IN PART I OF THE COURSE WE DEVELOPED A FRAMEWORK FOR STUDYING THE QUESTION: WHAT FUNCTIONS AND LANGUAGES ARE COMPUTABLE AND WHICH ONES ARE NOT? THIS LED TO THE CHARACTERIZATION OF ALL LANGUAGES AS: RECURSIVE, RE, NOT RE.

IN PART II, WE STUDIED THE FOLLOWING QUESTION. WHICH OF THE RECURSIVE LANGUAGES ARE <u>EFFICIENTLY</u> RECOGNIZABLE? THIS LED TO THE STUDY OF P (THE CLASS OF POLYNOMIALLY DECIDABLE LANGUAGES) AND NP (THE CLASS OF POLYNOMIALLY VERIFIABLE LANGUAGES. THE LANGUAGES IN P ARE, ROUGHLY SPEAKING, THE LANGUAGES THAT CAN BE EFFICIENTLY RECOGNIZED. WE SAW THAT MANY LANGUAGES IN NP ARE <u>NP-COMPLETE</u>, AND THUS PROBABLY NOT IN P.

IN PART III, WE WANT TO TRY TO DEVELOP TOOLS FOR SHOWING THAT CERTAIN LANGUAGES / FUNCTIONS ARE SOLVABLE IN POLYTIME. WE WILL STUDY TWO VERY COMMON METHODS, THE <u>GREEDY METHOD</u> AND

DYNAMIC PROGRAMMING.

THESE METHODS ARE ALGORITHMIC TECHNIQUES THAT CAN BE APPLIED TO A WIDE VARIETY OF PROBLEMS TO GIVE POLYNOMIAL TIME ALGORITHMS.

GREEDY ALGORITHMS

WE WILL START WITH AN EXAMPLE, THE

MINIMUM SPANNING TREE PROBLEM.

DEF'NS

LET G BE AN UNDIRECTED GRAPH. $G=(V,E)$

THE **DEGREE** OF A VERTEX $v \in V$ IS THE NUMBER OF EDGES TOUCHING v.

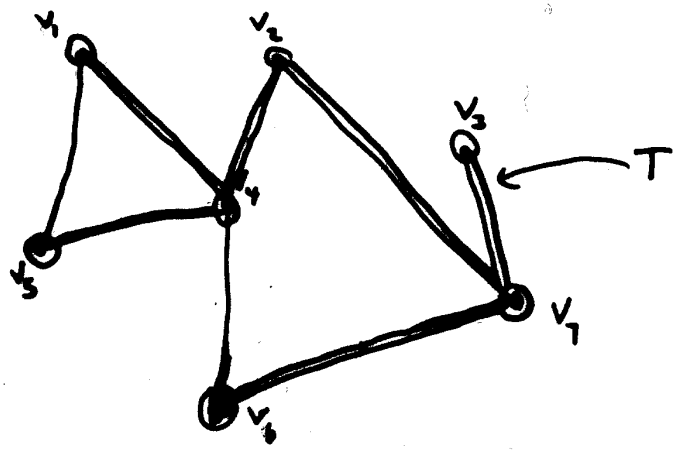
A **PATH** BETWEEN V_1, V_2 IS A SEQUENCE $V_1 V_2 .. V_k$ SUCH THAT $(V_i\ V_{i+1}) \in E\ \forall i$

G IS **CONNECTED** IF THERE IS A PATH IN G BETWEEN EVERY TWO DISTINCT VERTICES IN G.

G IS ACYCLIC IF IT HAS NO CYCLES. (A CYCLE IS A CLOSED PATH $V_1 V_2 .. V_k V_1$ IN G.)

A **TREE** IS A CONNECTED ACYCLIC GRAPH

A **SPANNING TREE** OF A CONNECTED GRAPH G IS A SUBSET $T \subseteq E$ SUCH THAT (V,T) IS A TREE.



