Algorithm Abstraction - Exam 1

2023-09-27

Contents

1 Chapter 1: Representative Problems and Gale-Shapley

1.1 Stable Matching Definitions

- **Algorithm**: A procedure that takes an input, transforms it, and then outputs the result
- **Unstable Matching**: A matching such that there exists a pair (x_1, y_1) in matching M where both x_1 and y_1 both prefer another partner to the one they currently have (unstable pair)
- **Stable Matching**: A matching such that there are **NO** unstable pairs
- **Perfect Matching**: A matching where every element of Set A is matched with exactly one element of set B

1.2 Gale-Shapley

Gale-Shapley: Algorithm that is guaranteed to find the same perfect matching and a stable matching every time

```
GALE{SHAPLEY (preference lists for hospitals and students)
    INITIALIZE M to empty matching.
    WHILE (some hospital h is unmatched and hasn't proposed to every student)
        s ← first student on h's list to whom h has not yet proposed.
    IF (s is unmatched)
        Add h{s to matching M.
    ELSE IF (s prefers h to current partner h')
        Replace h'-s with h-s in matching M.
    ELSE
        s rejects h.
    RETURN stable matching M.
```

2 Chapter 2: Algorithm Analysis

2.1 Big O Notation

- The upper bound of a function such that $f(n)$ is $O(g(n))$ if there exists constants $c > 0$ and $n_0 \geq 0$ such that $0 \leq f(n) \leq c * g(n)$ for all $n \geq n_0$

- Can be further expanded such that $f(n)$ is $O(g(n))$ if

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$$

- if f_1 is $O(g_1)$ and f_2 is $O(g_2)$, then $f_1 f_2$ is $O(g_1 g_2)$
- if f_1 is $O(g_1)$ and f_2 is $O(g_2)$, then $f_1 + f_2$ is $O(max\{g_1, g_2\})$

2.2 Big Omega(Ω) Notation

- The lower bound of a function such that $f(n)$ is $\Omega(g(n))$ if there exists constants $c > 0$ and $n_0 \geq 0$ such that $0 \leq c * g(n) \leq f(n)$ for all $n \geq n_0$

- Can be further expanded such that $f(n)$ is $\Omega(g(n))$ if

$$\lim_{n \to \infty} \frac{g(n)}{f(n)} = 0$$

2.3 Big Theta(Θ) Notation

- The tight bound of a function such that $f(n)$ is $\Theta(g(n))$ if there exists constants $c_1 > 0$, $c_2 > 0$, and $n_0 \geq 0$ such that $0 \leq c_1 * g(n) \leq f(n) \leq c_2 * g(n)$ for all $n \geq n_0$

- Can be further expanded such that $f(n)$ is $\Theta(g(n))$ if

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = c$$

3 Chapter 3: Graphs

3.1 Graph Types and Definitions

- **Undirected Graphs**: Set of nodes (vertices) and bidirectional edges between nodes
- **Directed Graphs**: Set of nodes and directional edges between nodes
- **Adjacency Matrix**: n-by-n matrix where $A_{uv} = 1$ if (u,v) is an edge
- **Adjacency Lists**: Node-indexed array of lists with only edges in the list
- **Path**: There is a path between two nodes if there is a sequence of edges and nodes that go from one node to another
- **Simple Path**: All nodes in the path are distinct
- **Connected Graph**: A graph is connected if there is a path for every pair of nodes
- **Cycles**: A cycle is a path $v_1, v_2, ..., v_k$ in which $v_1 = v_k$ and $k \geq 2$
- **Tree**: An undirected graph is a tree if its connected and does not contain a cycle

3.2 Connectivity and Traversal

- **Breadth-First Search**: Explore outward from starting node s in all possible directions one layer at a time. Runs in O(m+n) if graph is given as adjacency list

3.3 Bipartite Graphs

- **Bipartite Graphs**: An undirected graph is bipartite if the nodes can be colored blue or white such that every edge has one white and one blue end
- If graph G is bipartite, it cannot contain an odd-length cycle

3.4 DAGs and Topological Orderings

- **Directed Acyclic Graph (DAG)**: A DAG is a directed graph that contains no directed cycles
- **Topological Order**: An ordering of a directed graph such that for nodes $v_1, v_2, ...v_n$ and every edge (v_i, j_2), we have $i < j$

```
* Maintain the following information:
    - count(w) = remaining number of incoming edges
    - S = set of remaining nodes with no incoming edges
* Initialization: O(m + n) via single scan through graph.
* Update: to delete v
    -remove v from S
    - decrement count(w) for all edges from v to w;
      and add w to S if count(w) hits 0
    - this is O(1) per edge
```

