NOI 2012 – solutions to contest tasks.

School of Computing
National University of Singapore

Chang Ee-Chien

10 March 2012

Scientific Committee

Chang Ee-Chien (chair)

Steven Halim

Martin Henz

Lee Hwee Kuan

Frank Stephan

Sung Wing Kin, Ken

TASK 1: MODSUM
TASK 2: PANCAKE
TASK 3: FORENSIC
TASK 4: WALKING

Task 1: MODSUM

Problem

Given n, v_1, w_1, v_2, w_2, … , v_n, w_n, compute

```
t = 0;

for x1 = v1 to w1
        for x2 = v2 to w1
                …
                for xn = vn to wn
                        t = t + f ( x1, x2, …, xn )

Output t.
```

Problem

More precisely, output

$$\sum_{x_1=v_1}^{w_1} \sum_{x_2=v_2}^{w_2} \cdots \sum_{x_n=v_n}^{w_n} f(x_1, \ldots, x_n)$$

Partial solution

```
/*  store the input into the array V and W */

for i1= V[1]  to W[1]  do
   for i2= V[2]  to W[2]  do
        for i3= V[3]  to W[3] do
            for i4= V[4]  to W[4]  do
                for i5= V[5]  to W[5]  do
                    for i6= V[6]  to W[6] do
                        t=t + f( );
```

Able to solve input instance 1.

Solution 1: Recursion

Let S (i, $\langle x_1, x_2, \ldots, x_{i-1} \rangle$) be the output of

```
t = 0;
for  x_i = v_i  to w_i
        for  x_{i+1} = v_{i+1}  to w_{i+1}
                ...
                for  x_n = v_n  to w_n
                        t = t + f ( x_1,..., x_{i-1} , x_i, ..., x_n )
Output t.
```

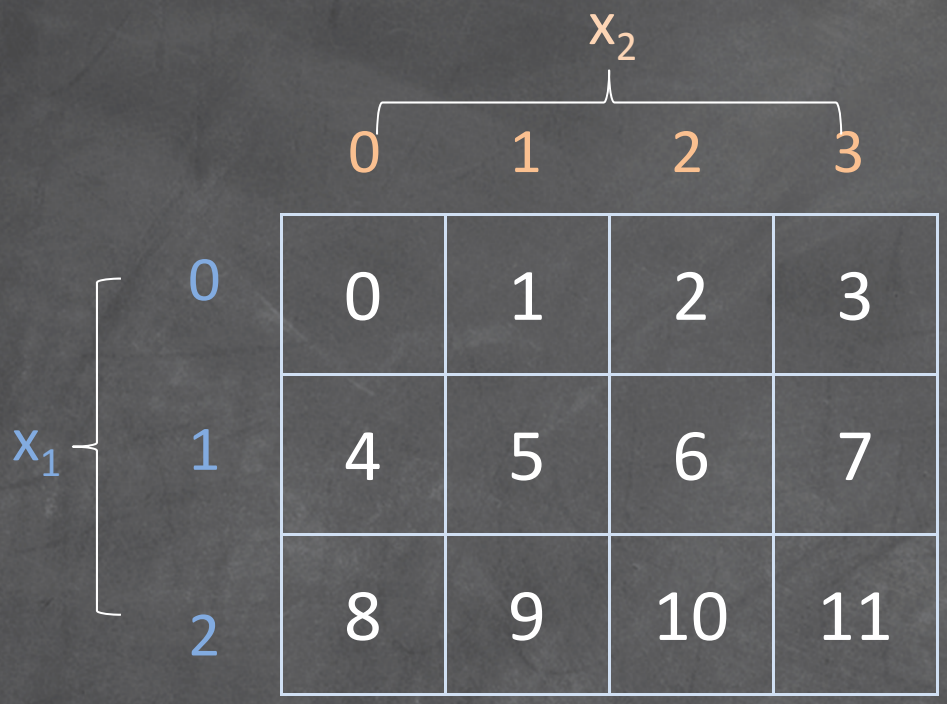
We want to compute S(1, $\langle\ \rangle$).

Note that S(i , 〈 $x_1, x_2, \ldots, x_{i-1}$ 〉) can be defined recursively.

S(i , 〈 $x_1, x_2, \ldots, x_{i-1}$ 〉)

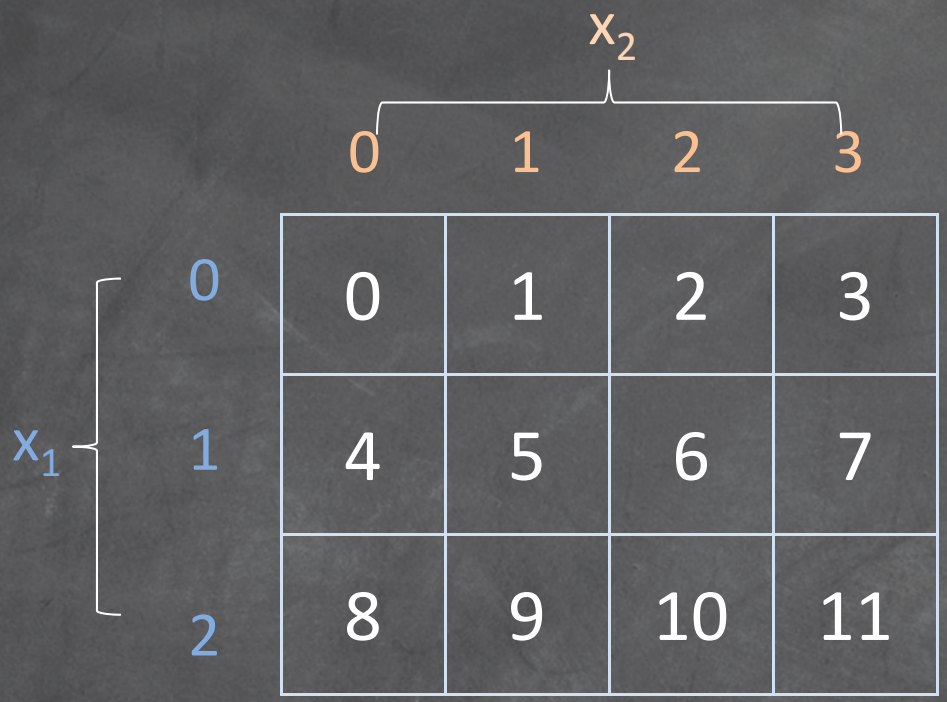
```
t = 0;
for x = v[ i ] to w[ i ]
        t = t + S( i +1, 〈 x1, x2, …, xi-1 , x 〉 )
Output t.
```

Solution 2: enumerating the entries




```
t=0;
for i=0  to 11 do
    t=t+  f(  index_x1 ( i ),    index_x2 ( i )  );
```

Solution 2: enumerating the entries



We want to map the co-ordinate (x_1, x_2) to a one-dimensional point i

$$i = 4 x_1 + x_2$$

The inverse mapping:

index_x1 (i) = floor(i / 4)

index_x2 (i) = i modulo 4

In general for n-dimensional array

$$i = x_n \left(\prod_{j=1}^{n-1} L_j \right) + x_{n-1} \left(\prod_{j=1}^{n-2} L_j \right) + \cdots + x_4 L_3 L_2 L_1 + x_3 L_2 L_1 + x_2 L_1 + x_1$$

$$\begin{aligned}
x_1 &= i \bmod L_1 \\
x_2 &= \left\lfloor \frac{i \bmod L_2 L_1}{L_1} \right\rfloor \\
x_3 &= \left\lfloor \frac{i \bmod L_3 L_2 L_1}{L_2 L_1} \right\rfloor \\
x_j &= \left\lfloor \frac{i \bmod \prod_{k=1}^{j} L_k}{\prod_{k=1}^{j-1} L_k} \right\rfloor
\end{aligned}$$

Solution 3: Just-for-fun

- Write a program to generate the program.

```
for (int x=0; x<n; x++)
   cout << "for ( int I" << x
        << "= V[" << x << "];"
        << "I"    << x << " <= "
        << " W["  << x << "];"
        << " I"   << x << "++ )"  << endl;
```

```
for (int I0=V[0]; I0<= W[0]; I0++)
for (int I1=V[1]; I1<= W[1]; I1++)
for (int I2=V[2]; I2<= W[2]; I2++)
for (int I3=V[3]; I3<= W[3]; I3++)
……
```

A speedup: Collapse the "null" loops

```
for x1 = 1 to 5
 for x2 = 2 to 2
  for x3 = 3 to 7
   for x4 = 9 to 9
    for x5 = 1 to 1
      t = t+f(x1,x2,x3,x4,x5)
```

```
for x1 = 1 to 5

  for x3 = 3 to 7

                t = t+f(x1,2,x3,9,1)
```

Further speedup

Due to a property of the function f.

```
for x1 = 1 to 5
 for x2 = 2 to 2
  for x3 = 3 to 7
   for x4 = 9 to 9
    for x5 = 1 to 1
     t = t+f(x1,x2,x3,x4,x5)
```

```
for x1 = 1 to 5

   for x3 = 3 to 7

                t = t+f(x1, x3, 12)
```

TASK 2: PANCAKE

Flipping pancakes

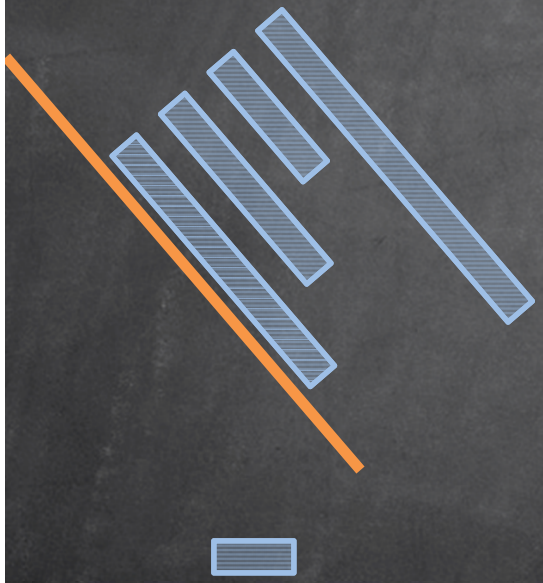
A stack of pancakes of different sizes can be *flipped.*

each flip operation lifts up the top few pancakes ….

Problem

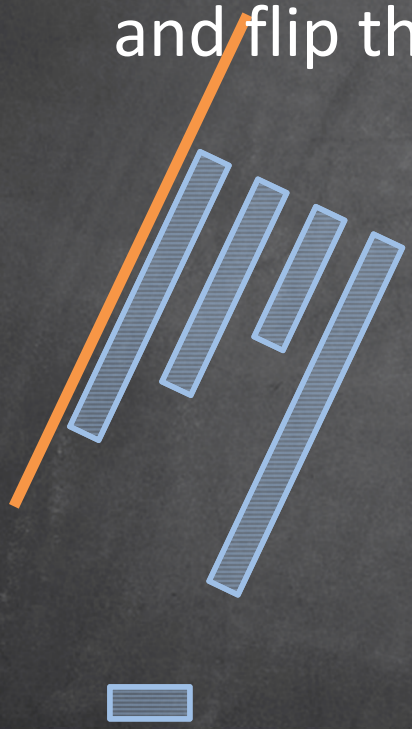
each flip operation lifts up the top few pancakes

....