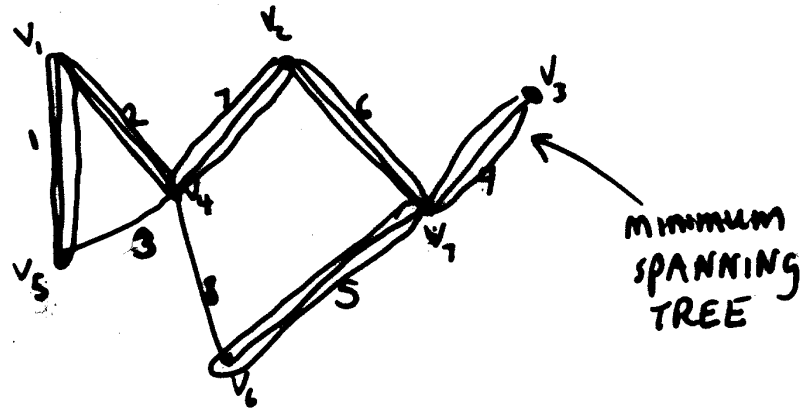
LEMMA EVERY TREE WITH n NODES/VERTICES HAS EXACTLY $n-1$ EDGES.

DEF'N
LET G BE A GRAPH $G=(V,E)$ WHERE EACH EDGE $e_i \in E$ HAS AN ASSOCIATED COST $c(e_i)$

THEN A MINIMUM SPANNING TREE FOR G IS A SPANNING TREE, T, FOR G SUCH THAT FOR ANY OTHER SPANNING TREE FOR G, T',

$$c(T') \geq c(T)$$

WHERE $c(T)$ IS THE SUM OF THE COSTS OF THE EDGES IN T.



MST (MINIMUM SPANNING TREE) PROBLEM

GIVEN A CONNECTED GRAPH $g=(V,E)$, n VERTICES, m EDGES, AND EDGE COSTS $c(e_1), .., c(e_m)$.

FIND A MINIMUM SPANNING TREE FOR g.

KRUSKAL'S GREEDY ALGORITHM

1. SORT EDGES SO THAT

$$c(e_1) \leq c(e_2) \leq \ldots \leq c(e_m)$$

2. INITIALIZE TREE T = empty

3. FOR $i=1 \ldots m$

ADD EDGE e_i TO T AS LONG AS NO CYCLE IS CREATED. (OTHERWISE DO NOT ADD e_i TO T)

4. OUTPUT T.

WHAT IS RUNTIME?

WHAT ARE DATA STRUCTURES USED?

HOW TO TEST FOR A CYCLE?

DATA STRUCTURES

1. T IS A SET OF EDGES

2. D IS AN ARRAY, $D[1], .., D[n]$

   D KEEPS TRACK OF THE CONNECTED COMPONENTS OF (V, T)

   $D[i] = D[j]$ (AT STEP i) IFF i AND j IN SAME COMPONENT.

KRUSKAL'S ALGORITHM (MORE DETAIL)

```
SORT EDGES SO c(e_1) ≤ c(e_2) ≤ .. ≤ c(e_m)
T ← φ                          } initialize T
FOR i = 1 ... n                }
    D[i] ← i                   } initialize D
END FOR                        }

FOR i = 1 .. m
    ASSIGN TO r AND s THE ENDPOINTS OF e_i
    IF D[r] ≠ D[s] THEN                    } IF ADDING e_i DOES NOT
        T ← T ∪ {e_i}                      } CREATE A CYCLE,
        k ← D[r]                           } ADD e_i TO T
        l ← D[s]                           } AND MERGE THE CONNECTED
        FOR j = 1 .. n                     } COMPONENTS CONTAINING
            IF D[j] = l THEN               } r, s INTO ONE
                D[j] ← k                   } CONNECTED COMPONENT
            END IF                         }
        END FOR                            }
    END IF
END FOR
```

RUNTIME:

1. SORTING TAKES TIME $O(m \log m)$

2. INITIALIZE D, $O(n)$

3. MAIN LOOP: $n-1$ MERGES, EACH MERGE TAKES $O(n)$ STEPS
   PLUS $M-n+1$ STEPS FOR NONMERGES

   $= O(n^2) + O(m)$

$\therefore$ Total $= O(m \log m) + O(m) + O(n) + O(n^2) = O(m \log m + n^2)$
$= O(m^2)$,
or $= O(n^2 \log n)$

= POLYTIME

CORRECTNESS

WE WANT TO SHOW THAT T OUTPUT BY KRUSKAL'S ALGORITHM IS A MIN. SPANNING TREE FOR G.

