THE WEIGHT OF A DIRECTED i-j PATH ($i, j \in V$) IS THE SUM OF THE WEIGHTS OF EACH OF ITS DIRECTED EDGES.

PROBLEM: (APSP)
FOR EACH PAIR $(i,j) \in V \times V$, DETERMINE AN i-j PATH OF MINIMUM WEIGHT. (ALSO CALLED A SHORTEST PATH.)

AGAIN THERE ARE REALLY TWO PROBLEMS

- DETERMINE THE MINIMUM PATH WEIGHTS FOR EACH (i,j)
- DETERMINE SHORTEST i-j PATHS.

WE CONCENTRATE ON THE FIRST PROBLEM, LEAVING THE SECOND AS AN EXCERCISE.

Floyd-Warshall Algorithm

AN INTERMEDIATE VERTEX OF A DIRECTED PATH $P = (v_1, v_2, \ldots, v_\ell)$ IS ANY VERTEX OTHER THAN v_1 OR v_ℓ, i.e. ONE OF THE VERTICES $\{v_2, \ldots, v_{\ell-1}\}$.

LET $G=(V,E)$ BE A DIRECTED GRAPH WITH $V=\{1,2,\ldots,n\}$. DEFINE SUBSETS V_K OF V AS FOLLOWS

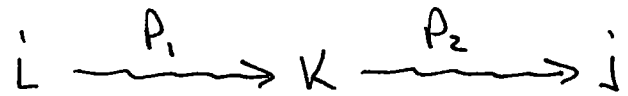
$$V_K = \begin{cases} \emptyset & K=0 \\ \{1,2,\ldots,K\} & 1 \le K \le n \end{cases}$$

LET $(i,j) \in V \times V$ AND $1 \le K \le n$. LET P DENOTE A MINIMUM WEIGHT PATH AMONGST ALL i-j PATHS WITH INTERMEDIATE VERTICES IN V_K.

NOW OBSERVE THAT WE HAVE TWO ALTERNATIVES

- K IS NOT AN INTERMEDIATE VERTEX OF P. IN THIS CASE P IS ALSO OF MINIMUM WEIGHT AMONGST ALL i-j PATHS WITH INTERMEDIATE VERTICES IN V_{K-1}.

- K IS AN INTERMEDIATE VERTEX OF P. WE CAN DECOMPOSE P INTO SUBPATHS P_1 AND P_2:



NOTE VERTEX K IS NOT INTERMEDIATE TO EITHER P_1 OR P_2.

THUS P_1 HAS MINIMUM WEIGHT AMONGST ALL i-k PATHS WITH INTERMEDIATE VERTICES IN V_{k-1}, AND LIKEWISE P_2 HAS MINIMUM WEIGHT AMONGST ALL k-j PATHS WITH INTERMEDIATE VERTICES IN V_{k-1}.

THESE OBSERVATIONS SHOW APSP EXHIBITS OPTIMAL SUBSTRUCTURE, NECESSARY FOR DYNAMIC PROGRAMMING.

LET $d_{ij}^{(k)}$ DENOTE THE WEIGHT OF A MINIMUM WEIGHT i-j PATH WITH ALL INTERMEDIATE VERTICES IN V_k.

WHEN $k=0$, SUCH A PATH HAS NO INTERMEDIATE VERTICES, HENCE AT MOST ONE EDGE. THUS $d_{ij}^{(0)} = w_{ij}$.

THE ABOVE OBSERVATIONS SHOW THAT FOR $1 \le k \le n$ WE HAVE

$$d_{ij}^{(k)} = \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right).$$

LET $D^{(k)}$ DENOTE THE MATRIX $\left(d_{ij}^{(k)}\right)$. THEN WE SEEK $D^{(n)}$ GIVEN $D^{(0)} = W$.

FLOYD-WARSHALL (W)

```
1.) n ← rows[W]
2.) D^(0) ← W
3.) for k ← 1 to n
4.)     for i ← 1 to n
5.)         for j ← 1 to n
6.)             d_ij^(k) ← min(d_ij^(k-1), d_ik^(k-1) + d_kj^(k-1))
7.) return D^(n)
```

SINCE (6) TAKES TIME $O(1)$, FLOYD-WARSHALL RUNS IN TIME $\Theta(n^3)$.