Problem

each flip operation lifts up the top few pancakes and flip them.



Problem

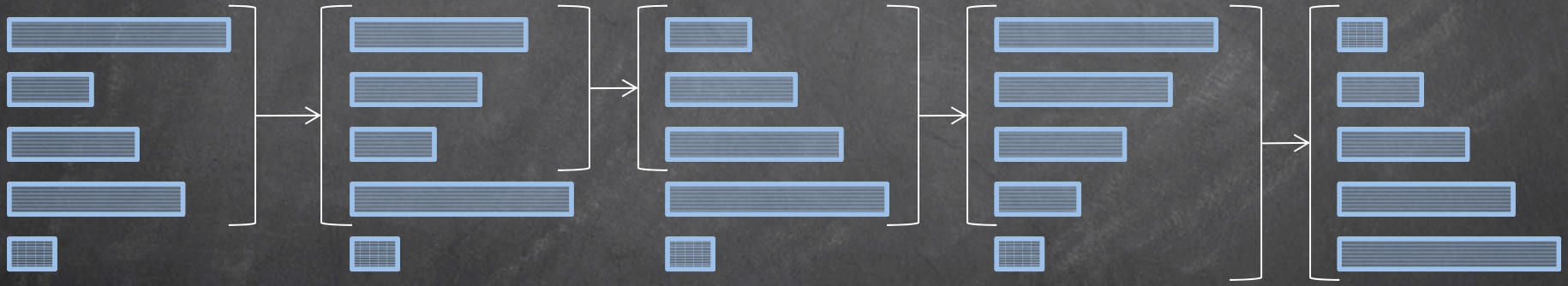
each flip operation lifts up the top few pancakes and flip them.

Problem

each flip operation lifts up the top few pancakes and flip them.

Sorting pancakes

We can sort this starting configuration using 4 flips. That is the best we can do (in term of number of flips).



Problem

Given many starting configurations, for each of them, determine the minimum number of flips required to sort it.

Partial Solution 1

Assume that ***at most*** 1 flip is required.

Check whether the input is already sorted. If so, then output 0, otherwise output 1.

Able to handle input instance 1.

Partial solution 2

Assume that ***at most*** 2 flips are required.

- Check whether the input is already sorted. If so, then output 0 and exit.
- For i= **2** to N,
  - flip the top i pancakes
  - if it is sorted, then output 1 and exit.
- Output 2.

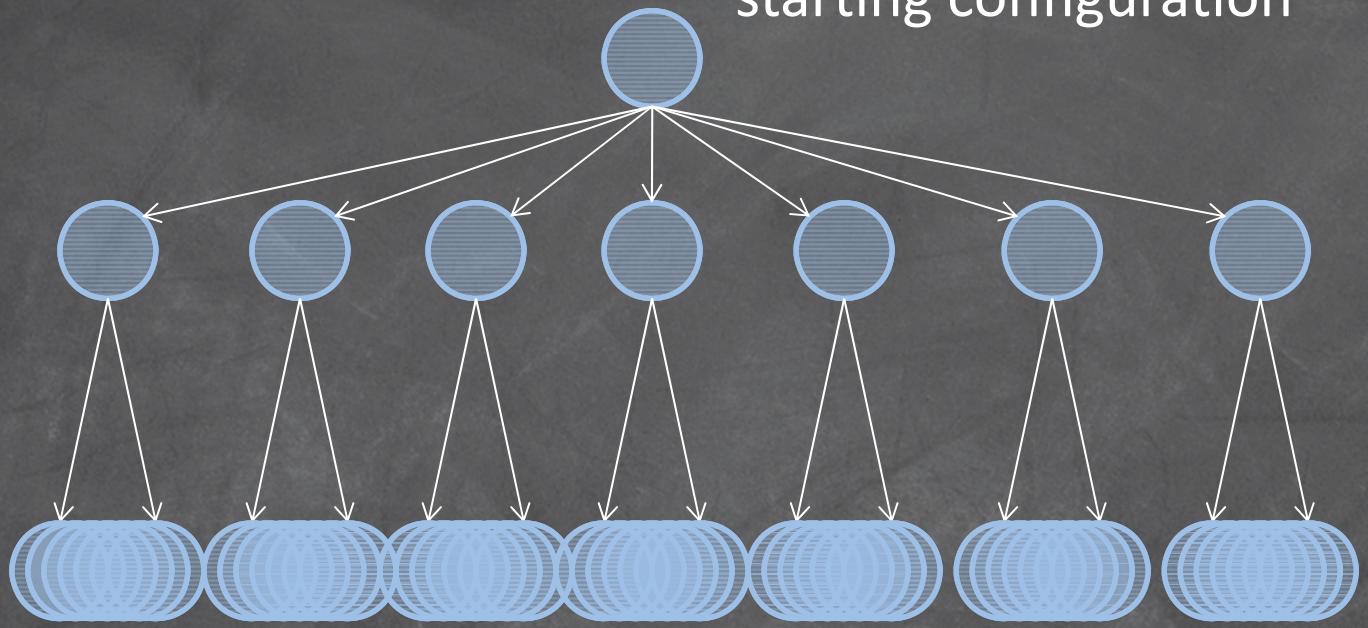
Able to handle input instance 2 & 3

Partial Solution 3: Breath-first-search

starting configuration

1 flip

2 flips



3 flips

...

Partial Solution 3: Breath-first-search

Let S= { starting_config }

t=0

While all configurations in S are not sorted,

t = t+1; S'= empty set;

for each s in S,

find the 7 configurations reachable from s in one flip and insert them into S' ;

S = S' ;

Output t.

It is not clear how long the algorithm would take.

Fortunately, it happens (to be discussed later) that the number of flips required is at most 9. So this algorithm is able to meet the 5 seconds requirement for one stack of pancakes.

Able to handle input instance 4

Solution

Total number of configurations is less than

$$8^8 = 2^{24} = 16\ M$$

which is small enough.

Let $T(s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8)$ be the min number of flips required to sort the stack of 8 pancakes of size $s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8$.

So, $T(1,2,3,4,5,6,7,8) = 1$.

Carry out breath-first-search, but use the table T to avoid repetitions.

Now, for each starting configuration, lookup the table T for the minimum number of flips required.

Remark

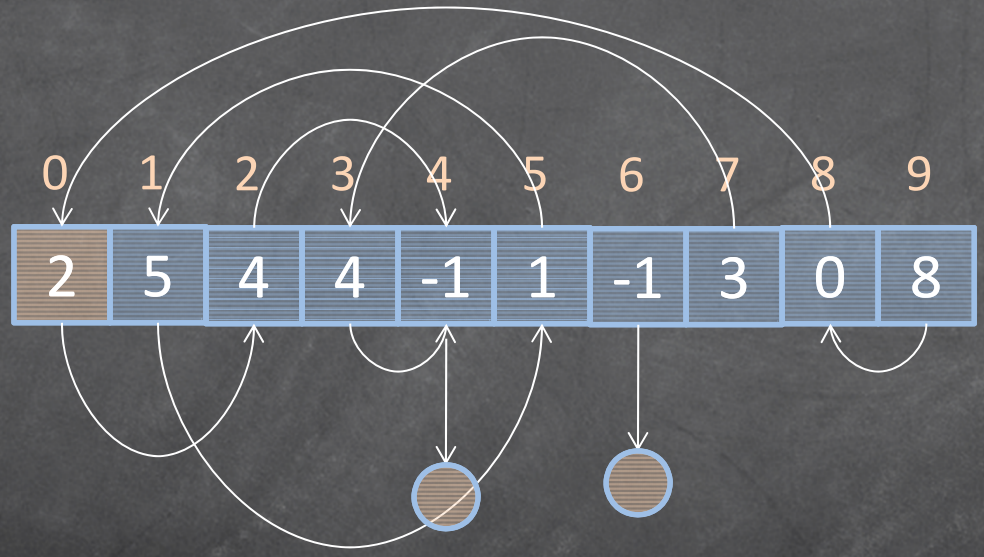
- Given *n*, what is the maximum number of flips required in the worst case?
- "Google" using "pancake number" as keyword to find out more.
- Let's denote the max as P_n. It is known that $P_8 = 9$.
- In 1979, William H. Gates and Christos H. Papadimitriou showed that

