算 hash 也有复杂度, 一般是个大于 $\log n$ 的常数.

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

The Boyer-Moore (BM) algorithm gives good performance most of the cases, best is $O(m/n)$.
BM algorithm slides P from left to right; however, BM compares P and T from right to left, i.e., If $P[n]$ matches with $T[i]$, then $P[n-1]$ with $T[i-1]$, etc.

Figure VI.1: BM Algorithm

Bad-character heuristics
 $T = \text{swim in the water ...}$
 $P = \text{device ?}$
 device
 If $P[j] \neq T[i]$ and $T[i]$ does not appear in P , P is advanced by j .

Good suffix heuristic
 $T = \text{cook in the stove}$
 $P = \text{onion ?}$
 onion
 Mismatch at $P[j]$ and $P[j+1, n]$ matches with T .
 Shift P such that T is aligned with the rightmost occurrence of $P[j+1, n]$.

Figure VI.2: Example

3.1 Bad Character Rule

Bad Character Rule

Example: (right-to-left matching)
 $GCTTCGTACCTTTTGGCGCGCGGAA$
 $CCTTTG$ || $s1=6, k=2$
 $s1=6, k=2$ skip 3

Slide the pattern to match with the mismatched character

Bad character rule
 Array $s1$ tells how far the pattern can skip for matching the first unmatched character (rightmost).
 Ex: $P=CCTTTG?$ where $? =$ first mismatch of P

Pattern	Index	Matched	Mismatch	skip
CCTTTG?	1	C	C	0
CCTTTG?	2	C	C	0
CCTTTG?	3	T	T	0
CCTTTG?	4	T	T	0
CCTTTG?	5	T	T	0
CCTTTG?	6	G	G	0
CCTTTG?	7	G	G	0
CCTTTG?	8	A	A	0

Pattern CCTTTG? length=8
 Index 12345678
 $s1 =$ length of string - index
 $skip = \min(1, s1 - k)$, where $k =$ number of matches

Figure VI.3: Bad Character Rule

3.2 Good Suffix Rules

Make use of the matched characters to skip

CGTGCCTACTTACTTACTACGCGAA...

CTTACTTAC

CTTACTTAC

CTTACTTAC

The pattern will slide (by s_2 characters) so as to align with the matched characters.

Case 1:

DBACCBACDEABBCEDCAB...

ABDABCABDCAB

ABDABCABDCAB

ABDABCABDCAB

Let u be the matched pattern e.g., $u = "AB"$

Skip this shift because same preceding character

There exists a substring in P which matches with pattern u and is preceded by a different character as u .

Case 2:

DBACCBACDEABBCEDCAB...

ABDABCABDCAB

ABDABCABDCAB

Case 3:

DBACCBACDEABBCEDCAB...

EDAC

EDAC

- No substring satisfies Case 1.
- Find the longest prefix of P , which is also a suffix of matched pattern u (e.g., AB and $u = DEAB$)

Case 3:

DBACCBACDEABBCEDCAB...

EDAC

EDAC

- No substring satisfies Cases 1 and 2, skip whole

$s_2(k)$ = distance between the matched suffix (size k) and its rightmost "occurrence" in the pattern

Figure VI.4: Good Suffix Rules

4.1 reTRIEval

reTRIEval

- Each edge is labeled with a character.
- Some nodes \bullet are marked as representing words.
- Tree children are sorted.
- A preorder traversal of the trie outputs all words in sorted order.
- DFS finds all words rooted at "compa", i.e., -ny, -ss
- Fail to find "compile"
- $O(m)$ time to build the trie
- $O(n)$ time to check if P is prefix of some string, say $P = "compu"$
- Assume TRIE is precomputed, time complexity to find all strings that starts with a pattern?

