The Knapsack Problem

Knapsack Problem

- Given an integer K, n items of different size (the i-th item has an integer size k_i)

  Find a subset of the items

  Such that sizes sum to exactly K
- e.g. Given $K = 7$, and 4 items of size $\{2, 3, 5, 6\}$

  Find a subset of the items

  Such that sizes sum to exactly 7

Given K=13, & 5 items of size {2, 3, 5, 7, 8, 9}

Find a subset of the items

Such that sizes sum to exactly 13

Brute Force

- Generate all combination of subset of the item set
- For each subset, test if feasible solution.
- Complexity O(2^n) , n: #(items)

Thinking

- For simplicity, we first concentrate on decision problem
- P(n, K): n items packing in size K
- P(i, k): first i items packing in size k
- Hypothesis: we know how to solve P(n-1, K)
- Induction
  - if ∃ solution for P(n-1, K), we have done

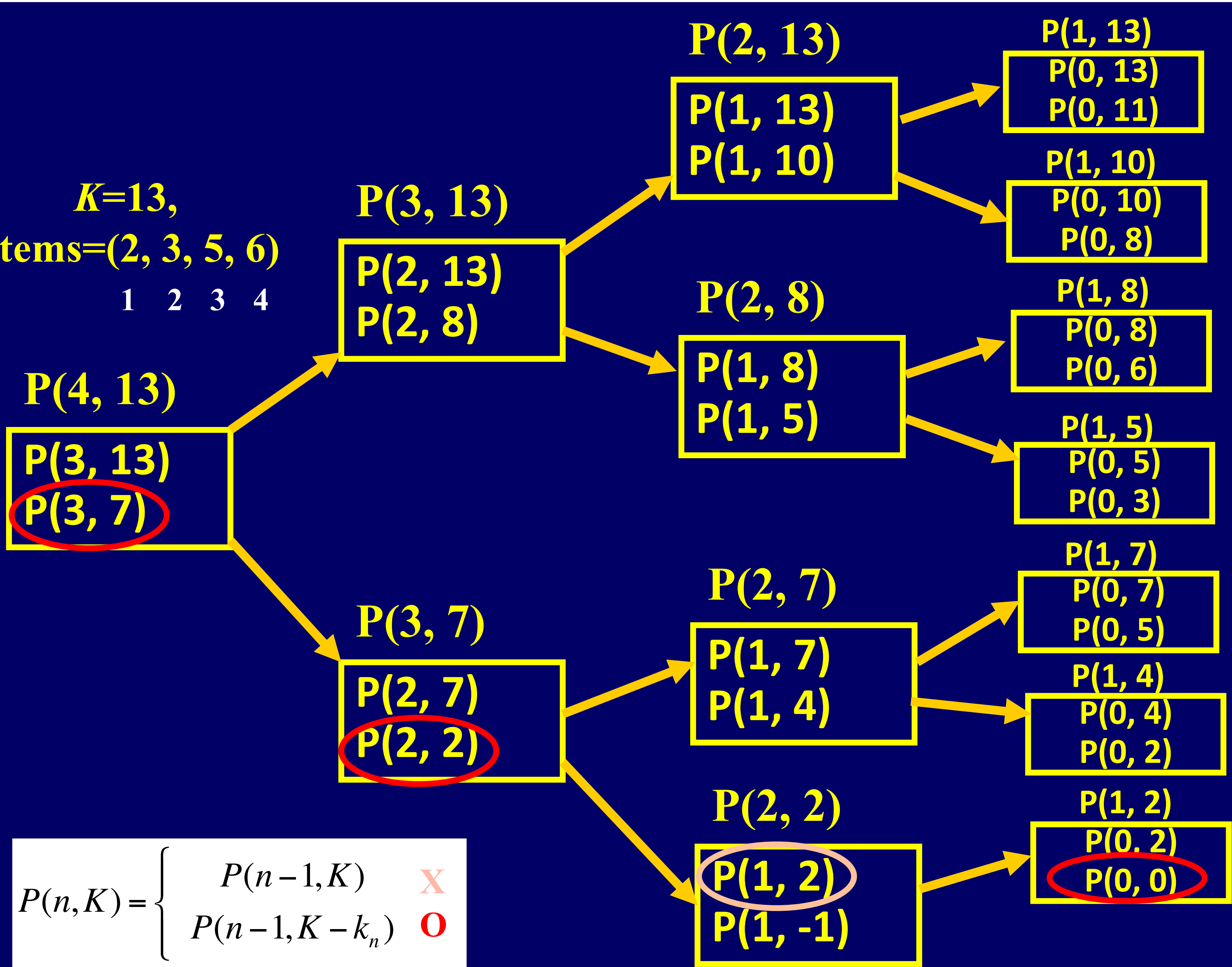
    the n-th item must be excluded
  - if not ∃ solution for P(n-1, K),
    - can we use the negative result ?
    - the n-th item must be included
    - Reduce P(n, K) problem to P(n-1,K) & P(n-1, K-k_n)