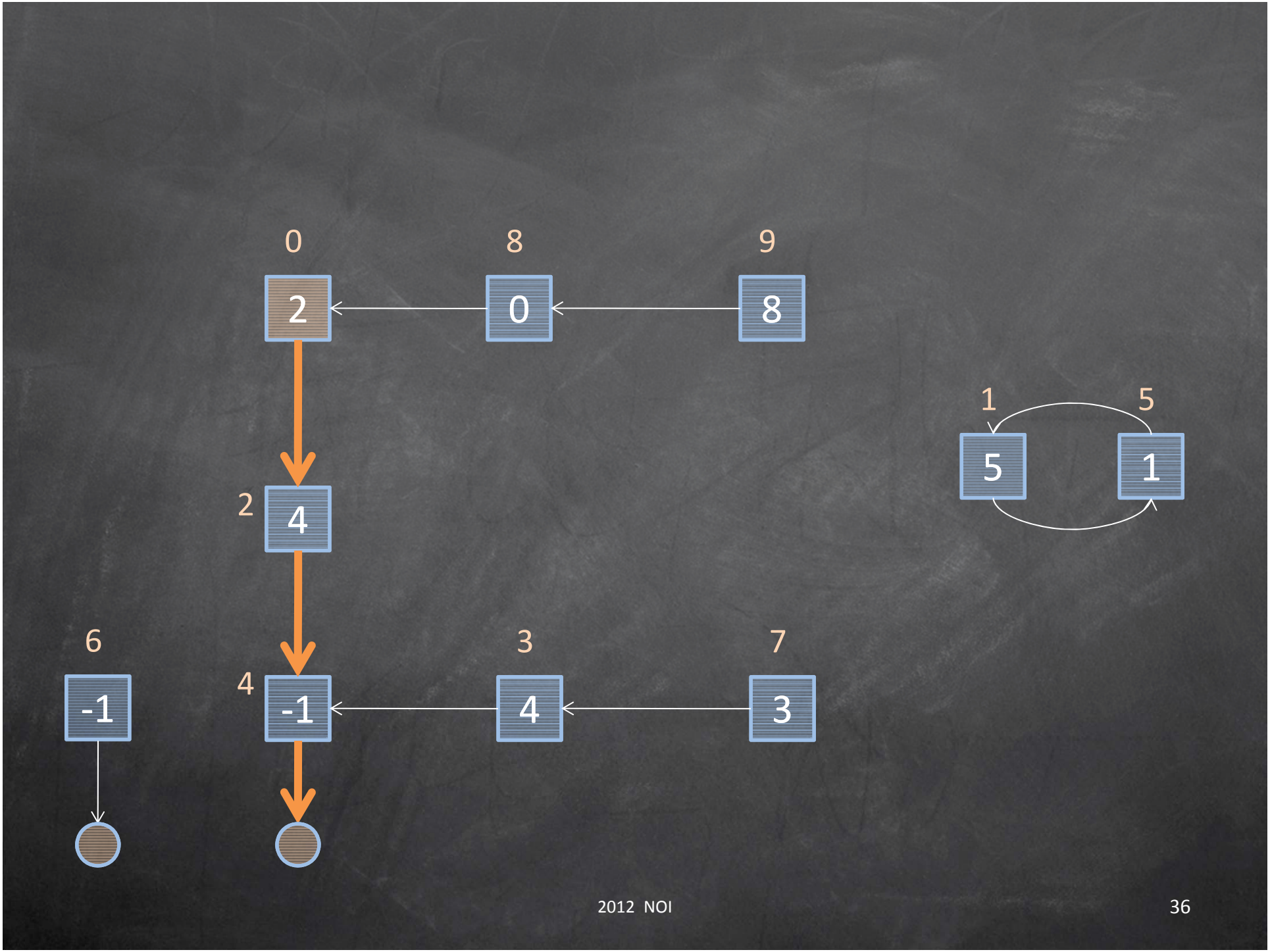
$$17n/16 \leq P_n \leq (5n + 5)/3$$

Task 3: FORENSIC

Linked list

The pointers of a linked list is stored in an array, with 0 as the header.

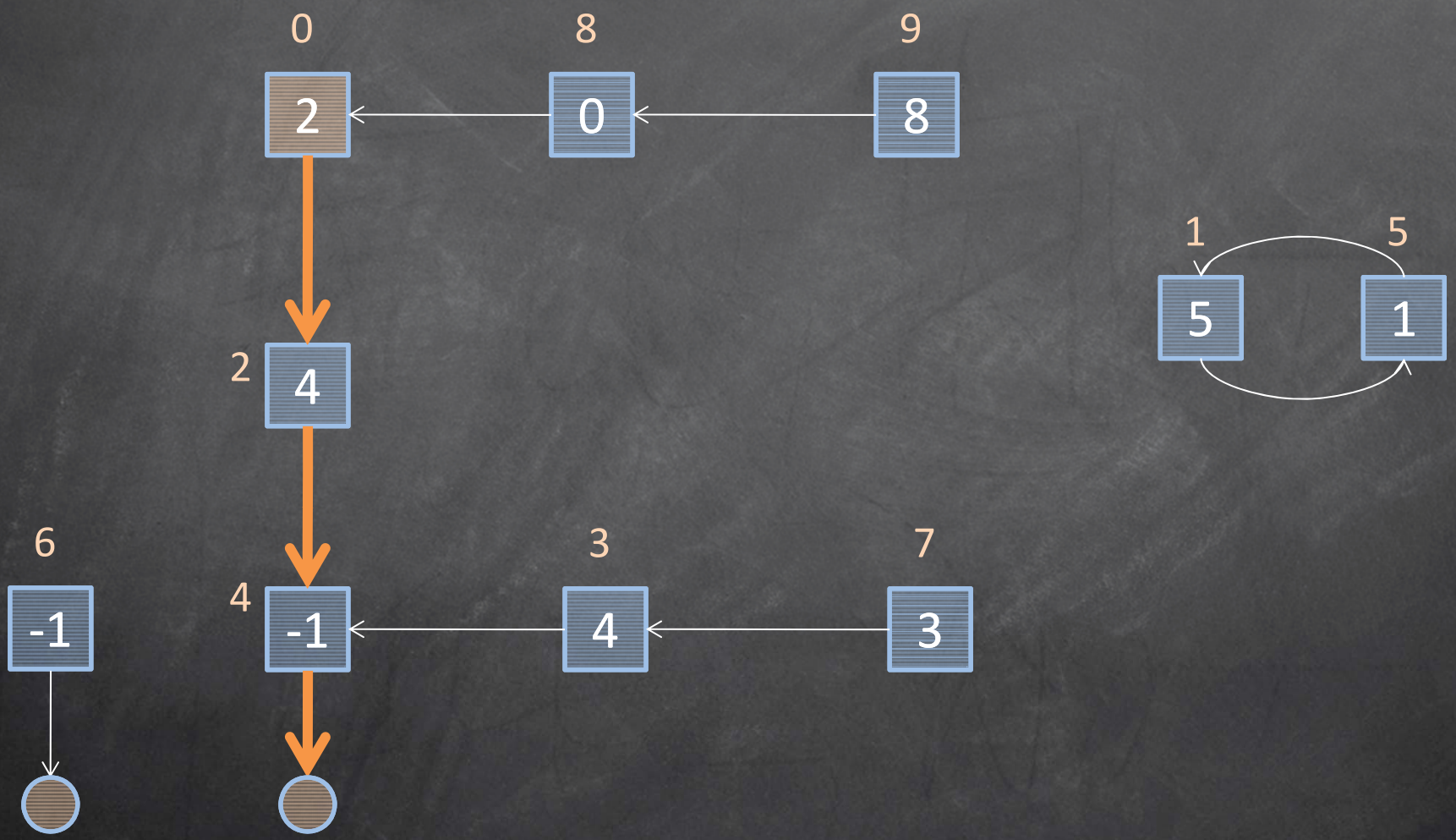

0 1 2 3 4 5 6 7 8 9
2 5 4 4 -1 1 -1 3 0 8


0
8
9
2
0
8
2
4
1
5
5
1
6
3
7
4
-1
-1
4
3

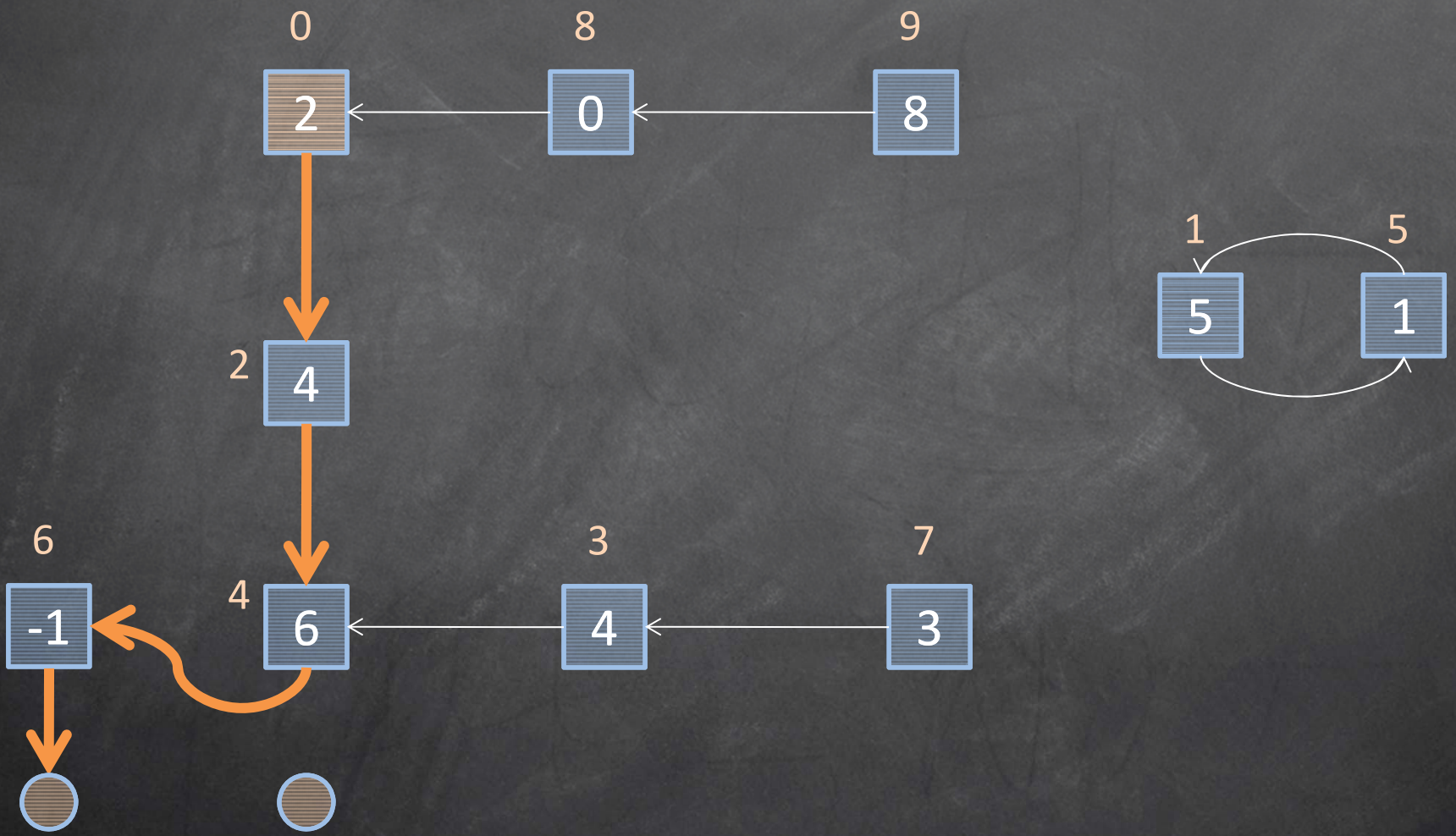
Problem

We want to modify one entry so as to obtain the longest linked list.