Answer: DFS takes $O(m+n)$

Ideally $O(n+z)$ where z = no of matches, independent of m

Figure VI.7: reTRIEval

4.2 Find all strings start with a pattern

Pattern = GTAGCGGCG

letter	G	T	A	C
s_1	2	7	6	1

skip = $\min(1, s_1 - k)$, where k = no of matches

letter	G	T	A	G	C	G	C	G
k	9	8	7	6	5	4	3	2
s_2	8	8	8	8	8	8	3	2

GTTATAGCTGATCGCGGCGTAGCGGCGAG

GTAGCGGCG bc=7, gc=0

GTAGCGGCG bc=1, gc=3

GTTATAGCTGATCGCGGCGTAGCGGCGAG

GTAGCGGCG bc=1, gc=8

GTAGCGGCG

The Boyer-Moore (BM) algorithm gives good performance most of the cases, best is $O(m/n)$.

Figure VI.5: Example

- Add an end-marker $\$$ at the ends of the strings where $\$$ is the smallest alphabet (character)
- Leaf nodes \bullet correspond to strings.
- number of leaves = number of strings
- All internal nodes correspond to routing structure

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

4. Tire

- Given a set of text strings, $\{T_1, T_2, \dots, T_k\}$, total length m i.e., $|T_1| + \dots + |T_k| = m$, and a pattern string P of length n
- Goal: Find all text strings that start with P .
- Naïve solution : Just scan over all the strings and see which ones start with P .

comp
company
compass
compete
composer
compute
computer
computing

$P = \text{comp}$

Many duplicate efforts to scan shared prefix

- Question: If we have a set of fixed text strings and varying patterns, can we speed this up?

Figure VI.6: Autocomplete Problem

- A node (not root) is silly if it has only one child.
- A Patricia Trie is a trie where all the silly nodes are merged into their parents
- Every internal node in a Patricia trie (except possibly the root) has two or more children.

Theorem: The number of nodes in a Patricia trie with k strings is always $O(k)$.

Proof: Since each internal node (non-root) has at least two children. This means there are at most k internal nodes. Thus, there is a total of $O(k)$ nodes.

