ICS 260—Fall, 2000–Class Notes 7: Greedy Algorithms

Notes Originally Developed by Sandy Irani
Modified by Michael B. Dillencourt

1 Greedy Algorithms

This material is taken from [CLR 17.1].

Optimization algorithms typically follow a series of steps in which there is a choice to be made at each step. Greedy algorithms are a special case of optimization algorithms which take the locally optimal step at each choice. While greedy algorithms do not always yield the optimum solution or even a solution which is close to the optimal, they do behave optimally for many important problems. In trying to design an algorithm for an optimization problem, it is often a good idea to examine the greedy algorithm first to see how well it performs. If it does not perform well, one can then go on to other methods.

We will examine a simple scheduling problem and show that for this problem, the greedy algorithm does yield an optimal solution.

Consider a set of activities which require a common resource. No two activities can use the common resource at the same time. Each activity i has a start time s_i and a finish time f_i. Two activities i and j are not compatible if the intervals $[s_i, f_i)$ and $[s_j, f_j)$ overlap. The goal is to select the largest set of mutually compatible activities.

We will give a greedy algorithm in the figure below. We will assume that the jobs are ordered by increasing finishing times:

$$f_1 \leq f_2 \leq \cdots \leq f_n.$$

If they are not given in this order, they can be sorted in time $O(n \log n)$.

```
Procedure GreedyActivitySelect:
A ← {1}
j ← 1
for i ← 2 to n
    if s_i ≥ f_j
        A ← A ∪ {i}
        j ← i
```

Figure 1: A greedy algorithm for activity selection.

- example

Although greedy algorithms in general do not always find the right solution, the greedy algorithm above gives the optimal answer for the activity selection problem.

Theorem 1 *The greedy activity selection algorithm produces a solution of maximum size of the activity-selection problem.*

Proof. Let S be the set of activities from which the algorithm is selecting. We will prove by induction that the following property holds: suppose that at some point in time, A is the current set of activities chosen by the greedy strategy. Let a_j be the activity in A with the latest finishing time f_j. Let B be the optimal solution when the set of jobs is $S' = \{a_i \in S \mid s_i \geq f_j\}$. Then $A \cup B$ is an optimal solution to S.

This fact is certainly true when $A = \emptyset$. Furthermore, if it is true when $S' = \emptyset$, then A is the optimal solution to for S.

Now assume that at some point in the algorithm, A is the set of chosen activities. $A \cup B$ is the optimal solution to S if B is the optimal solution to S'. The activity which the greedy algorithm will choose in the next iteration is the activity a_m which has the earliest finishing time in S'. We must prove that there is an optimal solution B for S' which also includes job a_m. If this is true, then $B - \{a_m\}$ is an optimal solution for $\{a_i \in S \mid s_i \geq f_m\}$ and $(A \cup \{a_m\}) \cup (B - \{a_m\}) = A \cup B$ is an optimal solution for S.

Let B' be an optimal solution S' which does not include a_m. Let a_k be the first activity in the ordering which is included in B'. Let $B = B' - \{a_k\} \cup \{a_m\}$. B is a valid selection of activities since $f_m \leq f_k$, it can not overlap with any of the rest of the activities in B'. Since B has as many activities as B' and B' is an optimal solution for S', B is also an optimal solution. ∎

There is a beautiful theory about greedy algorithms which describe problems for which the greedy algorithms yield optimal solutions. It involves a combinatorial structure called a matroid. It does not cover all situations for which the greedy algorithm works (the activity selection problem is an example of this), but it does cover many important problems. We will not cover matroids here, but we will mention a famous example of a matroid which is the problem of finding a minimum spanning tree.

$G = (V, E)$ is an undirected graph. Each edge (u, v) has a weight $w(u, v)$. Find $T \subseteq E$ such that T is a tree (i.e. T has no cycles) such that T connects all of the vertices and its weight

$$W(T) = \sum_{(u,v) \in T} w(u, v)$$

is minimized.

The algorithm can be implemented in $O(m \log m)$ time since the time to sort the edges dominates the running time of the algorithm. There are more efficient algorithms which solve the minimum spanning tree problem. The fastest runs in time $O(n \log n + m)$.

```
Procedure MinimumSpanningTree(G):
(1) Sort edges by weight so that w(e_1), ≤ w(e_2) ≤ ··· ≤ w(e_m)
(2) T ← ∅
(3) for i = 1 to m
(4)     if adding e_i to T does not create a cycle
            T ← T ∪ {e_i}
```

Figure 2: A greedy algorithm for activity selection.