Class Notes
CIS 502 Analysis of Algorithm
4-Greedy Algorithm

Da Kuang

University of Pennsylvania

1 Optimization Problem

Definition 1.1 ***Optimization Problem:*** *Minimize or maximize some function subject to some constrains.*

Now we start with a special class of optimization problem:

- Given a set of elements, pick a subset.
- Constrains tell you which subsets are allowed.
- Any allowed subset is a feasible solution.
- Objective function assigns a value to every feasible solution.
- Goal is to find the feasible solution with greatest/ least value.

ddd

1.1 Minimum Spanning Tree Problem

Minimum spanning tree problem is an example of optimization problem.

Input: Connected, undirected graph $G \in (V, E)$ together with a weight function $w : E \rightarrow \mathbb{R}^+$.

Definition 1.2 *The **feasible solution** is a set of edges forming an acyclic connected graph on all vertices.*

Definition 1.3 *The **cost** of a solution is the sum of the weights of the edges in the solution.*

The are problems for which the optimal solution can be pick by choosing one element at a time.

Definition 1.4 *The **greedy algorithm** builds up solution as by taking the next element to be one of the optimal cost value that can be added feasibly.*

Most of the time Greedy algorithm itself is simple but it is difficult to prove correctness.

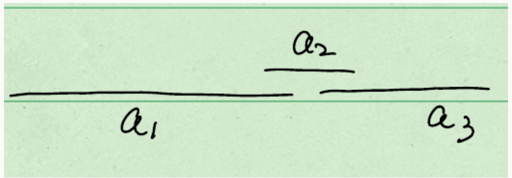
2 Activity Selection Problem

- **Input:** n activities $a_1, \cdots, a_n$, where a_i starts at time x_i and ends at time y_i.
- **Feasible Solution:** Any subset of these activities such that no two activities in the subset overlap.
- **Objective Function:** Maximize the number of activities we schedule.

2.1 Some Attempts

Criterions to be greedy on:

- Pick the activities with shortest duration.
  It dose not work. The counter example is as follows:



- Pick the activities what finish first.

  Sort the activities by finish time and then renumber them so that $f_1 \leq f_2 \cdots \leq f_n$

2.2 Proposed Greedy Algorithm

Given a set of activities,

- Pick the earliest finishing activities that remain.
- Remove all activities that conflict with the chosen activity.
- Repeat.

To prove the correctness, we start by arguing that the first choice algorithm is not wrong.

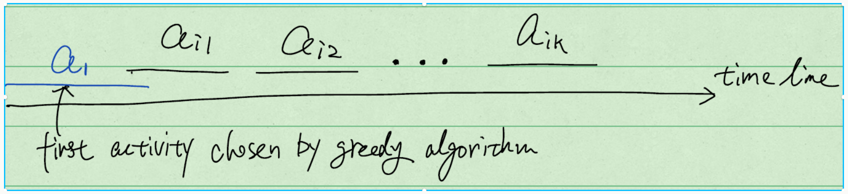
Claim: **Greedy Choice Property:** Fist choice made by greedy algorithm is not wrong. To be more specific, in activities selection problem, if greedy algorithm choose an activity at first, then there is an optimal feasible solution that contain a_1.

Proof: Suppose for contradiction that no optimal feasible solution uses a_1. Let O be the subset of activities in some optimal solution. We can order the activities in O by finish time.

Let $a_{i1}, a_{i2}, \cdots, a_{ik}$ be the activities in O so ordered. We can make an **exchange argument** as the following plot. Throw out a_i from O and include a_1 in instead to get a new set of activities O'.

There are some properties about O'.

- $|O'| = |O|$
- O' is feasible.
- O' is also optimal and contains a_1. $\Rightarrow\!\!\Leftarrow$



■

There is a way to construct an optimal solution starting with a_1. This optimal solution should certainly exclude activities that conflict with a_1. Recursively need to solve a smaller problem consisting of activities that do not conflict with a_1. In particular, the smaller problem is finding the optimal subset of activities out of the remaining activities.

<u>Claim:</u> **Optimal Substructure Property** In the set of activities, we need to pick an optimal feasible subset activities.

<u>Proof:</u>

- A: Original set of activities
- A': Set of activities that remain after throwing out a_1 and its conflicting activities.

