Hands-On 3: Algorithm Design A.Y. 2024

Giacomo Trapani   Leonardo Crociani

01 April 2024

1 Exercise 1

Design a space-efficient data structure D for prefix sums over a binary vector (bitvector) B of n bits. D replaces B, and should use O(n) bits; moreover, it must support the constant-time operation rank(i).

Given a binary vector B, such as:

Val	1	0	1	1	0	1	0	0	1	1	0	1	0	1	0	1
Pos	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

We can build D sampling B.

1.1 Sampling

The idea for the sampling is the following: we take a subset of the positions of B. The items are selected every l bits. For these items, we compute the rank() operation. We store these prefix sums in an array D.

With this approach we can answer in O(1) time every query that targets a position that is a multiple of l.

1.2 Lookup table, popcount

To simplify the following reasoning, we assume we can read a constant fraction $1/k$ of l consecutive bits in $O(1)$ time.

Let $\text{idx} = \left\lfloor \frac{i}{l} \right\rfloor$, $\text{offset} = i - l \cdot \text{idx}$.

For every position which is not a multiple of l, we read k bits starting from position idx+1 up to position $\lceil \text{offset}/k \rceil$, other operations which take $O(1)$ time.

In order to compute the prefix sum of those bits, we can make use of the popcount(i, j) instruction which calculates the prefix sum up to position j of bits in the binary representation of i in $O(1)$ time. This instruction is available on modern processors, and there are many fast implementations of this algorithm at software level (such as GCC's `__builtin_popcount`).

We can summarize the steps to compute the rank operation as such:

```
function rank(i):
  idx = floor(i/l)
  offset = i-D.l*idx
  result = D[idx];
  if (l*idx != i):
    for(index = idx+1; index < offset; index += k):
      i = B[index...min(index+k-1, B.length)]
      if index+k-1 >= B.length:
        j = B.length - index
      else:
        j = index + k - 1 < offset ? k : offset % k
      result += popcount(i, j)
  return result
```

Another option would be to use a lookup table `ppc_lookup_table` to memorize the result of the popcount operation up to a certain threshold. In this case, we are using additional space other than the one required by the prefix sum array D: assuming we are sampling every $l = \log(n)/2$ values, we can still get a space complexity of $O(n)$.

Specifically, we build a table of size:

$$2^l \times l$$

The table has a row for each value from 0 to $2^l - 1$ and a column for each value from 0 to l - 1.

In the cell [i, j] we store the prefix sum up to position j of the bits in the binary representation of i. So for a generic query rank(i), the answer will be the sum of the nearest sample in D and a value in the lookup table. We can compute the rank of the any i-th position in O(1) time using the following formula:

$$\text{rank(i)} = D[\text{idx}] + T[\text{idx+1:i, offset}]$$

As required, the data structure takes O(n) bits:

$$\begin{aligned}
&|T| = 2^l \cdot l \cdot \log(\log(n)) = O(n) \\
&|D| = O(n) \\
&|T| + |D| = O(n) + O(\log(n)) = O(n)
\end{aligned}$$

□

2 Exercise 2

Design an alternative to Bloom filter with failure probability f as an approximate dictionary.

First, we show how to reconstruct any value of B starting using D in constant time, then we show that we can get failure probability f and that D takes space $n \cdot \log_2\left(\frac{1}{f}\right) + o(n)$.

2.1 Reconstruct values in B

We can reconstruct B using D:

1. The first element of B is equal to the first element of D
$$B[1] = D[1]$$
2. Take $i \in \mathbf{N}, i \in [2; n]$. The i-th element of B is equal to the difference between the i-th element and the $(i-1)$-th element of D
$$B[i] = D[i] - D[i-1]$$

$\square$

2.2 Calculate the failure probability of D

We can take a universal hash function $h : U \to \mathbb{Z}_{n/f}$.

By definition of h we know that

$$\forall x, y \in U, x \neq y. \ \Pr[h(x) = h(y)] \leq \frac{1}{n/f} = \frac{f}{n}$$

We define n random indicator variables X_i s.t.

$$X_i = \begin{cases} 1 \text{ if we get a conflict for the } i\text{-th element} \\ 0 \text{ otherwise} \end{cases}$$

The failure probability of D is equal to the sum of the expected values of such r.i.d. variables:

$$\sum_{i=1}^{n} E[X_i] = n \cdot \Pr[h(x) = h(y)] = n \cdot \frac{f}{n} = f.$$

$\square$

2.3 Calculate the size of D

We shall give both a lower and an upper bound on the size of D to prove the thesis.

Throughout the following calculations, we make use of the following relation

$$\log_2 \binom{x}{y} \leq y \cdot \log_2 \frac{x}{y}$$

Lower Bound

From information theory, we know that the minimum number of bits required to describe a subset A of elements chosen from a set B is $\log_2 \binom{B}{A}$.

We shall call S the set of keys we are mapping using D, we know $|S| = n$.

Since D is an approximate dictionary for S, it implicitly gives an exact dictionary for S' and as seen in class we can compute its size:

$$
\begin{aligned}
&f = \frac{|S' \setminus S|}{|U|} \\
&|S'| = |S| + |S' \setminus S| \\
&|S'| = |S| + f|U| \\
&|S'| = n + fm
\end{aligned}
$$

We call E the exact dictionary for S, which is composed by D' the exact dictionary for S' and an additional number of bits to represent $S' \setminus S$.

We can now calculate $minsize(E)$

$$
\begin{aligned}
&minsize(E) = size(D') + \log_2 \binom{|S'|}{|S' \setminus S|} \\
&= b' + \log_2 \binom{n + fm}{n} \\
&\leq b' + n \log_2 \left(\frac{n + fm}{n}\right)
\end{aligned}
$$

We can combine the boundary given above with the theoretical bound from information theory mentioned above and get a lower bound on the size of D

$$
\begin{aligned}
&b' + n \log_2 \left(\frac{n + fm}{n}\right) \geq \log_2 \binom{m}{n} \\
&b' + n \log_2 \left(\frac{n + fm}{n}\right) \geq n \log_2 \frac{m}{n} \\
&b' \geq n \left(\log_2 \left(\frac{m}{n}\right) - \log_2 \left(\frac{fm}{n}\right)\right) \\
&b' \geq n \left(\log_2 \left(\frac{m}{n} \times \frac{n}{fm}\right)\right) \\
&b' \geq n \log_2 \left(\frac{1}{f}\right).
\end{aligned}
$$

Upper Bound

We know the size of D to be

$$
size(D) = \log_2 \binom{n}{k} + o(n)
$$

We can rewrite the logarithmic part of $size(D)$ to get our target value.

$$\begin{aligned}
&\log_2 \binom{n}{k} \leq k \cdot \log_2 \left(\frac{n}{k}\right) \\
&\leq k \cdot \log_2 \left(\frac{n/f}{k}\right) \text{ because } f < 1 \\
&\leq n \cdot \log_2 \left(\frac{n/f}{n}\right) \text{ because } k \leq n \\
&= n \cdot \log_2 \left(\frac{1}{f}\right)
\end{aligned}$$

We can take both the **lower bound** and the **upper bound** to get a bound on the size of D:

$$n \cdot \log_2 \left(\frac{1}{f}\right) \leq size(D) \leq n \cdot \log_2 \left(\frac{1}{f}\right) + o(n)$$

$\square$