Induction

- **Hypothesis: we know how to solve P(n-1, k) for all 0≤ k ≤ K**
- **Induction:**
  - **reduce p(n, K) problem to P(n-1, K) & P(n-1, K-k_n)**

$$P(n,K)=\begin{cases} P(n-1,K) & \text{if n-th item is excluded} \\ P(n-1,K-k_n) & \text{if n-th item is included} \end{cases}$$

K=13,
Items=(2, 3, 5, 6)
1 2 3 4



$$P(n,K)=\begin{cases} P(n-1,K) & \text{X} \\ P(n-1,K-k_n) & \text{O} \end{cases}$$

Problem of Recursion

$$P(n,K)=\begin{cases} P(n-1,K) \\ P(n-1,K-k_n) \end{cases}$$

- **Problem:**
  - **inefficient, reduce problem of size *n* to two subproblems of size (*n*-1)**
  - **time complexity: exponential O(2^n)**

Improvement

- **Observation:**
  - **P(i, k), i: n possibilities, k: K possibilities**
  - **$n*K$ different combinations**
- **Solutions (Dynamic Programming)**
  - **Remember all solutions and never solve the same problem twice**
- **Comment**
  - **dynamic programming can work only if total no. of possible subproblems is not too large.**

Knapsack Problem (cont.)

- **Solution: dynamic programming**

$$P(n,K)=\begin{cases} P(n-1,K) & \text{if n-th item is excluded} \quad \mathrm{O} \\ P(n-1,K-k_n) & \text{if n-th item is included} \quad \mathrm{I} \end{cases}$$

		0	1	2	3	4	5	6	7
	0	I	-	-	-	-	-	-	-
1	2	O	_	I	_	_	_	_	_
2	3	O	_	O	I	_	I	_	_
3	5	O	_	O	O	_	O	_	I
4	6	O	_	O	O	_	O	I	O

I included
O exclude
- no solution

Knapsack Problem (cont.)

- $$P(n,K) = \begin{cases} P(n-1,K) & \text{if n-th item is excluded} \\ P(n-1,K-k_n) & \text{if n-th item is included} \end{cases}$$

I: included, O: exclude, -:no solution

		0	1	2	3	4	5	6	7	8	9	10	11	12	13
	0	I	-	-	-	-	-	-	-	-	-	-	-	-	-
1	2	O	_	I	_	_	_	_	_	_	_	_	_	_	_
2	3	O	_	O	I	_	I	_	_	_	_	_	_	_	_
3	5	O	_	O	O	_	O	_	I	I	_	I	_	_	_
4	6	O	_	O	O	_	O	I	O	O	I	O	I	_	I

```
Algorithm Knapsack(S, K)
Input: S(array of size n storing the sizes of items)
          K(size of knapsack)
Output: P (  P[i, k].exist=true if there exists a solution to knapsack  problem with
                                                   the first i elements and a knapsack of size k
               P[i, k].belong=true if the ith element belongs to the solution            )
Begin
    p[0,0].exist:=true;
    for k:=1 to K do
        p[0, k].exist:=false;
    for i:=1 to n do
        for k:=0 to K do
            P[i, k].exist:=false;
            if P[i-1, k].exist:=true;                /* P(i, k)=P(i-1, k), item i不選
              P[i, k].exist:=true;                   /* P(i,k)有解
              P[i, k].belong:=false                  /* O (exclusive)
            else if k-S[i] >=0 then
                if P[i-1,k-S[i]].exist then          /* P(i, k)=P(i-1, k-si), item i必選
                  P[i, k].exist:=true;               /* P(i,k)有解
                  P[i, k].belong:=true;              /* I (inclusive)
End
```

$$P(n,K)=\begin{cases} P(n-1,K) \\ P(n-1,K-k_n) \end{cases}$$

Knapsack Problem: Backtracking

- $$P(n,K) = \begin{cases} P(n-1,K) & \text{if n-th item is excluded} \\ P(n-1,K-k_n) & \text{if n-th item is included} \end{cases}$$

I: included, O: exclude, -:no solution

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	0	**I**	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
1	**2**	**O**	_	**I**	_	_	_	_	_	_	_	_	_	_	_	_	_	_
2	**3**	**O**	_	**O**	**I**	_	**I**	_	_	_	_	_	_	_	_	_	_	_
3	**5**	**O**	_	**O**	**O**	_	**O**	_	**I**	**I**	_	**I**	_	_	_	_	_	_
4	**6**	**O**	_	**O**	**O**	_	**O**	**I**	**O**	**O**	**I**	**O**	**I**	_	**I**	**I**	_	**I**

Dynamic Programming

- developed by Richard Bellman in the 1950s
- typically applied to optimization problems
- simplifying a complicated problem by
  breaking it down into simpler sub-problems
  in a recursive manner.
- Using a table, instead of recursion
- solving every sub-problem just once & then saves its answer in a table
- avoiding the work of re-computing the answer every time the sub-problem is encountered.

