Recap: Greedy Algorithms

Minimize Lateness
- **Greedy Choice:** Sort by earliest deadline
- **Proof of Optimality:** can always optimal solution into one with fewer inversions (Greedy Choice has 0 inversions)
- **Running Time:** $O(n \log n)$
Optimal Offline Caching
- **Goal:** Minimize number of cache misses
- **Greedy Choice:** Evict item used furthest in future [Belady'60]
- **Proof of Optimality:** Invariant: $S_T$ is optimal through first $j+1$ requests.
- **Limitation:** Need to know sequence in advance.

Optimal Online Caching
- **Goal:** Minimize number of cache misses
- **Greedy Choice:** Evict item in cache with the highest page fault rate
- **Proof of Optimality:** Invariant: $S_T$ is optimal through first $j+1$ requests.
- **Limitation:** Need to know sequence in advance.

Minimum Spanning Tree

**Definition:** Given a connected graph $G = (V, E)$ with real-valued edge weights $c_e$, an MST is a subset of the edges $T \subseteq E$ such that $T$ is a spanning tree whose sum of edge weights is minimized.

**Cayley's Theorem:** There are $n^{n-2}$ spanning trees of $K_n$.

Applications

**MST is fundamental problem with diverse applications.**
- Network design.
- Telephone, electrical, hydraulic, TV cable, computer, road
- Approximation algorithms for NP-hard problems.
- Traveling salesperson problem, Steiner tree
- Indirect applications.
- Max bottleneck paths
- LP with min-sum objective
- Image registration with Renyi entropy
- Learning salient features for real-time face verification
- Reducing data storage in sequencing amino acids in a protein
- Model locality of particle interactions in turbulent fluid flows
- Autoconfig protocol for Ethernet bridging to avoid cycles in a network
- Cluster analysis.

Greedy Algorithms

**Kruskal’s algorithm.** Start with $T = \emptyset$. Consider edges in ascending order of cost. Insert edge $e$ in $T$ unless doing so would create a cycle.

**Reverse-Delete algorithm.** Start with $T = E$. Consider edges in descending order of cost. Delete edge $e$ from $T$ unless doing so would disconnect $T$.

**Prim’s algorithm.** Start with some root node $s$ and greedily grow a tree $T$ from $s$ outward. At each step, add the cheapest edge $e$ to $T$ that has exactly one endpoint in $T$.

**Remark.** All three algorithms produce an MST.
Greedy Algorithms

Simplifying assumption. All edge costs $c_e$ are distinct.

Cut property. Let $S$ be any subset of nodes, and let $e$ be the min cost edge with exactly one endpoint in $S$. Then the MST contains $e$.

Cycle property. Let $C$ be any cycle, and let $f$ be the max cost edge belonging to $C$. Then the MST does not contain $f$.

Cycle-Cut Intersection

Claim. A cycle and a cutset intersect in an even number of edges.

Proof. (by picture)

Greedy Algorithms

Simplifying assumption. All edge costs $c_e$ are distinct.

Cycle property. Let $C$ be any cycle in $G$, and let $f$ be the max cost edge corresponding to $S$. Then the MST $T^*$ does not contain $f$.

Proof (exchange argument)

- Suppose $f$ belongs to $T^*$, and let’s see what happens.
- Deleting $f$ from $T^*$ creates a cut $S$ in $T^*$.
- Edge $f$ is both in the cycle $C$ and in the cutset $D$ corresponding to $S$.
- There exists another edge, say $e$, that is in both $C$ and $D$.
- $T' = T^* - \{e\} - \{f\}$ is also a spanning tree.
- Since $c_e < c_f$, $\text{cost}(T') < \text{cost}(T^*)$.
- This is a contradiction.