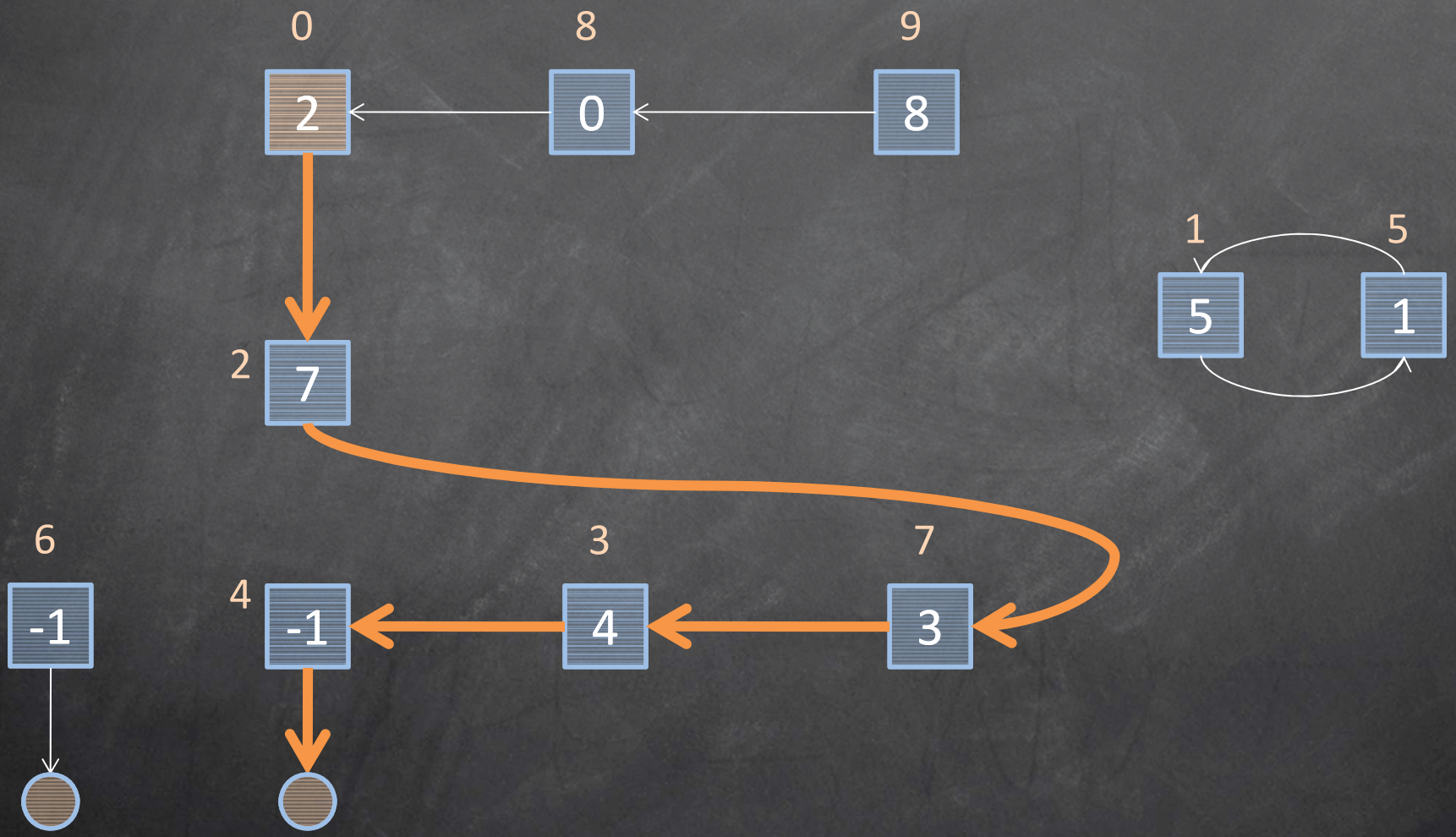
Example



4 nodes



5 nodes and that is the best we can do.



Partial Solution 1

Try all possibilities of changes.

location of entry to be changed

new value

```
for i=0 to N-1 do
    for x = -1 to N-1 do
        Change A[i] to x;
        s= length (A);  if s>max,  then max =s;
        Revert A[i] back to its original value;
```

The length() function can be computed using a for-loop. So this is essentially a $\Theta(n^3)$ algorithm.

Partial Solution 1

Try all possibilities of changes.

```
for i=0 to N-1 do
    for x = -1 to N-1 do
        Change A[i] to x;
        s= length (A);  if s>max,  then max =s;
        Revert A[i] back to its original value;
```

Able to solve input instance 1

The length() function can be computed using a for-loop. So this is essentially a $\Theta(n^3)$ algorithm.

Partial Solution 2

Observe that we don't have to try all entries. Only changes made to entries along the main-chain will have effect.


```
for i  in the main-chain
    for x = -1 to N-1 do
        Change A[i] to x;
        s= length (A);  if s>max,  then max =s;
        Revert A[i] back to its original value;
```

Able to solve input instance 2

Input instance 3

It is possible to have an $O(n^2)$ algorithm.

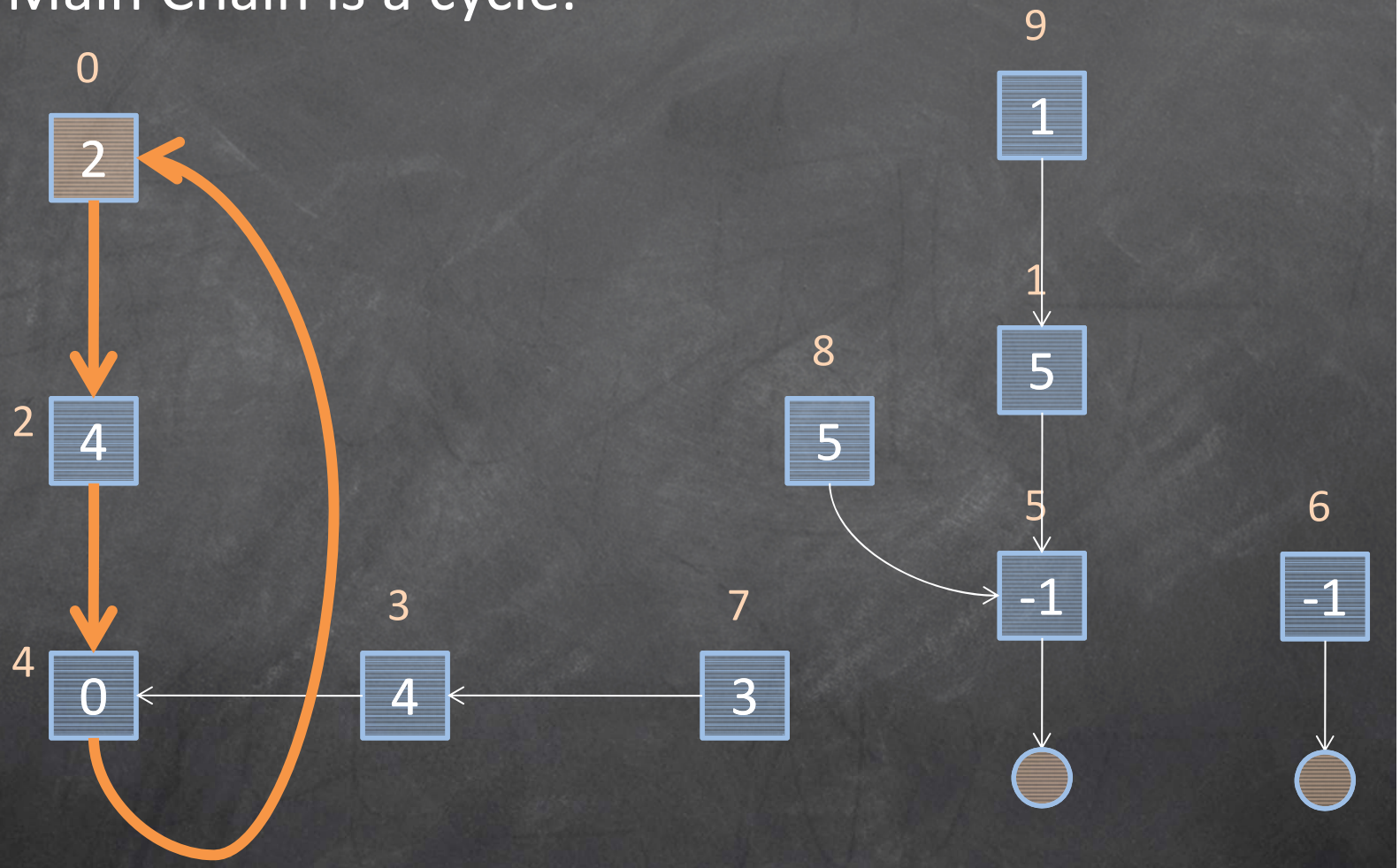
(details omitted)

Able to solve input instance 3

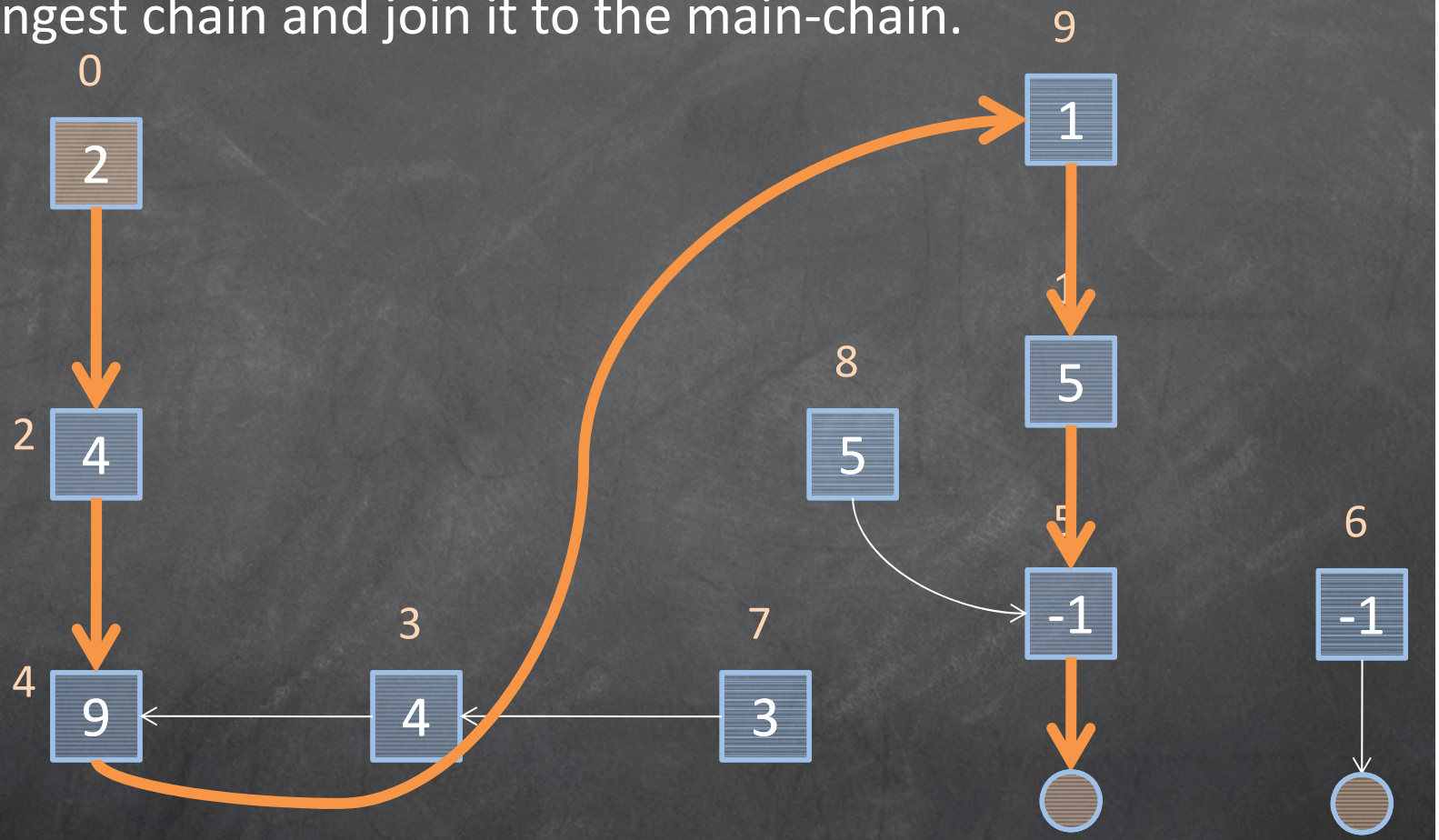
Solution

let’s call all nodes reachable from 0 the main-chain.