Figure VI.9: Patricia Tire

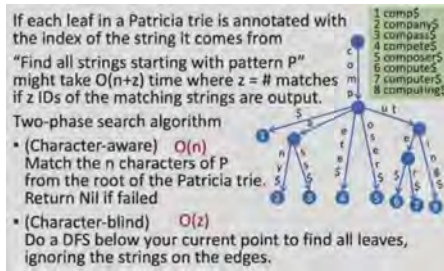


Figure VI.10: Indexed Patricia Tire

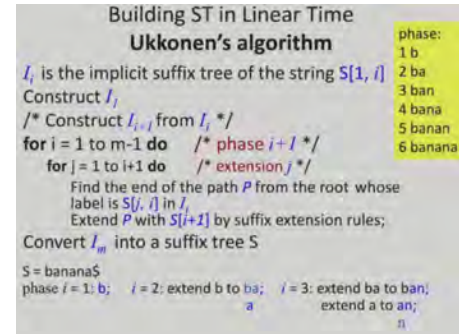


Figure VI.13: Ukkonen's Algorithm

5. Suffix Tree

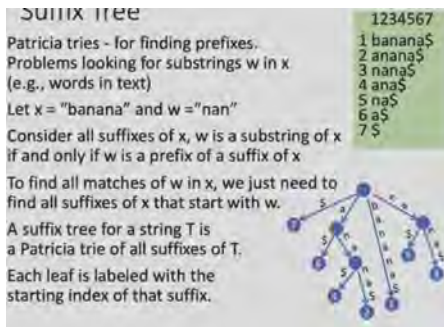


Figure VI.11: Suffix Tree

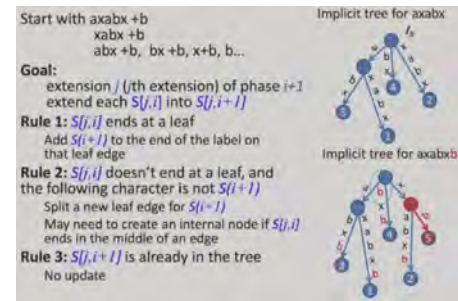


Figure VI.14: Extension Rules

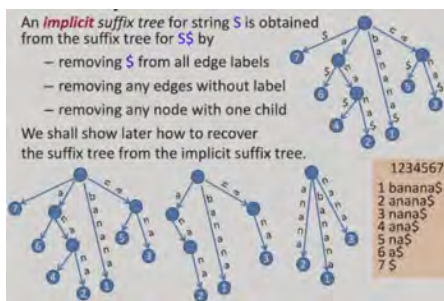


Figure VI.12: Implicit Suffix Tree

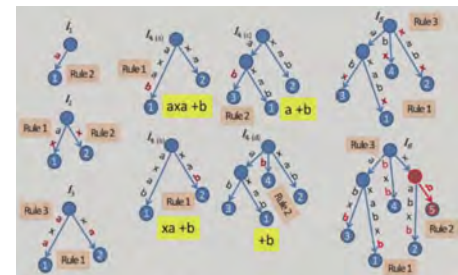
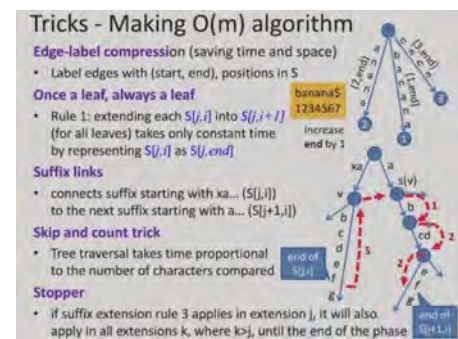
Figure VI.15: Naive Example for $axabxb$ 

