NOI 2010 – solutions to contest tasks.

School of Computing
National University of Singapore

Chang Ee-Chien

13 March 2010

Scientific Committee

Chang Ee-Chien (chair)

Martin Henz

Leong Wing Lup, Ben

Frank Stephan

Sung Wing Kin, Ken

Tan Keng Yan, Colin

Overview

- Two aspects in programming/algorithm design
  - ***Correctness***
  - ***Efficiency: Running times, memory size, etc.***

- Roles of ***data structure*.**

Tasks

1. CARD
2. BESTP
3. SAILING
4. PKMATCH
5. WEATHER

CARD

1. Problem

2. Algorithm

    I. Direct implementation

    II. Linked list

    III. Tracking the top-of-array

    IV. Tracking the targeted location

3. Remarks.

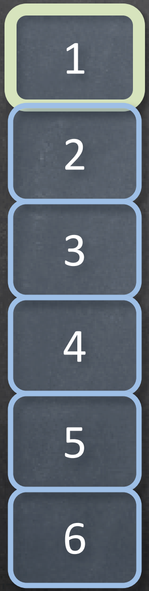
Problem description

Given a sequence of re-arrangements to a stack of *n* cards, and an index *k*, find the *k*-th element after all re-arrangements have been applied.

Problem description

Moves: A B A A

⇧



1

2

3

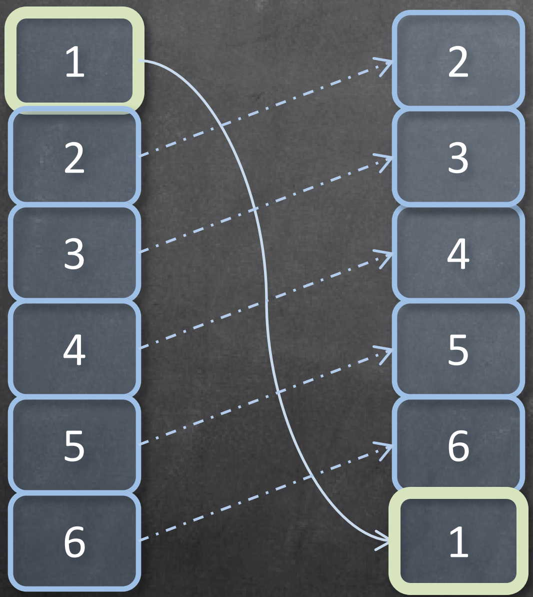
4

5

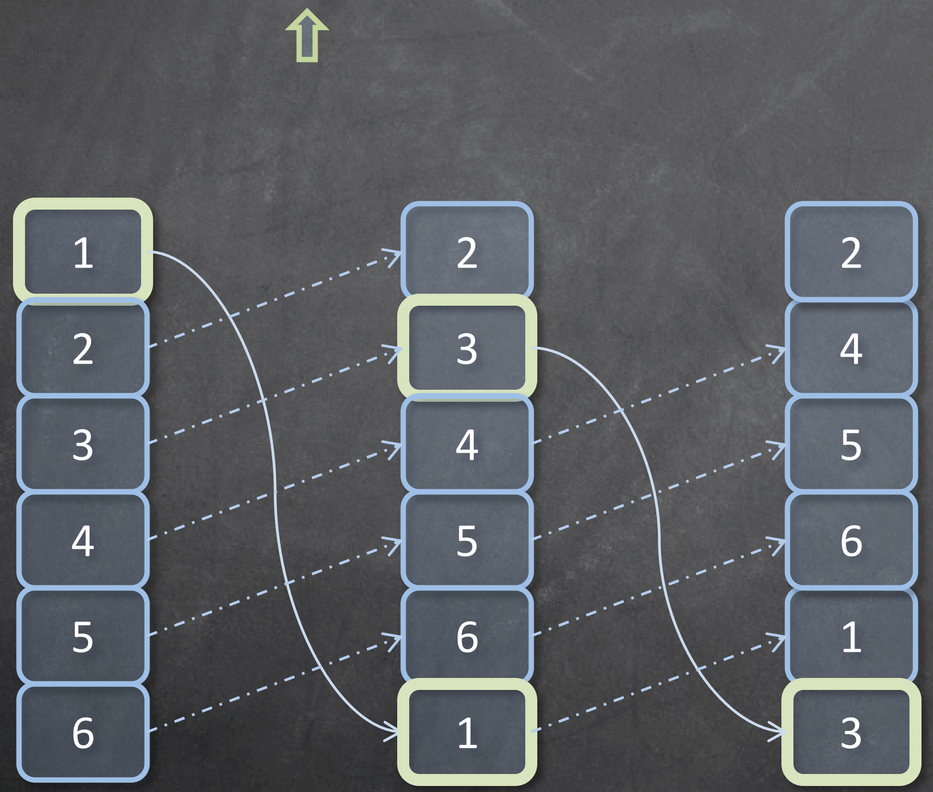
6

Moves: A B A A

⇧

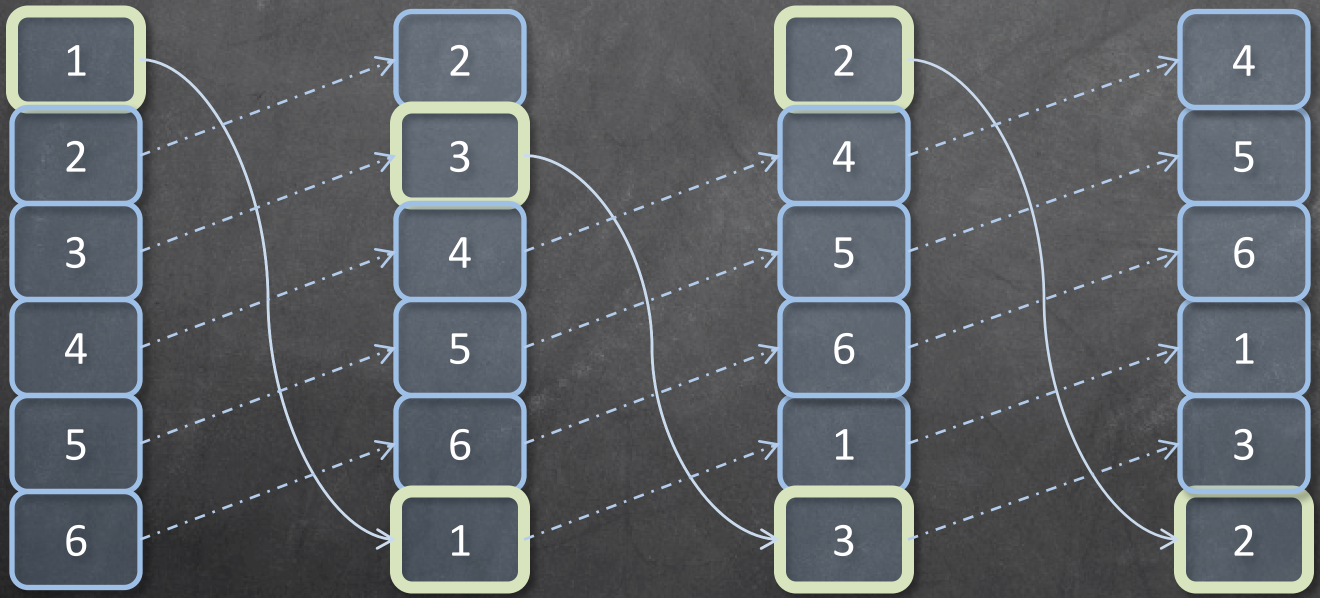


Moves: A B A A



Moves: A B A A

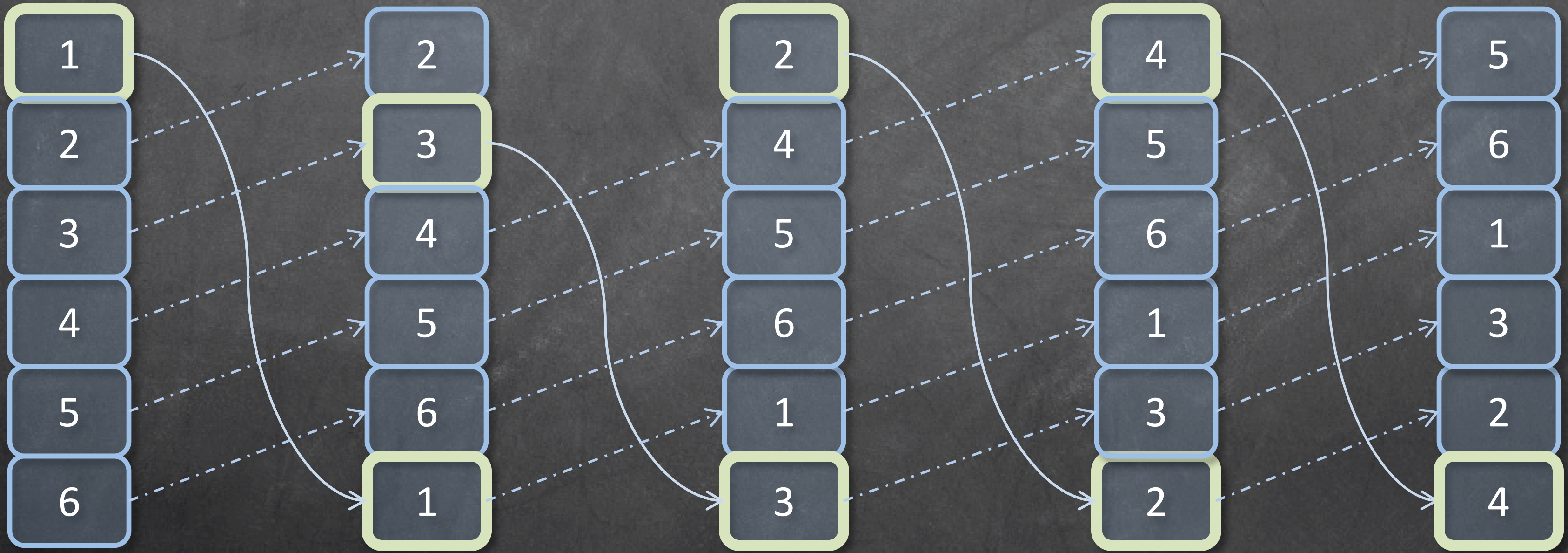
⇧



1	2	2	4
2	3	4	5
3	4	5	6
4	5	6	1
5	6	1	3
6	1	3	2

Moves: A B A A

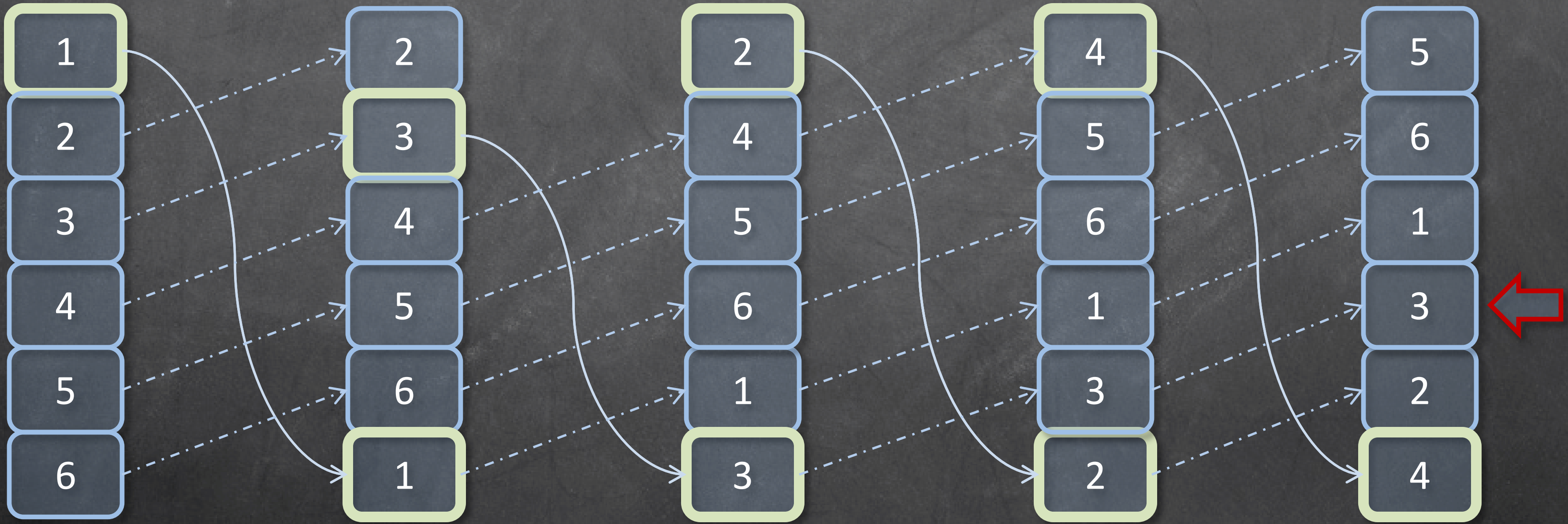
