tutorial #7 - sketch

Monday, October 30, 2017 8:14 PM

Greedy Algorithms

- make choice that looks best at the moment
- makes locally optimal choice in hopes of coming to a globally optimal solution
- does not always produce the optimal sol^n, sometimes it does

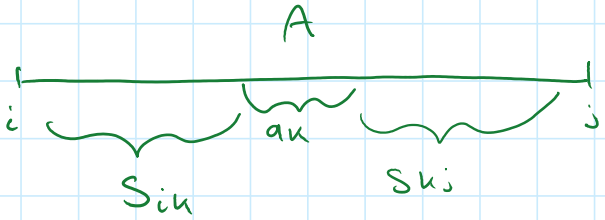
Activity Selection Problem

Goal: select max size subset of mutually compatible activities

S_{ij} : set of activities that start after activity a_i finishes and that finish before activity a_j starts.

↳ want the max set of mutually compatible activities in S_{ij}

A_{ij} is that set: contains a_k



$$A_{ij} = S_{ik} \cup \{a_k\} \cup S_{kj}$$

$$|A_{ij}| = |A_{ik}| + |A_{kj}| + 1 \text{ activities}$$

Dynamic perspective:

$$c[i,j] = c[i,k] + c[k,j] + 1$$

→ must examine all activities in S_{ij}

$$c[i,j] = \begin{cases} 0 & \text{if } S_{ij} = \emptyset \\ \end{cases}$$

$$c[i,j] = \begin{cases} 0 & \text{if } S_{ij} = \emptyset \\ \max_{a_k \in S_{ij}} \{c[i,k] + c[k,j] + 1\} & \text{if } S_{ij} \neq \emptyset \end{cases}$$

→ have to fill out the table in a similar fashion to the longest common sequence.

Lets be greedy:

- select an activity to add to our optimal soln w/o solving all the subproblems.

→ select activity that leaves resource available for as many other activities as possible
↳ select a_1 since they are sorted by finish time

→ all compatible activities must start after a_1 finishes.

all activities after a_k finishes: $S_k = \{a_i \in S : s_i \geq f_k\}$

→ if we greedy select a_1, then S_1 remains only problem to solve

Thm 16.1: Consider any nonempty subproblem S_k and let a_m be an activity in S_k w the earliest finish time. Then a_m is included in some max-size subset of mutually compatible activities of S_k

So far...

- Repeatedly choose activity that finishes 1st
- keep only activities compatible (no-overlap)
- repeat until no activities remain.

Top down: . put activity in optimal soln
↳ . solve what is left over.
↳ typical for greedy algs

s: start times array
f: finish times

k: S_k subproblem

n: size of original problem

```
RECURSIVE-ACTIVITY-SELECTOR(s, f, k, n)
1  m = k + 1
2  while m ≤ n and s[m] < f[k]      // find the first activity in S_k to finish
3      m = m + 1                    looking for s_m ≥ f_k
4  if m ≤ n
5      return {a_m} ∪ RECURSIVE-ACTIVITY-SELECTOR(s, f, m, n)
6  else return ∅
```

Θ(n)

add a_0 to solution, S_0 is remaing set of activities.

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

$\{a_1\} \cup$ Rec-Act-Sel$(s, f, 1, 11)$ S_1

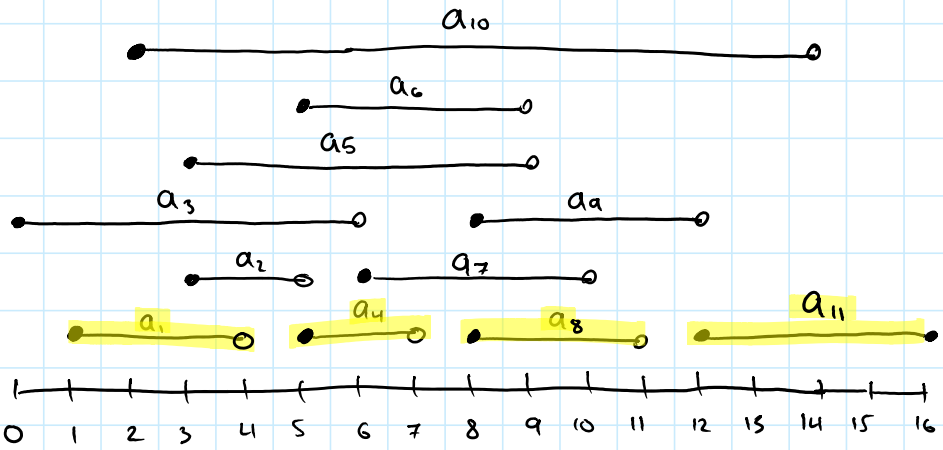
$\{a_1, a_4\} \cup$ Rec-Act-Sel$(s, f, 4, 11)$ S_4

$\{a_1, a_4, a_8\} \cup$ Rec-Act-Sel$(s, f, 8, 11)$ S_8

$\{a_1, a_4, a_8, a_{11}\} \cup$ Rec-Act-Sel$(s, f, 11, 11)$ S_{11}

$\emptyset$

Final answer = $\{a_1, a_4, a_8, a_{11}\}$



Iterative Version:

```
Greedy Act Selector (s, f):
    n = s.length
    A = {a_1}
    k = 1
    for m = 2 to n:
        if s[m] ≥ f[k]
            A = A ∪ {a_m}
            k = m
    return A
```

$\Theta(n)$