CPSC 320: Intermediate Algorithm Design and Analysis[1]

[1] Taught by Prof. Mehrdad Oveisi

Arpit Kumar

Fall 2023/24

These are my class notes, further research and a possibly random collection of facts from this course.

For further detail, see these notes: https://savreline.com/projects/320notes.pdf

Stable Matching Problem

Let's first consider a brute force approach. The idea is simple: generate all $n!$ perfect matchings, then loop through each of them and return the first stable matching found.

```
for each perfect matching M (i.e a potential solution) do
    if M is stable
        return M
return "no solution"
```

The algorithm for determining whether a solution is stable or not is described below.

```
for each e in E
    for each a in A
       if (e,a) is not in M
            find em such that (em, a) is in M
            find am such that (e, am) is in M
            if a prefers e over em and e prefers a over am
                return false
return true
```

Now we develop a worst case bound for the runtime of our brute force implementation. Before we do that we need to realize that our runtime bound is entirely dependent on the data structures we use for our implementation. If we are just given a list of matchings M, it is easy to see that most of our operations will run in linear time. So, we setup some structures to make our lives easier. We assume the following:

- We have an Employer-Matches$[1 \dots n]$ array whose eth entry is the applicant matches with employer e. This makes our if condition check as well as the first find call run in $O(1)$.
- We also create an Applicant-Matches$[1 \dots n]$.

- To store preferences, we create a Employer-Ranks[e][a] matrix. This makes checking for preferences run in $O(1)$.
- Similarly, we create a Applicant-Rank[a][e] matrix.

Now, building our arrays will take $O(n)$ time. Building our matrices take $O(n^2)$ time. We have a double for-loop, each of which runs n times. Since the operations within the loop are all constant time, we get our runtime to be

$$O(n + n^2 + n^2) = O(n^2)$$

Notice that this was just the function to check stability. We have $n!$ possible solutions, therefore, we call this function at worst $n!$ times, giving the final horrendous runtime of $O(n!n^2)$. Not good.

Gale-Shapely

```
// Initially all e in E and a in A are free
while a in A is free and has not applied to every employer
    let e be the highest ranked employer for a
    if (e is free) {
        (e,a) are now a tentative pair
    } else {
        let ae be the current pairing with e
        if e prefers a > ae {
            (e,a) are now a tentative pair
        }
        ae is free
    }
    remove e from a preference list
return all pairs at the end
```

The runtime for Gale-Shapely is $O(n^2)$. We can see this by observing that, in the worst case, we would need to traverse the full preference list for each of the n applicants. Since each preference list is also of size n, we get $O(n^2)$.

Next we provide a proof for correctness and termination.

Proof. @TODO □

Algorithm analysis

Formal definitions.

1. $g(n) \in \mathcal{O}(f(n))$ if $\exists c \in \mathbb{R}^+, \exists n_0 \in, n \geq n_0 \implies g(n) \leq c \cdot f(n)$

2. $g(n) \in \Omega(f(n))$ if $\exists c \in \mathbb{R}^+, \exists n_0 \in, n \geq n_0 \implies g(n) \geq c \cdot f(n)$
3. $g(n) \in \Theta(f(n))$ if $\exists c, d \in \mathbb{R}^+, \exists n_0 \in, n \geq n_0 \implies d \cdot f(n) \leq g(n) \leq c \cdot f(n)$

Graphs

A **graph** is a pair of sets (V, E) where V is the collection of vetices and E is the collection of edges. An **edge** is an unordered pair of vertices $(u, v) \in V \times V$.

Types of graphs

- **Simple graphs** have no self-edges or multi-edges.
- **Direction graph** is a graph where each edge is a directed edge.
- **Weighted graph** is one where each edge is assigned a weight.
- **Cyclic graph** contains atleast one cycle.
- **Connected graph** contains atleast one path between every pair of distinct vertices.
- **Complete graph** is one where every pair of vertices has an edge between them.
- **Strongly connected graph** is a directed graph where every pair of vertices is mutually reachable. A mutually reachable pair is one where there is a path from the head to tail and from the tail to the head.
- **Tree** is an undirected, connected, acyclic graph.

  Now consider the following mystery definition.

```
L_0 ← {s}
d ← 1
while L<d 6 = V do    // L<d is defined as ∪_{0≤i<d} L_i
    L_d ← N(L_d-1) - L_<d
    d ← d + 1
```

Claim: L_d is the set of nodes of distance exactly d from s.

Proof. □

Any two of the following statements implies the third:

1. G is a connected graph
2. G is acyclic, i.e is a tree
3. G has $n - 1$ edges

Reductions