⇧



1	2	2	4	5
2	3	4	5	6
3	4	5	6	1
4	5	6	1	3
5	6	1	3	2
6	1	3	2	4

Moves: A B A A

What is the 3rd card from the bottom?



CARD

1. Problem
2. Algorithm
    I. Direct implementation
    II. Linked list
    III. Tracking the top-of-array
    IV. Tracking the targeted location
3. Remarks.

Algorithms

Four approaches

I. Direct implementation. (slow[*])

II. Using doubly linked list.

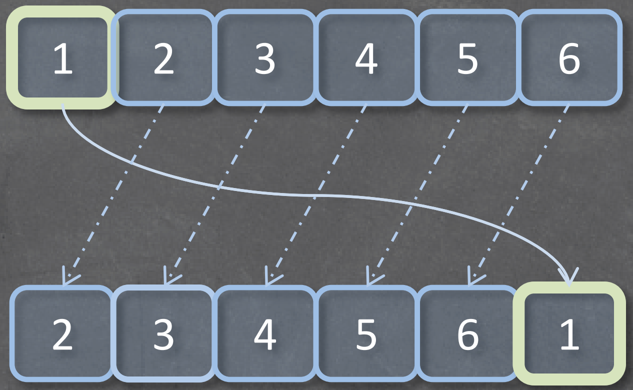
III. Keeping track of the top-of-array + circular buffer.

IV. Follow the targeted location -- backward.

[*] : Able to gain partial marks in this contest.

I. Direct implementation

```
Given an array S  and a move,
If move is “A”,
  begin
          temp ← S[0];
          for i = 1 to n-2,   S[i] ← S[i+1];
          S[n-1] ← temp;
  end
If move is “B”,
  begin
          temp ← S[1];
          for i = 2 to n-2,   S[i] ← S[i+1];
          S[n-1] ← temp;
  end
```

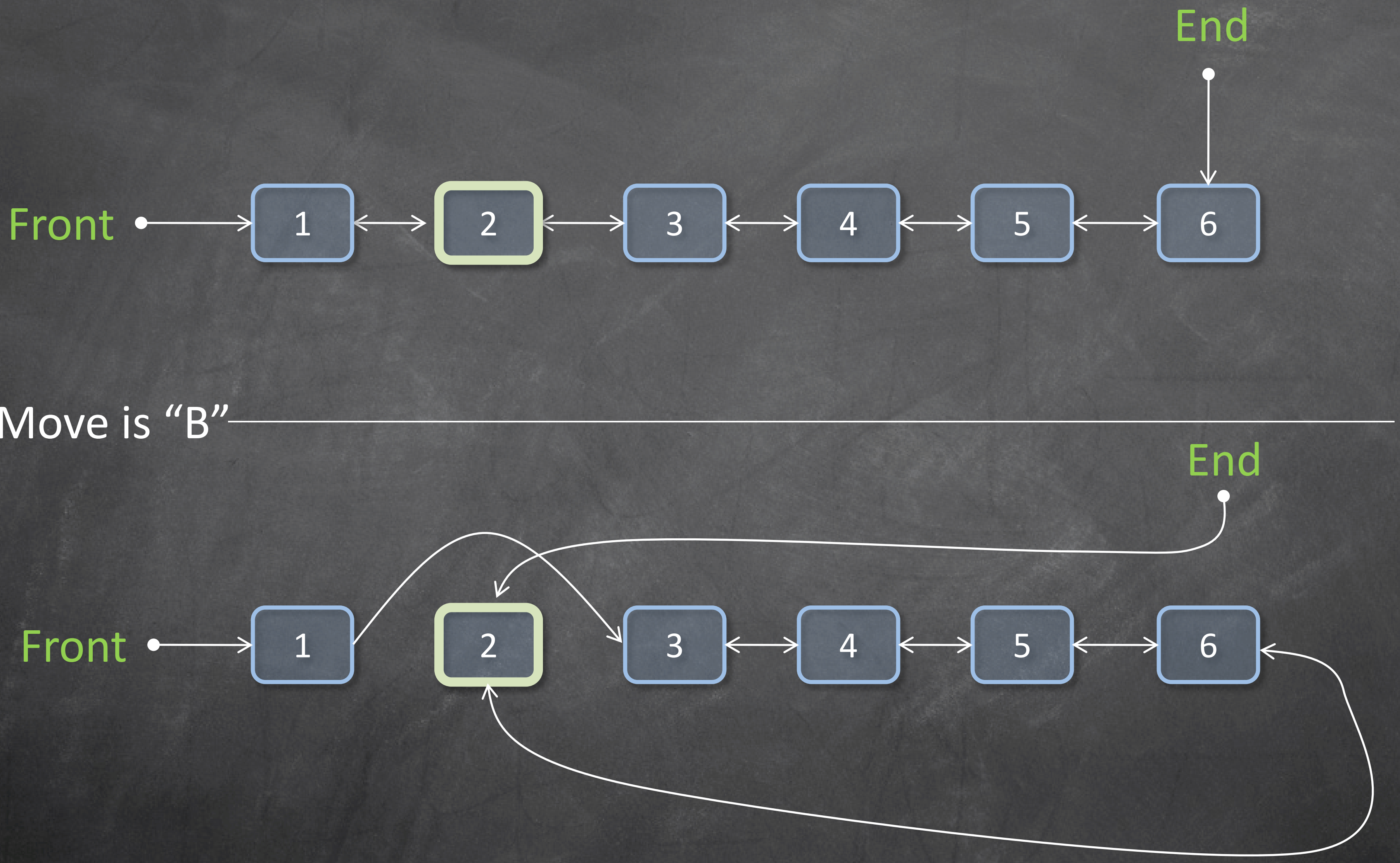


1 2 3 4 5 6

2 3 4 5 6 1

Number of shifts
(i.e. number of times the operation ← being carried out) is at least *m n* where *m* is the number of moves.

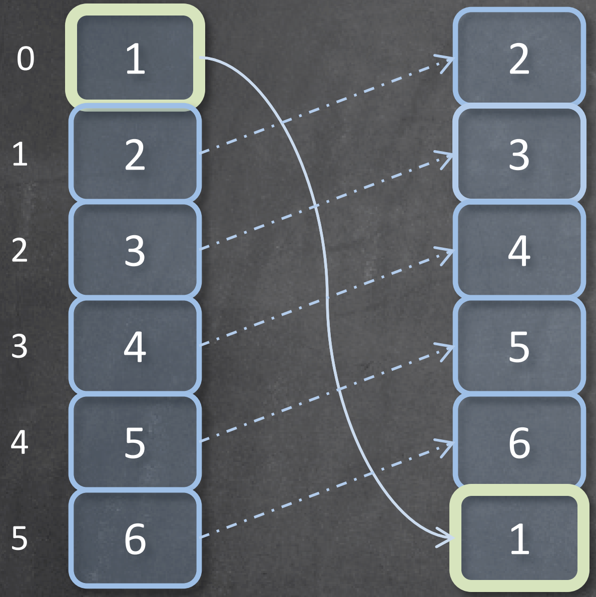
II. Using linked list.

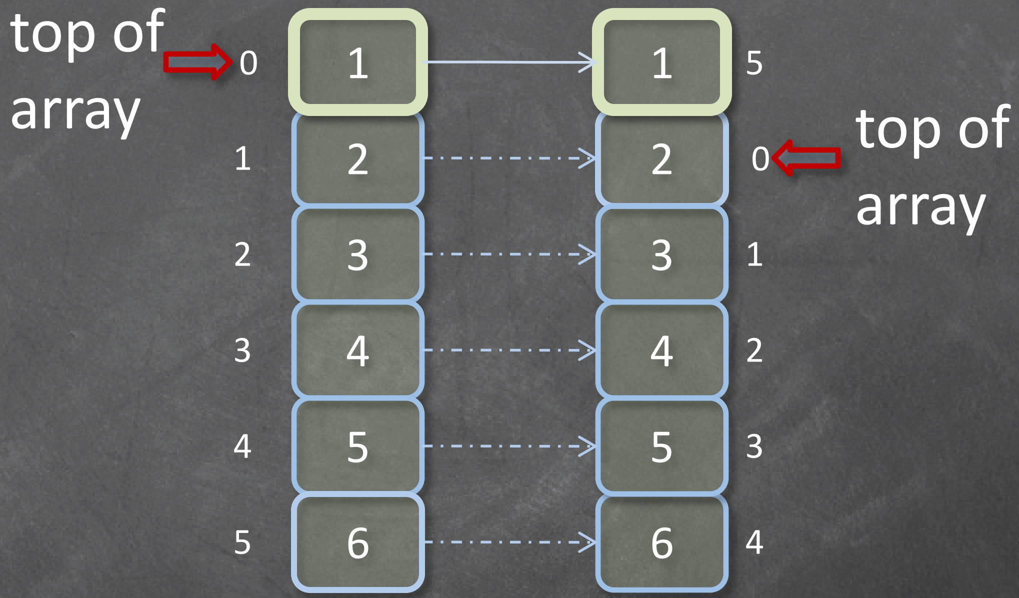


Constant number of operations per move.