Figure VI.16: Tricks

5.1 Ukkonen's Algorithm

Building Suffix Tree in Linear Time

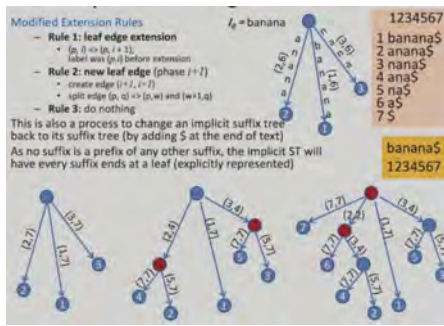


Figure VI.17: Full Example for banana

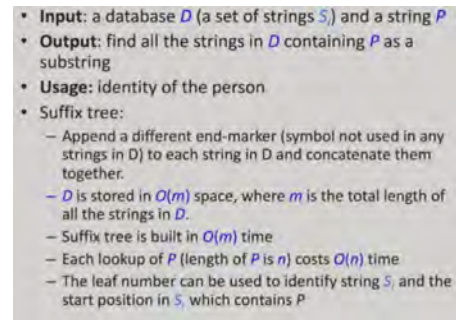


Figure VI.20: Substring Problem for a set of Strings

5.2 Exact String Matching

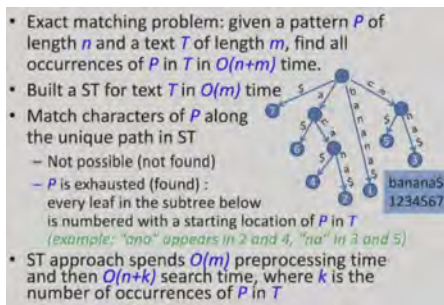


Figure VI.18: Exact String Matching

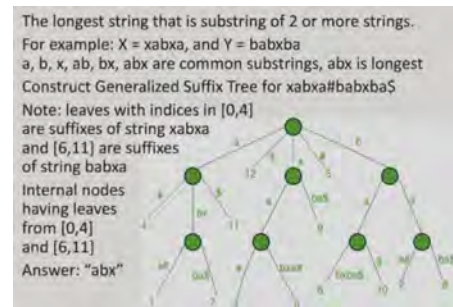


Figure VI.21: Longest Common Substring Problem

6. Suffix Array

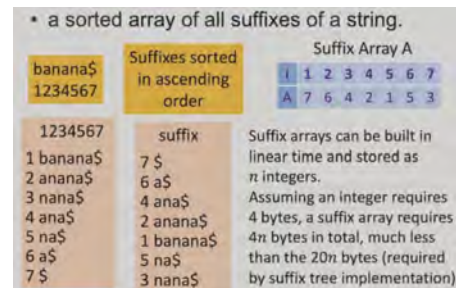


Figure VI.22: Suffix Array

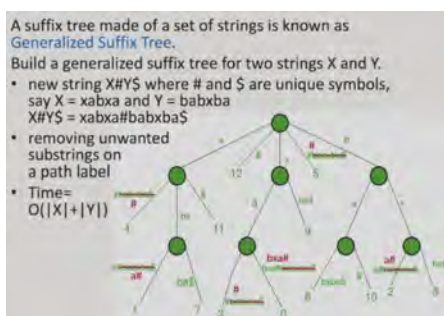


Figure VI.19: Generalized Suffix Tree

VII Dynamic Programming

1. Divide and Conquer

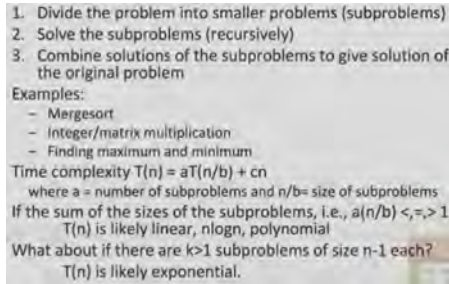


Figure VII.1: Divide and Conquer

2. Multistage Graph

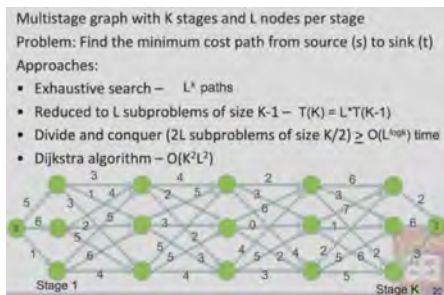


Figure VII.2: Shortest Path Problem

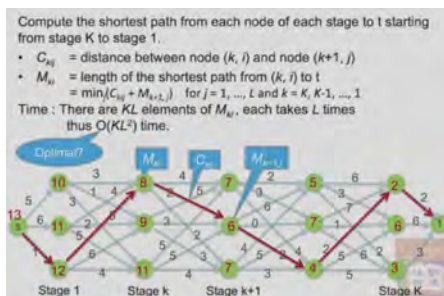


Figure VII.3: Multistage Graph - DP

Optimal

2.1 DP Implementation

