CSE 201 2-11-20

hw#4 #6d.)

find algorithm that K-sorts A[1....n] in time

$$\overline{T}(n) \sim n \log\left(\frac{n}{K}\right)$$

i.e. $\overline{T}(n) = n \log\left(\frac{n}{K}\right) + o(n \log n)$

continue... Dynamic Programming.

Recall: for $0 \le k \le n$

$$\binom{n}{k} = \begin{cases} 1 & k = 0 \text{ or } k = n \\ \binom{n-1}{k-1} + \binom{n-1}{k} & 0 < k < n \end{cases}$$

Proof.

Let S be a set with $|S| = n$.

Let $x \in S$ be arbitrary. So $|S - \{x\}| = n-1$.

Let k satisfy $0 < k < n$. The k-subsets of S fall into 2 classes. They are

(1) those that contain x,

and

(2) those that don't contain x.

To form a k-subset of S of type (1): Pick any $(k-1)$ elements of $S-\{x\}$, to that set, add x.

$$\#\text{ways to do} = \binom{n-1}{k-1}$$

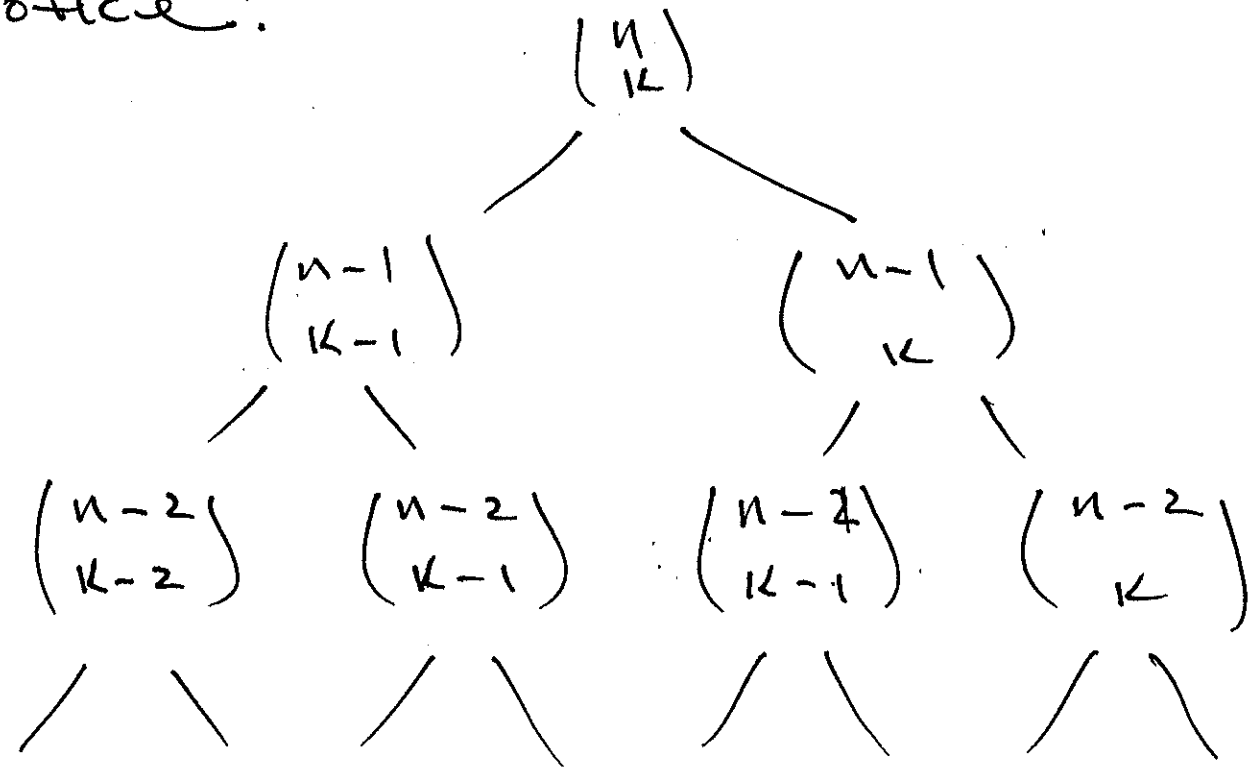
to form a k-subset of S of type (2): Pick any k elements from $S-\{x\}$.

