Greedy Algorithms

- Making decisions on the basis of information immediately at hand without worrying about the effect these decisions may have in the future
- A family of algorithms typically used to solve optimization problems
  - Knapsack
  - Scheduling
  - MST: minimum spanning tree
  - Single source shortest path

The knapsack problem

- Given
  - n objects numbered from 1 to n. Object i has a positive weight w_i and a positive value v_i
  - a knapsack that can carry a weight not exceeding W
- Problem
  - Fill the knapsack in a way that maximize the value of the included objects, while respecting the capacity constraints
  - Fractional Knapsack Problem
    - the objects can be broken into small pieces
  - 0-1Knapsack Problem:
    - An object cannot be broken into pieces
    - Either choose it or not

Formal description: Fractional Knapsack

- Given W; w_i, v_i
- Find an array x_i, $1<=i<=n$, $0<=x_i<=1$, to
  - Maximize $\sum_{i=1}^{n} x_i v_i$
  - And be subject to $\sum_{i=1}^{n} x_i w_i \le W$

A greedy algorithm

```
Knapsack(w[], v[], W)
{
   for (i=1; i<=n; i++)
     x[i] =0;
   weight = 0;

    while (weight < W) {
        i = select the best remaining object;
        if (weight + w[i] < W)
          x[i] = 1;
        else
          x[i] = (W-weight)/w[i];
    }
    return x;
}
```

The key is which object to select

fill the largest portion possible

Selection methods

1. Choose the most valuable remaining object
2. Choose the lightest remaining object
3. Choose the object with the highest value per weight unit.

n=5, W=100

w	10	20	30	40	50	
v	20	30	66	40	60	
v/w	2.0	1.5	2.2	1.0	1.2	
Method 1			1	3	2	146
Method 2	1	2	3	4		156
Method 3	2	3	1		4	164

Optimality of method 3

- Theorem:
  - If the objects are selected in order of decreasing v_i / w_i then the algorithm knapsack finds an optimal solution
- Observation:
  - After choose a part of or the whole object i, what left is still an optimization problem:
    - Fill the remaining knapsack using the remaining objects
  - Prove that the selected object/part object at each step is *safe*
    - It is always a part of some optimal solution
- Prove: the largest possible portion of the object with the "highest value per weight unit" must be in some optimal solution

Scheduling: an activity selection problem

- A set $S=\{a_1, a_2, \ldots, a_n\}$ activities wish to use a resource
  - The resource can be used by one activity at a time
  - Each activity has a start time s_i and a finish time f_i with $0 <= s_i < f_i$
  - If selected, activity take place at an half-open interval $[s_i, f_i)$.
  - Activities a_i, a_j are compatible if their intervals do not overlap: $s_i >= f_j \,||\, s_j >= f_i$
- The activity selection problem
  - Select a maximum-size subset of mutually compatible activities

An 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	9	10	11	12	13	14

- Compatible set
  - $\{a_3, a_9, a_{11}\}$
  - $\{a_1, a_4, a_8, a_{11}\}$
  - $\{a_2, a_4, a_8, a_{11}\}$

Some definitions

- Define S_{ij} as a subset of activities in that can start after a_i finishes and finish before a_j starts
  - $S_{ij}=\{a_k \in S: f_j \leq s_k < f_k \leq a_j\}$
- For simplicity
  - Assume all activities are sorted by their finish times
    - S_{ij} is empty when $i \geq j$
  - Add two fictitious activities S_0 and S_{n+1}
    - $f_j = 0$
    - $s_{n+1} = \infty$
  - $S = S_{0,n+1}$

The optimal structure

- Theorem: Consider any nonempty subproblem S_{ij} and let a_m be the activity in S_{ij} with the earliest finish time
  - Activity a_m is used in some maximum-size subset of mutually compatible activities of S_{ij}
  - The subproblem S_{im} is empty, so that leaves the subproblem S_{mj} as the only one that may be nonempty

The algorithm

```
GreedyActivitySelector(s, f) // s[] and f[] are start and finish times
                              // activities are sorted by finish time
{
    add a1 to A;

    lastSelected = 1;
    for (i=2; i<=n; i++) {
       if (s[i] >= f[lastSelected]) {
          add ai to A
          lastSelected = i;
       }
    }
    return A;
}
```

Summary: Greedy strategy

- Typical Steps
  - Cast the problem as one in which we make a choice and are left with one subproblem to solve
  - Optional:
    - Prove that there is always an optimal solution to the original problem, so that the greedy choice is always safe
    - Demonstrate that: an optimal solution to the subproblem combined with the greedy choice we have made is an optimal solution to the original problem