WE WILL SHOW: FOR EACH STEP OF LOOP, THE SET T OF EDGES (SO FAR) CAN BE EXTENDED TO A MST USING ONLY THE REMAINING EDGES NOT YET CONSIDERED

DEFINITION A SET T OF EDGES IS PROMISING AFTER STAGE i IF T CAN BE EXPANDED TO A MST FOR G USING EDGES FROM $\{e_{i+1} \dots e_m\}$.

THAT IS, IF THERE IS A MST T_{opt} SUCH THAT $T \subseteq T_{opt} \subseteq T \cup \{e_{i+1}, \dots, e_m\}$

LEMMA 4 FOR $0 \le i \le m$, LET T_i BE THE VALUE OF T AFTER i STEPS. (AFTER EXAMINING $e_1 .. e_i$)
THEN P(i) HOLDS FOR ALL i, $0 \le i \le m$,

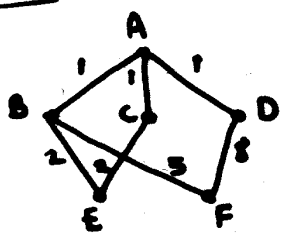
P(i): T_i IS PROMISING AFTER STAGE i

THE PROOF USES THE FOLLOWING LEMMA

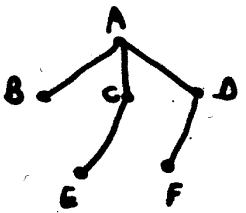
EXCHANGE LEMMA
LET G BE CONNECTED. LET T_1 BE ANY SPANNING TREE OF G, AND T_2 BE A SET OF EDGES NOT CONTAINING A CYCLE. THEN FOR EVERY EDGE $e \in T_2 - T_1$ THERE IS AN EDGE $e' \in T_1 - T_2$ SUCH THAT $\underbrace{T_1 \cup \{e\} - \{e'\}}_{T'}$ IS A SPANNING TREE OF G.

EXAMPLE

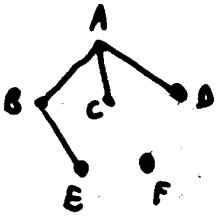
G:



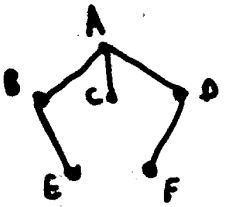
T_1: (T_{opt})



T_2: (T_{i+1})



T':



Proof of Exchange Lemma

Fix T_1, T_2 as in lemma. (T_1: spanning tree, T_2: set of edges, no cycle)

Let $e \in T_2 - T_1$, say $e = [u\ v]$

Since there is a path from u to v in T_1,
$T_1 \cup \{e\}$ has a simple cycle C
and C is the only simple cycle in $T_1 \cup \{e\}$.

Since T_2 acyclic, $\exists e'$ on C that is not in T_2
hence $e' \in T_1 - T_2$

Removing a single edge of C from $T_1 \cup \{e\}$
leaves resulting graph acyclic but connected

$\therefore T_1 \cup \{e\} - \{e'\}$ is a spanning tree of g.

PROOF OF LEMMA 4

BY INDUCTION ON i, WE'LL SHOW THAT $P(i)$ HOLDS $\forall i$.