Prim’s Algorithm: Proof of Correctness

Prim’s algorithm. (Jarník 1930, Dijkstra 1959, Prim 1957)

- Initialize $S = \{\}$.
- Apply cut property to $S$.
- Add min cost edge in cutset corresponding to $S$ to tree $T$, and add one new explored node $u$ to $S$. 
Implementation. Use a priority queue ala Dijkstra.
- Maintain set of explored nodes \( S \).
- For each unexplored node \( v \), maintain attachment cost \( a[v] = \text{cost of cheapest edge } v \rightarrow \text{a node in } S \).
- \( O(n^2) \) with an array; \( O(m \log n) \) with a binary heap.
- \( O(m + n \log n) \) with Fibonacci Heap

Prim(G, c) {
    \( \text{foreach } (v \in V) a[v] \leftarrow \infty \)
    Initialize an empty priority queue \( Q \)
    \( \text{foreach } (v \in V) \text{ insert } v \text{ onto } Q \)
    Initialize set of explored nodes \( S \leftarrow \emptyset \)
    \( \text{while } (Q \text{ is not empty}) \{ \)
        \( u \leftarrow \text{delete min element from } Q \)
        \( S \leftarrow S \cup \{u\} \)
        \( \text{foreach } (\text{edge } e = (u, v) \text{ incident to } u) \)
        \( \text{if } (v \not\in S) \text{ and } (c_e < a[v]) \)
        \( \text{decrease priority } a[v] \text{ to } c_e \)
    \}
}

Kruskal's Algorithm: Proof of Correctness
Kruskal's algorithm. \([\text{Kruskal, 1956}]\)
- Consider edges in ascending order of weight.
- Case 1: If adding \( e \) to \( T \) creates a cycle, discard \( e \) according to cycle property.
- Case 2: Otherwise, insert \( e = (u, v) \) into \( T \) according to cut property where \( S \) is set of nodes in \( u \)'s connected component.

Kruskal(G, c) {
    Sort edges weights so that \( c_1 \leq c_2 \leq \ldots \leq c_m \).
    \( T \leftarrow \emptyset \)
    \( \text{foreach } (u \in V) \text{ make a set containing singleton } u \)
    \( \text{for } i = 1 \text{ to } m \)
        \( (u, v) = e_i \)
        \( \text{if } (u \text{ and } v \text{ are in different sets}) \{ \)
            \( T \leftarrow T \cup \{e_i\} \)
            \( \text{merge the sets containing } u \text{ and } v \)
        \}
    return \( T \)
}

Lexicographic Tiebreaking
To remove the assumption that all edge costs are distinct: perturb all edge costs by tiny amounts to break any ties.

Impact. Kruskal and Prim only interact with costs via pairwise comparisons. If perturbations are sufficiently small, MST with perturbed costs is MST with original costs.

boolean less(i, j) {
    if \( (\text{cost}(e_i) < \text{cost}(e_j)) \) return true
    else if \( (\text{cost}(e_i) > \text{cost}(e_j)) \) return false
    else if \( (i < j) \) return true
    else return false
}

MST Algorithms: Theory
Deterministic comparison based algorithms.
- \( O(m \log n) \). \([\text{Jarník, Prim, Dijkstra, Kruskal, Beruška}]\)
- \( O(m \log \log n) \). \([\text{Cheriton–Tarjan 1976}, \text{Yao 1975}]\)
- \( O(m \log(n, m)) \). \([\text{Fredman-Tarjan 1987}]\)
- \( O(m \log \log(n, m)) \). \([\text{Gabow-Gall-J-Scaper–Tarjan 1986}]\)
- \( O(m = \log(n, m)) \). \([\text{Chazelle 2000}]\)

Holy grail. \( O(m) \).

Notable.
- \( O(m) \) randomized. \([\text{Karger–Klein–Tarjan 1995}]\)
- \( O(m) \) verification. \([\text{Dixon-Rauch-Tarjan 1992}]\)

Euclidean.
- 2-d: \( O(n \log n) \). compute MST of edges in Delaunay
- \( k \)-d: \( O(kn^2) \). dense Prim

4.7 Clustering
Clustering

Clustering. Given a set $U$ of $n$ objects labeled $p_1, \ldots, p_n$, classify into coherent groups.

Distance function. Numeric value specifying "closeness" of two objects.

photos, documents, micro-organisms

Clustering of Maximum Spacing

$k$-clustering. Divide objects into $k$ non-empty groups.

Distance function. Assume it satisfies several natural properties.
- $d(p, p) = 0$ (identity of indiscernibles)
- $d(p, p) \geq 0$ (nonnegativity)
- $d(p, p) = d(p, p)$ (symmetry)

Spacing. Min distance between any pair of points in different clusters.

Clustering of maximum spacing. Given an integer $k$, find a $k$-clustering of maximum spacing.