A Simple Example of Dynamic Programming

- Fibonacci Number F(n) = ?
- F(n) = F(n-1) + F(n-2)
- Two approaches
  - Recursive program

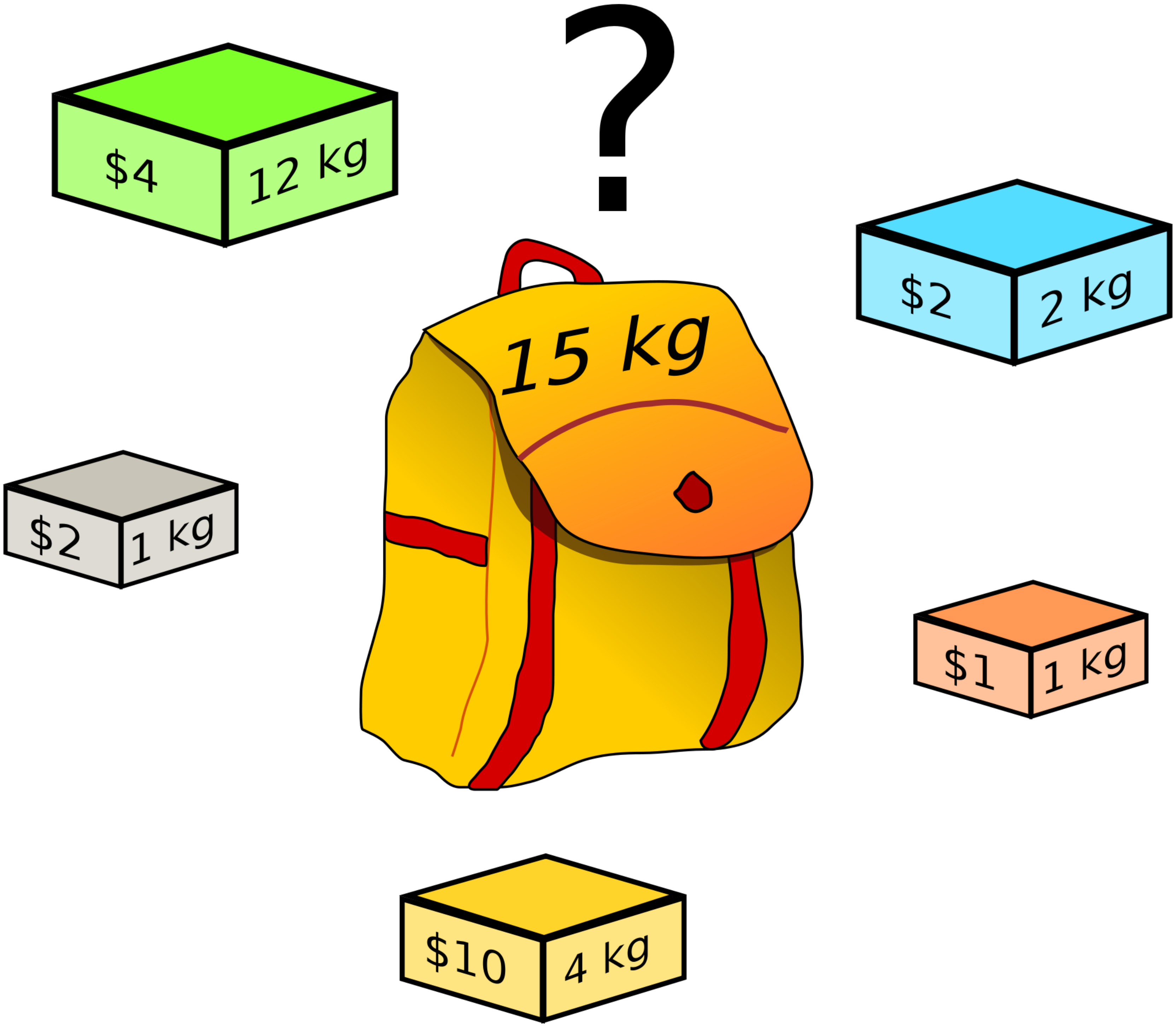
```
int fib(int n)
{
    if ( n <= 1)
        return (n);
    else
        return( fib(n-1) + fib(n-2) );
}
```

  - Dynamic Programming

1	1	2	3	5	8	13	21	34	…
1	2	3	4	5	6	7	8	9	…

Maximal Value Knapsack Problem

- Given an integer K, n items of different size & value v_i
  (the i-th item has an integer size k_i)
  Find a subset of the items
  Such that total size is not larger than K, &
  total value is as large as possible.
- e.g. Given K=13, &
  5 items of size {1, 1, 2, 4, 12} & value {1, 2, 2, 10, 4}
  Find a subset of the items
  Such that total size is not larger than 13 &
  total value is as large as possible.