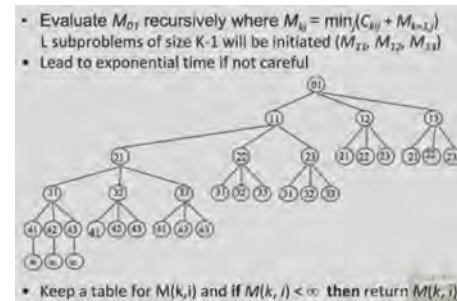


Figure VII.4: DP Implementation

3. Principle of Optimality

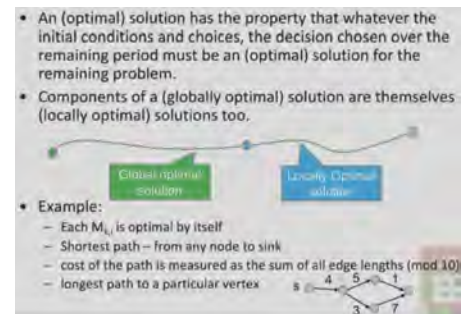


Figure VII.5: Principle of Optimality

3.1 integer Decomposition

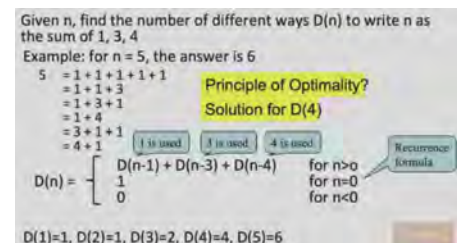


Figure VII.6: integer Decomposition

3.2 Number of Tilings

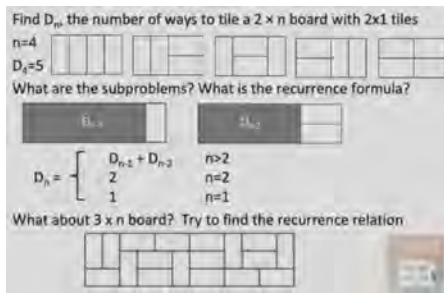


Figure VII.7: Number of Tilings

4. How to Solve Problems by DP

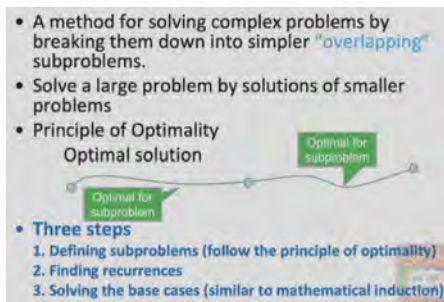


Figure VII.8: How to Solve Problems by DP

4.2 Palindrome Problem

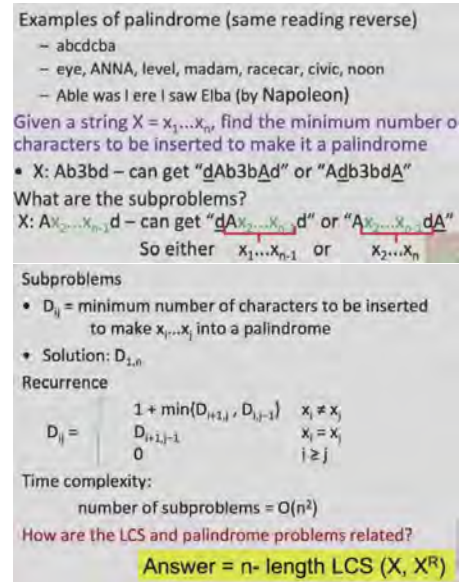


Figure VII.10: Palindrome Problem

4.1 Longest Common Subsequence

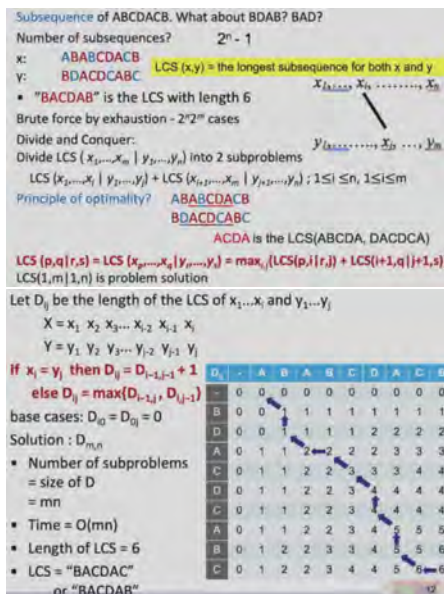


Figure VII.9: Longest Common Subsequence

4.3 Longest Increasing Subsequence

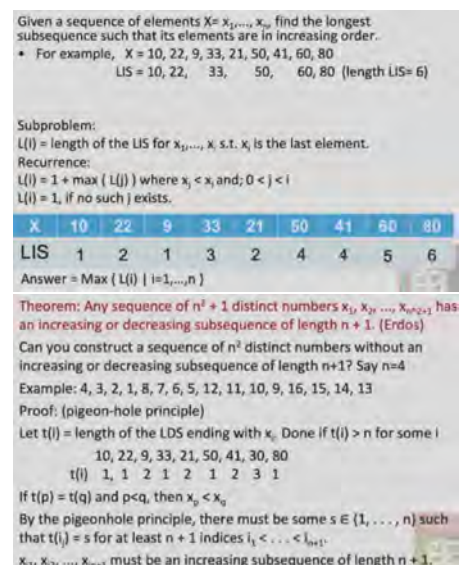


Figure VII.11: Longest Increasing Subsequence

4.4 Tree Coloring

Maximum Color: Given a tree (say, rooted at r), color nodes black as many as possible without coloring two adjacent nodes black.