Greedy Clustering Algorithm

Single-link $k$-clustering algorithm.
1. Form a graph on the vertex set $U$, corresponding to $n$ clusters.
2. Find the closest pair of objects such that each object is in a different cluster, and add an edge between them.
3. Repeat $n-k$ times until there are exactly $k$ clusters.

Key observation. This procedure is precisely Kruskal’s algorithm (except we stop when there are $k$ connected components).

Remark. Equivalent to finding an MST and deleting the $k-1$ most expensive edges.

Greedy Clustering Algorithm: Analysis

Theorem. Let $C^*$ denote the clustering $C^*_1, \ldots, C^*_k$ formed by deleting the $k-1$ most expensive edges of a MST. $C^*$ is a $k$-clustering of max spacing.

Proof. Let $C$ denote some other clustering $C_1, \ldots, C_k$.
1. The spacing of $C^*$ is the length $d^*$ of the $(k-1)^{th}$ most expensive edge (in MST).
2. Let $p, q$ be in the same cluster in $C^*$, say $C^*_r$, but different clusters in $C$, say $C_s$ and $C_t$.
3. Some edge $(p, q)$ on the path spanning two different clusters in $C$.
4. All edges on the path have length $\leq d^*$ since Kruskal chose them.
5. Spacing of $C$ is $\geq d^*$ since $p$ and $q$ are in different clusters.

Union-Find Data-Structure

Three Operations
- `MakeUnionFind(S)`
  - Initialize a Union-Find data structure where all elements in $S$ are in separate sets
- `Find(u)`
  - Input: $u \in S$
  - Output: Name of the set $A$ containing $u$
  - Require: If $u, v$ in the same set $A$ then $\text{Find}(u) = \text{Find}(v)$
- `Union(A,B)`
  - Input: Names of sets $A$ and $B$ in the Union-Find data structure
  - No Output: Merge the sets $A$ and $B$ into a single set $A \cup B$
  - Require: If we had $u \in A$ and $v \in B$ then we require that $\text{Find}(u) = \text{Find}(v)$ after this operation is completed
Union-Find Applications

- Efficient Implementation of Kruskal's Algorithm
- Initially all nodes are in different sets (no edges added to T)
  - Find(u) = u for each node u
  - Indicates that each node is its own connected component (initially)
  - Add edge (u,v) to T
- Merges two connected components containing u and v respectively
- Union(A,B) where A = Find(u) and B=Find(v)

  - Check if adding edge (u,v) induces a cycle in T
  - Observation: (u,v) induces a cycle if and only if u and v are already in the same connected component.
  - Test: Find(u) = Find(v)?
    - Yes → u,v are in same component (u,v) would induce cycle
    - No → u,v are not in same component (u,v) won't induce cycle

Union-Find Implementation

MakeUnionFind(S)
Initialization: S={1,...,n}

Node 1  Node 2  Node n
1 | null | 1 | null | ...... | n | null
Pointers to parent in rooted tree
Size of set

node Find(v) {
  if (v.parent == null)
    return v
  else
    vRoot =Find(v.parent)
    return vRoot
}

Example: Find(v) = x

Union(Node u, Node v){
uRoot = Find(u),  vRoot=Find(v)
  if (uRoot==vRoot) return
  else if (uRoot.size > vRoot.size)
    vRoot.Parent = uRoot; uRoot.size+= vRoot.size;
  else
    uRoot.Parent = vRoot; vRoot.size+= uRoot.size;
}

Example: Union(u,v)

- uRoot is new root of Merged set
- vRoot is new root of Merged set
Path Compression

Example: Find(y)

```
node Find(v) {
    if (v.parent == null)
        return v
    else
        vRoot = Find(v.parent);
        v.parent = vRoot;
        return vRoot;
}
```

Path Compression

Example: Find(y) - every node on path from y to root x now points directly to x

Union Find: Running Time

(Path Compression + Union by Size)

- Amortized Running Time: $O(\alpha(n))$ per operation
- $\alpha(n)$ - Inverse Ackermann Function (Grows Incredibly Slowly)
- $\alpha(n) \leq 5$ for any value of n you will ever use on a computer!
- Could achieve same result with union by rank (height of tree)

MST Algorithms: Theory

Deterministic comparison based algorithms.
- $O(m \log n)$ [Jarník, Prim, Dijkstra, Kruskal, Boruvka]
- $O(m \log \log n)$ [Cheriton-Tarjan 1976, Yao 1975]
- $O(m (\log \log n)^2)$ [Fredman-Tarjan 1987]
- $O(m \alpha(m, n))$ [Gabow-Galil-Spencer-Tarjan 1986]
- $O(m \alpha(m, n))$ [Chazelle 2000]

Holy grail. $O(m)$.

