Algorithm Design and Analysis(H)
Southern University of Science and Technology
Mengxuan Wu
12212006

Final Review

Mengxuan Wu

1 Algorithm Analysis

1.1 Models of Computation

1. **Deterministic Turing Machine:** Run time is the number of steps. Memory is the number of cells. No random access to memory or any other random operations.
2. **Word RAM:** Each memory location and input/output cell stores a w-bit integer/word. Run time is the number of primitive operations. Memory is the number of memories used.

1.2 Computational Complexity

We measure the efficiency of an algorithm as a function of the input size, where time complexity is the number of computation steps and space complexity is the number of memory cells used.

1.2.1 Desirable Scaling Properties

If the input size is increased by a constant factor c_1, the run time should increase by at most a constant factor c_2. A polynomial-time algorithm, whose run time is bounded by an^b for some constants $a > 0$ and $b > 0$, obeys this property.

1.2.2 Efficient Algorithms

An algorithm is efficient if it runs in polynomial time. And this definition is insensitive to models of computation.

In real life, polynomial-time algorithms often have a small exponent and constant. Hence, they are efficient compared to brute-force algorithms.

Exceptions For polynomial-time algorithms with a large exponent or constant, the algorithm may not be efficient in practice. Also, some exponential-time algorithms may be widely used because their worst-case instances are rare.

1.2.3 Asymptotic Notation

$T(n)$ is in $O(f(n))$ if there exist constants $c > 0$ and $n_0 > 0$ such that $T(n) \leq cf(n)$ for all $n \geq n_0$. $T(n)$ is in $\Omega(f(n))$ if there exist constants $c > 0$ and $n_0 > 0$ such that $T(n) \geq cf(n)$ for all $n \geq n_0$. $T(n)$ is in $\Theta(f(n))$ if $T(n)$ is both in $O(f(n))$ and in $\Omega(f(n))$.

Common Properties

1. Logarithms<Polynomials<Exponentials: For every $a > 1$ and $c > 0$, we have $\log_a n = O(n^c)$ and $n^c = O(a^n)$.
2. Factorials: $n! = 2^{\Theta(n \log n)}$, i.e., $\log n! = \Theta(n \log n)$.
3. Factorials>Exponentials: For every $r > 1$, we have $r^n = O(n!)$.

2 Stable Matching

2.1 Definition

A **perfect matching** is when everyone is matched monogamously. An unstable pair is a pair of people who prefer each other over their current partners. A **stable matching** is a perfect matching with no unstable pairs.

In some cases, a stable matching may not exist. While in other cases, there may be multiple stable matchings.

2.2 Gale-Shapley Algorithm

Algorithm 1: Gale-Shapley Algorithm

```
Initialize all people to be free;
while Some man is free and hasn't proposed to every woman do
    Choose such man m;
    Let w be the highest ranked women in m's preference list to whom m has not
     yet proposed;
    if w is free then
        (m, w) become engaged;
    end
    else if w prefers m to her current partner m' then
        (m, w) become engaged and (m', w) become free;
    end
    else
        w rejects m;
    end
end
```

2.2.1 Termination

Proof. Each man proposes to at most n women. Since there are at most n^2 proposals, the algorithm terminates in $O(n^2)$ time. □

2.2.2 Perfection

Proof by Contradiction. Assume someone is unmatched. Then, since the number of man and woman are equal, there must be another person of the opposite sex who is also unmatched.

For the unmatched woman, she must have been proposed by any man, otherwise she would have been matched. But since the unmatched man has proposed to every woman, he must have proposed to the unmatched woman.

Hence, we have a contradiction. □

2.2.3 Stability

Proof. Suppose (m, w) is an unstable pair.

Case 1 m never proposed to w. Then, m prefers his current partner to w. This contradicts the assumption that (m, w) is unstable.

Case 2 m proposed to w and w rejected m. Then, w prefers her current partner to m. This contradicts the assumption that (m, w) is unstable. □

2.2.4 Other Properties

Since multiple stable matchings may exist, which one is chosen by the algorithm?

We define a **valid partner** of a person as a partner that the person receives in some stable matching. The algorithm produces the following stable matchings:

Man Optimal Each man receives his most preferred valid partner.

Proof by Contradiction. Assume there exists some men who do not receive their most preferred valid partners.

Let m be the first men who do not receive his most preferred valid partner. Let w be m's most preferred valid partner.

Since m does not receive w, w must prefer her current partner to m, denoted as m'. But since m is the first men who do not receive his most preferred valid partner, w must be the most preferred valid partner of m'.

Then, in the stable matching where m is matched with w (this match must exist since w is m's most preferred valid partner), m' prefers w to his current partner. And w prefers m' to her current partner. This contradicts the assumption that the matching is stable. □

Woman Pessimal Each woman receives her least preferred valid partner.

Proof by Contradiction. Assume there exists some woman who does not receive her least preferred valid partner.

Let w be the woman who does not receive her least preferred valid partner. Let m be w's least preferred valid partner. Let m' be w's current partner.

In some stable matching, w is matched with m. Then, m' prefers w to his current partner (man optimal), and w prefers m' to m. This contradicts the assumption that the matching is stable. □

2.3 Extensions

1. Some people declare unacceptable partners.
2. Unequal number on both sides.
3. Limited positions (one to many).

3 Greedy Algorithms

3.1 Interval Scheduling

3.1.1 Definition

Given a set of intervals, how to select the maximum number of non-overlapping intervals?

3.1.2 Solution

Greedy Approach Select the compatible interval with the earliest finish time.

3.2 Interval Partitioning

3.2.1 Definition

Given a set of intervals, how to partition them into the minimum number of groups such that no two intervals in the same group overlap?

3.2.2 Solution

Greedy Approach Consider the intervals in increasing order of their start times. Assign each interval to the group with the earliest finish time. If no group is available, create a new group.

3.3 Minimizing Lateness

3.3.1 Definition

Given a set of jobs with processing time and deadline, how to minimize the total lateness (the amount of time a job finishes after its deadline)?

3.3.2 Solution

Greedy Approach Consider the jobs in increasing order of their deadlines. Schedule each job as early as possible.