III. Tracking the top-of-array

move is “A”

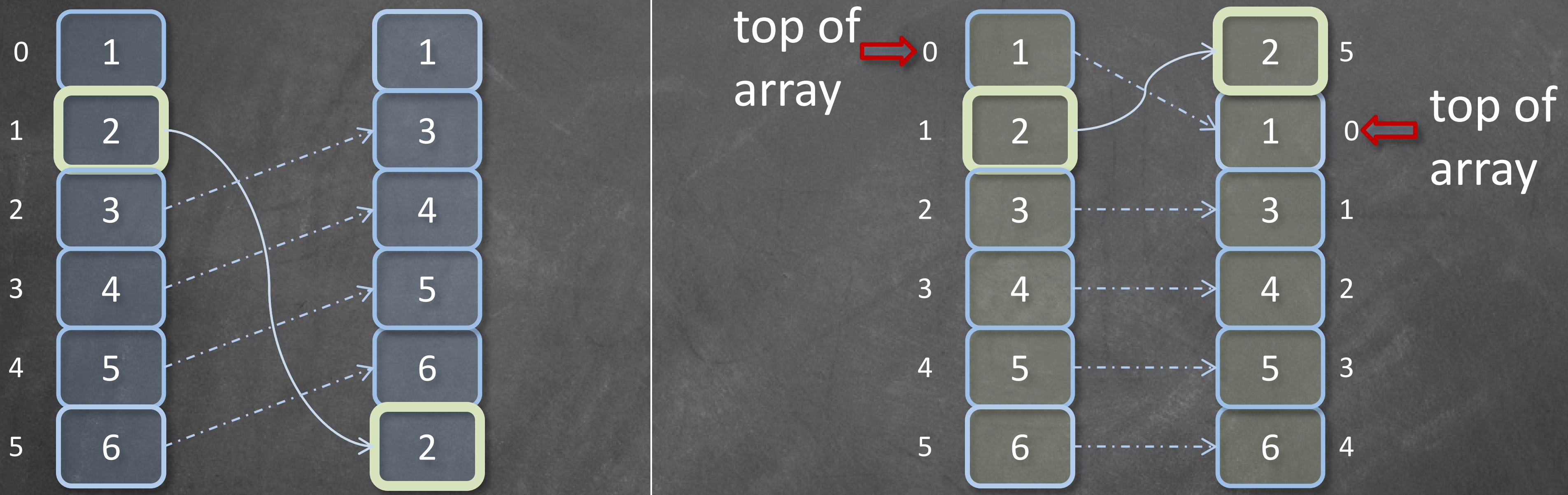




No shifting! Just modify the top-of-array pointer.

Constant number of operations per move.

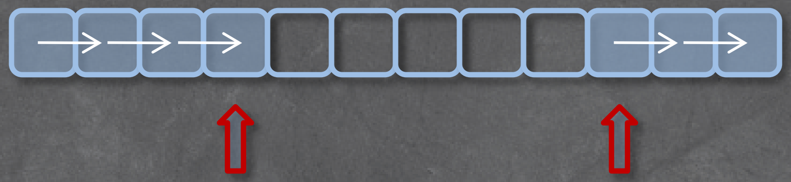
move is “B”



Constant number of operations per move.

Circular buffer

The previous method can be viewed as an implementation of linked-list using circular buffer.



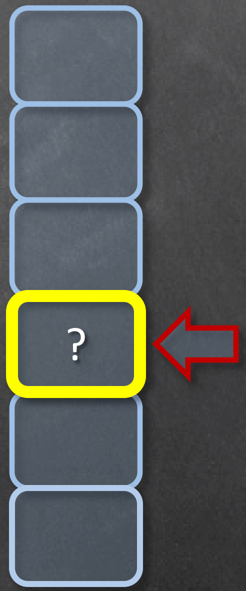
last element in queue

first element in queue

This data-structure supports constant time insertion/removal of elements near the two ends of the queue. However, there are large fraction of waste space. In the task CARD, the size is fixed and hence no issue on waste space.

