EXERCISE:
MODIFY THIS ALGORITHM TO DEAL WITH THE SITUATION IN WHICH THE SUPPLY OF COINS IN SOME DENOMINATIONS IS LIMITED. LET $l[1..n]$ BE ANOTHER INPUT ARRAY, AND REQUIRE AT MOST $l[i]$ COINS OF TYPE i BE USED. NOTE $0 \le l[i] \le \infty$ FOR $1 \le i \le n$.

ONCE THE TABLE $C[1..n; 0..N]$ HAS BEEN FILLED THE SECOND PROBLEM CAN BE SOLVED, i.e. EXACTLY WHICH COINS ARE TO BE DISBURSED.

IF $C[i,j] = C[i-1,j]$, THEN NO COINS OF TYPE i ARE NEEDED TO PAY j UNITS WHEN RESTRICTED TO TYPES $\{1,..,i\}$. WE MOVE UP ONE ROW TO $C[i-1,j]$ TO SEE WHAT TO DO NEXT.

IF $C[i,j] = 1 + C[i, j-d_i]$, WE PAY OUT ONE COIN OF TYPE i THEN MOVE LEFT TO $C[i, j-d_i]$ TO SEE WHAT TO DO NEXT

IF $C[i,j]$ EQUALS BOTH $C[i-1,j]$ AND $1 + C[i, j-d_i]$ THEN EITHER ACTION IS ACCEPTABLE.

EXERCISE:

WRITE A RECURSIVE ALGORITHM WHICH GIVEN THE FILLED TABLE $C[1..n; 0..M]$, PRINTS A SEQUENCE OF $C[n,M]$ COIN TYPES WHOSE VALUE ADDS TO N. IF $C[n,M] = \infty$ PRINT AN APPROPRIATE MESSAGE.

THE 0-1 KNAPSACK PROBLEM

A THIEF WISHES TO STEAL n OBJECTS INDEXED $i = 1$ TO n. LET

V_i = VALUE OF OBJECT i
W_i = WEIGHT OF OBJECT i

THE THIEF HAS A KNAPSACK WHICH CAN CARRY A MAXIMUM WEIGHT OF W. HIS GOAL IS TO FILL THE KNAPSACK IN A WAY WHICH MAXIMIZES THE TOTAL VALUE OF THE GOODS STOLEN, WHILE RESPECTING ITS CAPACITY CONSTRAINT.

LET

$$x_i = \begin{cases} 0 & \text{IF OBJECT } i \text{ IS NOT TAKEN} \\ 1 & \text{IF OBJECT } i \text{ IS TAKEN} \end{cases}$$

THUS THE PROBLEM IS TO CHOOSE $x_i \in \{0,1\}$ $(1 \le i \le n)$ TO

$$\text{MAXIMIZE} \quad \sum_{i=1}^{n} x_i v_i$$

$$\text{SUBJECT TO} \quad \sum_{i=1}^{n} x_i w_i \le W$$

WHERE $v_i > 0$, $w_i > 0$, $W > 0$.

TO SOLVE THIS PROBLEM WE CREATE A TABLE $V[1..n; 0..W]$ WHERE $V[i,j]$ IS THE MAXIMUM VALUE OF THE OBJECTS IN THE SET $\{1,..,i\}$ WHOSE TOTAL WEIGHT DOES NOT EXCEED j. $(1 \le i \le n, 0 \le j \le W)$.

TO DETERMINE $V[i,j]$ WE HAVE IN GENERAL TWO ALTERNATIVES

- DO NOT INCLUDE OBJECT i. IN THIS CASE AT MOST VALUE $V[i-1,j]$ CAN BE STOLEN.
- INCLUDE OBJECT i. THIS INCREASES THE VALUE OF THE LOOT BY v_i, AND REDUCES THE REMAINING CAPACITY BY w_i. THUS IN THIS CASE AT MOST VALUE $v_i + V[i-1, j-w_i]$ CAN BE STOLEN.

Choosing the best alternative yields

$$V[i,j] = \max\left(V[i-1,j],\ v_i + V[i-1, j-w_i]\right)$$

Including boundary and out of bounds entries we have

$$V[i,j] = \begin{cases} 0 & i=0,\ j\geq 0 \\ \max\left(V[i-1,j],\ v_i + V[i-1, j-w_i]\right) & i>0,\ j\geq 0 \\ -\infty & j<0 \end{cases}$$

Ex. $n=5$, $W=10$

		0	1	2	3	4	5	6	7	8	9	10
$w_1=1$,	$v_1=1$	0	1	1	1	1	1	1	1	1	1	1
$w_2=3$,	$v_2=5$	0	1	1	5	6	6	6	6	6	6	6
$w_3=5$,	$v_3=12$	0	1	1	5	6	12	13	13	17	18	18
$w_4=6$,	$v_4=25$	0	1	1	5	6	12	25	26	26	30	31
$w_5=7$,	$v_5=30$	0	1	1	5	6	12	25	30	31	31	35

Exercise.

Write an algorithm which given the input arrays $v[1..n]$ and $w[1..n]$, and the weight limit W, fills in the table $V[1..n; 0..W]$ and returns $V[n,W]$. (Or if you prefer, return the whole table.)

EXERCISE

WRITE AN ALGORITHM WHICH GIVEN THE FILLED TABLE $V[1 \ldots n; 0 \ldots W]$ PRINTS OUT A LIST OF EXACTLY WHICH OBJECTS TO INCLUDE

THE PRINCIPLE OF OPTIMALITY

AN OPTIMIZATION PROBLEM SATISFIES THE PRINCIPLE OF OPTIMALITY IF THE OPTIMAL SOLUTION TO ANY (NON-TRIVIAL) INSTANCE IS A COMBINATION OF SOME OF ITS SUBINSTANCES.

i.e. AN OPTIMAL SOLUTION CONTAINS WITHIN IT OPTIMAL SOLUTIONS TO CERTAIN SUBPROBLEMS.

i.e. IN AN OPTIMAL SEQUENCE OF CHOICES, EACH SUBSEQUENCE IS ALSO OPTIMAL.

