Advanced Algorithms

Lecture 7: Dynamic programming (cont.)

Announcements

- HW 2 is out! Due next Friday — start early!
- <u>Policy about citation</u>

(cite all sources for hints, etc.).

Last lecture: subset sum

Problem: given n **non-negative** integers A[0], A[1], …, A[n-1] and a "target" S, find if there is a subset of the A[i] that add up to S

- Wrote down recursive algorithm:
  - A[0] is either included or not — split into two "sub-trees"/sub-problems
  - Key obs: if S is small, same sub-problem is solved many times
  - Store answers, look-up before computing!

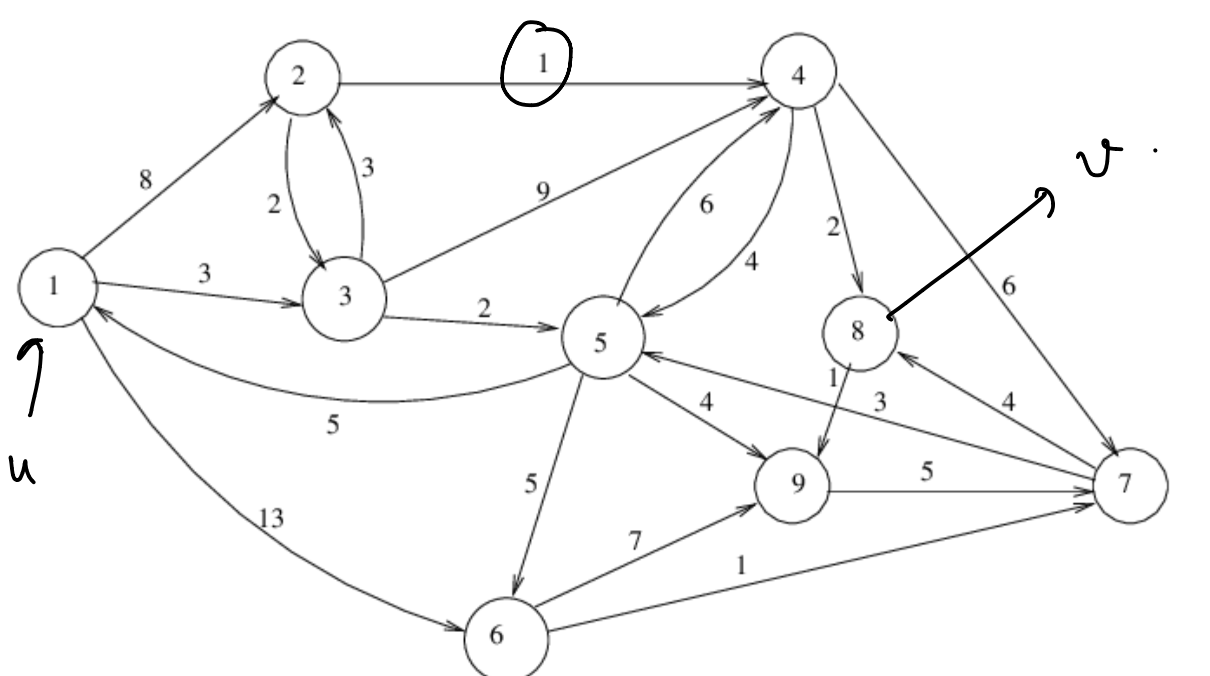
running time = $n \cdot S$.

(memoization).

Last lecture: shortest path

(See notes).

Problem: given a directed graph G = (V, E), two nodes u, v, and a parameter *L*, find the shortest path with *L* "hops" from u —> v



- Recursive algorithm:
  - Any L-hop path == (L-1)-hop to a neighbor of v + one edge
  - Key obs: only a small number of distinct sub-problems $\leq n \cdot (L+1)$
  - Again, look-up before making call, and store answers!

$n \cdot m$.

running time: $(L+1)|E|$; memory $= n(L+1)$.

Common features

ϕ ✓ × ✓ | × × . . .

Sequence of decisions to be made ; "reward" only in the end.

- Sequential decision making
- Some resource "depleting" (steps remaining / #remaining)
- **Key:** past decisions lead to some "state"; we can then solve sub-problem starting at the state (ignoring past)

[not true in some applications — MDPs.]

More examples

- Cake-eating
- Traveling salesman problem

Eating schedule

Problem: given *k* pieces of cake, figure out how to maximize "total satisfaction". Constraints: ....

$\beta = 0.8$

Day 1 ; Day 2 ; Day 3 ; . . .

given k pieces $\quad k - n_1$

$\underline{\underline{n_1}} \quad n_2 \quad . \quad .$

$\log(1+n_1) \quad \beta \log(1+n_2) \quad \beta^2 \{\log(1+n_3) \quad . . .$

First come up with a recursive alg ; remember answers.