IV. Follow the targeted location -- backward

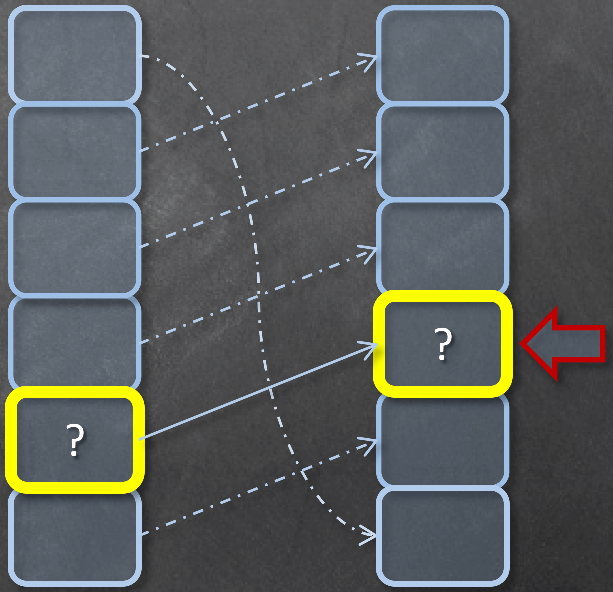
Moves: A B A A



IV. Follow the targeted location -- backward

Moves: A B A A

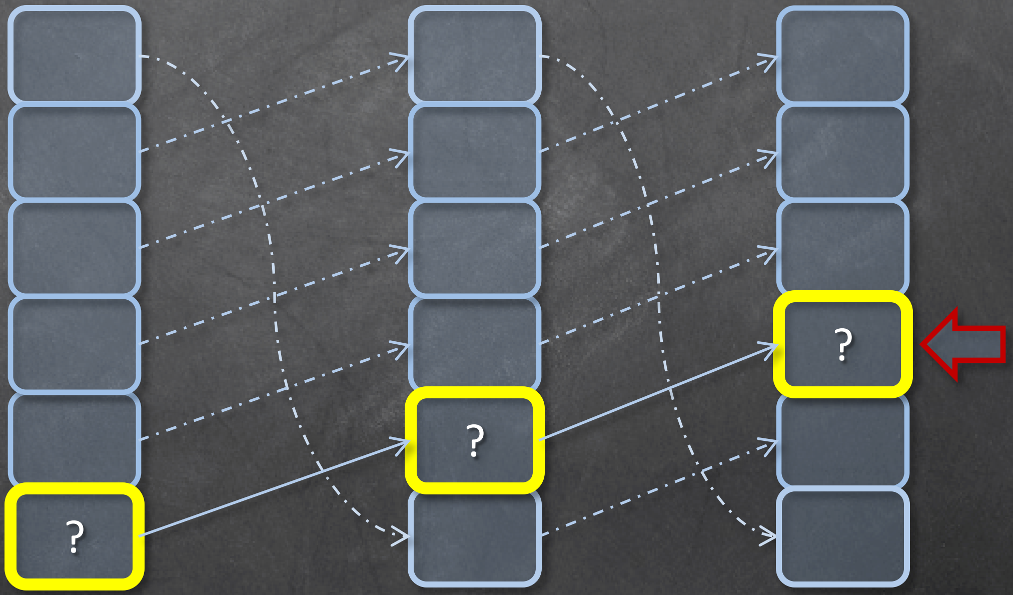
⇧



IV. Follow the targeted location -- backward

Moves: A B A A

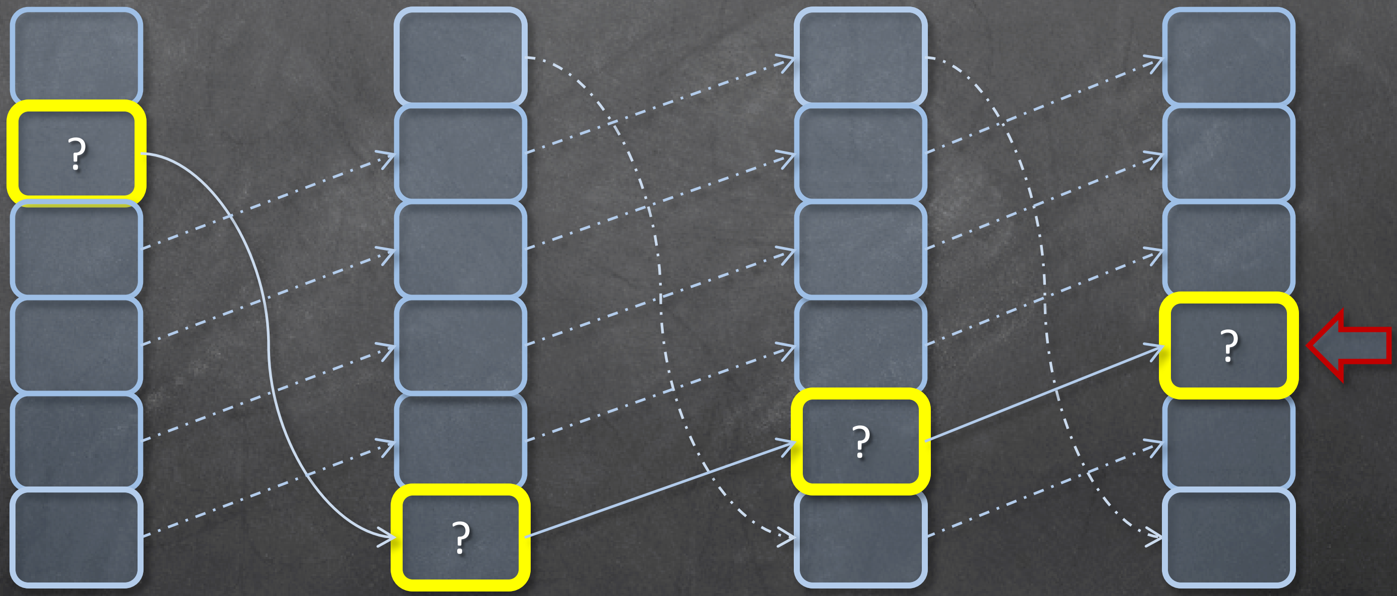
⇧



IV. Follow the targeted location -- backward

Moves: A B A A

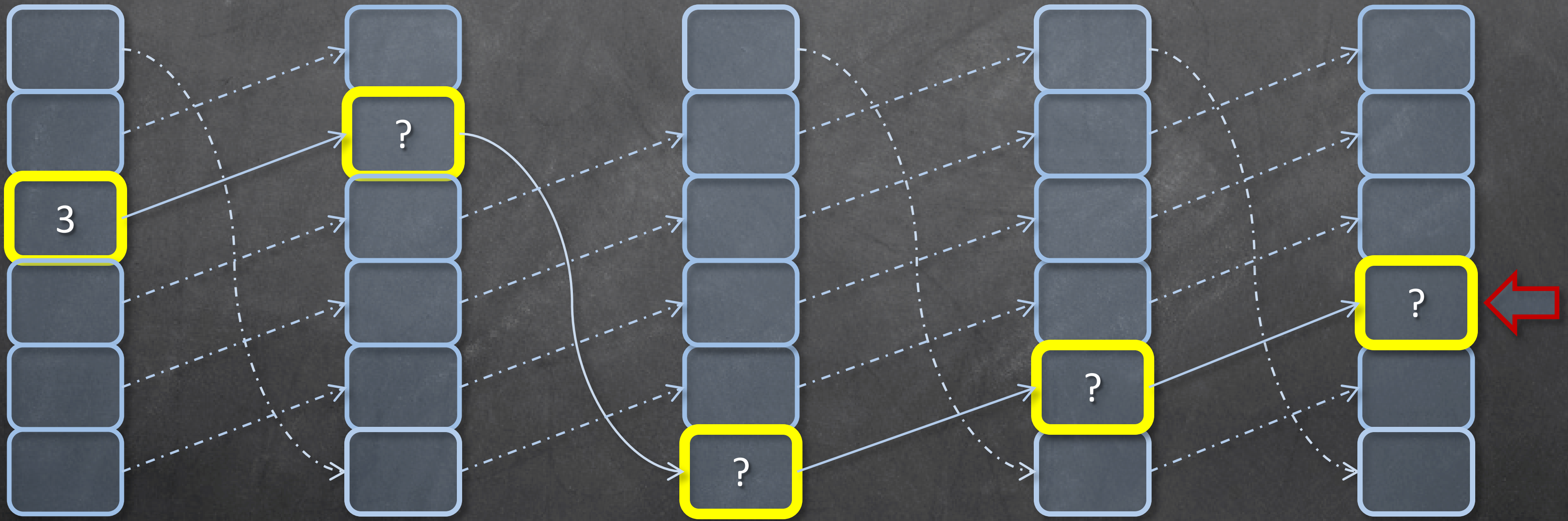
⇧



IV. Follow the targeted location -- backward

Moves: A B A A

⇧



CARD

1. Problem
2. Algorithm
    - I. Direct implementation
    - II. Linked list
    - III. Tracking the top-of-array
    - IV. Tracking the targeted location
3. Remarks.

Remark: Shuffling & Encryption

How good is Alice as a card dealer?

Very bad.

When the number of moves *m* is small, even without knowing the actual moves, from *m*, one can correctly guess the content in the targeted location with high chance.

How many “shuffles” are suffice for 52 cards?

7 “riffle” shuffles[1].

[1] Brad Mann, *How many times should you shuffle a deck of card?* *in* Topics in contemporary probability and its applications, 1995.

A popular encryption algorithm RC4 (Rivest Cipher 4) encrypts message in the following way:

1) It first derives a shuffled array based on the short secret key.

2) Next, the message is encrypted using the shuffled array.

(The progra ask CIPHER, 2009 NOI is RC4).

Adi Shamir, Ronald Rivest, and Leonard Adleman
Recipients of 2002 Turing award

BESTP

1. Problem
2. Solutions
    - I. Exhaustive Search
    - II. Divide-and-conquer
    - III. Scanning from left to right

Problem description

- Given a sequence of *n* integers $b_1, b_2, b_3, \ldots, b_n$, find the maximum value of

$$(b_{i+1} + b_{i+2} + \ldots + b_{j-1} + b_j)$$

among all possible i, j where $i < j$.

2 -1 3 -8

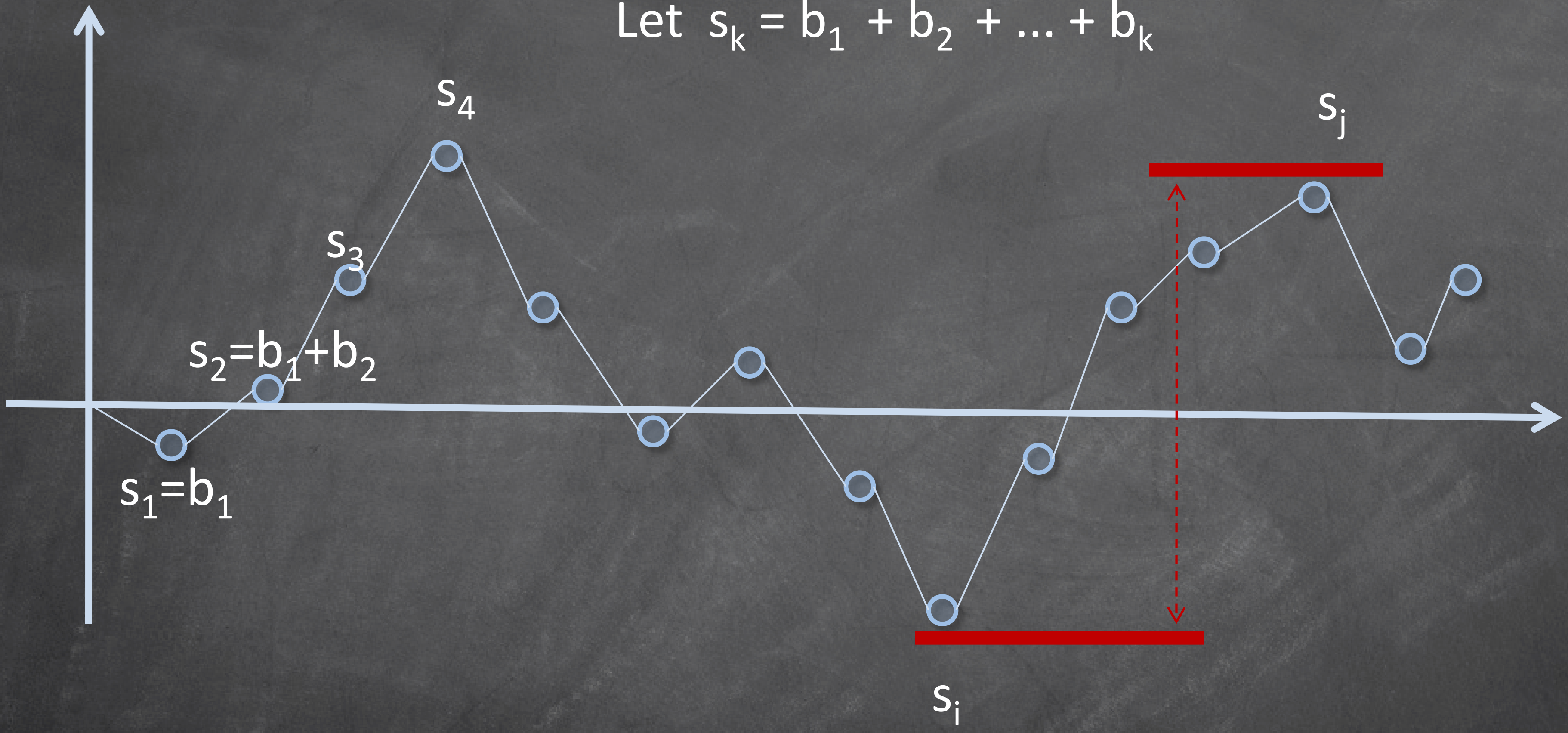
All possible consecutive subsequences and their sum:

2 -1 3 -8	(-4)
2 -1 3	(4)
2 -1	(1)
2	(2)
-1 3 -8	(-6)
-1 3	(2)
-1	(-1)
3 -8	(-6)
3	(2)
-8	(-8)

Max is 4.

view from another perspective.

Let $s_k = b_1 + b_2 + \ldots + b_k$



$$
\begin{aligned}
s_i &= b_1 + b_2 + \ldots + b_i \\
s_j &= b_1 + b_2 + \ldots + b_i + b_{i+1} + \ldots + b_j \\
s_j - s_i &= 0 + b_{i+1} + \ldots + b_j
\end{aligned}
$$

Equivalent problem

Given a sequence $s_0=0$, $s_1, s_2, s_3, \ldots, s_n$, find the maximum value of

$$(s_j - s_i)$$

among all possible ***i, j*** where $0 \leq i < j \leq n$

BESTP

1. Problem
2. Solutions
   - I. Exhaustive Search
   - II. Divide-and-conquer
   - III. Scanning from left to right

Solutions

Three approaches

I. Three for-loops. Two for-loops. (slow*)

II. Divide-and-conquer.

III. Scanning from left to right.

* : Able to gain partial marks in this contest.

I. Three for-loops, two for-loops

Search all possible i, j's.

three loops required

Given a sequence of n integers $b_1, b_2, b_3, \ldots, b_n$, find the maximum value of

$$(b_{i+1} + b_{i+2} + \ldots + b_{j-1} + b_j)$$

among all possible i, j where $i < j$.

Number of additions > $(1/8)\ n^3$

two loops required

Given a sequence $s_0=0, s_1, s_2, s_3, \ldots, s_n$, find the maximum value of

$$(s_j - s_i)$$

among all possible i, j where $0 \leq i < j \leq n$

Number of subtractions > $(1/4)\ n^2$

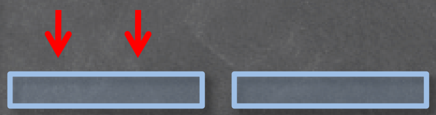
II. Divide-and-conquer

Want to find the *i, j* s.t $s_j - s_i$ is max.

[Divide] Divide the sequence into two halves.

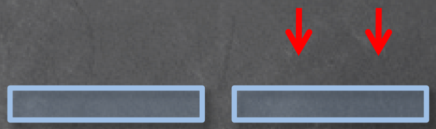
[Conquer] There are 3 steps corresponding to 3 cases:

Both s_i , s_j are in the left halve.



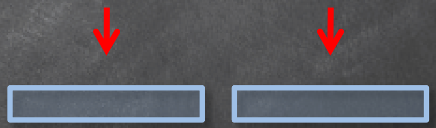
Recursively find them in the left.

Both s_i , s_j are in the right halve.



Recursively find them in the left.

Otherwise



s_i must be the min in the left. Find it.

s_j must be the max in the right. Find it.

[Combine] Choose the largest among the above 3 results.

Running time: O (n log n).

Can be improved to O(n) with some modifications.

