Unit 5: Greedy Algorithms

**Course contents:**
- Elements of the greedy strategy
- Activity selection
- Knapsack problem
- Huffman codes
- Task scheduling
- Matroid theory

**Reading:**
- Chapter 16

---

Greedy Algorithm: Vertex Cover

A vertex cover of an undirected graph \( G = (V, E) \) is a subset \( V' \subseteq V \) such that if \((u, v) \in E\), then \( u \in V' \) or \( v \in V' \), or both.

- The set of vertices covers all the edges.
- The size of a vertex cover is the number of vertices in the cover.
- The vertex-cover problem is to find a vertex cover of minimum size in a graph.

**Greedy heuristic:**
- Cover as many edges as possible (vertex with the maximum degree) at each stage and then delete the covered edges.

- The greedy heuristic cannot always find an optimal solution!
- The vertex-cover problem is NP-complete.

---

A Greedy Algorithm

- A greedy algorithm always makes the choice that looks best at the moment.

An Activity-Selection Problem:
- Given a set \( S = \{1, 2, \ldots, n\} \) of \( n \) proposed activities, with a start time \( s_i \) and a finish time \( f_i \) for each activity \( i \), select a maximum-size set of mutually compatible activities.
- If selected, activity \( i \) takes place during the half-open time interval \([s_i, f_i)\).
- Activities \( i \) and \( j \) are compatible if \([s_i, f_i) \) and \([s_j, f_j) \) do not overlap (i.e., \( s_i \geq f_j \) or \( s_j \geq f_i \)).

**The Activity-Selection Algorithm**

```
Greedy-Activity-Selector(s, f)
1. n = s.length
2. A = \{1\} // A in 3rd Ed.
3. j = 1
4. for i = 2 to n
5. if s_i \geq f_j
6. A = A \cup \{i\}
7. j = i
8. return A
```

  - (Greedy-choice property) Suppose \( A \subseteq S \) is an optimal solution. Show that if the first activity in \( A \) activity \( \neq 1 \), then \( B = A - \{k\} \cup \{1\} \) is an optimal solution.
  - (Optimal substructure) Show that if \( A \) is an optimal solution to \( S \), then \( A' = A - \{1\} \) is an optimal solution to \( S' = \{i \in S : s_i \geq f_1\} \).

- Proof by induction on the number of choices made.

---

Optimality Proofs

- (Greedy-choice property) Suppose \( A \subseteq S \) is an optimal solution. Show that if the first activity in \( A \) activity \( \neq 1 \), then \( B = A - \{k\} \cup \{1\} \) is an optimal solution.
  - Exp: \( A' = \{4, 8, 11\}, S' = \{4, 6, 7, 8, 9, 11, 12\} \) in the Activity Selection example
  - Proof by contradiction: If \( A' \) is not an optimal solution to \( S' \), we can find a "better" solution \( A' \). Then, \( A'(1) \) is a better solution than \( A \) to \( S \), contradicting the original claim that \( A \) is an optimal solution to \( S \).
Elements of the Greedy Strategy

- When to apply greedy algorithms?
  - Greedy-choice property: A global optimal solution can be arrived at by making a locally optimal (greedy) choice.
  - Dynamic programming needs to check the solutions to subproblems.
- Optimal substructure: An optimal solution to the problem contains within its optimal solutions to subproblems.
  - E.g., if A is an optimal solution to S, then A' = A - {1} is an optimal solution to S' = {i ∈ S: si ≥ f1}.
- Greedy heuristics do not always produce optimal solutions.
- Greedy algorithms vs. dynamic programming (DP)
  - Common: optimal substructure
  - Difference: greedy-choice property
  - DP can be used if greedy solutions are not optimal.

Knapsack Problem

- Knapsack Problem: Given n items, with ith item worth vi dollars and weighing wi pounds, a thief wants to take as valuable a load as possible, but can carry at most W pounds in his knapsack.
- The 0-1 knapsack problem: Each item is either taken or not taken (0-1 decision).
- The fractional knapsack problem: Allow to take fraction of items.

Coding

- Is used for data compression, instruction-set encoding, etc.
- Binary character code: character is represented by a unique binary string
  - Fixed-length code (block code): a: 000, b: 001, ..., f: 101
  - Variable-length code: frequent characters ⇒ short codeword; infrequent characters ⇒ long codeword

Optimal Prefix Code Design

- Coding Cost of T: B(T) = Σ c frequciency · dT(c)
  - c: character in the alphabet C
  - c frequciency: frequency of c
  - dT(c): depth of c's leaf (length of the codeword of c)
- Code design: Given c frequciencies, construct a binary tree with n leaves such that B(T) is minimized.
  - Idea: more frequently used characters use shorter depth.