Subproblems

- B_v : optimal solution for a subtree rooted at black v
- W_v : optimal solution for a subtree rooted at uncolored v
- Solution = $\max(B_r, W_r)$

Recurrence

- $B_v = 1 + \sum_{u \in \text{children}(v)} W_u$
- $W_v = \sum_{u \in \text{children}(v)} \max(B_u, W_u)$

Time complexity:
size of B_u and $W_u = O(\text{size of tree})$




Figure VII.12: Tree Coloring

4.5 Traveling Salesman Problem

Find shortest (open) path that visits every node exactly once

Subproblems

- $D_{S,v}$: length of the optimal path that visits every node in S exactly once and ends at v
- Solution:**
 $\min_{v \in V} D_{S,v}$, where V is a given set of nodes

Recurrence

- path must come from some u in $S - \{v\}$ before v
- that subpath is optimal (visits $S - \{v\}$ and ends at u)
- $D_{S,v} = \min_{u \in S - \{v\}} (D_{S-\{v\},u} + \text{cost}(u, v))$

Time:

- About n^2 subproblems, total $O(n^2)$ instead of $O(n!)$

TSP for $S = \{v_1, v_2, v_3, v_4\}$

Solution: $\min \{D_{S,v_1}, D_{S,v_2}, D_{S,v_3}, D_{S,v_4}\}$

$$D_{S,v_1} = \min \{D_{S-\{v_1\},v_2} + 2, D_{S-\{v_1\},v_3} + 3, D_{S-\{v_1\},v_4} + 1\}$$

$$D_{S,v_2} = \min \{D_{S-\{v_2\},v_1} + 2, D_{S-\{v_2\},v_3} + 4, D_{S-\{v_2\},v_4} + 5\}$$

$$D_{S,v_3} = \min \{D_{S-\{v_3\},v_1} + 3, D_{S-\{v_3\},v_2} + 4, D_{S-\{v_3\},v_4} + 6\}$$

$$D_{S,v_4} = \min \{D_{S-\{v_4\},v_1} + 1, D_{S-\{v_4\},v_2} + 5, D_{S-\{v_4\},v_3} + 6\}$$

$$D_{S-\{v_1\},v_2} = \min \{D_{S-\{v_1,v_2\},v_3} + 4, D_{S-\{v_1,v_2\},v_4} + 5\} = 10$$

$$D_{S-\{v_1\},v_3} = \min \{D_{S-\{v_1,v_3\},v_2} + 4, D_{S-\{v_1,v_3\},v_4} + 6\} = 9$$

$$D_{S-\{v_1\},v_4} = \min \{D_{S-\{v_1,v_4\},v_2} + 5, D_{S-\{v_1,v_4\},v_3} + 6\} = 9$$

$$D_{S-\{v_2\},v_1} = \min \{D_{S-\{v_2,v_1\},v_3} + 3, D_{S-\{v_2,v_1\},v_4} + 1\} = 7$$

$$D_{S-\{v_2\},v_3} = \min \{D_{S-\{v_2,v_3\},v_1} + 3, D_{S-\{v_2,v_3\},v_4} + 6\} = 4$$

$$D_{S-\{v_2\},v_4} = \min \{D_{S-\{v_2,v_4\},v_1} + 1, D_{S-\{v_2,v_4\},v_3} + 6\} = 4$$

$$D_{S-\{v_3\},v_1} = \min \{D_{S-\{v_3,v_1\},v_2} + 2, D_{S-\{v_3,v_1\},v_4} + 5\} = 6$$

$$D_{S-\{v_3\},v_2} = \min \{D_{S-\{v_3,v_2\},v_1} + 2, D_{S-\{v_3,v_2\},v_4} + 5\} = 3$$

$$D_{S-\{v_3\},v_4} = \min \{D_{S-\{v_3,v_4\},v_1} + 1, D_{S-\{v_3,v_4\},v_2} + 5\} = 3$$

