

Summary of CPSC 320

Tom Wang

Summer, 2024

1 Review of CPSC 221

1.1 Asymptotic Analysis

- O -notation: $f(n) = O(g(n))$ if there exists $c > 0$ and n_0 such that $f(n) \leq cg(n)$ for all $n \geq n_0$.
- Ω -notation: $f(n) = \Omega(g(n))$ if there exists $c > 0$ and n_0 such that $f(n) \geq cg(n)$ for all $n \geq n_0$.
- Θ -notation: $f(n) = \Theta(g(n))$ if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.
- o -notation: $f(n) = o(g(n))$ if for all $c > 0$, there exists n_0 such that $f(n) < cg(n)$ for all $n \geq n_0$.
- ω -notation: $f(n) = \omega(g(n))$ if for all $c > 0$, there exists n_0 such that $f(n) > cg(n)$ for all $n \geq n_0$.

Some notes:

$$\log n < \sqrt{n} < n < n \log n < n^a < a^n < n!, \forall a > 1$$

1.2 Graph

Definition. an *articulation point* in an undirected graph is a vertex whose removal increases the number of connected components in the graph.

Definition. The *diameter* of an undirected, unweighted graph is the largest possible value of the following quantity:

- The smallest number of edges on any path between two nodes.
- In other words, it's the largest number of steps required to get between any two nodes in the graph.

2 Stable Matching

We have exactly n Employers and n Workers. Each Employer has a preference list of the Workers, and each Worker has a preference list of the Employers. We want to find a stable matching between the Employers and Workers.

A matching is a set of pairs (e, w) where e is an Employer and w is a Worker. A matching is stable if there is no pair (e, w) and (e', w') such that e prefers w' over w and w' prefers e over e' .

2.1 Gale-Shapley Algorithm

- 1: set all $s \in S$ and $w \in W$ to be free
- 2: **while** there is a free employer e **do**
- 3: $w = e$'s most preferred worker to whom e has not yet proposed
- 4: **if** w is free **then**
- 5: (e, w) become engaged

```

6:   else
7:       if  $w$  prefers  $e$  to her current employer  $e'$  then
8:            $e'$  becomes free
9:            $(e, w)$  become engaged
10:      end if
11:  end if
12: end while

```

Complexity: $O(n^2)$

2.2 Residence Hospital Matching

We have n Residents and m Hospitals, $m < n$. Each hospital has different number of slots s_i . The total number of slots is equal to the number of residents. Each resident has a preference list of the hospitals, and each hospital has a preference list of the residents. We want to find a stable matching between the residents and hospitals.

2.2.1 Reduction to Stable Matching

1. Create new hospital H_{ij} for each slot the hospital H_i has. Copy the preference list of H_i to H_{ij} .
2. Update the preference list of the residents to include the new hospitals, make sure the new hospitals are at the same rank as the original hospital. (expand the preference list without changing the order)
3. Apply the Gale-Shapley algorithm.
4. The matching is stable.
5. Transform the matching back to the original problem by replacing all H_{ij} with H_i .
6. The matching is stable.

3 Reductions

Definition. A *reduction* from problem A to problem B is a transformation of problem A into problem B in such a way that a solution to problem B can be used to solve problem A .

3.1 How reductions work

1. Show how to transform an arbitrary instance I_A of problem A into an instance I_B of problem B .
2. Show that the answer to I_A is “yes” if and only if the answer to I_B is “yes”.
3. Show how to transform the solution to I_B into a solution to I_A .

The solver is a black box that solves problem B .

4 Greedy Algorithms

Definition. A *greedy algorithm* is an algorithm that makes a sequence of choices, each of which is the best choice at the time (local maximum), without regard for the future.

4.1 Greedy stays ahead

Theorem. *If a greedy algorithm always makes a choice that stays ahead (at least as good) of the optimal solution, then the greedy algorithm is optimal.*

- Essentially a proof by induction.
- Compare to an optimal solution, show that the greedy solution is at least as good.

5 Divide and Conquer

Definition. *Divide and Conquer* is a problem-solving strategy that breaks a problem into smaller, simpler subproblems, solves the subproblems, and then combines the solutions to the subproblems to solve the original problem.

5.1 Recurrence Relations

The running time $T(n)$ of a recursive function can be described using a recurrence relation:

- $T(n)$ is defined in terms of one or more terms of the form $T(m)$ where $m < n$.
- For example: $T(n) = 2T(n/2) + O(n)$ with a base case $T(1) = O(1)$.

5.2 Recurrence Tree

- A tree that represents the recursive calls of a function.
- Each node represents a recursive call. The size of the subproblem is shown at the node.
- Next to each node, we write the cost of the work done at that level besides the recursive calls.
- Compute the total cost of the work done at each level of the tree.
- The total cost of the algorithm is the sum of the costs at each level.

5.3 Master Theorem

Theorem. *Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on the non-negative integers by the recurrence relation:*

$$T(n) = aT(n/b) + f(n)$$

where n/b means either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then $T(n)$ has the following asymptotic bounds:

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a} \log^k n)$, then $T(n) = \Theta(n^{\log_b a} \log^{k+1} n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq kf(n)$ for some constant $k < 1$ and sufficiently large n , then $T(n) = \Theta(f(n))$.

5.4 Prune and Search

Definition. *Prune and Search* is a problem-solving strategy that divides the search space into two parts, one of which is pruned (discarded) and the other is searched.

```
function QUICKSELECT(A[1:n], k)                                ▷ Returns the element of rank k in an array of n numbers
  if n = 1 then
    return A[1]
  end if
  Choose a random pivot element p from A
  Partition A into  $L = \{x \in A : x < p\}$ ,  $E = \{x \in A : x = p\}$ ,  $G = \{x \in A : x > p\}$ 
  if  $k \leq |L|$  then
    return QuickSelect(L, k)
  else if  $k \leq |L| + |E|$  then
    return p
  else
    return QuickSelect(G, k - |L| - |E|)
  end if
end function
```

6 Dynamic Programming

7 NP-Completeness