$P(0)$ EASY

IND STEP LET $0 \le i < m$, AND ASSUME $P(i)$

SINCE T_i PROMISING AFTER STEP i, LET T_{opt} BE MST

S.T. $T_i \subseteq T_{opt} \subseteq T_i \cup \{e_{i+1} .. e_m\}$

① IF $T_i \cup e_{i+1}$ CONTAINS A CYCLE, THEN e_{i+1} REJECTED

AND e_{i+1} NOT IN T_{opt} EITHER, SO

$T_{i+1} \subseteq T_{opt} \subseteq T_{i+1} \cup \{e_{i+2} .. e_m\}$

② OTHERWISE e_{i+1} IS ADDED TO T_i. $T_{i+1} = T_i \cup e_{i+1}$

IF $e_{i+1} \in T_{opt}$, THEN $T_{i+1} \subseteq T_{opt} \subseteq \{e_{i+2} .. e_m\}$.

OTHERWISE $e_{i+1} \notin T_{opt}$. THEN BY EXCHANGE LEMMA,

∃ AN EDGE $e_j \in T_{opt} - T_{i+1}$ SUCH THAT

$T' = T_{opt} \cup \{e_{i+1}\} - \{e_j\}$ IS A SPANNING TREE.

CLEARLY $T_{i+1} \subseteq T' \subseteq T_{i+1} \cup \{e_{i+2} .. e_m\}$

ALSO T' IS A MST BECAUSE $T_{opt} \subseteq T_i \cup \{e_{i+1} .. e_m\}$

AND $e_j \in T_{opt} - T_{i+1}$ SO $j > i+1$, SO $c(e_{i+1}) \le c(e_j)$

AND THUS $cost(T') = c(T_{opt}) + c(e_{i+1}) - c(e_j) \le cost(T_{opt})$.

■

GREEDY ALGORITHM: HIGH LEVEL VIEW

1. SORT INPUT DATA IN SOME WAY
2. ALGORITHM BUILDS UP A SOLUTION, ONE STAGE AT A TIME

   AT EACH STAGE, WE HAVE A PARTIAL SOLUTION

   AT EACH STAGE, MAKE SOME DECISION

   (USUALLY TO INCLUDE OR EXCLUDE SOME ELEMENT FROM SOLUTION)

   NEVER BACKTRACK OR CHANGE OUR MIND.

GREEDY ALGORITHM ANALYSIS ~ HIGH LEVEL VIEW

SHOW FOR EACH STAGE i

THAT PARTIAL SOLUTION SO FAR IS PROMISING

(ie THAT IT CAN BE EXTENDED TO AN OPTIMAL SOLUTION USING ELEMENTS THAT HAVE NOT YET BEEN CONSIDERED.)

↑

BY INDUCTION ON i.

HARD PART: IF S_{OPT} DOES NOT EXTEND S_{i+1}, NEED TO SHOW HOW TO MODIFY S_{OPT} TO S'_{OPT} S.T. S'_{OPT} IS ALSO OPTIMAL AND EXTENDS S_{i+1}.

A GREEDY ALGORITHM FOR SCHEDULING JOBS WITH DEADLINES AND PROFITS

INPUT : $(d_1, g_1)\ (d_2, g_2), \ldots, (d_n\ g_n)$

n jobs, one processor
each takes one time unit
g_i is profit from job i
d_i is deadline for job i (must be completed by time unit d_i or NO profit)

A SCHEDULE S is an array $S(1)\ S(2) \ldots S(n)$
$S(i) \in \{0, 1, \ldots n\}$ is job scheduled for time unit t
$S(i) = 0$ means NO job scheduled for time unit t

S is feasible if
(a) $S(t) = i > 0 \Rightarrow t \leq d_i$ (every scheduled job meets its deadline)
(b) $t_1 \neq t_2$ and $S(t_1) \neq 0 \Rightarrow S(t_1) \neq S(t_2)$
(can schedule a job at most once)

The profit of a feasible schedule S, $P(S) = g_{S(1)} + \ldots + g_{S(n)}$

OUTPUT: A feasible schedule S whose profit P(S) is optimal (as large as possible)