Any solution to A' that gives value k can be extended to a solution to A of value $(k+1)$ by adding a_1. So need optimal solution to A' ■

In general. Optimal Substructure Property inductively assumes that greedy solves problem with fewer than n activities optimally.

Greedy Choice Property and Optimal Substructure Property imply that greedy solve n-activity problem optimally.

2.2.1 Time

We sort the activities by finish time take $O(n \log n)$. Then the rest of steps can be done in linear or constant time.

3 Linear Algebra

Definition 3.1 *V is a **vector space over** $\mathbb{R}$ if*

- *For v_1, $v_2 \in V$, $v_1 + v_2 \in V$.*
- *For any $\alpha \in \mathbb{R}$, $v \in V$, $\alpha v \in V$.*

Definition 3.2 *Given a finite set of vectors, $v_1, v_2, \cdots, v_k$, then* ***span*** *S is as follows*

$$S(v_1, v_2, \cdots, v_k) = \{v : v = \alpha_1 v_1 + \alpha_2 v_2 + \cdots + \alpha_k v_k,\ \alpha_i \in \mathbb{R}\}$$

$S(v_1, v_2, \cdots, v_k)$ is a vector space.

Definition 3.3 *$\alpha_1 v_1 + \alpha_2 v_2 + \cdots + \alpha_k v_k$ is a* ***linear combination*** *of vectors.*

Definition 3.4 *The set of vector $v_1, \cdots, v_k$ is called* ***linear dependent*** *if there exist some coefficient $\alpha_1, \cdots, \alpha_k$ not all 0, so that $\sum \alpha_i v_i = 0$*

Definition 3.5 *A set of vector is said to be* ***linearly independent*** *if it is not linearly dependent.*

Definition 3.6 *If $v_1, \cdots, v_k$ are linearly independent and their span in V, then $v_1, \cdots, v_k$ form a* ***basis*** *of V.*

Definition 3.7 *If $v_1, \cdots, v_k$ is a basis for v and $u_1, \cdots, u_m$ is another basis. Then $m = k$.*

Proof. Suppose for contradiction that $m > k$. Since v_i's from a basis,

$$\begin{aligned} u_1 &= \alpha_{11} v_1 + \cdots + \alpha_{1k} v_k \\ u_2 &= \alpha_{21} v_1 + \cdots + \alpha_{2k} v_k \\ &\vdots \\ u_m &= \alpha_{m1} v_1 + \cdots + \alpha_{mk} v_k \end{aligned}$$

Will prove $(u_1, \cdots, u_m)$ is linearly dependent. Need to show $\sum x_i u_i = 0$ for some $(x_1, \cdots, x_m)$ not all zero.

$$x_1 u_1 + x_2 u_2 + \cdots + x_m u_m = 0$$

substitue u_i with v_i's,

$$\begin{aligned} &(x_1 \alpha_{11} + x_2 \alpha_{21} + \cdots + x_m \alpha_{m1}) v_1 \\ + &\cdots \\ + &(x_1 \alpha_{1k} + x_2 \alpha_{2k} + \cdots + x_m \alpha_{mk}) v_k = 0 \end{aligned}$$

All the coefficients above should be 0. There are k coeficientes, m equations and we assume $m > k$. Therefore, there are infinite possible combinations of x_i's. Therefore, there must be a solution which is not all zeros. Because if there is not such solution, then there should be only one solution which is all zeros.

4 Maximum Total Weight Problem

- **Inputs:** vectors $(v_1, v_2, \cdots, v_n)$ with weights $(w_1, w_2, \cdots, w_n)$.
- **Goal:** a basis for the space spanned by $(v_1, v_2, \cdots, v_n)$ of maximum total weights.

Note. A single vector is linear independent to any other vector if it is a zero vector.

4.1 Greedy Algorithm

- Sort vectors by descending order of weights.
- S is an empty set of vectors initially.
- For each vectors v_i in this order, if $S \cup \{v_i\}$ is linearly independent and v_i is an non-zero vector, then $S = S \cup \{v_i\}$.

4.2 Correctness

- Suppose greedy returns vector $(v_1, v_2, \cdots, v_k)$.
- Suppose there is an optimal solution that return OPT = $(v_{j1}, v_{j2}, \cdots, v_{jk})$.