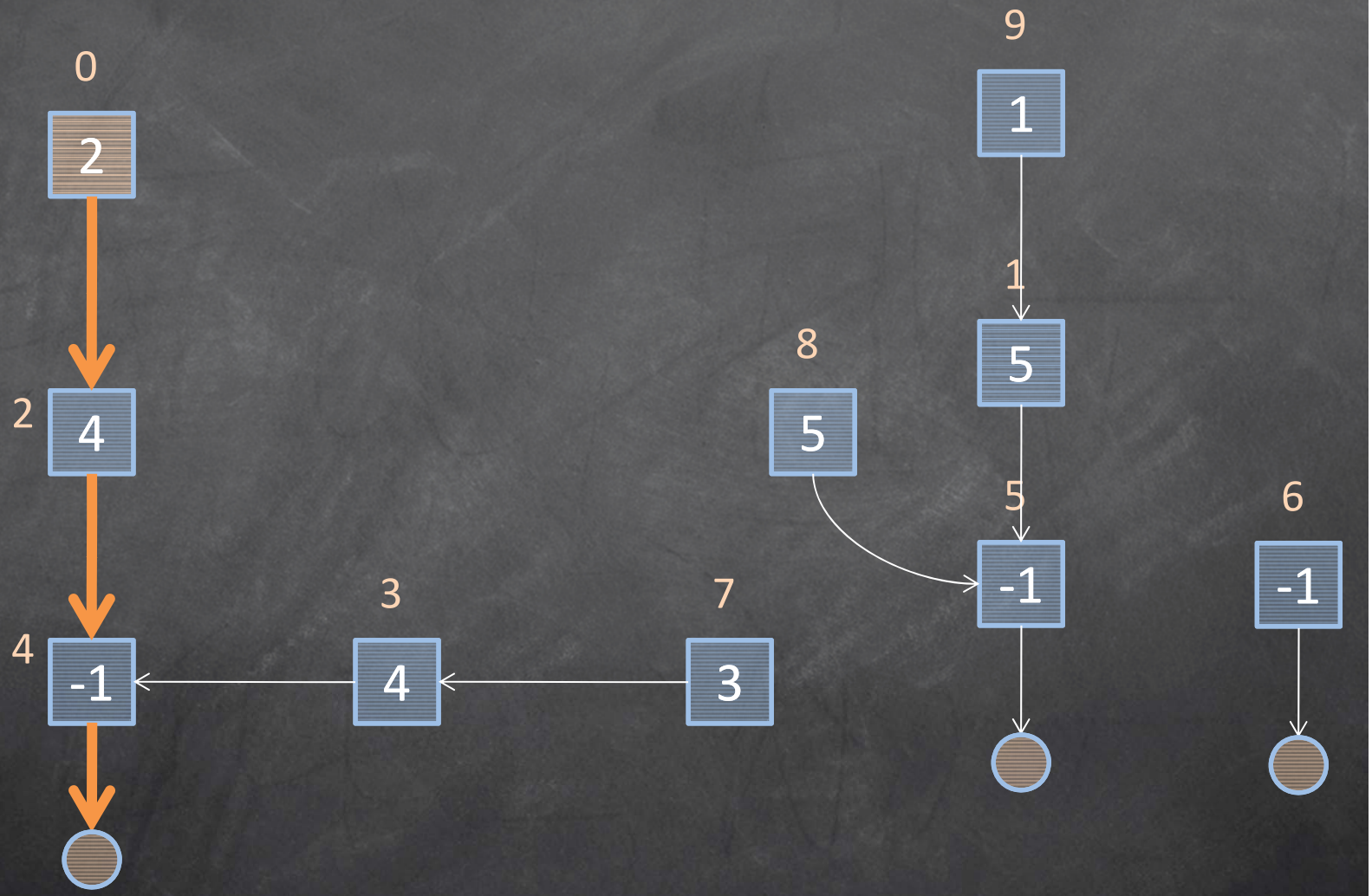
Case 1: Main Chain is a cycle.



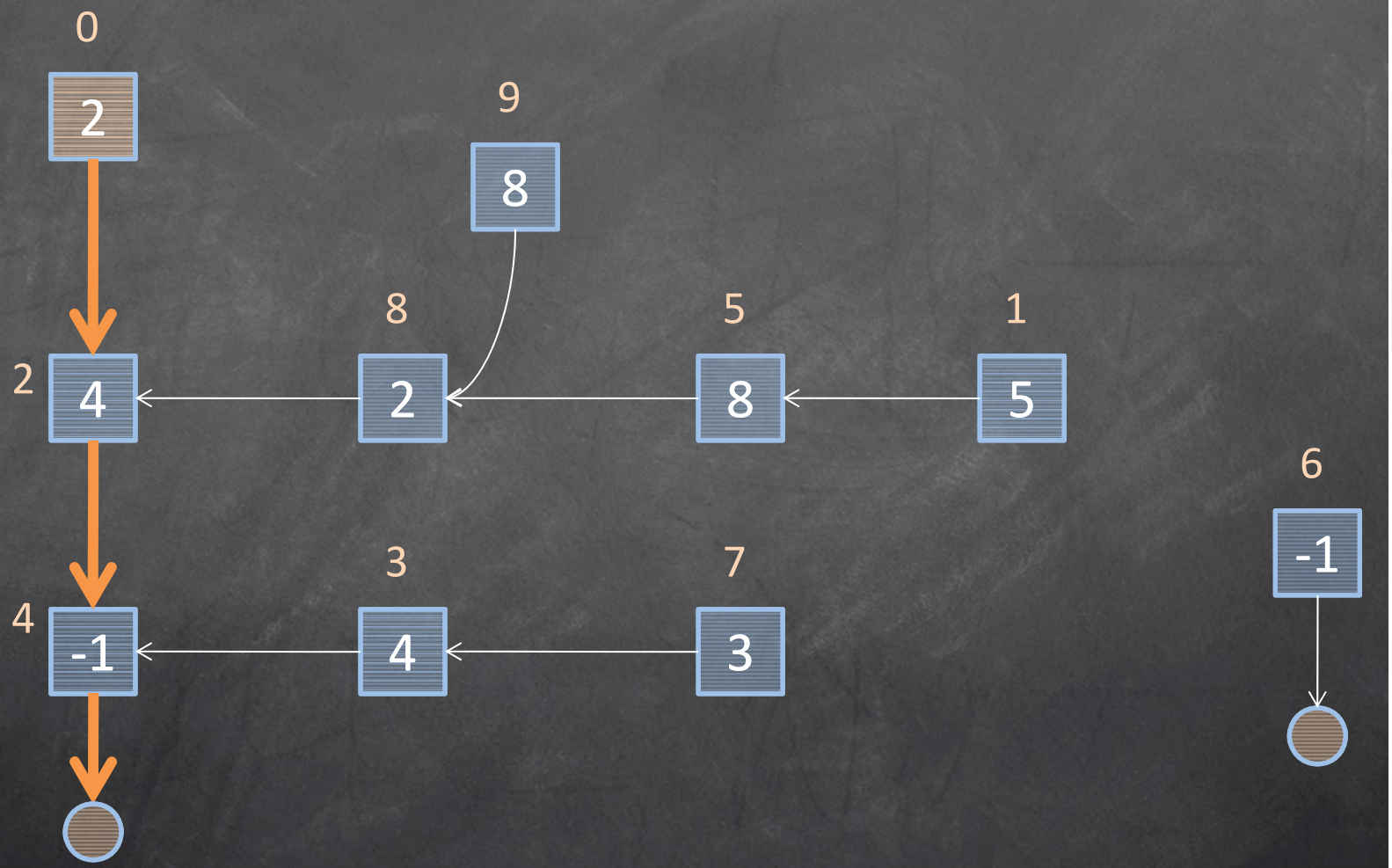
Case 1: Main Chain is not valid – ignore branches along the main chain.
Find the longest chain and join it to the main-chain.



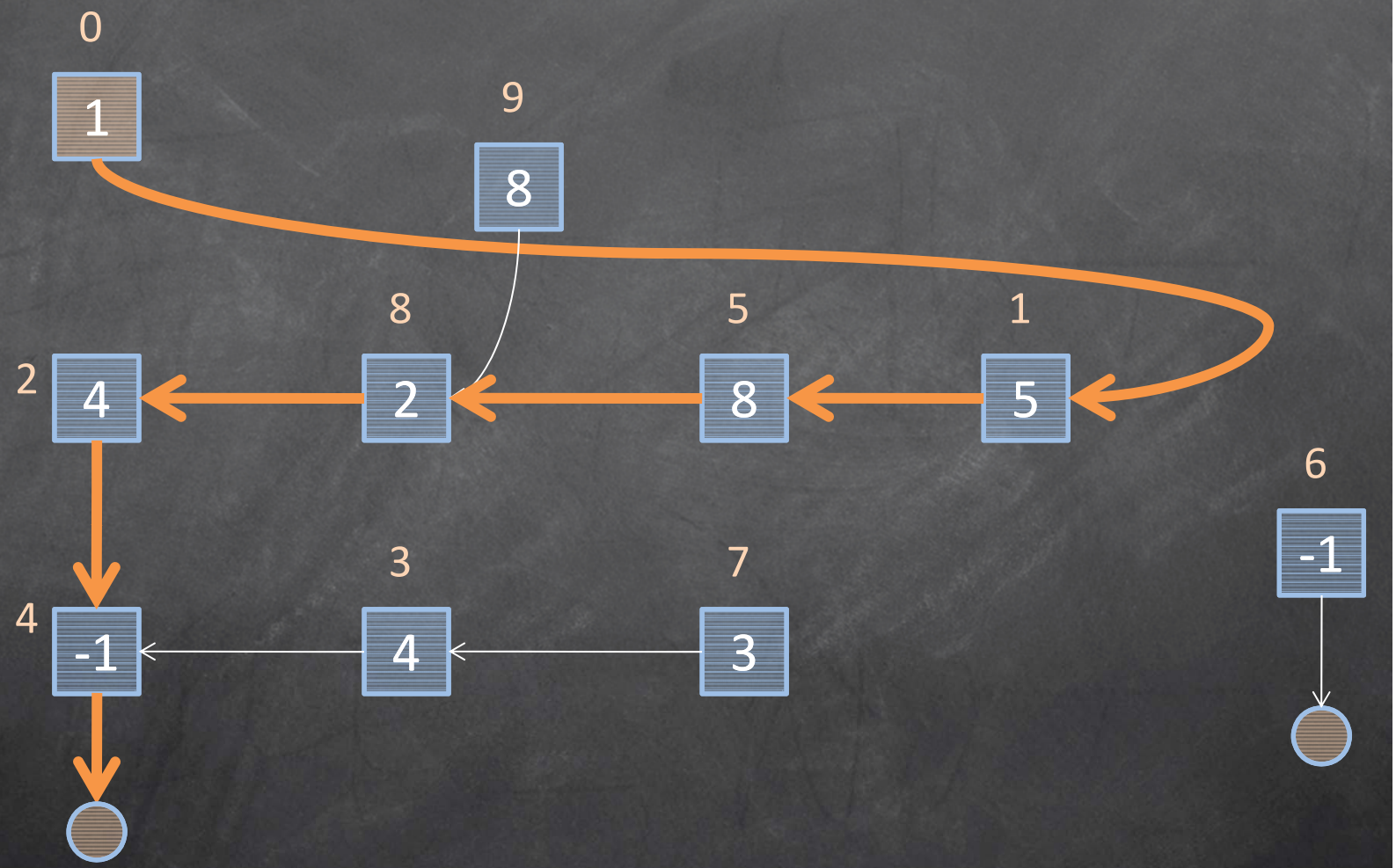
Case 2: Main Chain is valid. Check two sub-cases.



