Homework 8

Thomas Kim tsk389 51835

David Munoz dam2989 51840

CS331 Algorithms and Complexity

Problem Q1: 13.2. Consider a county in which 100,000 people vote in an election. There are only two candidates on the ballot: a Democratic candidate (denoted D) and and Republican candidate R. As it happens, this county is heavily Democratic, so 80,000 people go to the polls with the intention of voting for D, and 20,000 go to the polls with the intention of voting for R.

However, the layout of the ballot is a little confusing, so each voter, independently and with probability $\frac{1}{100}$, votes for the wrong candidate - that is, the one that he or she *didn't* intend to vote for. (Remember that in this election, there are only two candidates on the ballot.)

Let X denote the random variable equal to the number of votes received by the Democratic candidate D when the voting is conducted with this process of error. Determine the expected value of X, and give an explanation of your derivation of this value.

Proof. Let A_i be a binary random variable where $A_i = 1$ if voter i voted for D and $A_i = 0$ if voter i voted for R.
By this, $X = \sum_1^{100000} A_i$
For $1 \leq i \leq 80000$, $E[A_i] = 0.99 \times 1 + 0.01 \times 0 = 0.99$
For $80000 < i \leq 100000$, $E[A_i] = 0.99 \times 0 + 0.01 \times 1 = 0.01$
Since each event is independent, the cumulative expectation is linear with respect to each individual event.
Therefore, $E[X] = \sum_1^{80000} 0.99 + \sum_{80001}^{100000} 0.01 = 79400$

□

Problem Q2(a): 13.11(a). *Load balancing algorithms* for parallel or distributed systems seek to spread out collections of computing jobs over multiple machines. In this way, no one machine becomes a "hot spot." If some kind of central coordination is possible, then the load can potentially be spread out almost perfectly. But what if the jobs are coming from diverse sources that can't coordinate? As we saw in Section 13.10, one option is to assign them to machines at random and hope that this randomization will work to prevent imblanaces. Clearly , this won't generally work as well as a perfectly centralized solution, but it can be quite effective. Here we try analyzing some variations and extensions on the simple load balancing heuristic we considered in Section 13.10.

Suppose you have k machines, and k jobs show up for processing. Each job is assigned to one of the k machines independently at random (with each machine equally likely).

Let $N(k)$ be the expected number of machines that do not receive any jobs, so that $\frac{N(k)}{k}$ is the expected fraction of machines with nothing to do. What is the value of the limit $lim_{k\to\infty}\frac{N(k)}{k}$? Give a proof of your answer.

Proof. Pr[Machine i doesn't get any jobs] $= (1 - \frac{1}{k})^k$
$\implies N(k) = k(1 - \frac{1}{k})^k$
$\implies \frac{N(k)}{k} = (1 - \frac{1}{k})^k$
According to WolframAlpha, $lim_{k\to\infty}(1 - \frac{1}{k})^k = \frac{1}{e}$ □

Problem Q2(b): 13.11(b). Suppose that machines are not able to queue up excess jobs, so if the random assignment of jobs to machines sends more than one jobs to a machine M, then M will do the first of the jobs it receives and reject the rest. Let $R(k)$ be the expected number of rejected jobs; so $\frac{R(k)}{k}$ is the expected fraction of rejected jobs. What is $lim_{k\to\infty}\frac{R(k)}{k}$? Give a proof of your answer.

Proof. The number of accepted jobs is $k - N(k)$ by definition.
Therefore, the number of rejected jobs is $k - (k - N(k))$
$R(k) = k - (k - N(k)) = k(1 - \frac{1}{k})^k \implies \frac{R(k)}{k} = (1 - \frac{1}{k})^k$
$lim_{k\to\infty} 1 - \frac{1}{k})^k = \frac{1}{e}$

□

Problem Q2(c): 13.11(c). Now assume that machines have slightly larger buffers; each machine M will do the first two jobs it receives, and reject any additional jobs. Let $R_2(k)$ denote the expected number of rejected jobs under this rule. What is $lim_{k\to\infty} \frac{R_2(k)}{k}$? Give a proof of your answer.