?
15 kg
$4 12 kg
$2 2 kg
$2 1 kg
$1 1 kg
$10 4 kg

Given K=13, &

6 items of size {1, 2, 3, 5, 5, 8} &

value {2, 3, 1, 5, 6, 7}

Find a subset of the items

such that total size is not larger than 13 &

total value is as large as possible.

設計Dynamic Programming演算法解 Maximal Value Knapsack Problem?

Summary of Design Algorithm by Induction

Evaluating Polynomials

- Given a sequence of real no. $a_n, a_{n-1}, \ldots a_1, a_0$
  and a real number x
  Compute value of polynomial
  $$P_n(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0$$

- Given 10, 5, 8, 2, 6, (i.e., $P_4(x) = 10x^4+5x^3+8x^2+2x+6$)
  and 2, (i.e., $x=2$)
  Compute $P_4(2) = 10*2^4+5*2^3+8*2^2+2*2+6$

Evaluating Polynomials (cont.)

- Approach 3
  - $10x^4+5x^3+8x^2+2x+6=$

    $\{[(10x+5)x+8]x+2\}x+6$
  - Hypothesis: we know how to compute $P'_{n-1}(x)$

    $P'_{n-1}(x) = a_n x^{n-1}+a_{n-1}x^{n-2}+\ldots+a_1$
  - Induction: $P_n(x) = x \bullet P'_{n-1}(x) + a_0$

    $a_n x^n+a_{n-1}x^{n-1}+\ldots+a_1x+a_0=$

    $((\ldots(a_nx+ a_{n-1})x+ a_{n-2})\ldots)x+a_1)x+a_0$
  - Complexity: n multiplication & n addition

Finding One to One Mapping

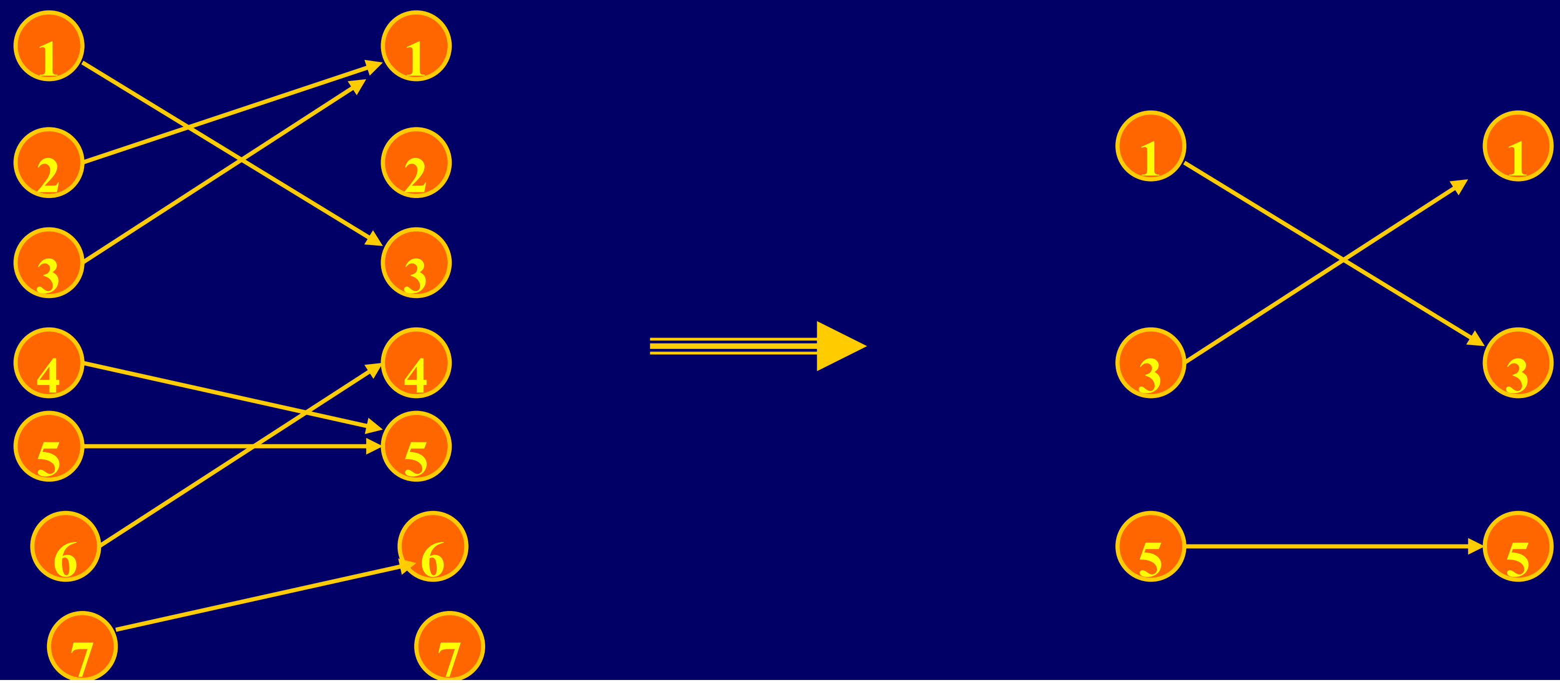
- **Given** a finite set A & a function *f* from A to itself