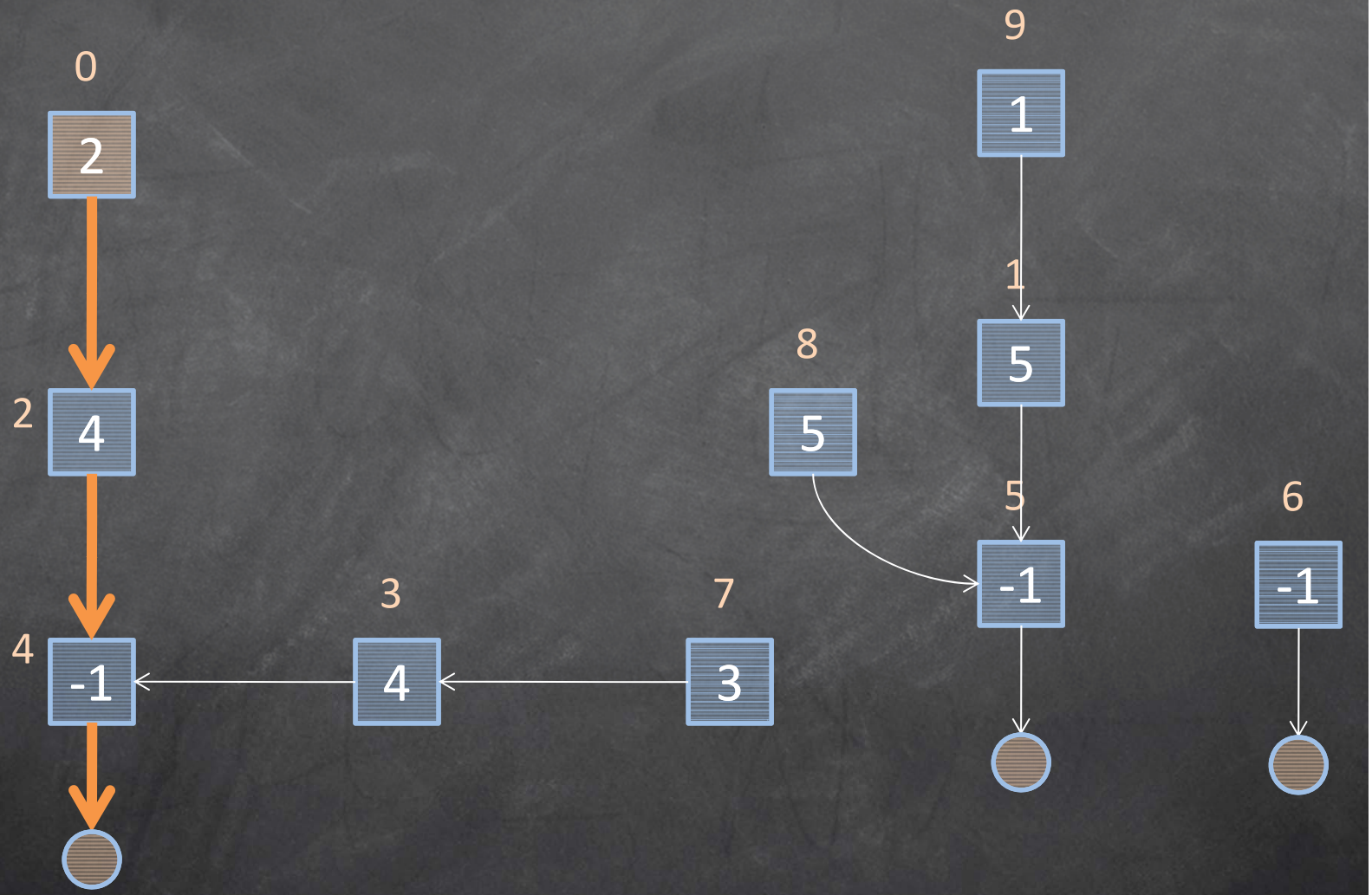
Sub-case 2(a): include the longest branches along main-chain.



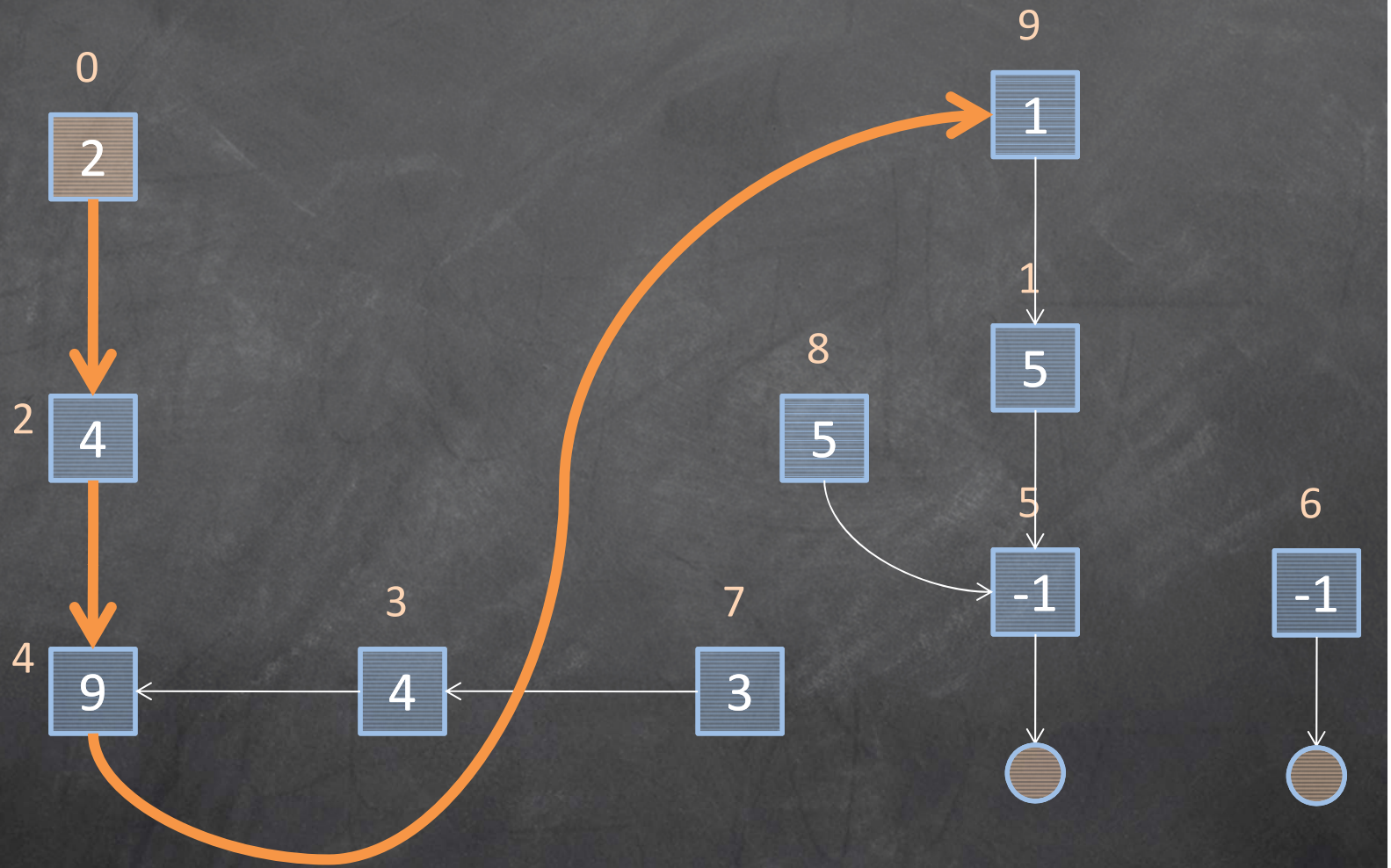
Sub-case 2(a): include the longest branches along main-chain.



Sub-case 2(b): Append the longest chain to the main-chain.



Sub-case 2(b): Append the longest chain to the main-chain.



Hence, the problem can be solved in “linear” time.

