Greedy algorithms

- “A greedy algorithm is an algorithm that follows the problem solving heuristic of making the locally optimal choice at each stage with the hope of finding a global optimum.” - Wikipedia

- It makes the choice that seems best at the moment and then solves the subproblem that remains.

Change-making problem

- Using only R1, R2 and R5 coins, make change for a given amount using as few coins as possible. A greedy algorithm works:

```
x ← amount required
while x > 0:
   select largest y from {R1, R2, R5} such that y ≥ x
   add y to solution
   x ← x – y
```

- Consider the similar problem: Replace the R2 coin with a R4 coin. The greedy algorithm makes change for R8 with R5 + R1 + R1 + R1, while the best solution is R4 + R4.

Comparison to DP

- Greedy algorithms are usually simpler and more efficient, but do not always produce an optimal solution.

- Both have optimal substructure: An optimal solution to the problem contains optimal solutions to subproblems.

- In dynamic programming, each choice depends on the solutions to subproblems, so the solution is calculated bottom-up. In a greedy algorithm, the solutions to subproblems are not considered and the solution can be calculated top-down.

Properties to prove

- Optimal substructure: An optimal solution to the problem must contain optimal solutions to subproblems.

- Greedy choice property: By making the choice that seems best at the moment and solving the subproblem that remains later, an optimal solution must be produced. It may depend on choices made so far, but not on the solution to a future subproblem.

Interval scheduling

- Given a set of tasks, each with a start time *s(i)* and finish time *f(i)*, schedule as many tasks as possible, without any overlapping intervals.

- Define two tasks to be compatible if their intervals do not overlap.

```
T ← set of all tasks
while T not empty:
   select i from T with earliest finish time
   add i to solution
   remove all tasks not compatible with i from T
```

Staying ahead

- If you consider the progress of the algorithm at each step, it must always be at least as good as the optimal solution.

- Let $i_1, i_2, \ldots, i_m$ be the solution produced by the greedy algorithm. Assume there is a better solution $j_1, j_2, \ldots, j_n$. Since the second is a better solution, $m < n$. (Assume both sequences are sorted in increasing order of start and finish times.)

Staying ahead

- We will prove the greedy algorithm stays ahead by inductively proving $f(i_k) \leq f(j_k)$ for each $k \leq m$.

- This is obviously true for $k = 1$, since the algorithm chooses the task with the lowest finish time.

- For the induction hypothesis, assume that $f(i_{k-1}) \leq f(j_{k-1})$. We must prove that it is impossible that $f(i_k) > f(j_k)$. Since j_k is compatible with j_{k-1}, and therefore also i_{k-1}, the algorithm would rather have selected j_k, which means it is impossible.

Staying ahead

- Now that we have proven that $f(i_m) \leq f(j_m)$ (in other words, the greedy algorithm stays ahead), we must show that this implies the solution is optimal.

- Since we assumed that $m < n$, there must be a j_{m+1} which is compatible with j_m. It must then also be compatible with i_m.

- Since j_{m+1} is compatible with i_m, the algorithm would never have removed it from the set T, but the algorithm only stops when the set is empty. This contradiction proves the greedy solution is optimal.

Scheduling with deadlines

- This is similar to the previous problem, but instead of a start and finish time, each of the n tasks have a deadline d_i and a required duration t_i. We must assign the start and finish times such that $f(i) = s(i) + t_i$.

- All the tasks must be scheduled without overlapping intervals, but they are allowed to finish after their deadlines.

- The lateness of a task is defined as $f(i) – d_i$. We must minimize the maximum lateness.

Scheduling with deadlines

```
sort the tasks so that d_1 ≤ d_2 ≤ … ≤ d_n
let t = 0
for i = 1 to n:
   s(i) ← t
   f(i) ← t + t_i
   t ← t + t_i
```

- Observe that there is no idle time in the schedule.

- No tasks are scheduled before other intervals with earlier deadlines (there are no inversions).

Exchange argument

- Consider an optimal solution to the problem and transform it into the solution produced by the greedy algorithm, without affecting its quality. Then the greedy solution must be optimal.

- Note that any two solutions without idle time and inversions have the same maximum lateness. Only tasks with the same deadline can be scheduled in a different order, and the last one determines the maximum lateness.

- We must prove that there is an optimal solution without idle time or inversions. Obviously there exists an optimal solution without idle time.

Exchange argument

- An optimal solution with inversions must have a task *i* scheduled directly after a task *j*, such that $d_i < d_j$.

- Swapping *i* and *j* will result in a schedule with one less inversion.

- The new schedule has a maximum lateness no greater than that of the original schedule. Only *j*'s lateness can increase. It finishes where *i* finished before, so its lateness is now $f(i) - d_j$. Since $d_i < d_j$, $f(i) - d_i > f(i) - d_j$.

- Since the optimal solution can be transformed into the greedy solution, without increasing its maximum lateness, the greedy solution is optimal.

Set systems

- Mathematical set systems have been proven to allow greedy algorithms in some cases. Matroids are one of these cases.

- The example used here will be an algorithm you already know – Kruskal's algorithm for finding minimum spanning trees.

- The set system *(E, F)* consists of a ground set *E* and a family of subsets of *E* called *F*.

- As an example *E* could be the edges of a graph, and *F* could be defined as all subsets which do not contain cycles.

Matroids

- A matroid is an independence system. This means that if $A \in F$, for every $B \subseteq A$, $B \in F$ (B can also be the empty set).

- Additionally, it must satisfy the exchange property: If A, $B \in F$, and $|A| > |B|$, there must be an element $x \in A$, $x \notin B$, such that $B \cup \{x\} \in F$.

- Given the weight function $w(x)$, which gives the positive weight of element x, a greedy algorithm can be used to find a maximum-size $A \in F$ such that the total weight of the elements of A is maximized or minimized.

Matroids

- The following algorithm can be used to maximize the total weight. It is simple to modify this algorithm to minimize the total weight.

```
let A be an empty set
sort E so in monotonically decreasing order of weight
for each x ∈ E, in sorted order:
   if A ∪ {x} ∈ F:
      A ← A ∪ {x}
```

Kruskal's algorithm

- Kruskal's algorithm can be represented by a matroid. Let E be the edges of the graph, and F all subsets of edges which do not contain cycles.

- F is clearly independent, since removing an edge cannot create a cycle.

- Suppose that, with $A, B \in$ F, for each $x \in A$, $x \notin B$, $B \cup \{x\} \notin$ F. To prove the exchange property, we must show that $|A| \leq |B|$.

- If V is the set of vertices of the graph, our assumption implies that for each edge $(u, v) \in A$, u and v are in the same connected component of (V, B). If they were not, adding (u, v) to B would not create a cycle.

Kruskal's algorithm

- Since each edge in A connects two vertices in the same connected component of (V, B), each connected component of (V, A) is a subset of a connected component of (V, B).

- Since (V, A) has $|V| - |A|$ connected components and (V, B) has $|V| - |B|$ components, this implies that $|V| - |A| \geq |V| - |B|$.

- This means that $|A| \leq |B|$, which proves the exchange property.

- Since both properties have been proven, the minimum spanning tree problem can be expressed as a matroid, which proves that the greedy matroid algorithm produces an optimal solution.

Generalizations

- There are generalizations of matroids which can prove more greedy algorithms, but their definitions are also more complex and they require more work to prove.

- Two of these generalizations are called greedoids an matroid embeddings.