  Find a subset S of A with maximum number of elements

  Such that

(1) *f* maps S to itself

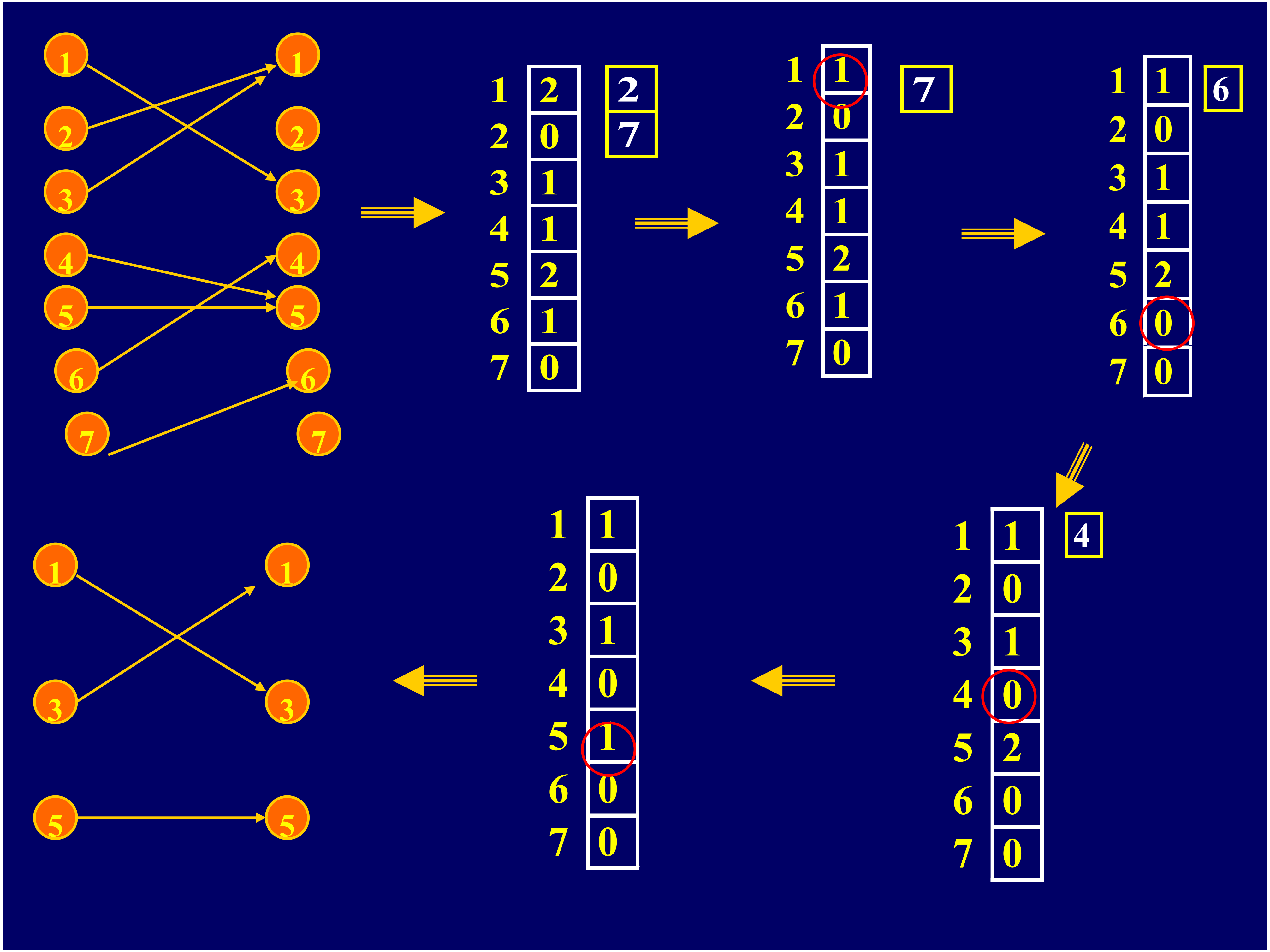
(2) *f* is one to one when restricted to S