III. Scanning from left to right

Main Idea:

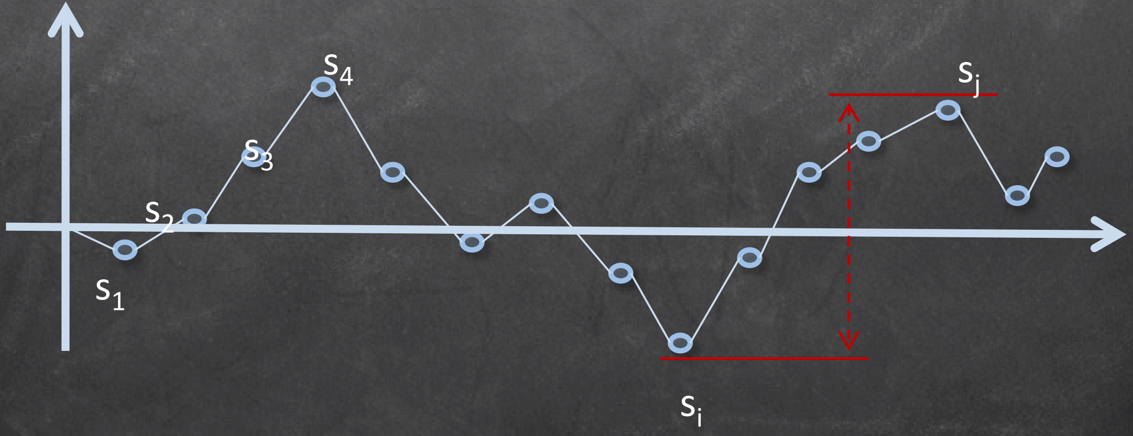
Let ($i_{,} j$) be the indices corresponds to the max. Then,

1. s_i must be the minimum among { $s_{1,}$ $s_{2,}$ ...., $s_{j\text{-}1}$ }. For each *j*, let M(*j*) be the *i* s.t. s_i is min among { $s_{1,}$ $s_{2,}$ ...., $s_{j\text{-}1}$ }.

   So, we can exhaustive try all possible *j*, and for each *j*, find M(*j*). Nevertheless, the number of operations required is still in $\Omega(n^2)$

2. M(*j* -1) ≤ M(*j*).

   So, instead of computing M(*j*) for each *j*, we can re-use previous result of M(*j* -1).

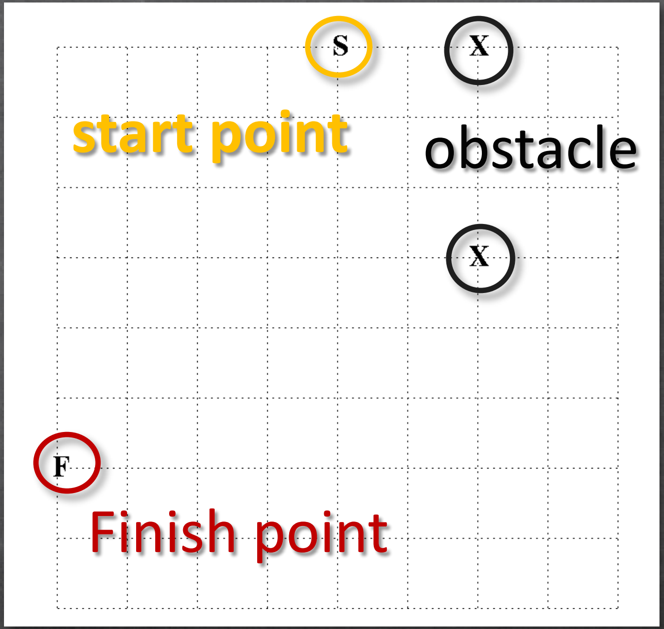


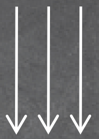
SAILING

1. Problem
2. Solution: Dijkstra's algorithm
   - I. with linear search
   - II. with binary search tree
   - III. with priority queue

Problem description

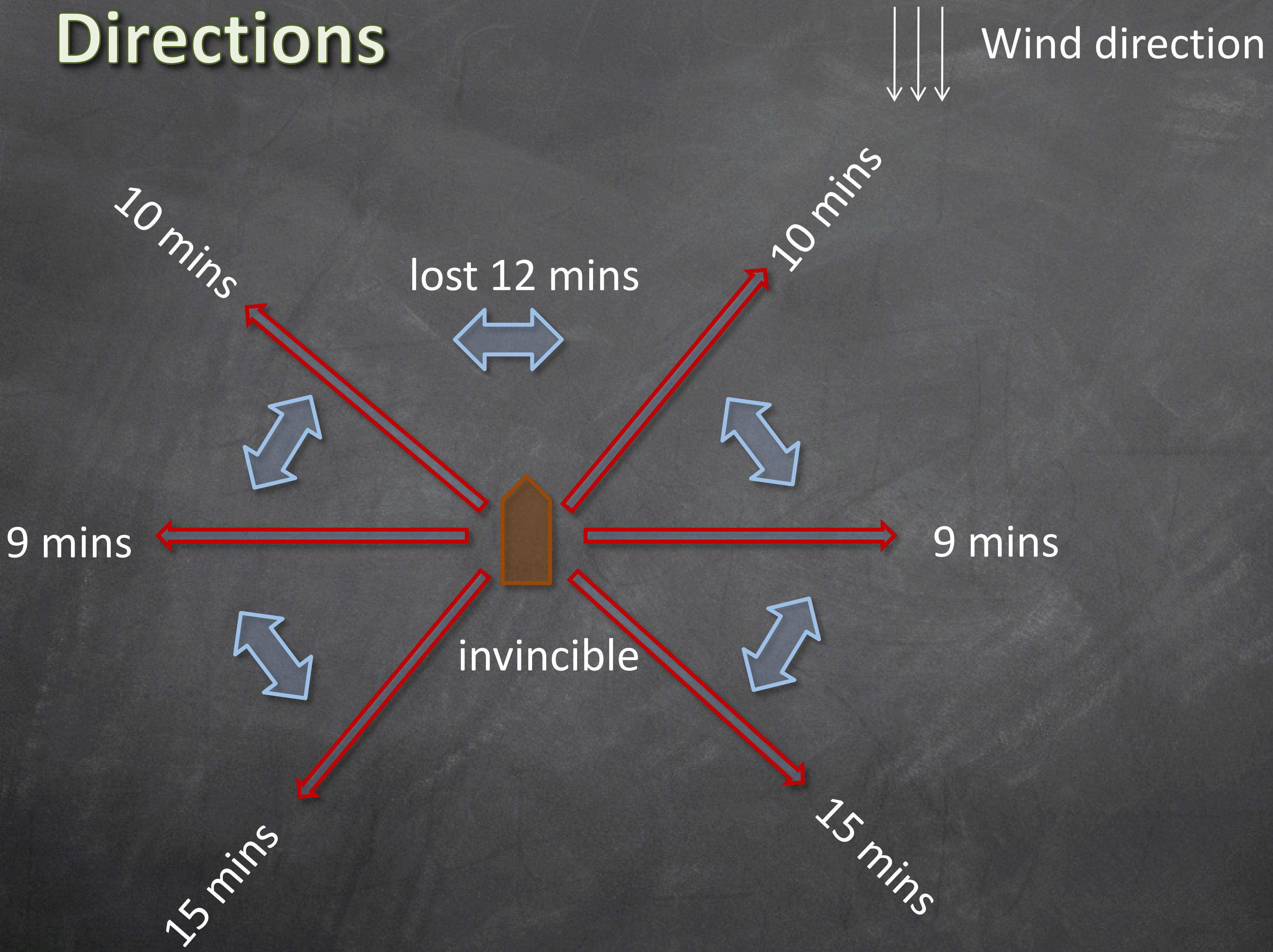
Given two grid points S and F and the locations of the obstacles, find the path for the sailing boat Invincible to sail from S to F in the shortest time.

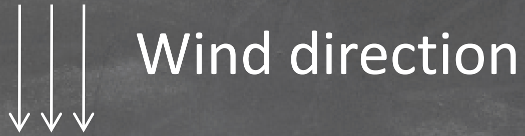


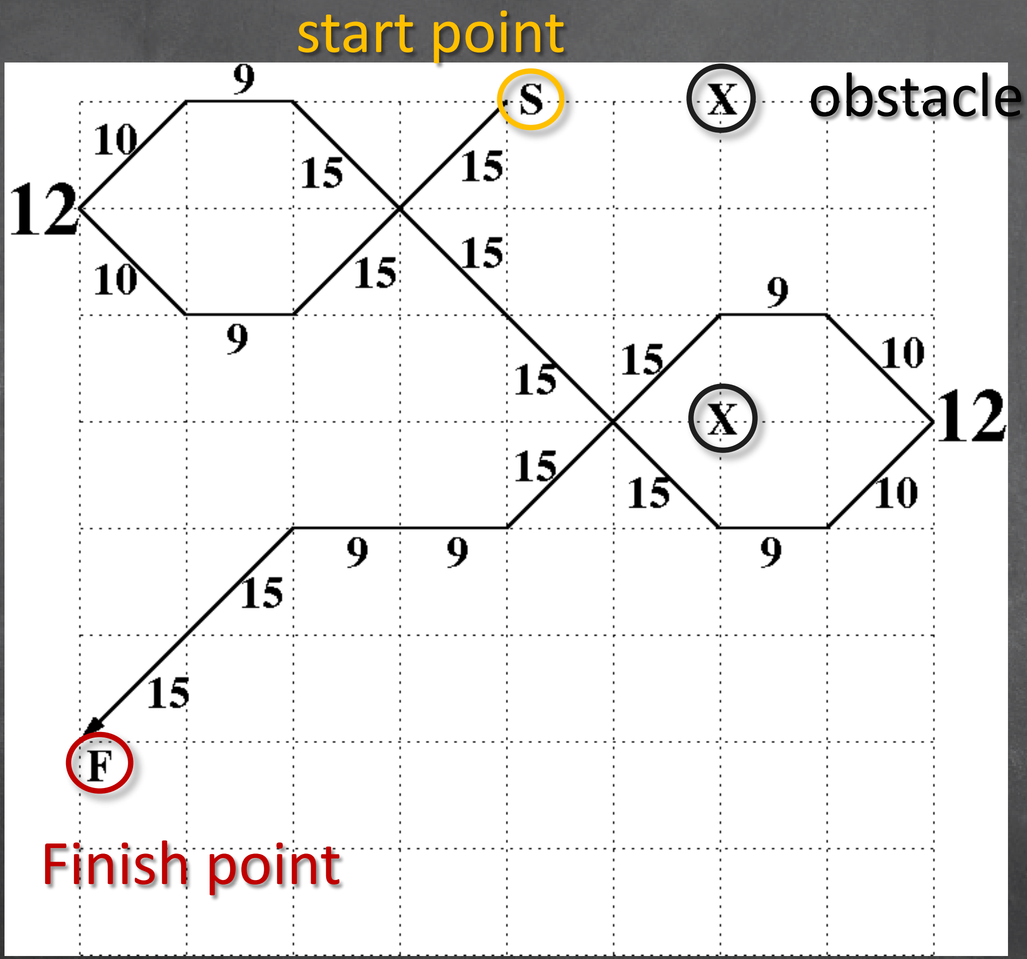


The constrains on the sailing directions w.r.t. the wind direction, the time taken in changing directions and the sailing speed are stated in the task description.

