Problem Set 5 Solutions

Exercise 5-1. Do exercise 8.3-4 on page 173 of CLRS.

Solution:
View each number as composed of 2 digits where each digits is composed of $\lg n$ bits and can take on $n$ possible values. Radix sort takes $\Theta(d(n+k))$ time where $d$ is the number of digits and $k$ is the number of possible values for the digits. In this case, the running time is $\Theta(n)$ as $d$ is a constant and $k = n$.

Exercise 5-2. Do exercise 11.2-5 on page 229 of CLRS.

Solution:
There are $m$ slots and $nm$ possible keys. Using pigeonhole principle, there are must be one slot with at least $n$ keys regardless of the hash function used.

Exercise 5-3. Do exercise 16.2-4 on page 384 of CLRS.

Solution:
Professor Midas should repeatedly drive to the farthest station that he can reach with his gas tank before refueling. To show that the greedy choice works, assume that the optimal set of stations does not contain the furthest station reachable from the original station. Replacing the first stop $x$ with the furthest station $z$ results in another optimal solution as the second station is also reachable from $z$.

Problem 5-1. Longest-probe bound for hashing

(a) The table is at most half full. Hence under the uniform hashing assumption, the probability of the first probe failing is at most $1/2$. The ratio of empty to filled slots increases with each failed probe, hence the conditional probability of failing at the $j$-th probe given that all probes up to $j-1$ has failed is less than $1/2$. Hence, the probability of a sequence of $k$ failures is at most $(1/2)^k$.

(b) Substituting $k = 2\log n = \log n^2$ into part (a) gives a probability of failure of no more than $2^{-\log n^2} = 1/n^2$.

(c) We have
\[ P\{X > 2\log n\} \leq \sum_{i=1}^{\frac{n}{\log n}} P\{X_i > 2\log n\} \leq \frac{n}{n^2} = 1/n. \]
(d) The expected length of the longest probe

\[
E[X] = \sum_{i=1}^{n} i P\{X = i\} \\
\leq 2\lg n P\{X \leq 2\lg n\} + n P\{X > 2\lg n\} \\
\leq 2\lg n + 1 \\
= O(\lg n).
\]

Problem 5-2. Planning a company party

Consider an optimal solution. If the person at the root is absent, then the optimal value is the sum of the optimal values of all the subtrees rooted at its children. To see this, if any of the subtree does not have the optimal value, the solution can be improved by using the optimal value for that subtree. Similarly, if the person at the root is present, then the optimal value is the sum of the person’s conviviality with the optimal values of all the subtrees rooted at its children when none of the people associated with the children nodes are present.

Add three fields to each node: \(a[i]\) and \(p[i]\) which stores the optimal value in the subtree rooted at \(i\) if \(i\) is absent and present respectively and \(guest[i]\) which indicates whether \(i\) is a guest. Finding the optimal sum can be done by running the following algorithm \textsc{Calc-Max} from the root and checking whether \(a[root]\) or \(p[root]\) is larger. To fill in the \textit{guest} field, run \textsc{List-Guest} from the root after \textsc{Calc-Max}.

\begin{verbatim}
CALC-MAX(x)
1 a[x] = 0, p[x] = conviviality[x]
2 y = left-child[x]
3 while y \neq nil
4 Calc-Max(y)
5 y = right-sibling[y]
6 if x is not the root
7 a[parent[x]] = a[parent[x]] + max{a[x], p[x]}
8 p[parent[x]] = p[parent[x]] + a[x]
\end{verbatim}
Handout 6: Problem Set 5 Solutions

List-Guest(x)

1. If \(x = root\)
2. If \(p[x] > a[x]\)
   - \(guest[x] = true\)
3. Else \(guest[x] = false\)
4. Else
5. If \(guest[parent[x]] = true\)
   - \(guest[x] = false\)
6. Else
7. If \(p[x] > a[x]\)
   - \(guest[x] = true\)
8. Else \(guest[x] = false\)
9. \(y = left\text{-}child[x]\)
10. While \(y \neq \text{nil}\)
11. List-Guest(y)
12. \(y = right\text{-}sibling[y]\)

Since each node is processed once and all operations in the processing run in constant time, the running time is \(O(n)\).