Induction of One to One Mapping

- Hypothesis: solve problem for set of (n-1) elements
- Base:
- Induction:
  - any element i that has no other element mapped to it cannot belong to S
  - remove i, A'=A-{i}, A' has (n-1) elements

  * condition in A (n element) is the same as that in A'=A-{i} (n-1 element), except size
- Complexity: O(n)


1 2 3 4 5 6 7
1 2 3 4 5 6 7
1 2
2 0
3 1
4 1
5 2
6 1
7 0
2
7
1 1
2 0
3 1
4 1
5 2
6 1
7 0
7
1 1
2 0
3 1
4 1
5 2
6 0
7 0
6
1 1
2 0
3 1
4 0
5 2
6 0
7 0
4
1 1
2 0
3 1
4 0
5 1
6 0
7 0
1 3 5
1 3 5

The Celebrity Problem

- Celebrity: someone who is known by everyone

  but does not know anyone
- Celebrity problem

  Identify the celebrity by asking questions

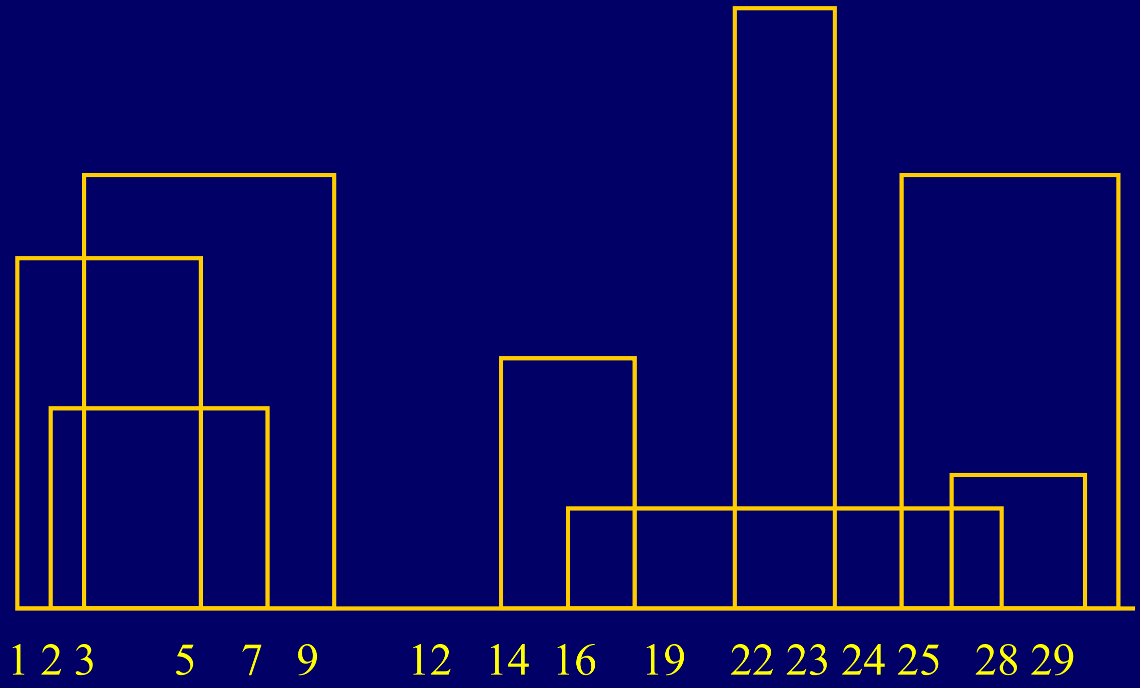
  “Do you know the person over there?”