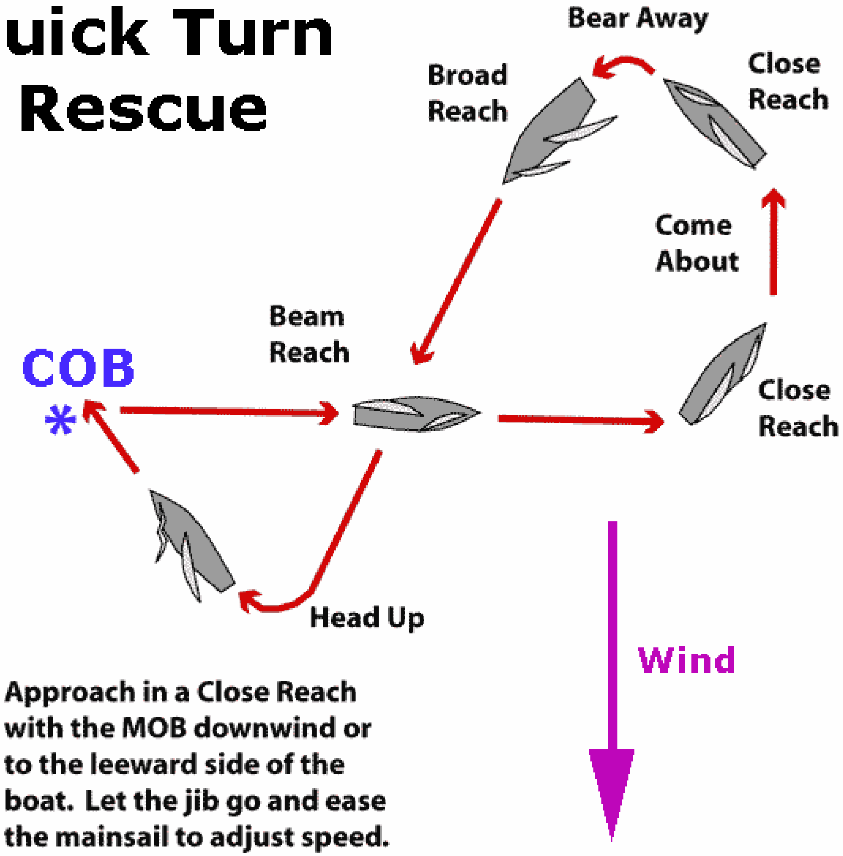
Directions


Wind direction
10 mins
10 mins
lost 12 mins
9 mins
9 mins
invincible
15 mins
15 mins


Wind direction


start point
S
X
obstacle
9
10
12
10
15
15
15
15
9
9
15
10
12
X
15
15
15
10
9
9
9
15
15
F
Finish point

Quick Turn Rescue



Approach in a Close Reach with the MOB downwind or to the leeward side of the boat. Let the jib go and ease the mainsail to adjust speed.

Image from http://www.sailingusa.info/

SAILING

1. Problem
2. Solution: Dijkstra’s algorithm
   1. I. with linear search
   2. II. with binary search tree
   3. III. with priority queue

Formulation

Possible states of invincible:

- The location of Invincible, and
- the direction of Invincible.

Transition Time from state S to state D:

- Distance of the two locations, and
- time lost in changing direction.

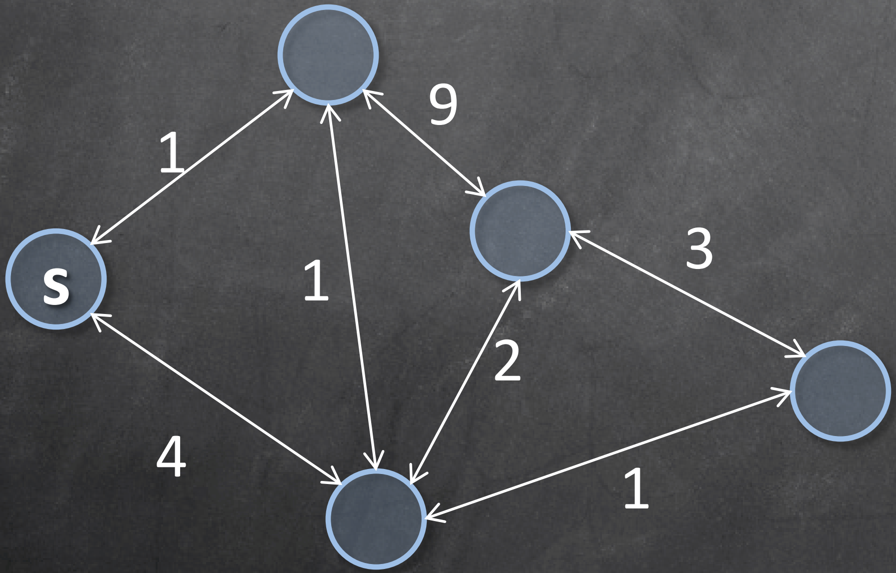
Weighted Graph:

State ↔ Vertex

Transition time ↔ Edge wt.

Dijkstra's shortest path algorithm

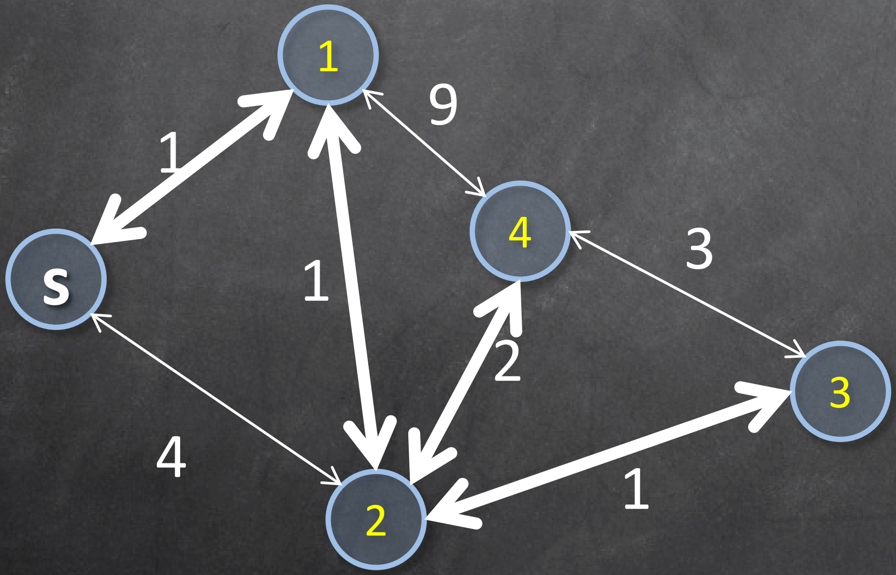
Given a weighted graph **G** and a vertex **s**, find the shortest path from **s** to each vertex **w** in **G**.



E.W. Dijkstra, 1930 – 2002, recipient of 1972 Turing Award

Dijkstra's shortest path algorithm

Given a weighted graph **G** and a vertex **s**, find the shortest path from **s** to each vertex **w** in **G**.



E.W. Dijkstra, 1930 – 2002,
recipient of 1972 Turing Award

Detailed description of Dijkstra’s algorithm can be found in most algorithm textbooks.

```
Let T be set of all vertices.
For each v in T,  Dist[v] $\leftarrow \infty$
Dist[ s ] $\leftarrow$ 0;
    Repeat
        Find and remove the v from T  where Dist[v] is smallest.
        For each w adjacent  to v,
            Dist[w] $\leftarrow$ min { Dist[w,   Dist[v] + weight of the edge (v,w) }
    Until T is empty
//   Dist now keeps the length of the shortest path to each vertex
```

Detailed description of Dijkstra’s algorithm can be found in most algorithm textbooks.

Note that it requires a data-structure that support:

(1) Extract the element with the smallest value.

(2) Given an element, reduce its value.

Data-structures that supports the above two operations are known as priority queue.

```
Let T be set of all vertices.
For each v in T,  Dist[v] ← ∞
Dist[ s ] ← 0;
    Repeat
         Find and remove the v from T  where Dist[v] is smallest.
         For each w adjacent  to v,
                Dist[w] ← min { Dist[w,   Dist[v] + weight of the edge (v,w) }
    Until T is empty
//   Dist now keeps the length of the shortest path to each vertex
```

Data Structures

Shortest path algorithm with one of the following as priority queue:

I. Array. Use linear search to find min (slow*)

Extract min in Θ(n), reduce key in Θ(1)

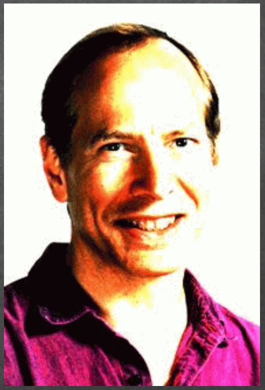
II. Binary search tree, heap (fast)

Extract min in Θ(log n), reduce key in Θ(log n)

III. Fibonacci heap (very fast)

Introduced by Fredman and Tarjan in 1998.

Extract min in Θ(log n), reduce key in Θ(1) (amortized time)



Robert Tarjan,
recipient of 1986 Turing Award,
together with John Hopcroft.

* : Able to gain partial marks in this contest.

PKMATCH

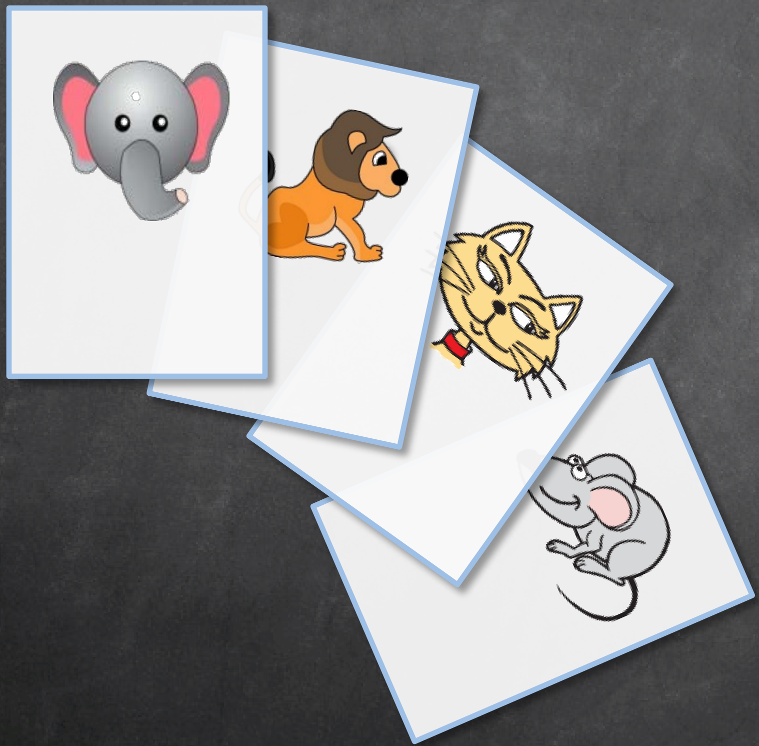
1. Problem
2. Solution
    I. Random player
    II. Prioritize the monsters.
    III. Dynamically find the strongest
    IV. Optimal.