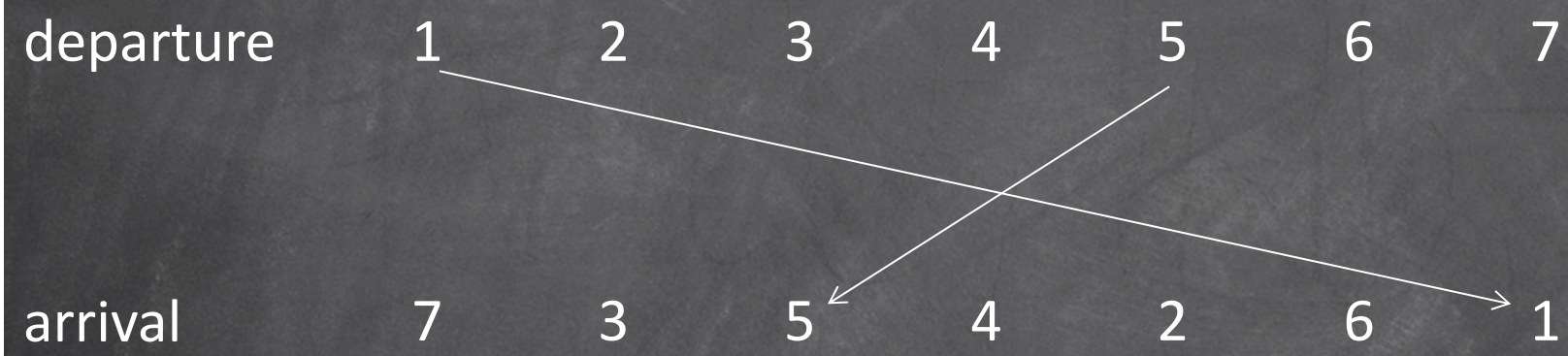
Task 4: WALKING

Friends

Given two sequences: departure and arrival sequences of *n* persons. Let's assign an unique number to each person.

departure	1	2	3	4	5	6	7
arrival	7	3	5	4	2	6	1

Given two sequences: departure and arrival sequences of *n* persons. Let's assign an unique number to each person.



Two persons x, y are friends ***iff*** their positions in the two sequences being swapped.

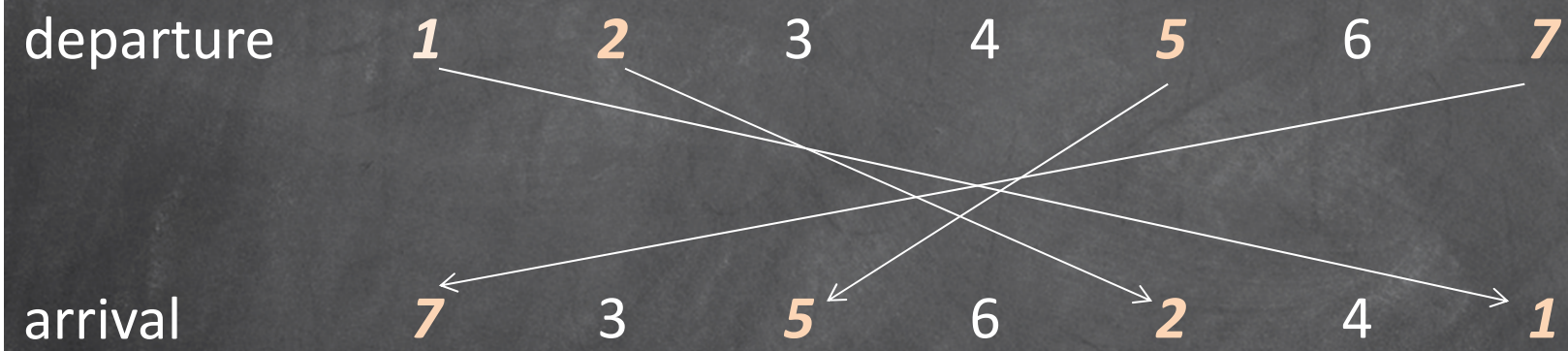
That is, if x leaves earlier than y, but arrives later than y, then they are friend.

Problem

Find the largest set of persons where any two persons in that set are friends.

Example

A set of 4 persons is the largest possible.



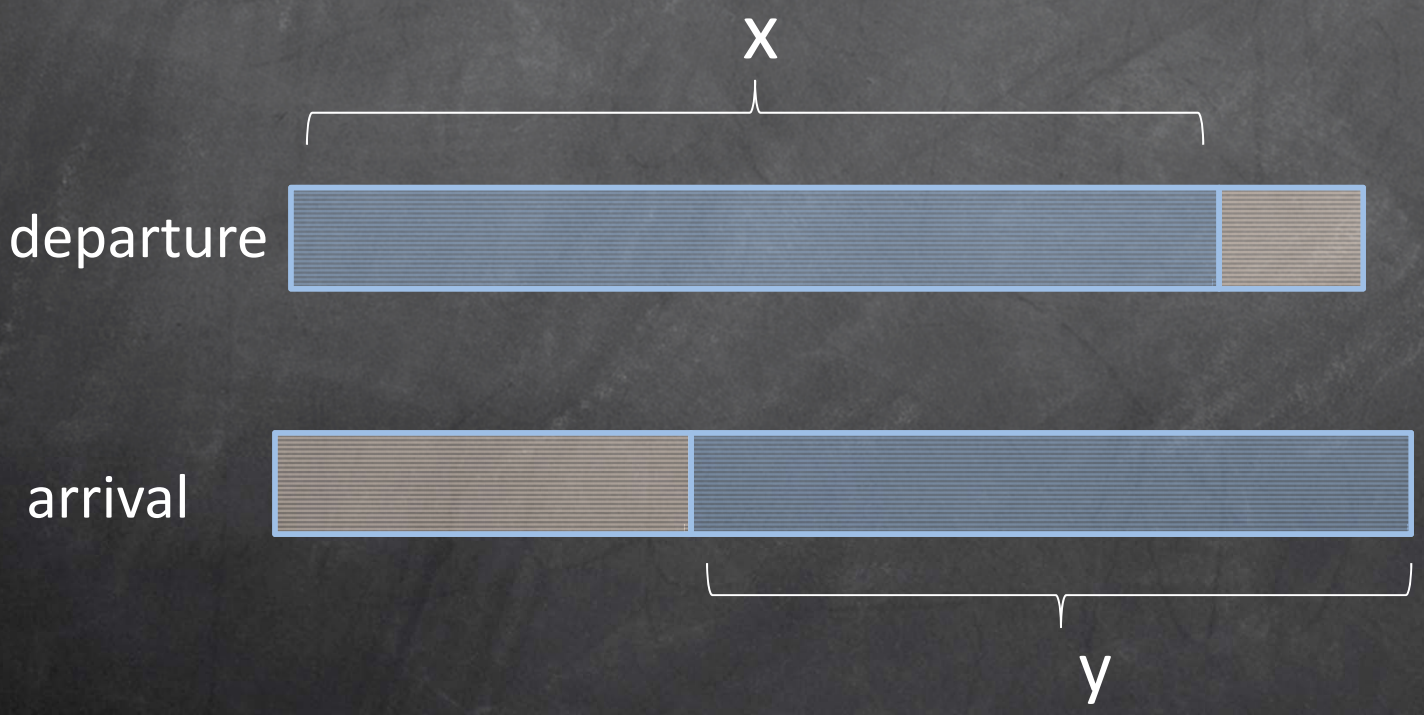
Max Clique

Yes, this is the max clique problem. In general, it is NP-hard, and thus best known algorithm takes exponential time.

But, this is a special case which interestingly has fast algorithm.

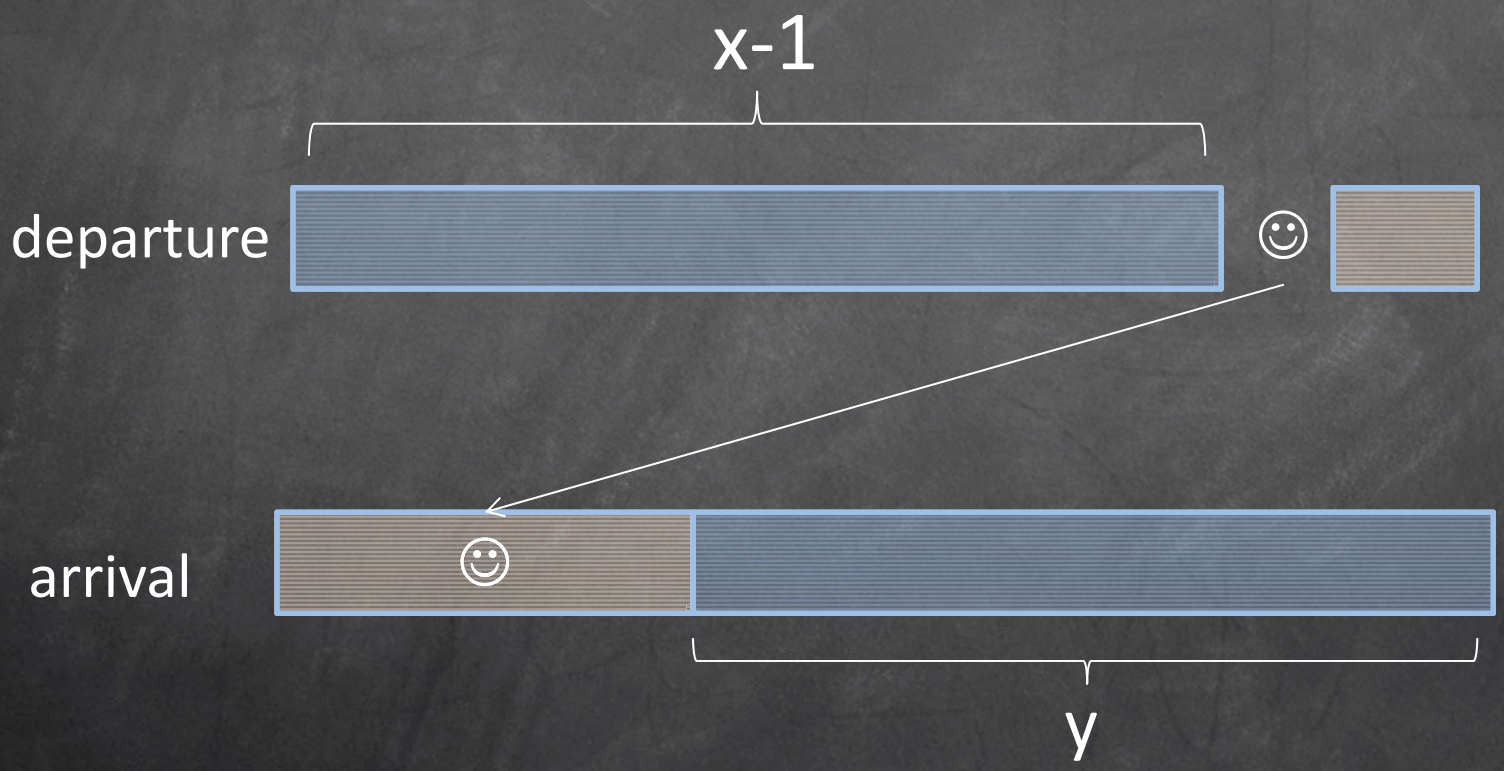
Try Dynamic Programming

Let M(x, y) be the size of the max clique among the first x persons to depart, and the last y persons to arrive. We want to find M(n,n).



