Math 182 Lecture 13

Chapter 7 Greedy Algorithms

§7.1 An activity selection problem

Greedy: making locally optimal choice

Standard example: cashier making change.

$S = \{a_1, \dots, a_n\}$ proposed activities

a_i start time s_i finish time f_i

$0 \le s_i < f_i < \infty$

a_i takes place during $[s_i, f_i)$

We say a_i, a_j compatible if

$[s_i, f_i) \cap [s_j, f_j) = \emptyset$.

The **activity-selection problem**, we want to find a maximum-size subset of $S = \{a_1, \dots, a_n\}$ of mutually-compatible activities. Note that we want to maximize the *number* of activities scheduled, not necessarily the total *duration* of the activities which are scheduled.

By convention we'll assume

$f_1 \le f_2 \le \cdots \le f_{n-1} \le f_n$.

Example:

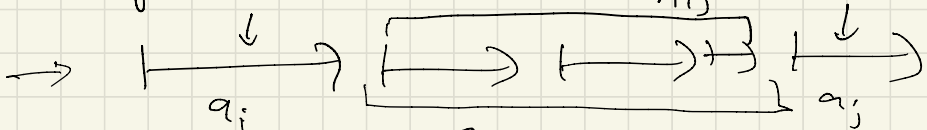
i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	8	8	10	11	12	14	16

$\{a_3, a_9, a_{11}\}$ mut. comp. size 3

$\{a_1, a_4, a_8, a_{11}\}$ mut. comp size 4

or $\{a_2, a_4, a_9, a_{11}\}$ optimal sols.

Optimal substructure $S = \{$all activities$\}$

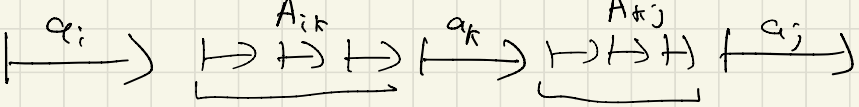


Let $S_{ij} = \{$all activities starting after a_i and ending before $a_j\}$

Want activities between a_i and a_j to be optimal sol to S_{ij}.

A_{ij} = optimal sol to S_{ij}.

Further sps A_{ij} contains a_k



define $A_{ik} := A_{ij} \cap S_{ik}$

$A_{kj} := A_{ij} \cap S_{kj}$

Then A_{ik}/A_{kj} optimal sol to S_{ik}/S_{kj}.

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

$$\rightsquigarrow |A_{ij}| = |A_{ik}| + |A_{kj}| + 1.$$

Let $c[i,j] :=$ size of optimal sol'n of S_{ij}

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

(This would work although we can do better bcs we can say which k gives us a max)

Making greedy choice

Sps we wanted to commit to one activity which is guaranteed to be part of some optimal sol'n.

Idea: commit to activity w/ earliest finish time.

Need to show activity w/ earliest stopping time *always* part of *some* optimal solution.

Define: $S_k := \{a_i \in S : s_i \geq f_k\}$

$= \{$all activities which start after a_k finishes$\}$.

Theorem Consider any nonempty subproblem S_k, and let a_m be an activity in S_k w/ earliest finish time.

Then a_m is included in some maximize-size subset of mutually compatible activities of S_k.

Proof: "Exchange argument"

(Idea: the greedy choice is just as good as any other choice which is part of optimal sol.)

Let $A_k \subseteq S_k$ be maximum size compatible subset. Let $a_j \in A_k$ be activity w/ first start time.

Case 1: $a_j = a_m$. Done ✓.

Case 2: $a_j \neq a_m$. Consider "exchanged" set $A'_k := A_k \setminus \{a_j\} \cup \{a_m\}$.

disjoint = mut. comp.

A'_k activities are disjoint bcs activities in $A_k \setminus \{a_j\}$ are disjoint. Also since $f_m \leq f_j \leq s_i \; \forall a_i \in A_k \setminus \{a_j\}$ a_m is disjoint w/ $A_k \setminus \{a_j\}$.

Also $|A_k| = |A'_k|$ thus A'_k also optimal sol. which contains a_m. ▨

Remark: Since we know a choice guaranteed to work, better to implement as top-down instead of bottom-up.

Recursive greedy algorithm

$s = \langle 0, s_1, s_2, \ldots, s_n \rangle$

$f = \langle 0, f_1, f_2, \ldots, f_n \rangle$

n = # of original activities

k = index for subproblem S_k to solve.

```
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
4  if m ≤ n
5        return {a_m} ∪ RECURSIVE-ACTIVITY-SELECTOR(s, f, m, n)
6  else return ∅
```

(call Rec-Act-Sel$(s, f, 0, n)$ will find optimal sol for original problem S)

Line 1 we first consider activity a_{k+1}.

Lines 2-3 iterate through $a_{k+1}, \ldots, a_m$ until find comp. activity w/ earliest start time

If this happens, make this greedy choice in line 5.

O/w if no activities left, return $\emptyset$.

Running time: $\Theta(n)$ bcs m takes values $1, \ldots, n+1$ exactly once across all recursive calls.