Problem description

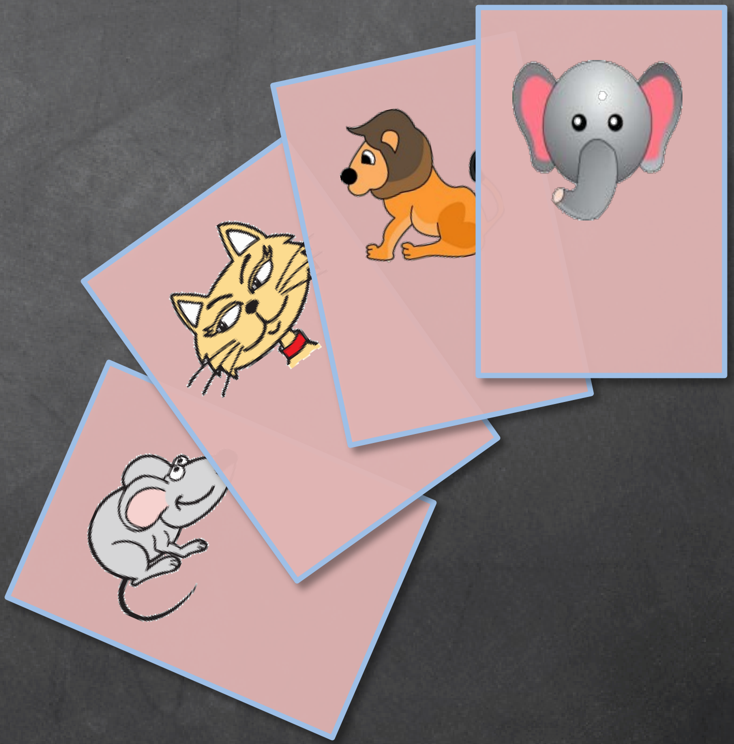
I’m going to play a multiple-rounds game with an opponent and I know the opponent’s algorithm. What is my optimal strategy?

A multiple-rounds Card Game

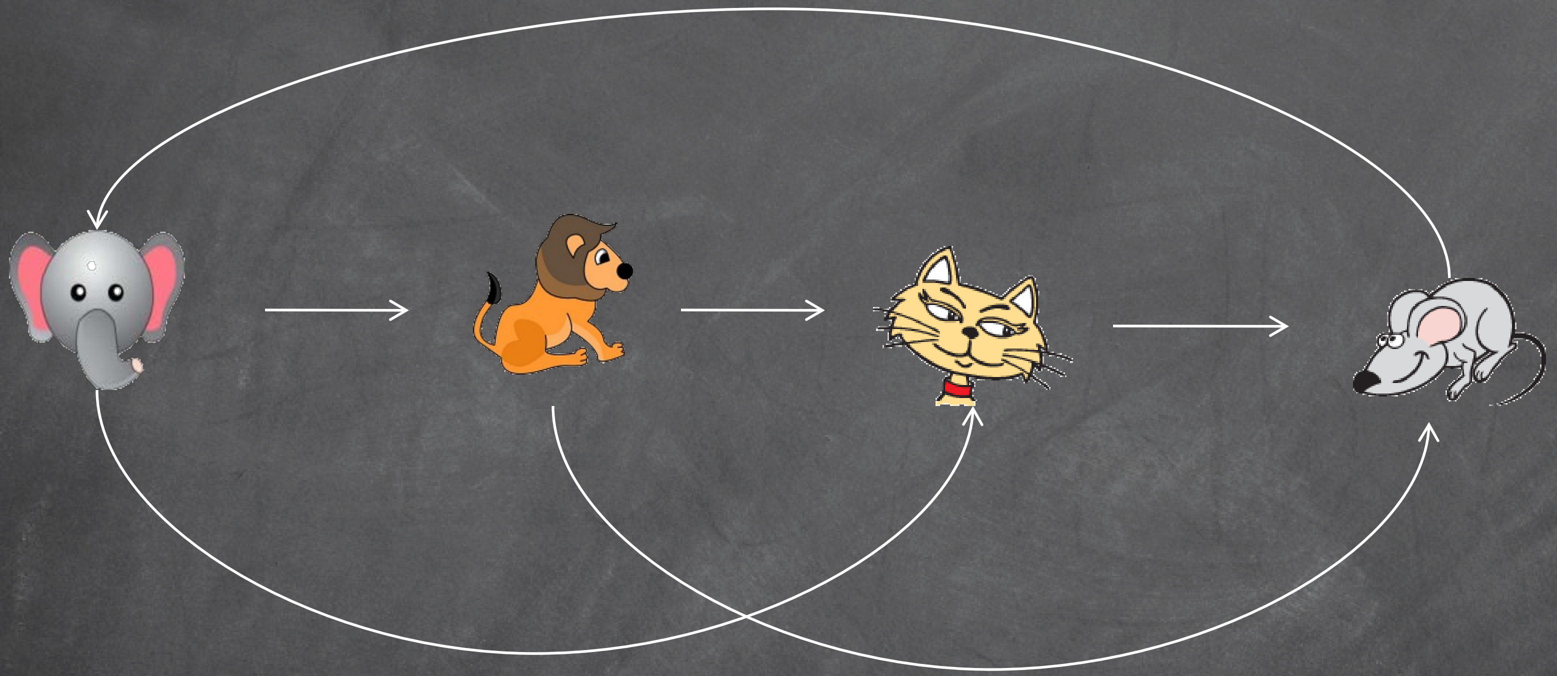
Me



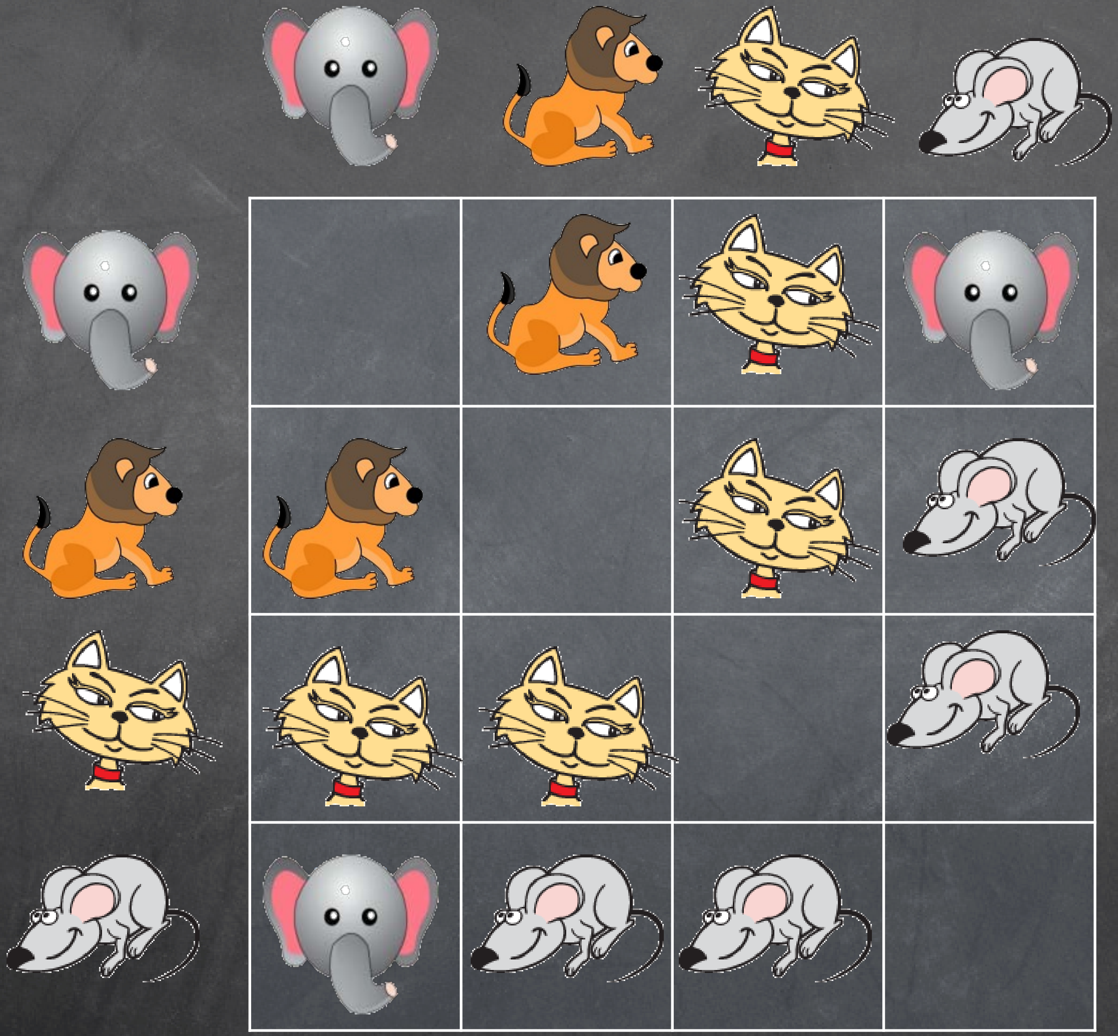
Randy



Knocked-out rule



Strength table



	🐘	🦁	🐱	🐭
🐘		🦁	🐱	🐘
🦁	🦁		🐱	🐭
🐱	🐱	🐱		🐭
🐭	🐘	🐭	🐭	

Strength table

	0	1	2	3
0	0	1	2	0
1	1	1	2	3
2	2	2	2	3
3	0	3	3	3

In this task, the strength table is randomly generated in each game.

PKMATCH

1. Problem
2. Solution
    - I. Random player
    - II. Prioritize the monsters
    - III. Dynamically find the strongest
    - IV. Optimal

Strategies

I. Random *.

II. Before the game starts, derive a sorted list of monsters, starting from the “strongest” down to the “weakest”. In every round, always play the strongest monster among the remaining *.

III. In every round, find the “strongest” monster *.

IV. An optimal strategy.

* : Able to gain partial marks in this contest.