Solution: $v_4 - v_2 - v_3 - v_1$, cost = $1 + 2 + 4 = 7$




Figure VII.13: Traveling Salesman Problem

VIII Matching

1. Bipartite matching

Input: undirected, bipartite graph $G = (L \cup R, E)$.
 $M \subseteq E$ is a matching if each node appears in at most one edge in M .
 Max matching: find a max cardinality matching.

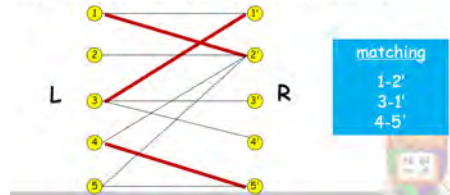
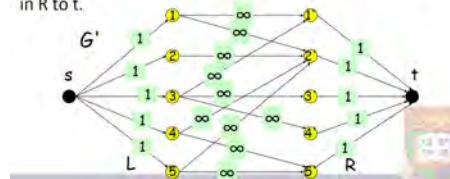


Figure VIII.1: Bipartite matching

1.1 Bipartite Matching and Max Flow

Bipartite Matching and Max Flow

Create digraph $G' = (L \cup R \cup \{s, t\}, E')$.
 Direct all edges from L to R , and assign infinity capacity.
 Add source s , and unit capacity edges from s to each node in L .
 Add sink t , and unit capacity edges from each node in R to t .



- Max cardinality matching $G = \text{Value of max flow } G'$.
- Where is the minimum cut?

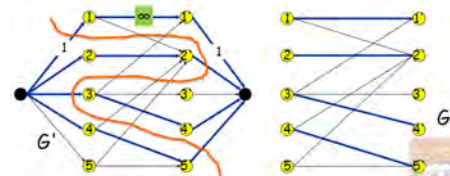


Figure VIII.2: Bipartite matching and Max Flow

1.2 Alternating Path Approach

An **alternating path** is a path from an unmatched $v \in V$ to another unmatched $u \in U$ with edges alternatively in $E-M$ and M .

No of unmatched edges
= 1 + no. of matched edges

Replace all edges in M by edges in $E-M$

A match is maximum iff it has no alternating paths.

At most N searches, each takes $O(N^2)$

Using BFS, $\forall N$ stages, so $O(N^{2.5})$

Figure VIII.3: Alternating Path Approach

2. Perfect Matching

A matching $M \subseteq E$ is **perfect** if each node appears in exactly one edge in M .

When does a bipartite graph have a **perfect matching**?

Structure of bipartite graphs with perfect matching

- $|L| = |R|$.
- Necessary conditions?
- Sufficient conditions?

Let S = a subset of nodes, and

$N(S)$ = the set of nodes adjacent to nodes in S .

Bipartite graph $G = (L \cup R, E)$ with $|L| = |R|$

has a perfect matching if and only if

$|S| \leq |N(S)|$ for all subsets $S \subseteq L$.

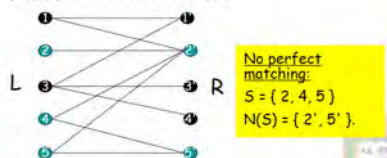


Figure VIII.4: Perfect Matching

2.1 Marriage Theorem

Let $G = (L \cup R, E)$ be a bipartite graph with $|L| = |R|$. Then, G has a perfect matching if and only if $|S| \leq |N(S)|$ for all subset $S \subseteq L$.