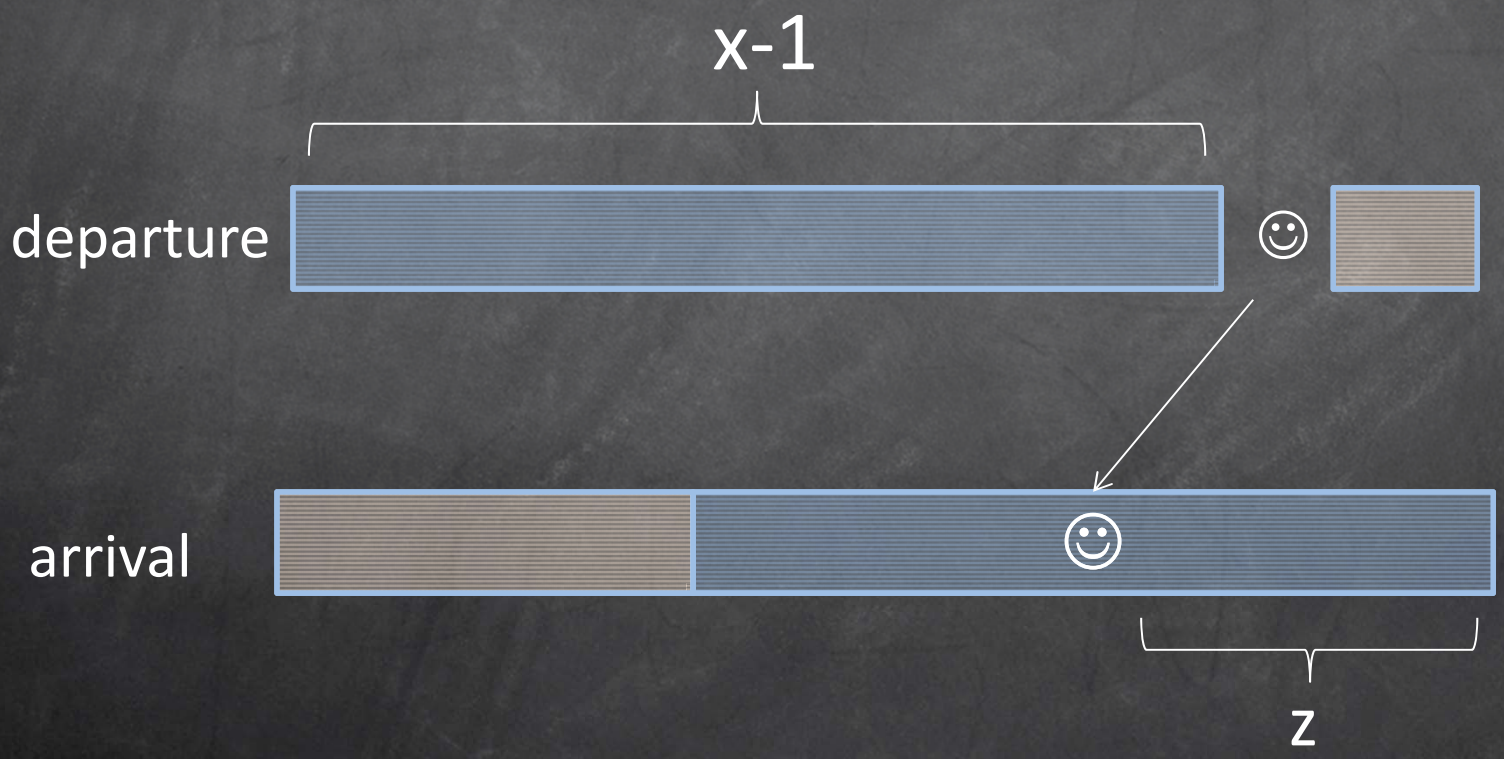
A recurrence equation: 3 cases

(a) M(x,y) = M(x-1,y) if the following holds:



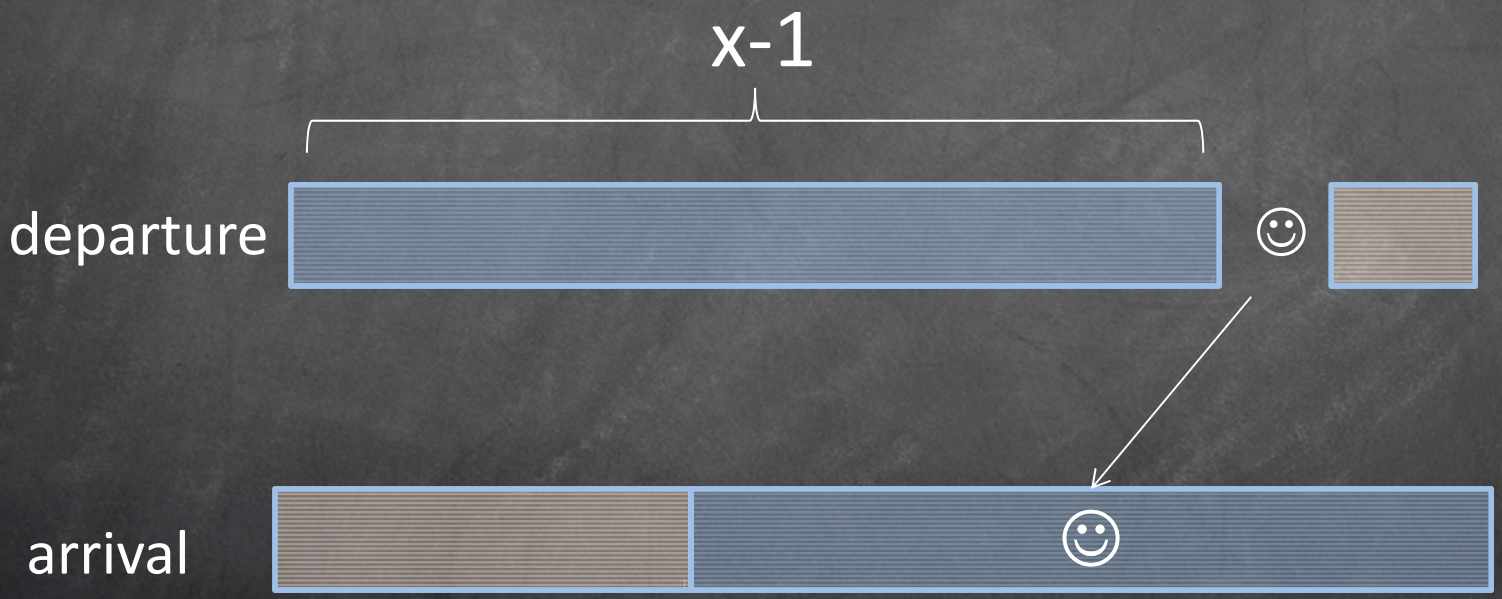
(b) M(x,y) = 1+ M(x-1, z) if the following holds and ☺ contributes the value of M(x,y)

to



(c) M(x,y) = M(x-1, y)

if the following holds and ☺ does not contributes to the value of M(x,y)



Recurrence equation

$M(x,y) = \max \{ M(x-1,y), 1 + M(x-1,z) \}$,

where z is as defined in the previous slide

Base cases:

$M(1,x) = 1$ for all x

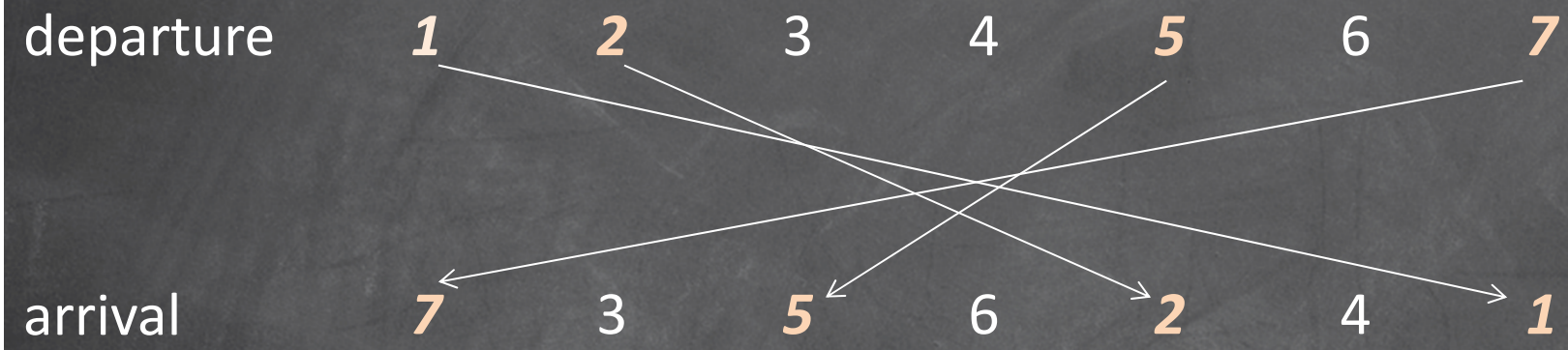
$M(x,1) = 1$ for all x.

Solution

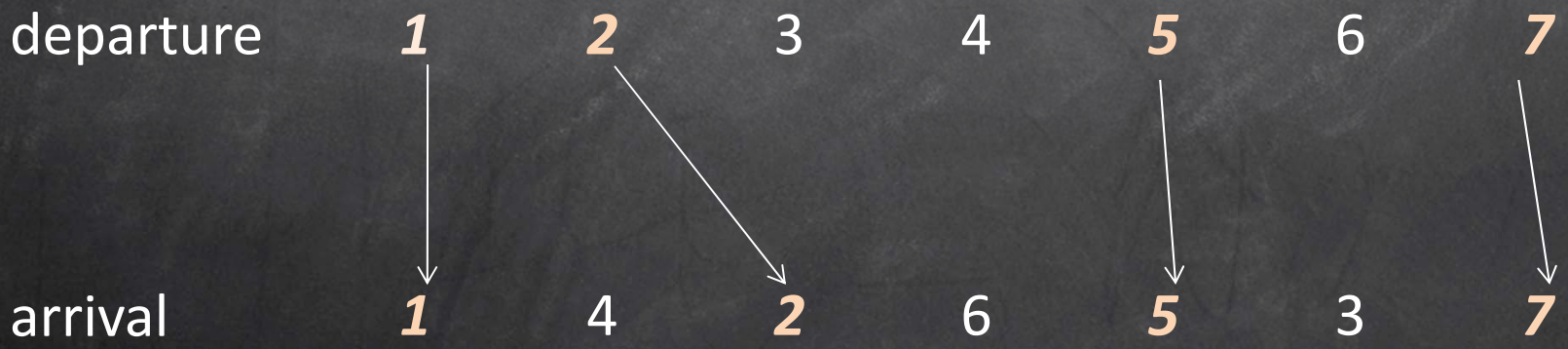
Using dynamic programming based on the previous recurrence equation, we can have a $O(n^2)$ algorithm.

Remark

Original problem: each edge crosses every other edges.



Flip the arrival seq. Now, all edges are non-crossing.



- Hence, the problem is can be “reduced” to this problem: given two sequences of integers, find the longest common subsequence.