Huffman's Procedure

- Pair two nodes with the least costs at each step.

Binary Tree vs. Prefix Code

- Prefix code: No code is a prefix of some other code.
### Huffman's Algorithm

1. \( n = |C| \)
2. \( Q = C \)
3. for \( i = 1 \) to \( n - 1 \)
   4. Allocate a new node \( z \)
   5. \( z.\text{left} = x = \text{Extract-Min}(Q) \)
   6. \( z.\text{right} = y = \text{Extract-Min}(Q) \)
   7. \( z.\text{freq} = x.\text{freq} + y.\text{freq} \)
   8. Insert \( (Q, z) \)
9. return \( \text{Extract-Min}(Q) \)

- Time complexity: \( O(n \lg n) \).
  - Extract-Min \((Q)\) needs \( O(\lg n) \) by a heap operation.
  - Requires initially \( O(n \lg n) \) time to build a binary heap.

### Huffman’s Algorithm: Greedy Choice

- Greedy choice: Two characters \( x \) and \( y \) with the lowest frequencies must have the same length and differ only in the last bit.

### Huffman’s Algorithm: Optimal Substructure

- Optimal substructure: Let \( T \) be a full binary tree for an optimal prefix code over \( C \). Let \( z \) be the parent of two leaf characters \( x \) and \( y \). If \( z.\text{freq} = x.\text{freq} + y.\text{freq} \), tree \( T' = T - \{x, y\} \) represents an optimal prefix code for \( C' = C - \{x, y\} \cup \{z\} \).

\[
B(T) = B(T') + x.\text{freq} + y.\text{freq}
\]

If \( T' \) is not optimal, find \( T'' = \frac{T'}{2} \) and \( \frac{T'}{2} < T'' \), i.e. \( z \) is a leaf of \( T'' \).

\[
B(T'') = B(T') + x.\text{freq} + y.\text{freq}
\]

If \( B(T'') < B(T') \), \( z \) must be a leaf of \( T '' \), contradiction!!

### Task Scheduling

- The task scheduling problem: Schedule unit-time tasks with deadlines and penalties s.t. the total penalty for missed deadlines is minimized.
  - \( S = \{1, 2, ..., n\} \) of \( n \) unit-time tasks.
  - Deadlines \( d_1, d_2, ..., d_n \) for tasks, \( 1 \leq d_i \leq n \).
  - Penalties \( w_1, w_2, ..., w_n \): \( w_i \) is incurred if task \( i \) misses deadline.

- Set \( A \) of tasks is independent if there is a schedule with no late tasks.
  - \( N_t(A) \): number of tasks in \( A \) with deadlines \( t \) or earlier, \( t = 1, 2, ..., n \).
  - Three equivalent statements for any set of tasks \( A \)
    - \( A \) is independent.
    - \( N_t(A) \leq t, t = 1, 2, ..., n \).
    - If the tasks in \( A \) are scheduled in order of nondecreasing deadlines, then no task is late.

### Greedy Algorithm: Task Scheduling

1. Sort penalties in non-increasing order.
2. Find tasks of independent sets: no late task in the sets.
3. Schedule tasks in a maximum independent set in order of nondecreasing deadlines.
4. Schedule other tasks (missing deadlines) at the end arbitrarily.

<table>
<thead>
<tr>
<th>Task</th>
<th>( d_i )</th>
<th>( w_i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>60</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>40</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>30</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>20</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>10</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>50</td>
</tr>
</tbody>
</table>

N\(_i(A)\) = \begin{cases} 0, i \leq 1 \\ 1, i \leq 2 \\ 2, i \leq 3 \\ 3, i \leq 4 \\ 4, i \leq 5 \\ 5, i \leq 6 \\ 0, i \leq 7 \end{cases}

optimal scheduling: \{2, 4, 1, 5, 3, 7, 6\}
penalty: 30 + 30 = 60

### Matroid

Let \( S \) be a finite set, and \( F \) a nonempty family of subsets of \( S \), that is, \( F \subseteq P(S) \).