Notable.
- $O(m)$ randomized [Karger-Klein-Tarjan 1995]
- $O(m)$ verification [Dixon-Rauch-Tarjan 1992]

Euclidean.
- 2-d: $O(n \log n)$ compute MST of edges in Delaunay
- k-d: $O(k n^2)$ dense Prim

Divide and Conquer

Divide-and-Conquer

- Break up problem into several parts.
- Solve each part recursively.
- Combine solutions to sub-problems into overall solution.

Most common usage.
- Break up problem of size n into two equal parts of size $\frac{n}{2}$.
- Solve two parts recursively.
- Combine two solutions into overall solution in linear time.

Consequence.
- Brute force: $n^2$
- Divide-and-conquer: $n \log n$.

Divide et impera.
Veni, vidi, vici.
-Julius Caesar
5.1 Mergesort

Mergesort

- Divide array into two halves.
- Recursively sort each half.
- Merge two halves to make sorted whole.

A Useful Recurrence Relation

Def. \( T(n) \) = number of comparisons to mergesort an input of size \( n \).

Mergesort recurrence.

\[
T(n) \leq \begin{cases} 
0 & \text{if } n = 1 \\
\frac{T(n/2^k)}{2} + \frac{T(n/2^k)}{2} + \frac{n}{2} & \text{otherwise}
\end{cases}
\]

Solution. \( T(n) \leq O(n \log_2 n) \).

Assorted proofs. We describe several ways to prove this recurrence. Initially we assume \( n \) is a power of 2 and replace \( \leq \) with \( = \).

Amer Hani

Copyright 2000, Kevin Wayne
Claim. If \( T(n) \) satisfies the following recurrence, then \( T(n) = n \log_2 n \).

\[
T(n) = \begin{cases} 
0 & \text{if } n = 1 \\
T(2) + a & \text{otherwise}
\end{cases}
\]

Pf. (by induction on \( n \))
- Base case: \( n = 1 \).
- Inductive hypothesis: \( T(n) = n \log_2 n \).
- Goal: show that \( T(2n) = 2n \log_2 (2n) \).

\[
T(2n) = 2T(n) + 2a
\]
\[
= 2n \log_2 (2n)
\]

More General Analysis

\[
T(n/b^k) = \frac{T(n)}{b^k} + \frac{a}{b^k}
\]

\[
T(n) \leq \frac{1}{\log n} \sum_{i=0}^{k-1} \left( \frac{\log n}{2} \right)^{i} \left( \frac{\log n}{2} \right)^{i} = \frac{1}{\log n} \sum_{i=0}^{k-1} \left( \frac{\log n}{2} \right)^{i} \left( \frac{\log n}{2} \right)^{i}
\]

Case 1: \( \gamma = \left( \frac{\log n}{2} \right) = 1 \)

\[
T(n) \leq \frac{1}{\log n} \sum_{i=0}^{k-1} \left( \frac{\log n}{2} \right)^{i} \left( \frac{\log n}{2} \right)^{i} = \frac{1}{\log n} \sum_{i=0}^{k-1} \left( \frac{\log n}{2} \right)^{i} \left( \frac{\log n}{2} \right)^{i}
\]

Case 2: \( \gamma < \left( \frac{\log n}{2} \right) < 1 \)

\[
T(n) \leq \frac{1}{\log n} \sum_{i=0}^{k-1} \left( \frac{\log n}{2} \right)^{i} \left( \frac{\log n}{2} \right)^{i} \leq \frac{1}{\log n} \sum_{i=0}^{k-1} \left( \frac{\log n}{2} \right)^{i} \left( \frac{\log n}{2} \right)^{i}
\]
More General Analysis

\[ T(n) = T(n/b) + T(n/b^2) + \ldots + T(n/b^k) \]

Case 3: \( k \geq 1 \)

\[ T(n) \leq \sum_{i=0}^{k-1} \left( \frac{n}{b^i} \right)^c \]

\[ \Theta \left( \frac{n^c}{\log bn} \right) \]