Proof. $\frac{N(k)}{k} = \frac{1}{e} \implies N(k) = \frac{k}{e}$
$\implies \frac{k}{e}$ processors will do 0 jobs.
Consider the following event E: job j is assigned to processor p, and jobs $l \neq j$ are not assigned to processor p.
The probability of this is $(1 - \frac{1}{k})^{j-1}(\frac{1}{k})(1 - \frac{1}{k})^{k-j}) = \frac{1}{k}(1 - \frac{1}{k})^{k-1}$
There are k such possibilities for j and k such possibilities for $p \implies Pr[E] = \frac{k^2}{k}(1-\frac{1}{k})^{k-1} = k(1-\frac{1}{k})^{k-1}$
We see $lim_{k\to\infty} k(1 - \frac{1}{k})^{k-1} = \frac{1}{e} \implies \frac{k}{e}$ processors will have exactly 1 job.
The remainder of the processors will process 2 jobs and reject the rest, so $\frac{k}{e}$ do 0 jobs, $\frac{k}{e}$ do 1 job, $k - \frac{2k}{e}$ do 2 jobs.
Therefore, the number of jobs that get done is $\frac{k}{e} + 2k - 2\frac{2k}{e} = 2k - \frac{3k}{e}$.
Therefore, the number of jobs that don't get done is $k - 2k + \frac{3k}{e} = -k + \frac{3k}{e} = \frac{3k-ek}{e} = \frac{k(3-e)}{e}$
Now we see the fraction of jobs that doesn't get done is $\frac{\frac{k(3-e)}{e}}{k} = \frac{3-e}{e}$ The limit of $R_2(k)$ specified in the problem describes the number of rejected jobs as k approaches infinity.
The above calculation calculates the number of 1-accept and 2-accept processors as k approaches infinity.
Therefore, the number of rejected jobs is simply the difference between k and the sum of the 1-accept and 2-accept processors.
Therefore, $lim_{k\to\infty} \frac{R_2(k)}{k} = \frac{3-e}{e}$

□

Problem Q3(a): 8.1(a). For each of the two questions below, decide whether the answer is "Yes," "No," or "Unknown, because it would resolve $\mathcal{P} = \mathcal{NP}$" Give a brief explanation of your answer. Let's define the decision version of the interval scheduling problem from chapter 4 as follows: Given a collection of intervals on a time-line, and a bound k, does the collection contain a subset of nonoverlapping intervals of size at least k? Question: Is it the case that Interval Scheduling $\leq_p$ Vertex Cover? **Yes**
Interval Scheduling has a polynomial-time greedy solution (earliest finish time). Vertex Cover is NP-Complete.

Problem Q3(b): 8.1(b). Question: Is it the case that Independent Set $\leq_p$ Interval Scheduling? **Unknown** Independent Set is NP-complete. If Independent Set $\leq_p$ Interval Scheduling, there would exist a polynomial algorithm to decide Independent Set, which by extension would mean there exists a polynomial-time algorithm to solve all NP problems.

Problem Q4: 8.22. Suppose that someone gives you a black-box algorithm $\mathcal{A}$ that takes an undirected graph $G = (V, E)$ and a number k, and behaves as follows:

- If G is not connected, it simply returns "G is not connected"
- If G is connected and has an independent set of size at least k, it returns "yes"

- If G is connected and does not have an independent set of size at least k, it returns "no"

Suppose that the algorithm $\mathcal{A}$ runs in time polynomial in this size of G and k.

Show how, using calls to $\mathcal{A}$, you could then solve the Independent Set Problem in polynomial time: Given an arbitrary undirected graph G, and a number k, does G contain an independent set of size at least k?

1: Use BFS to find a set of connected subgraphs of G $\mathcal{G} = \{G_1, G_2, \ldots, G_m\}$
2: $SUM := 0$
3: **for** $\forall G_i \in \mathcal{G}$ **do**
4: $MAX := 0$;
5: **for** $\forall 1 \leq j \leq |G_i|$ **do**
6: **if** $\mathcal{A}(G_i, j) =$ "yes" **then**
7: $MAX := j$
8: **end if**
9: **end for**
10: $SUM := SUM + MAX$
11: **end for**
12: **if** $SUM \geq k$ **then**
13: **return** "YES"
14: **else**
15: **return** "NO"
16: **end if**

Runtime

Proof. BFS can be done in polynomial time $O(|E|)$.
$\mathcal{A}$ is called on each connected component up to $|V|$ times.
This is $O(|V|^2)$ time, which is polynomial.
Therefore, the total runtime is polynomial.

□

Correctness

Proof. The algorithm identifies the maximum size independent set for each connected component.
By definition, different connected subgraphs do not share any edges.
Therefore, the union of two independent sets for two different connected components is an independent set for the graph composed of these two components.
By line 5, only the largest independent set for each connected component is used in the final calculation.
By lines 7 and 10, the algorithm sums each of the independent sets for each connected component.
By the previous claim, the union of these sets is a valid independent set for each connected component.
By the previous claim, the largest independent set for the entire graph is the union of the aforementioned sets.
By line 12, this cardinality for the largest independent set is compared to k.
Therefore, the algorithm will always accurately decide the Independent Set Problem, since it always finds the largest independent set.

□