We call \((S, F)\) a matroid if and only if

M1) If \( B \subseteq F \) and \( A \subseteq B \), then \( A \in F \).
  [The family \( F \) is called hereditary]

M2) If \( A, B \subseteq F \) and \( |A| < |B| \), then there exists \( x \) in \( B \setminus A \) such that \( A \cup \{x\} \in F \).
  [This is called the exchange property]
Example 1 (Matric Matroids)

Let $M$ be a matrix.
Let $S$ be the set of rows of $M$ and
$F = \{ A \mid A \subseteq S, A$ is linearly independent $\}$

Claim: $(S,F)$ is a matroid.
Clearly, $F$ is not empty (it contains every row of $M$).
M1) If $B$ is a set of linearly independent rows of $M$, then any subset $A$ of $M$ is linearly independent. Thus, $F$ is hereditary.
M2) If $A, B$ are sets of linearly independent rows of $M$, and $|A| < |B|$, then dim $\text{span } A < \text{dim span } B$. Choose a row $x$ in $B$ that is not contained in $\text{span } A$. Then $A \cup \{x\}$ is a linearly independent subset of rows of $M$. Therefore, $F$ satisfied the exchange property.

Undirected Graphs

Let $V$ be a finite set, $E$ a subset of $\{ e \mid e \subseteq V, |e|=2 \}$
Then $(V,E)$ is called an undirected graph.
We call $V$ the set of vertices and $E$ the set of edges of the graph.

Induced Subgraphs

Let $G=(V,E)$ be a graph.
We call a graph $(V,E')$ an induced subgraph of $G$ if and only if its edge set $E'$ is a subset of $E$.

Spanning Trees

Given a connected graph $G$, a spanning tree of $G$ is an induced subgraph of $G$ that happens to be a tree and connects all vertices of $G$. If the edges are weighted, then a spanning tree of $G$ with minimum weight is called a minimum spanning tree.

Example 2 (Graphic Matroids)

Let $G=(V,E)$ be an undirected graph.
Choose $S = E$ and
$F = \{ A \mid H = (V,A)$ is an induced subgraph of $G$ such that $H$ is a forest $\}$.

Claim: $(S,F)$ is a matroid.
M1) $F$ is a nonempty hereditary set system.
M2) Let $A$ and $B$ in $F$ with $|A| < |B|$. Then $(V,B)$ has fewer trees than $(V,A)$. Therefore, $(V,B)$ must contain a tree $T$ whose vertices are in different trees in the forest $(V,A)$. One can add the edge $x$ connecting the two different trees to $A$ and obtain another forest $(V,A \cup \{x\})$.

Weight Functions

A matroid $(S,F)$ is called weighted if it equipped with a weight function $w: S \to \mathbb{R}^+$, i.e., all weights are positive real numbers.

If $A$ is a subset of $S$, then
$w(A) := \sum_{a \in A} w(a)$.

Weight functions of this form are sometimes called “linear” weight functions.
Greedy Algorithm for Matroids

Greedy(M=(S,F),w)
A := ∅;
Sort S into monotonically decreasing order by weight w.
for each x in S taken in monotonically decreasing order do
    if A∪{x} in F then A := A∪{x}; fi;
od;
return A;

Correctness

Theorem: Let M= (S,F) be a weighted matroid with weight function w. Then Greedy(M,w) returns a set in F of maximal weight.

[Thus, even though Greedy algorithms in general do not produce optimal results, the greedy algorithm for matroids does! This algorithm is applicable for a wide class of problems. Yet, the correctness proof for Greedy is not more difficult than the correctness for special instance such as Huffman coding. This is economy of thought!]

Complexity

Let n = |S| = # elements in the set S.
Sorting of S: O(n log n)
The for-loop iterates n times. In the body of the loop one needs to check whether A∪{x} is in F. If each check takes f(n) time, then the loop takes O(n f(n)) time.
Thus, Greedy takes O(n log n + n f(n)) time.

Minimizing or Maximizing?

Let M=(S,F) be a matroid.
The algorithm Greedy(M,w) returns a set A in F maximizing the weight w(A).
If we would like to find a set A in F with minimal weight, then we can use Greedy with weight function
w'(a)=m-w(a), where m > max_{s in S} w(s).

Matric Matroids

Let M be a matrix. Let S be the set of rows of the matrix M and F = { A | A⊆S, A is linearly independent }.
Weight function w(A)=|A|.
What does Greedy((S,F),w) compute?
The algorithm yields a basis of the vector space spanned by the rows of the matrix M.

Graphic Matroids

Let G=(V,E) be an undirected connected graph.
Let S = E and F = { A | H = (S,A) is an induced subgraph of G such that H is a forest }.
Let w be a weight function on E.
Define w'(a)=m-w(a), where m > w(a), for all a in A.
Greedy((S,F), w') returns a minimum spanning tree of G. This algorithm is known as Kruskal’s algorithm.
Kruskal's MST algorithm

Consider the edges in increasing order of weight, add an edge iff it does not cause a cycle.

Matroids characterize a group of problems for which the greedy algorithm yields an optimal solution.

Kruskals minimum spanning tree algorithm fits nicely into this framework.