We study randomized algorithms that take as input a set $S$ of $n$ keys from some large universe $U$ or a set of $n$ key-value pairs, associating each key from $S$ with a specific value, and build a data structure that solves one of the following tasks. On "lookup" for some key $x \in U$: \begin{itemize} \item Decide set membership with respect to $S$ (membership tester). \item If $x \in S$, then return the value associated with $x$. If $x \in U-S$, then return either some specific value "not in $S$" (dictionary), or some arbitrary value (retrieval data structure). \item If $x \in S$, then return a natural number associated with $x$, where for all elements of $S$ the numbers are pairwise distinct and the numbers for elements $x \in U-S$ are arbitrary (perfect hash function). \end{itemize} The data structures that we cover have the same simple structure. They consist of a table with $m$ cells, each capable of holding entries of size $r$ bits, as well as of a constant number of hash functions, which are used to map elements from $U$ to a constant number of table cells. Assuming fully random hash functions, we will discuss how such data structures can be constructed in time linear in $n$, and what load $c=n/m$ or space utilization $m \cdot r$, respectively, can be achieved in trade-off with the number of cell probes for "lookup". This leads to the question if a random bipartite graph with $n$ nodes (keys) on the left, $m$ nodes (cells) on the right, and edges determined by the hash functions, with high probability has a matching of a certain type, and furthermore, how such a matching can be calculated efficiently.