Iterative greedy algorithm

$s = \langle s_1, \ldots, s_n \rangle$
$f = \langle f_1, \ldots, f_n \rangle$

GREEDY-ACTIVITY-SELECTOR(s, f)

```
1  n = s.length
2  A = {a_1}
3  k = 1
4  for m = 2 to n
5       if s[m] ≥ f[k]
6            A = A ∪ {a_m}
7            k = m
8  return A
```

$\Theta(n)$

Line 2 make first greedy choice
Line 3 says first subproblem to consider is S_1.
Lines 4-7 iterates through activities
k keeps track of S_k
if a_m compatible w/ a_k
choose (greedily) a_m in line 6.

Chapter 8 Elementary graph algorithms

§8.1 Representations of graphs

Given a graph $G = (V, E)$

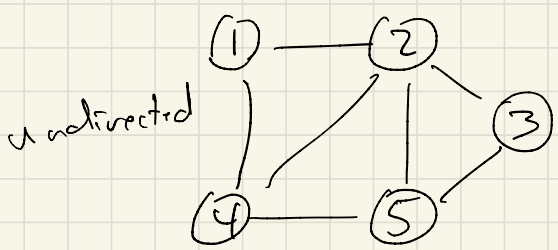
$V = \{\text{vertices}\}$

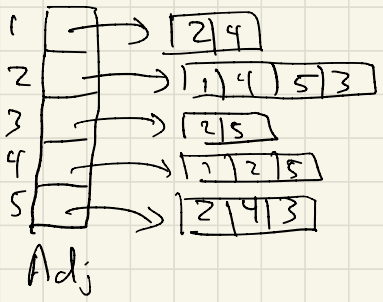
$E = \{\text{edges}\} \subseteq V \times V$

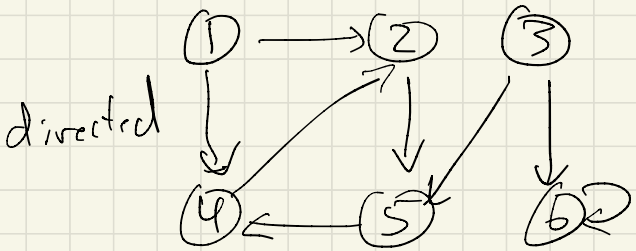
can represent G in two ways:

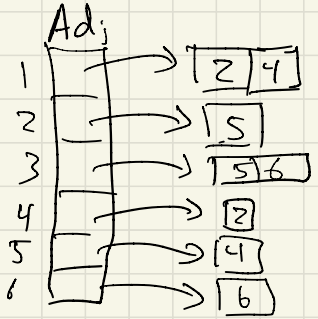
1: Adjacency-list representation

Graph has an array Adj of length $|V|$ s.t. for every $v \in V$ $Adj[v]$ is a list of all vertices v is adjacent to.









In pseudocode will write

$G.Adj$

or $G.Adj[u]$

Remarks · In general we'll use adjacency-lists.

- They use $\Theta(V+E)$ memory.
- Good way of representing "sparse" graphs w/ $|E| \ll |V|^2$
- Disadvantage: asking "$(u,v) \in E$?" can't be done in constant time.

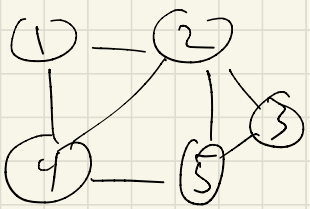
2: Adjacency-matrix representation.

Given $G=(V,E)$ define the adjacency matrix $A=(a_{ij})$ of size $|V| \times |V|$ by

$$a_{ij} := \begin{cases} 0 & \text{if } (i,j) \notin E \\ 1 & \text{if } (i,j) \in E \end{cases}$$

where $V = \{1, \ldots, |V|\}$.

Ex:



	1	2	3	4	5
1	0	1	0	1	0
2	1	0	1	0	1
3	0	1	0	0	1
4	1	1	0	0	1
5	0	1	1	1	0

Remarks · $\Theta(V^2)$ memory

- Question "$(i,j) \in E$?" can be answered in $\Theta(1)$ time.

§8.2 Breadth-first search (BFS)

Goal: Given $G = (V, E)$ and distinguished $s \in V$ called the source. Want to visit all nodes which are "reachable" from s.

BFS does the following:

(1) Compute distance to s of all vertices reachable from s.

(2) It produces a "breadth-first tree" of all vertices reachable from s w/ root s. (i.e., will produce spanning tree of connected component containing s)

(3) For any v reachable from s, the unique simple path from s to v in tree is a shortest path from s to v in G.

"Breadth-first" refers to fact that algorithm will visit all vertices distance 1 away, then distance 2 away, then distance 3, etc.

BFS tags vertices with three attributes:

color: vertex white if undiscovered
gray or black if discovered already
$v.color$
grey vertices are on the frontier
might have white neighbors.
black vertices have gray or black neighbors.

distance best known distance to s.
$v.d$ initialized to ∞.

predecessor the neighbor which first discovered v.
$v.\pi$ $v.\pi$ will be initialized to NIL.

```
BFS(G, s)
 1  for each vertex u ∈ V, V \ {s}
 2        u.color = WHITE
 3        u.d = ∞
 4        u.π = NIL
 5  s.color = GRAY
 6  s.d = 0
 7  s.π = NIL
 8  Q = ∅
 9  ENQUEUE(Q, s)
10  while Q ≠ ∅
11        u = DEQUEUE(Q)
12        for each v ∈ G.Adj[u]
13              if v.color == WHITE
14                    v.color = GRAY
15                    v.d = u.d + 1
16                    v.π = u
17                    ENQUEUE(Q, v)
18        u.color = BLACK
```