Proof: \Rightarrow obvious

\Leftarrow (sketch) By contradiction

Suppose G does not have a perfect matching.

Formulate as a max flow problem with (A, B) as min cut.

Then there always exists a S , such that $|S| > |N(S)|$

Figure VIII.5: Marriage Theorem

3. Perfect Matching for Dense Graph

4. Stable Marriage

There are n men and n women

Each man has a preference list, so does each woman.

Devise a system by which each of the n men and n women can end up getting married.

A marriage is "stable"

- If there are no two people of opposite sex who would like each other more than their current partners.

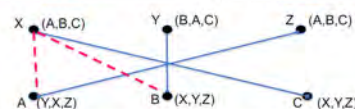


Figure VIII.6: Stable Marriage

4.1 Example Preference Lists

Man	1 st	2 nd	3 rd
X	A	B	C
Y	B	A	C
Z	A	B	C

Woman	1 st	2 nd	3 rd
A	Y	X	Z
B	X	Y	Z
C	X	Y	Z



Unstable pairs: (X, C) and (B, Y)
B and X prefer each other to current partners.

Another Unstable pairs: (X, C) and (A, Z)
A and X prefer each other to current partners.

Man	1 st	2 nd	3 rd
X	A	B	C
Y	B	A	C
Z	A	B	C

Woman	1 st	2 nd	3 rd
A	Y	X	Z
B	X	Y	Z
C	X	Y	Z

No Pairs creating *instability*

Because X and Y have already got their best partners and Z is ranked very low by partners of X and Y

Figure VIII.7: Example Preference Lists

4.2 Gale-Shapley (GS) Algorithm

Men Propose (Women dispose)

Initialize each person to be free,
while (some man m is free, i.e., no partner)
- must haven't proposed to every woman
 w = first woman on m 's list to whom
 m has not yet proposed
if (w is free)
match m and w
else if (w prefers m to her fiancé m')
match m and w , and free m'
else w rejects m

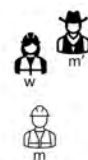


Figure VIII.8: Gale-Shapley (GS) Algorithm

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。

Improvement Lemma

- If a woman has a committed partner, then she will always have someone at least as good, from that point in time onwards.

Corollary

- Each woman will marry her absolute favorite of the men who proposed to her.

Demotion Lemma

- The sequence of women to whom m proposes gets worse and worse (in terms of his preference list)

Lemma 1: No Man can be rejected by all Women

Proof: ??

Contradiction

- Suppose Bob is rejected by all the women.
- At that point:
 - Each woman must have a partner other than Bob
 - (By Improvement Lemma, once a woman has a partner she will always have one)
 - The n women have n partners, Bob not among them
 - Thus, there must be at least $n+1$ men !

Corollary:

If m is free at some point, then there is a woman to whom he has not yet proposed.

From Lemma 1: No Man can be rejected by all Women

- The algorithm returns a matching.
- The algorithm returns a perfect matching.

Lemma 2

- Assume G-S algorithm returns a set of pairs S . The set S is a stable matching.

Proof?

Intuitively, consider any woman w matched with m , then consider w with all other men

- set of men proposed to w
 - w didn't find them better than m
- set of men never proposed to w
 - matched with some women better than w

So can't be unstable

The traditional marriage algorithm (e.g., G-S algorithm) always produces a man-optimal and woman-pessimal pairing.

- Theorem 1: GS produces man-optimal pairing.

Intuitively, man always tries to choose from his best choice

- Theorem 2: GS produced pairing is woman-pessimal.

Intuitively, woman can get her better man only when the other man also ranks her high (she is passive).

She cannot get any worse man because the match will be unstable.

Figure VIII.9: Facts

A Examination

Examination

June 13, 2024 (Thursday)

2 hours 30 minutes

Five questions

- Reduction/adversary/lower bound
- Selection
- Dynamic programming
- String matching
- Matching and Stable marriage

Figure A.1: exam

5. Assignment Problem