  Goal: minimize the number of questions
- In graph theory: celebrity = sink

  sink: vertex with indegree (n-1) & outdegree 0

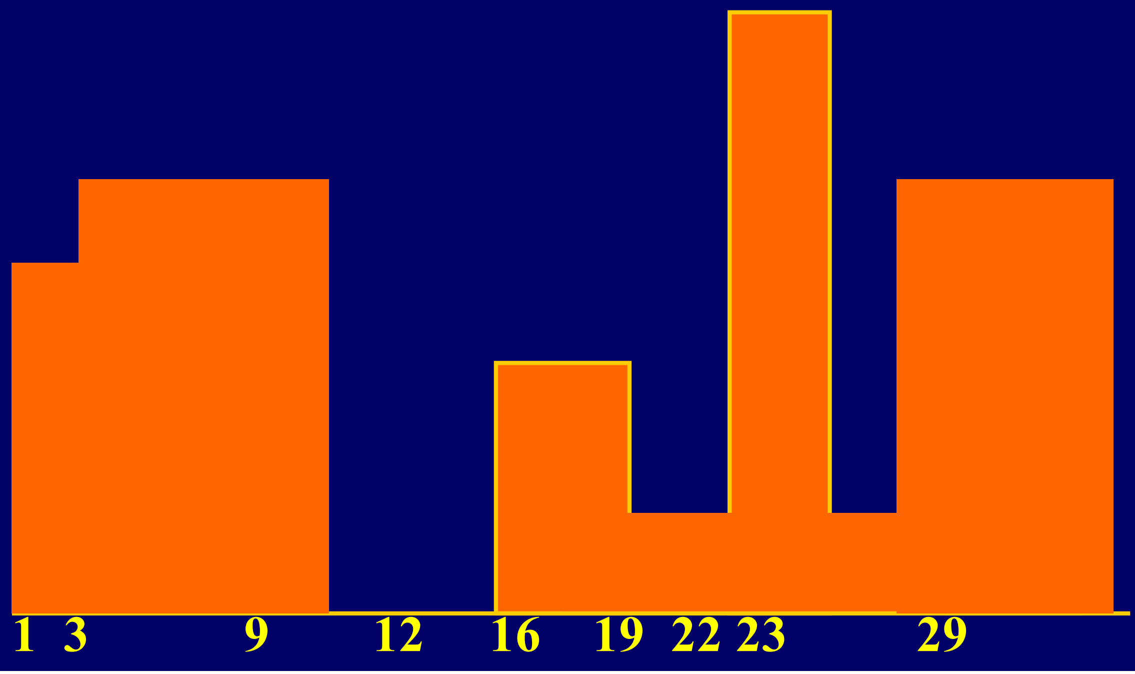
Induction

- Hypothesis: we know how to find celebrity among *n*-1 persons
  (if there exists, celebrity is among the (*n*-1) person)
- Induction:
  - eliminate someone who is non-celebrity (n -> (n-1))
    - ask someone X whether he/she knows Y
    - if X knows Y, X is not celebrity, eliminate X
    - if X does not know Y, Y is not celebrity, eliminate Y
  - three possibilities
    - Case 3: no celebrity among (n-1) persons
      no celebrity among n persons
    - Case 2: not exist (since celebrity is not the *n*-th person)
    - Case 1: two more questions to verify the celebrity among (n-1)



(左端,	高度,	右端)
(1,	11,	5)
(2,	6,	7)
(3,	13,	9)
(12,	7,	16)
(14,	3,	25)
(19,	18,	22)
(23,	13,	29)
(24,	4,	28)

(1,11,3,13,9,0,12,7,16,3,19,18,22,3,23,13,29,0)



Induction

- Hypothesis: we know how to solve for n/2 buildings
- Induction: from n/2 to n
  - merge two n/2 skylines: similar to add one building
  - merge: $O(n)$
- Algorithm
  - similar to merge sort
  - complexity: $T(n)=2*T(n/2)+O(n) = O(n\log n)$

