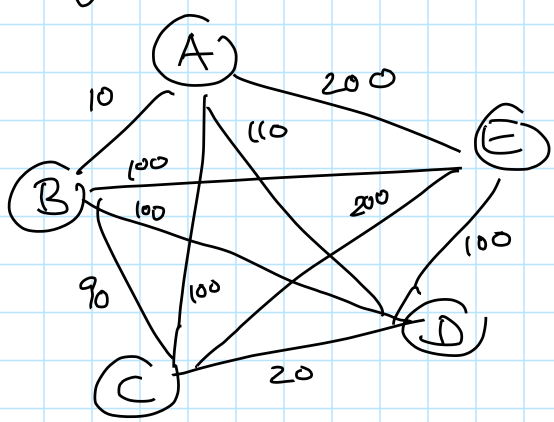
CS 340 - lecture 32 - Nov 30

Greedy algorithms

... are not always guaranteed to find optimal / correct solution.

Example 5.3

(a) finding the shortest Traveling Salesman tour:



greedy choice:

	cost
(A, B)	10
(B, C)	90
(C, D)	20
(D, E)	100
~~(A, E)~~ (E, A)	200
	420

cost of [A, B, E, D, C, A] : 330 → cheaper!

→ greedy method does not produce optimal solution

Theorem: For each number of vertices, there is a choice of edge weights such that the greedy method produces the unique worst TS tour!

(b) coin change problem, if coins are valued

15, 10, 1

give 20 cents change using as few coins as possible.

optimal solution: 2 coins (2×10)

greedy method: 6 coins (1×15, 5×1)

greedy approximation solution

for some problems, all known algorithms producing optimal solution are too inefficient, but a greedy algorithm achieves a "fairly good" suboptimal solutions efficiently.

6.3. Divide-and-Conquer Algorithms

divide: solve smaller problem instances recursively

conquer: combine solutions to smaller problem instances to a solution to a solution to the original instance

e.g., Mergesort, Quicksort

analysis of D & C algorithms: typically recurrence relations...

6.4 Dynamic Programming

sometimes recursive solutions are natural, yet inefficient

Example 54. (cf. Ex. 2) Fibonacci numbers

$$F(1) = F(2) = 1, \quad F(n) = F(n-1) + F(n-2) \text{ for } n > 2$$

recursive program very inefficient!

idea: rather than re-computing values several times, use memory to store such values in tables, then re-access them as needed.

record the two most recent Fibonacci number in a "table"

Example 55. All – Pairs Shortest Paths

(10.3.4 in book)

given: weighted directed graph $G=(V,E)$ with cost function c

assume G has no negative-cost cycles

$c(v,w) = \infty$ if $(v,w) \notin E$

task: for each pair of vertices $v,w \in V$, compute a shortest path from v to w. (if one exists)

Dijkstra's alg. solves the "single-source" version of this problem.

updates: if u known, w unknown, $(u,w) \in E$:

```
if (w.distance > u.distance + c(u,w))
{
    w.distance = u.distance + c(u,w);
    w.previous = u;
}
```

$\rightsquigarrow$ $\boxed{w.\text{distance} = \min\{w.\text{distance},\ u.\text{distance} + c(u,w)\}}$

w.distance refers to distance of w from source

NOW we need $\text{dist}(v,w)$ for any v,w.

Let $V = \{v_1, \dots, v_n\}$

idea (0) compute SP from v to w without intermediate states

i.e., if $(v,w) \in E$, then $[v,w]$ cost $c(v,w)$
else "no path" cost ∞

$$\rightsquigarrow \text{dist}_{(0)}(v,w) = \begin{cases} \text{cost } c(v,w) \\ \infty \end{cases}$$

(1) compute SP from v to w with intermediate state v_1 allowed

$\rightsquigarrow$ $dist_{(1)}(v,w)$

(2) compute SP from v to w, allowing intermediate states v_1, v_2

$\rightsquigarrow$ $dist_{(2)}(v,w)$

(3) allow interm. states v_1, v_2, v_3

$\rightsquigarrow$ $dist_{(3)}(v,w)$ ...

$\vdots$

(n) allow all states as intermediate states

$\rightsquigarrow$ $dist_{(n)}(v,w) = dist(v,w)$

$$dist_{(0)}(v,w) = c(v,w)$$

$$dist_{(1)}(v,w) = \min\{dist_{(0)}(v,w),\ dist_{(0)}(v,v_1) + dist_{(0)}(v_1,w)\}$$

$$dist_{(2)}(v,w) = \min\{dist_{(1)}(v,w),\ dist_{(1)}(v,v_2) + dist_{(1)}(v_2,w)\}$$

$$\vdots$$

$$dist_{(i+1)}(v,w) = \min\{dist_{(i)}(v,w),\ dist_{(i)}(v,v_{i+1}) + dist_{(i)}(v_{i+1},w)\}$$

$\rightsquigarrow$ keep a single $|V| \times |V|$ matrix for storing dist values, update in stages

see below for pseudocode

```
for all v ∈ V
    for all w ∈ V
        {
            dist(v,w) = c(v,w);        // this is ∞ if no edge
                                          from v to w exists
            path(v,w) = EMPTY;
        }
for all v_i ∈ V
    for all v ∈ V
        for all w ∈ V
            if (dist(v,w) > dist(v,v_i) + dist(v_i,w))
                {
                    dist(v,w) = dist(v,v_i) + dist(v_i,w);
                    path(v,w) = v_i;
                }
```

$\rightarrow$ running time $O(|V|^3)$