Since the v_j's form a basis,

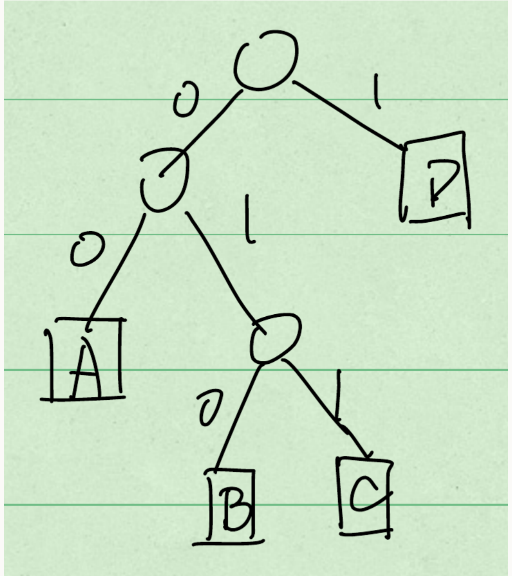
$$v_{i1} = \alpha_1 v_{j1} + \alpha_2 v_{j2} + \cdots + \alpha_k v_{jk}$$

v_{i1} is chosen by greedy algorithm so it is not a zero vector. Therefore, there must be some α_l is non-zero.

We can add v_{i1} to the set $(v_{j1}, v_{j2}, \cdots, v_{jk})$ and kick out v_{jl}.

<u>Claim:</u> This is also a basis whose weight is at least as good as OPT.

Binary code can be thought as a binary tree. As example:



Note that the symbols at the leaves are prefix code. The tree corresponding an optimal tree is a full binary tree, i.e. every internal node has two children. It is because if you have a internal node which is not full, then you are able to move everything below it one level up and reduce the code length.

Suppose we know the shape of the optimal tree which is a full binary tree.

Claim: Any full binary tree has at least a pair of sibling leaves at the deepest leaves.

Proof: Choose one leaf x at the deepest level. Its parent must have two children (say x and y). y cannot be internal node since x is the deepest leaf. Therefore, y must be the sibling leaf of x. ■

Claim: **Greedy Choice**: Pick the two symbols with least frequency and put them in the sibling leaves at the deepest level.

Proof: Prove the claim by exchange property. Suppose x and y are the two symbols of the least frequency. Suppose the sibling leaves at the deepest level have symbols a, b and $\{a, b\} \neq \{x, y\}$.

By exchanging the positions of a with x and b with y, we can bring x, y to the deepest level. One can prove that this is no worst than the original tree.

■

Make a new symbol xy in place of x and y so that $f(xy) = f(x) + f(y)$. Suppose the length from root to x is $(d+1)$. So the contribution of code length that x and y make is

$$\begin{aligned} & f(x)(d+1) + f(y)(d+1) \\ =& (f(x) + f(y))(d+1) \\ =& f(xy)(d+1) \end{aligned}$$

So the same tree with xy made a symbol has cost that of $f(xy)$ lower.

4.3 Algorithm

Combine two least frequent symbol x and y with one symbol xy with $f(xy) = f(x) + f(y)$.

Recursively solve the problem on the $(n-1)$ symbols.

Dynamic programming is on the halfway of the continuum between brute-force algorithms and greedy algorithms. Brute-force is a strategy to use when you have no idea what to do. So you look at the problem and try every possible solution. Then see the best among the results. On the other extreme, greedy algorithm is used when you have a perfect sense what to do. So that you only need to do whatever looks cheapest to do now. By comparison, brute-force tries everything while greedy goes in a very directed way.

Interestingly, dynamic programming is a bit directed but is not that sure. Therefore, it breaks down the optimization problem into a series of decisions. For each decision, unlike greedy knowing the right thing to do, dynamic programming tries out every possible way within this decision. It is brute force locally but in a controlled so the algorithm not inefficient.

The performance of algorithm goes from greedy to dynamic programming then to brute force getting worse and worse. But the proofs is getting easier and easier. In brute force, you do not have to prove anything since it just simply tries everything and get the best answer. There is nothing clever that needs justification. Greedy algorithm, on the extreme, make a commitment to make a decision and we need to prove the decision is correct and we do not miss anything while only solving the local optimal.