- if G has a Topological Order, then G is a DAG

4 Chapter 4 - Greedy Algorithms (1)

4.1 Coin Change

- Given a currency, find a way for a cashier to give the customer the fewest possible number of coins
- **Cashiers Algorithm**: At each iteration, add coin of the largest value possible that does not take us past the remaining amount to be paid
- Cashiers Algorithm is not always optimal depending on the denomination of coins present

4.2 Interval Scheduling

- Job j starts at s_j and finishes at f_j
- Two jobs are compatible if they don't overlap
- Goal: Find maximum subset of mutually compatible jobs
- **Earliest-Finish-Time-First**

```
SORT jobs by finish times and renumber so that f1 <= f2 <= ... <= fn.
S <- 0 .
FOR j = 1 TO n
    IF (job j is compatible with S)
    S <- S U { j }.
RETURN S
```

- Optimal algorithm that takes O(nlogn) time

4.3 Interval Partitioning

- Goal: find minimum number of classrooms to schedule all lectures so that no two lectures occur at the same time in the same room
- **Earliest-Start-Time-First Algorithm**

```
SORT lectures by start times and renumber so that s1 <= s2 <= ... <= sn.
d <- 0.
FOR j = 1 TO n
    IF (lecture j is compatible with some classroom)
        Schedule lecture j in any such classroom k.
    ELSE
        Allocate a new classroom d + 1.
        Schedule lecture j in classroom d + 1.
        d <- d + 1.
RETURN schedule
```

- Can be implemented in O(nlogn) time and is optimal

4.4 Minimize Lateness

- Single resource processes one job at a time
- Job j requires tj units of time and due at time dj
- if j starts at time sj, finishes at time fj = sj + tj
- Goal: Goal: schedule all jobs to minimize maximum lateness $L = max_j l_j$

- **Earliest-Deadline-First**

```
SORT jobs by due times and renumber so that d1 <=  d2 <= ... <= dn.
t <- 0.
FOR j = 1 TO n
    Assign job j to interval [t, t + tj ].
    sj <- t ; fj <- t + tj.
    t <- t + tj.
RETURN intervals [s1, f1], [s2, f2], ..., [sn, fn]
```

- Earliest-Deadine-First Schedule S is optimal

4.5 Optimal Caching

- **Cache Hit**: Item already in cache when requested
- **Cache Miss**: Item not already in cache when requested, must be brought into cache and evict some existing item if cache is full
- Goal: Eviction Schedule that minimizes number of cache misses
- A reduced schedule is a schedule that only inserts an item into the cache in a step in which that item is requested
- Farthest-in-Future is an optimal eviction algorithm in offline caching
- **Offline Algorithm**: Full sequence of request is known beforehand
- **Online Algorithm**: Full sequence of request is **NOT** known beforehand
- Some other caching methods are **LIFO** which evicts page brought in most recently and **LRU** which evicts page whose most recent access was earliest (FF with direction of time reversed)

4.6 Dijkstras Algorithm

- **Dijsktras Algorithm**: Algorithm to find the shortest possible path from starting point s to destination point t where the edge weights are greater than 0.

```
DIJKSTRA (V, E, ℓ, s)

FOREACH v ≠ s : π[v] ← ∞, pred[v] ← null; π[s] ← 0.
Create an empty priority queue pq.
FOREACH v ∈ V : INSERT(pq, v, π[v]).
WHILE (IS-NOT-EMPTY(pq))
  u ← DEL-MIN(pq).
  FOREACH edge e = (u, v) ∈ E leaving u:
    IF (π[v] > π[u] + ℓe)
      DECREASE-KEY(pq, v, π[u] + ℓe).
      π[v] ← π[u] + ℓe ; pred[v] ← e.
```

4.7 Minimum Spanning Tree

- **Cut**: A partition of nodes into two nonempty subsets S and V-S
- **Cutset**: Cutset of a cut S is the set of edges with exactly one endpoint in S
- **Spanning Tree**: Let H=(V,T) be a subgraph of an undirected graph G=(V,E). H is a spanning tree of G is H is both acyclic and connected
- **Minimum Spanning Tree (MST)**: Given a connected, undirected graph G=(V,E) with edge costs c_e, a MST(V,T) is a spanning tree of G such that the sum of the edge costs in T is minimized