$$\#\text{ways to do} = \binom{n-1}{k}$$

Thus

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$$

Notice:



Runtime $= \binom{n}{k}$

Recall: if $n = 2k$

$$\binom{n}{k} = \binom{2k}{k} = \Theta\left(\frac{4^k}{\sqrt{k}}\right)$$

- Recursive solution: Memoization
  Pass a table as fcn argument, check if each subinstance has already been computed.

- Iterative solution: Dynamic Programming

See DynBinCoef(n, k) on p. 2 of handout.

Runtime = $\Theta(nk)$

Usually Dynamic Programming is used to solve optimization Problems.

Problem: Coin changing.

Given unlimited supply of coins in denominations $\{d_1, d_2, \ldots, d_n\}$, and given an amount N of units to be disbursed determine how to Pay N using least # of coins.

Actually 2 problems

- find least # of coins needed to pay N units

  (Value of an opt. solution)

- find which (i.e. how many coins in each denomination) are necessary to achieve min. # of coins

  an optimal solution

Define 2-dimensional table

$$C[1 \ldots n;\ 0 \ldots N]$$

where $(1 \le i \le n,\ 0 \le j \le N)$

$C[i,j]$ = least # of coins necessary to pay j units using only coins from $\{d_1, \ldots, d_i\}$.

Goal: $C[n,N]$

We have 2 choices for $C[i,j]$:

(1) use no coins of denomination d_i. in this case, min. # coins is $C[i-1, i]$

(2) use at least one coin of denomination d_i. Then min # of coins is

$$1 + C[i, i-d_i]$$

Thus

$$C[i,i] = \min\left(C[i-1, i], 1 + C[i, i-d_i]\right)$$

Note: If either $i=1$ or $j<d_i$, then one or more of cells we need are out-of-bounds.

Define:

$$C[i,j] = +\infty$$

if either $i<1$ or $j<0$

Note: $C[i,0] = 0$ for $1 \le i \le n$

Ex. $n=4$, $d=(1,3,5,6)$, $N=8$

$d_1\ d_2\ d_3\ d_4$

i	d	0	1	2	3	4	5	6	7	8
		0	1	2	3	4	5	6	7	8
1	1	0	1	2	3	4	5	6	7	8
2	3	(0)	1	2	(1)	2	3	2	3	4
3	5	0	1	2	(1)	2	1	2	3	(2)
4	6	0	1	2	1	2	1	1	2	(2)

find optimal solution itself

Pay: 1 coin of Val. 3

1 ------ 5

Exercise

- write pseudo-code for the algorithm that fills the table. (p.4 of handout.)
- modify algorithm to deal with situation where some coins may run out. (hard)
- write a recursive algorithm that backtracks through the filled table to find optimal coin disbursment.
- modify recursive backtrack algorithm to print all optimal solutions (also hard)

Problem Discrete Knapsack

A thief wishes to steal n objects labeled 1 ---- n.

let

v_i = value of obj i

w_i = weight of obj i

for $1 \le i \le n$. Thief has a knapsack with capacity W.

Goal: steal max. poss. value of goods, not exceeding weight W.

Define bitstring

$$x = (x_1, x_2, x_3, \ldots, x_n)$$

where

$$x_i = \begin{cases} 1 & \text{if obj } i \text{ is stolen} \\ 0 & \text{" " " " not stolen} \end{cases}$$

Problem is to find $x \in \{0,1\}^n$, so as to

$$\text{maximize: } \sum_{i=1}^{n} x_i v_i$$

$$\text{subject to constraint: } \sum_{i=1}^{n} x_i w_i \leq W$$