Common features

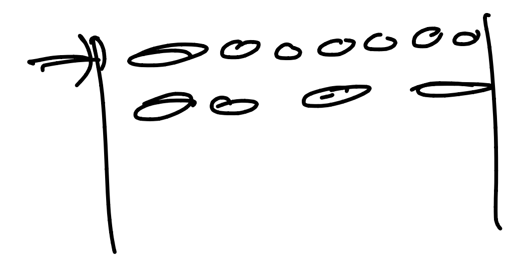
- Sequential decision making
- Some resource "depleting" (steps remaining / #remaining)
- <u>**Key:**</u> past decisions lead to some "state" (that defines sub-problem); we can then solve sub-problem (ignoring past); if # of sub-problems is small, can "store answers"

Other examples

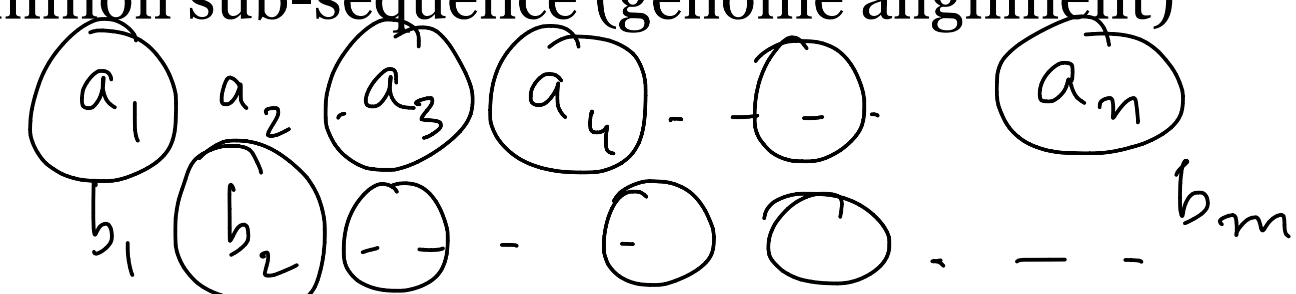
- fib(n) = fib(n-1) + fib(n-2)

define fib(n):
    return fib(n-1) + fib(n-2);

- Type-setting text (how does LaTeX split words across lines?)



- Longest common sub-sequence (genome alignment)



- Control theory, optimization, scheduling, ...

(see course web-page).

Common issues

(HW2, HW3 ...)

- Defining the "right" sub-problems
- Memory usage! (usually no "smooth" way to trade-off against accuracy) → subset sum used $n \cdot S$ space.
- Recursion "done right"

no good way to overcome this in general!

$nS^{1/2}$ space

$n^{1/2}$

2 ..

— Some problems have strong lower bounds.

Example: traveling salesmen

2^n

Naive soln: time = $O(n!)$; Space : n^2

Problem: suppose we have *n* "cities" with d_{ij} being distance between cities *i* and *j*. Salesman starts at city 1 (home), needs to travel to each city precisely once and return home. The goal is to minimize the total distance traveled.

1, 5, 4, 3, 2, 1

1, 2, 4, 3, 5, 1

$(p_1, p_2, \ldots, p_n)$

- Naive solution?

↳ for each permutation of $1, \ldots, n$, compute the total length to travel in the order given by p.

$n!$

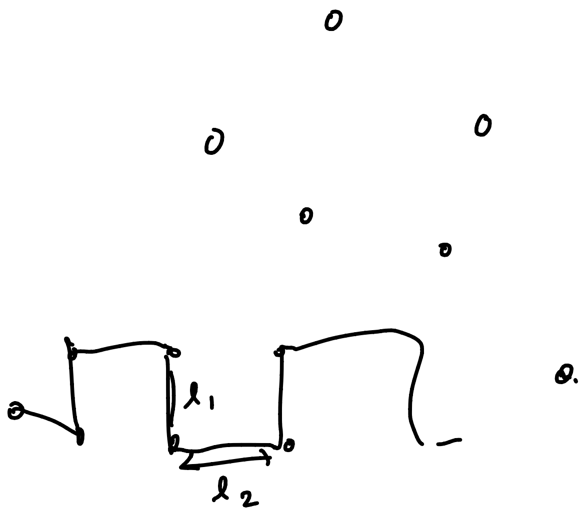
$d_{p_1 p_2} + d_{p_2 p_3} + \ldots + d_{p_n p_1}$

Candidate 1: Greedy strategy:

visit the closest unvisited node.

↓

turns out it can lead to sub-optimal tours.



$l_2 > l_1$

Recursive formulation?

Sub-problem in words:

starting at u, visit all the vertices in S precisely once, then go to "1": - minimize total cost

TSP-Sub(u, S)

* Key idea: Sub-problem must involve the "set of unvisited vertices". $\rightarrow S$
* must remember the "start" $\rightarrow 1$.
* where are we at? $\rightarrow u$.
* what is the total cost so far?

TSP-Sub(u, S) :

if $S = \phi$, return dist$(u, 1)$.

captures return to start.

for all $v \in S$:

compute val = dist(u, v) +

TSP-Sub$(v, S \setminus \{v\})$.

- look-up -

return the smallest of the 'val' values.

\# of sub-problems?

Defining sub-problems

– Defined by $(u, \boxed{S})$

u: current vertex

S: unvisited set.

Initially, $u = 1$, $S = \{2, \ldots, n\}$

# of candidates for u: precisely n.

# of candidates for S: $|\text{Subsets of } \{2, 3, \ldots, n\}|$

$= 2^{n-1}$

sub-problems $= n \cdot 2^{n-1} \ll (n-1)!$

Running time

Total running time = $O(n^2 \cdot 2^n)$.

Correctness