Computing Balance Factors in Binary Tree

- Height: H(v)
- Balance factor: B(v)=|H(vl)-H(vr)|,

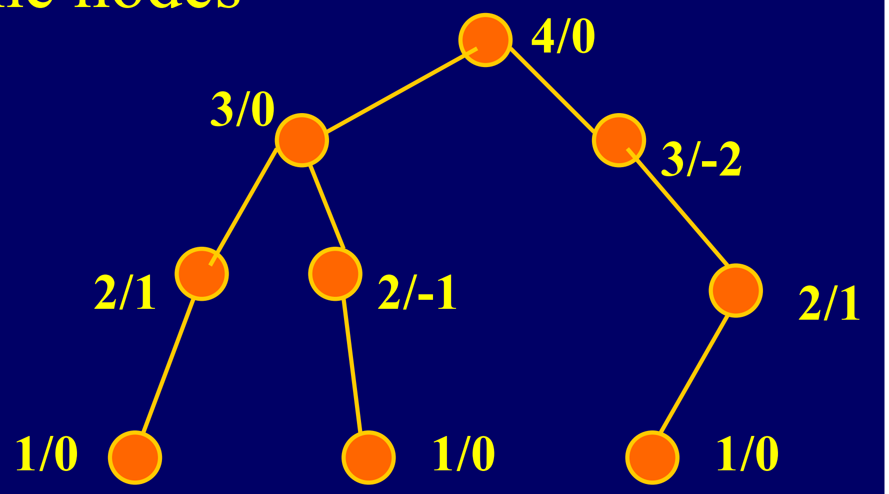
  * vl, vr: left, right children of v

  * AVL tree: balance factors of -1, 0, 1

- Problem

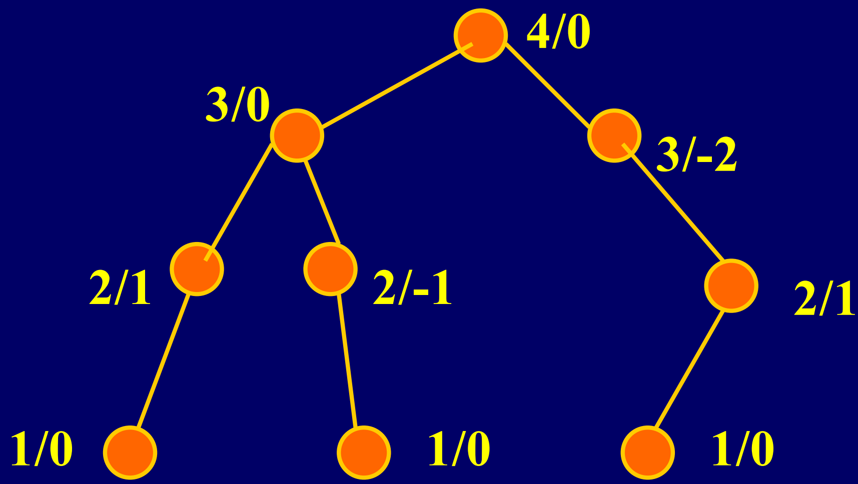
  Given a binary tree T with n nodes

  Compute the balance factors of all the nodes



Induction

- Hypothesis: we know how to compute balance factors & heights of all nodes in trees that have < n nodes
- Induction: from < n nodes to n nodes
  - Base
  - root:
    - calculate difference between heights of children
    - height: maximal height of two children + 1



Finding Maximum Consecutive Subsequence

- Subsequence (of consecutive elements)

  e.g. (3, -2, -3) is a subsequence of (2, -3, 1.5, -1, 3, -2, -3, 3)

- Maximum subsequence: maximum sum of subsequence

  e.g. maximum subsequence of (2, -3, 1.5, -1, 3, -2, -3, 3)=(1.5, -1, 3)

- Problem

  Given a sequence $x_1, x_2, \ldots, x_n$ of real numbers

  find a subsequence $x_i, x_{i+1}, \ldots, x_j$

  such that sum of the numbers in it is maximum over all subsequence of consecutive elements

Induction

- Hypothesis: we know how to find, in sequence of size < n
  - (1) maximum subsequence overall (global maximum)
  - (2) maximum subsequence that is a suffix (maximum suffix)
- Induction
  - If (x_n + maximum suffix) > global maximum,

    new global

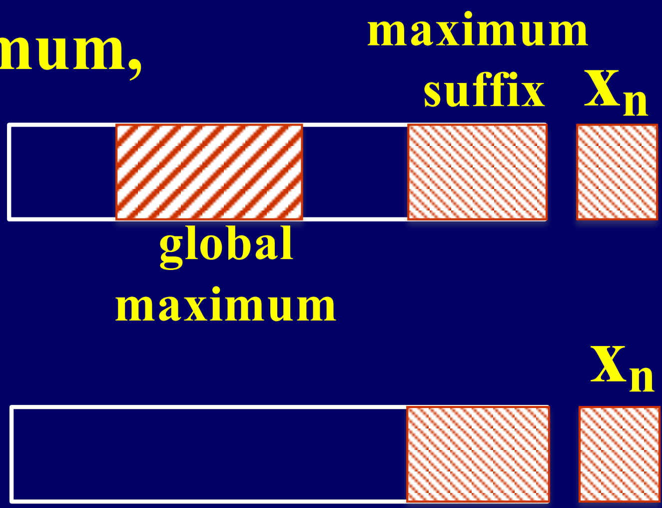
    Else retain previous global
  - maintain maximum suffix

    If maximum suffix + x_n <= 0,

    maximum suffix is empty

    Else (maximum suffix + x_n > 0)

    maximum suffix = maximum suffix + x_n



Knapsack Problem

- Given an integer *K*, *n* items of different size (the i-th item has an integer size k_i)

  Find a subset of the items

  Such that sizes sum to exactly K
- e.g. Given K=7, & 4 items of size {2, 3, 5, 6}

  Find a subset of the items

  Such that sizes sum to exactly 7

Induction

- **Hypothesis: we know how to solve P(n-1, k) for all 0≤ k ≤ K**
- **Induction:**
  - **reduce p(n, K) problem to P(n-1, K) & P(n-1, K-k_n)**

$$P(n,K)=\begin{cases} P(n-1,K) & \text{if n-th item is excluded} \\ P(n-1,K-k_n) & \text{if n-th item is included} \end{cases}$$