GREEDY ALGORITHM

SORT JOBS IN ORDER OF NONINCREASING PROFITS $g_1 \geq g_2 \geq \ldots \geq g_n$
INITIALIZE $S(1) = S(2) = \ldots = S(n) = 0$
CONSIDER JOBS ONE AT A TIME, $i = 1, \ldots, n$
If job i can be feasibly added, add it to S
in the latest possible feasible slot

EXAMPLE

Job 1: $d_1 = 3$ $g_1 = 9$

Job 2: $d_2 = 2$ $g_2 = 7$

Job 3: $d_3 = 3$ $g_3 = 7$

Job 4: $d_4 = 1$ $g_4 = 2$

s(1)	s(2)	s(3)	s(4)
3	2	1	0

$P(s) = 7 + 7 + 9 = 23$

Theorem

The schedule output by the greedy algorithm is optimal (has largest profit).

Proof of Theorem

A schedule S' extends (feasible) schedule S iff
$\forall t \quad 1 \le t \le n$ if $S(t) \neq 0$ then $S'(t) = S(t)$

A feasible schedule is promising after stage i
if it can be extended to an optimal feasible schedule by adding only jobs $\{i+1, \ldots, n\}$

Lemma Let S_i be the value of S after i stages of the greedy algorithm. Then $\forall i \; 0 \le i \le n$, S_i is promising after stage i.

↑
this completes the proof of theorem. ■

Proof of Lemma

S_0 is promising.

For inductive step, assume that P(i) holds; that is, S_i is promising after i stages, $0 \le i < n$.

To show P(i+1):
By INDUCTIVE ASSUMPTION, S_i CAN BE EXTENDED TO S_{OPT} using only jobs from $\{i+1 .. n\}$

Case 1 Job $i+1$ cannot be scheduled, so $S_{i+1} = S_i$
Then S_{opt} extends S_{i+1} using jobs from $\{i+2 .. n\}$

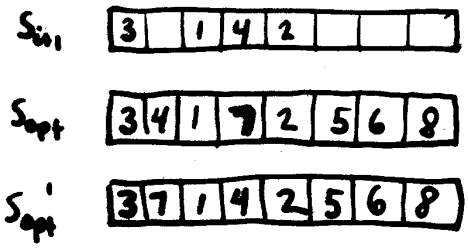
Case 2A Job $i+1$ is scheduled by greedy alg at time t_0 where t_0 is latest free slot in S_i that is $\le d_{i+1}$
and job $i+1$ occurs in S_{opt} at some time t_1

Then $t_1 \le t_0$ because S_{opt} extends S_i and t_0 is as large as possible.

Clearly if $t_1 = t_0$ we are done since S_{opt} extends S_{i+1}

Otherwise $t_1 < t_0$. Form S'_{opt} by interchanging values in slots t_1 and t_0.

S'_{opt} is feasible, optimal, and extends S_{i+1} using $\{i+2 .. n\}$.



Case 2B

Job $i+1$ scheduled by greedy algorithm at time t_0
where t_0 is latest free slot in S_i that is $\leq d_{i+1}$
and Job $i+1$ does not occur in S_{opt}.

Define S'_{opt} to be same as S_{opt}, except at time t_0
where $S'_{opt}(t_0) = i+1$.

Then S'_{opt} feasible and extends S_{i+1} using only $\{i+2 \ldots n\}$.

Also S'_{opt} optimal:

If $S_{opt}(t_0) = 0$, $P(S'_{opt}) = P(S_{opt}) + g_{i+1} \geq P(S_{opt})$

If $S_{opt}(t_0) = j > 0$. Then $j > i+1$ so $g_j \leq g_{i+1}$

Thus $P(S'_{opt}) = P(S_{opt}) + g_{i+1} - g_j \geq P(S_{opt})$.

■

RUNTIME

SORTING is $O(n \log_2 n)$

PLUS $O(n^2)$

$\therefore O(n^2)$ total.