WE ALSO SAY THAT SUCH A PROBLEM EXHIBITS OPTIMAL SUBSTRUCTURE.

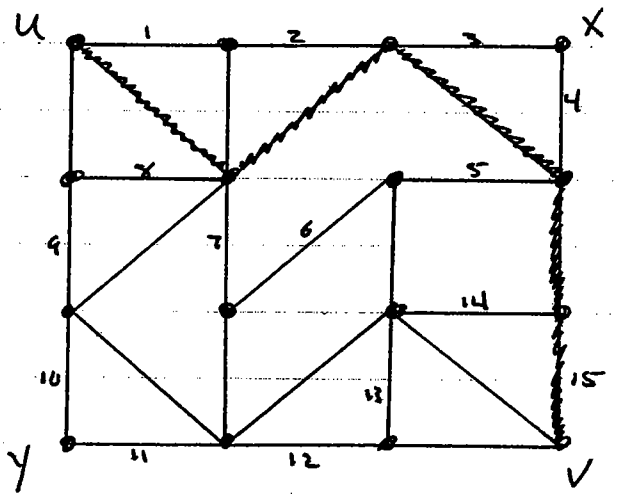
e.g. COIN CHANGE:

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

e.g. 0-1 KNAPSACK:

$$V[i,j] = \max(V[i-1,j],\ v_i + V[i-1, j-w_i])$$

EX. SHORTEST PATHS IN GRAPH



$d(u,v) = 5$

$\ell(u,v) = 15$

DEFN: A *u-v PATH* IS A SEQUENCE OF ALTERNATING VERTICES AND INCIDENT EDGES STARTING AT u AND ENDING AT v, IN WHICH NO VERTEX IS REPEATED (EXCEPT POSSIBLY $u=v$).

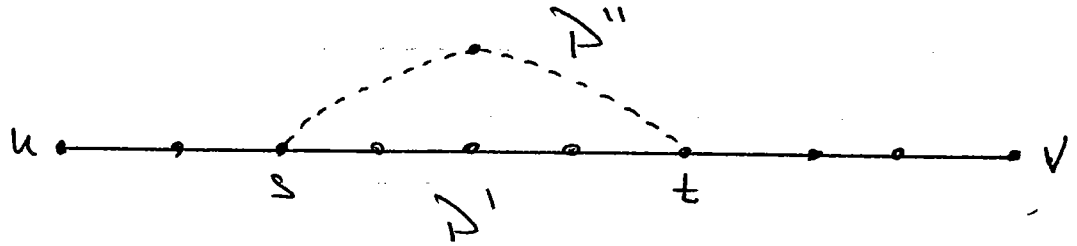
THE *LENGTH* OF A PATH IS THE NUMBER OF EDGES IN THE SEQUENCE

PROBLEM: DETERMINE A SHORTEST u-v PATH

LET $d(u,v)$ DENOTE THE LENGTH OF SUCH A PATH.

OBSERVE: ANY SEGMENT OF A SHORTEST PATH IS ALSO A SHORTEST PATH.

P:



PROOF: LET P BE A SHORTEST u-v PATH, s AND t TWO INTERMEDIATE VERTICES ON P, AND P' THE SEGMENT OF P FROM s TO t. IF P' WERE NOT A SHORTEST s-t PATH WE COULD SPLICE P' OUT OF P AND REPLACE IT WITH A SHORTER s-t PATH P''. THE RESULTING u-v PATH WOULD THEN BE SHORTER THAN P, CONTRADICTING THAT P WAS SHORTEST.

///

THUS THE SHORTEST PATH PROBLEM SATISFIES THE PRINCIPLE OF OPTIMALITY.

ONE MIGHT EASILY COME TO BELIEVE THAT ALL OPTIMALITY PROBLEMS EXHIBIT OPTIMAL SUB-STRUCTURE, BUT THIS IS FALSE. IF A PROBLEM FAILS TO SATISFY THE OPTIMALITY PRINCIPLE, IT IS PROBABLY IMPOSSIBLE TO SOLVE IT USING DYNAMIC PROGRAMMING.

EX. LONGEST PATH IN GRAPH

PROBLEM: DETERMINE A LONGEST u-v PATH

LET $l(u,v)$ DETERMINE THE LENGTH OF SUCH A PATH.

OBSERVE THAT A SEGMENT OF A LONGEST u-v PATH MAY NOT BE A LONGEST PATH. IN THE PREVIOUS EXAMPLE ONE CHECKS THAT $l(x,y) = 8$, BUT A LONGEST u-v PATH TRAVELS FROM x TO y IN 7 STEPS.

THIS PROBLEM THEREFORE VIOLATES THE PRINCIPLE OF OPTIMALITY.

GENERAL PROCEDURE

1.) CHARACTERIZE THE STRUCTURE OF AN OPTIMAL SOLUTION. (i.e. SHOW THAT YOUR PROBLEM INVOLVES SOME CHOICE(S) WHICH LEAVE ONE OR MORE SUBPROBLEMS.)

2.) RECURSIVELY DEFINE AN OPTIMAL SOLUTION (i.e. IN TERMS OF OPTIMAL SUBPROBLEM SOLUTIONS.)

3.) COMPUTE THE VALUE OF AN OPTIMAL SOLUTION IN A BOTTOM UP FASHION (i.e. CONSTRUCT TABLE)

4.) CONSTRUCT AN OPTIMAL SOLUTION (USING THE TABLE GENERATED IN (3).)