I. Random

A random player is expected to win around 60% of the games.

II. Prioritize the monsters

Before the game starts, from the table, decide an ordering of the monsters, starting from the strongest. In each round, always plays the strongest monster in the remaining cards.

	🐘	🦁	🐯	🐶	🐱	🐭	
🐘	0	1	2	3	4	0	→ k-o 5 monsters
🦁	1	1	2	3	4	5	→ k-o 5 monsters
🐯	2	2	2	3	4	5	→ k-o 4 monsters
🐶	3	3	3	3	4	5	→ k-o 3 monsters
🐱	4	4	4	4	4	5	→ k-o 2 monsters
🐭	0	5	5	5	5	5	→ k-o 2 monsters

III. Dynamically choose the strongest

In each round, decides which is the strongest among the remaining cards.

Randy

Me

🟥	1	2	3	🟥	0	→ k-o 3 monsters
🟥	1	2	3	🟥	5	→ k-o 4 monsters
🟥	2	2	3	🟥	5	→ k-o 2 monsters
🟥	🟥	🟥	🟥	🟥	🟥	
🟥	4	4	4	🟥	5	→ k-o 2 monsters
🟥	5	5	5	🟥	5	→ k-o 1 monsters

IV. Optimal

Probability (I win if I play 🐘 In the first round)

= 0.25 × Randy plays 🐘 × (chances that I win in the resulting configuration.)

+ 0.25 × Randy plays 🦁 × (chances that I win in the resulting configuration.)

+ 0.25 × Randy plays 🐱 × (chances that I win in the resulting configuration.)

+ 0.25 × Randy plays 🐭 × (chances that I win in the resulting configuration.)

Repeat the above computation for each card.

Play the card with the highest probability of wining.

Probability (I win) =
max { Probability (I win if I play 🐘 In the first round),
Probability (I win if I play 🦁 In the first round),
Probability (I win if I play 🐱 In the first round),
Probability (I win if I play 🐭 In the first round) }

- For a “randomly” chosen strength table, the probability of winning is approx 77%
- For this table[*],

	🐘	🦁	🐯	🐶	🐱	🐭
🐘	0	1	2	3	4	0
🦁	1	1	2	3	4	5
🐯	2	2	2	3	4	5
🐶	3	3	3	3	4	5
🐱	4	4	4	4	4	5
🐭	0	5	5	5	5	5

the probability of winning is 79.1% the optimal first move is 4.

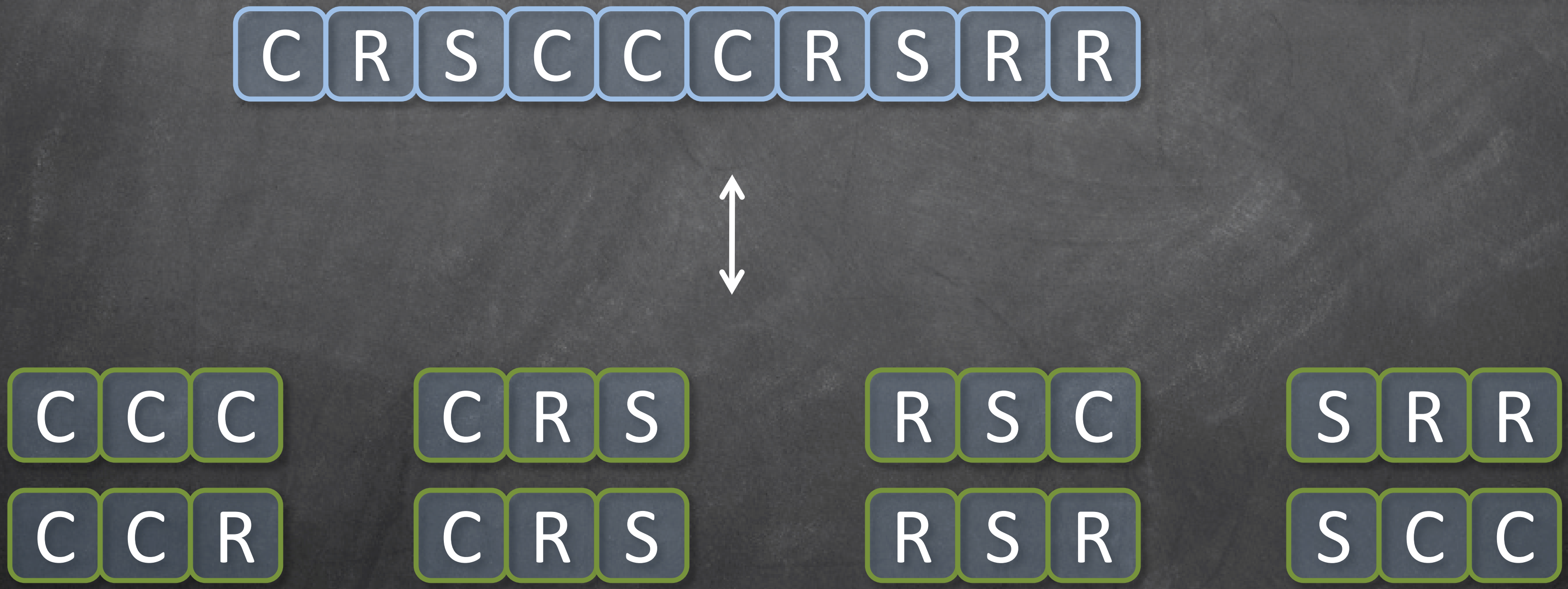
[*] : Note that this table is slightly different from example given in the task description.

WEATHER

1. Problem
2. Solution
3. Remark: DNA sequencing

Problem

- Given the (multi)-set of all length-*d* substrings of a string S, find the first and last character in S.



C R S C C C R S R R

C C C C R S R S C S R R

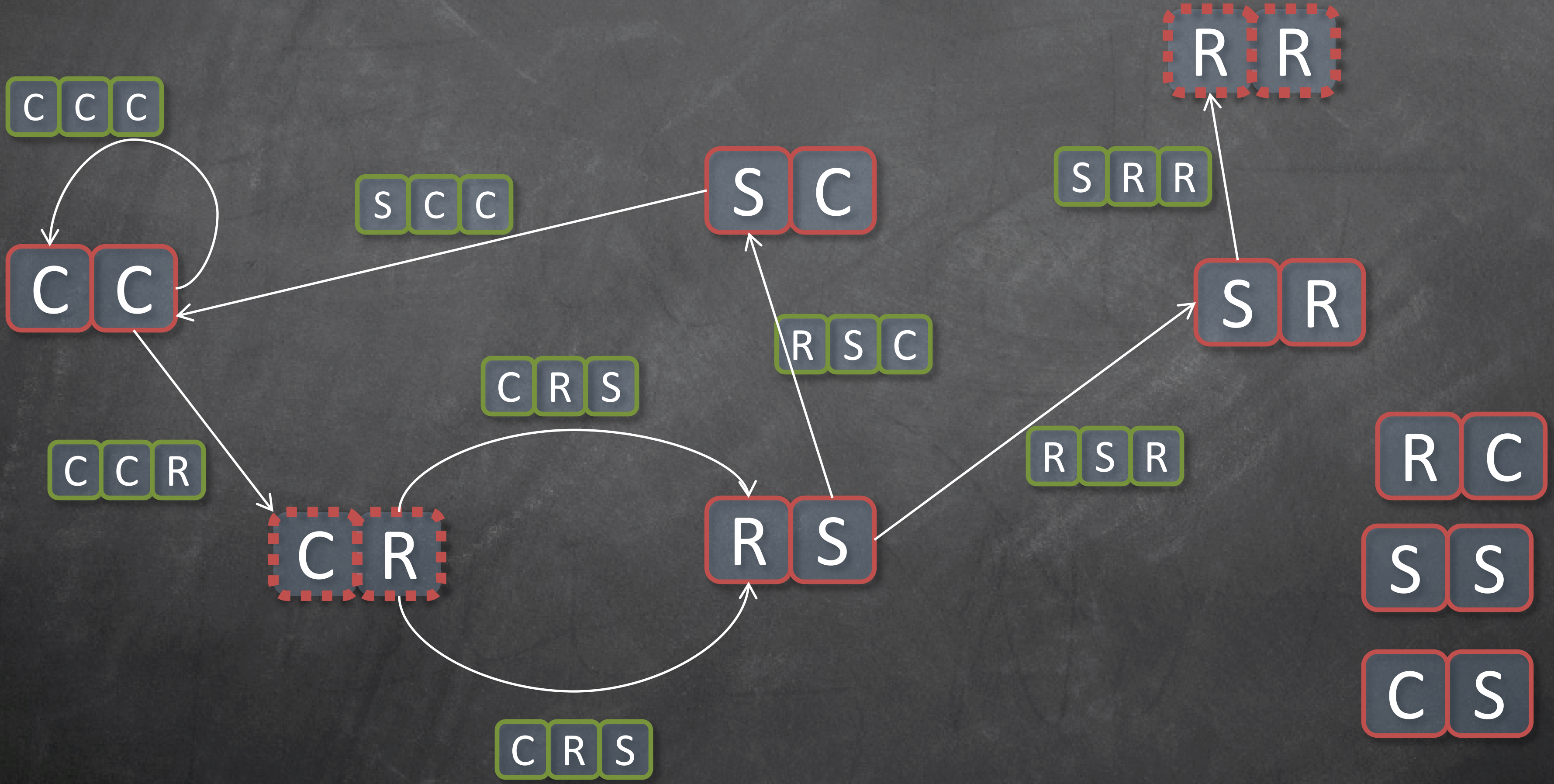
C C R C R S R S R S C C

WEATHER

1. Problem
2. Solution
3. Remark: DNA sequencing

Main Idea

Following the hint given in the task description:



Observations:

- The first node: # of edges pointing to it, is less than # of arrows coming out.
- The last node: # of edges pointing to it, is more than # of arrows coming out.

Programming:

- For large *d*, say d = 10, the number of nodes is $(26)^{10}$. An algorithm that enumerates all the nodes is not feasible*.
- It is not necessary to explicitly construct the graph.

* : Able to gain partial marks in this contest.

WEATHER

1. Problem
2. Solution
3. Remark: DNA sequencing

Remark

- Although this task can be easily solved, there are many variants of this problems that can be shown to be **NP**-hard* .

$$\mathbf{P} \underset{?}{=} \mathbf{NP}$$

- Cook formalized the notion of NP-completeness. He received Turing Award in 1982.

Stephen A. Cook

*: The original slides shown during the presentation mistakenly indicated that finding the string S is NP-hard. There is a polynomial time algorithm to find the S.

Remark

This task belongs to a class of “sequencing problems”. Typically, the input of a sequencing problem consists of a set C of short fragments/substrings from a long string. The goal is to find a long string S that is consistent with C.

Applications: DNA sequencing.