NOTE THE ABOVE ALGORITHM ALSO USES MEMORY n^3. IT IS POSSIBLE TO ACCOMPLISH THIS WITH JUST n^2 MEMORY (EXERCISE.)

TO CONSTRUCT SHORTEST PATHS WE COULD USE $D = D^{(n)}$ TO DETERMINE THE PREDECESSOR MATRIX $\Pi = (\pi_{ij})$, WHERE

π_{ij} = PREDECESSOR OF j ALONG A SHORTEST i-j PATH

ALTERNATIVELY WE COULD DETERMINE INTERMEDIATE PREDECESSOR MATRICES $\Pi^{(k)} = (\pi_{ij}^{(k)})$ $(0 \le k \le n)$

$\pi_{ij}^{(k)}$ = PREDECESSOR OF j ALONG A SHORTEST i-j PATH AMONGST THOSE WITH INTERMEDIATE VERTICES IN V_k.

(SEE P. 632 FOR DETAILS.)

EXERCISE

- RUN FLOYD-WARSHALL ON THE WEIGHTED DIGRAPH IN PRECEEDING EXAMPLE.
- WRITE AN ALGORITHM TO DETERMINE Π FROM $D = D^{(n)}$
- ALTER FLOYD-WARSHALL TO BUILD $\Pi^{(k)}$ $(0 \le k \le n)$ AS YOU GO.
- WRITE AN ALGORITHM TO PRINT A SHORTEST i-j PATH GIVEN $\Pi = \Pi^{(n)}$.

READ

- LONGEST COMMON SUBSEQUENCE (15.4)
- OPTIMAL BINARY SEARCH TREES (15.5)

Greedy Algorithms

Ex. Continuous Knapsack

As before we seek to

(1) Maximize $\sum_{i=1}^{n} x_i v_i$

(2) subject to $\sum_{i=1}^{n} x_i w_i \leq W$

where $W > 0$, $v_i > 0$, and $w_i > 0$ $(1 \leq i \leq n)$. However instead of $x_i \in \{0, 1\}$ we now allow $0 \leq x_i \leq 1$ for $1 \leq i \leq n$.

A vector $x = (x_1, \ldots, x_n)$ will be called a <u>feasible solution</u> if (2) is satisfied without regard to the optimality condition (1).

A <u>greedy strategy</u> consists of making a locally optimal (greedy) choice, then solving the subproblem arising from this choice.

In this problem that means including the "best" object which does not exceed the capacity constraint, then doing the same thing with the remaining objects and remaining capacity.

```
Knapsack (v, w, W)
1.) n ← length[v]
2.) for i ← 1 to n
3.)       x[i] ← 0
4.) weight ← 0
5.) while weight < W
*6.)      i ← THE "BEST" REMAINING OBJECT
7.)       if weight + w[i] ≤ W
8.)             x[i] ← 1
9.)             weight ← weight + w[i]
10.)            mark i AS INCLUDED.
11.)      else
12.)            x[i] ← (W - weight) / w[i]
13.)            weight ← W
14.) return x
```

GREEDY LOOP

THE "BEST" OBJECT IN (6) CAN BE INTERPRETED IN SEVERAL WAYS.

IN GENERAL WE DEFINE A <u>SELECTION FUNCTION</u> $f(i)$ WHICH ENCODES THE DESIRABILITY OF OBJECT i. LINE (6) THEN MAXIMIZES f OVER ALL REMAINING (UNMARKED) OBJECTS.

SOME POSSIBLE CHOICES FOR f IN THIS EXAMPLE ARE

- $f(i) = v_i$ (PROCESS OBJECTS IN ORDER OF DECREASING VALUES IN GREEDY LOOP.)
- $f(i) = \frac{1}{w_i}$ (INCREASING WEIGHTS)
- $f(i) = \frac{v_i}{w_i}$ (DECREASING VALUE TO WEIGHT RATIO.)

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

i	1	2	3	4	5
v_i	2	3	6.6	4	6
w_i	1	2	3	4	5
v_i/w_i	2	1.5	2.2	1	1.2

$f(i)$	x					VALUE
v_i	0	0	1	.5	1	14.6
$1/w_i$	1	1	1	1	0	15.6
v_i/w_i	1	1	1	0	.8	16.4

IN THIS EXAMPLE (AT LEAST) $f(i) = v_i/w_i$ GIVES THE BEST RESULT. IS THIS OPTIMUM?