Algorithm Analysis and Design Homework 8

10192100415 李瑞鹏

2022 年 11 月 30 日

• Suppose we perform a sequence of n operations on a data structure in which the i-th operation costs i if i is an exact power of 2, and 3 otherwise. Use (1) aggregate analysis, (2) accounting method, (3) potential method to determine the amortized cost per operation.

Aggregate analysis

$$\begin{aligned}\sum_{i=1}^{n} c_i &= \sum_{i=0}^{\lfloor \lg n \rfloor} 2^i + 3(n - \lfloor \lg n \rfloor - 1) \\ &< 2^{1+\lg n} - 1 + 3n \\ &= 5n - 1 \\ &= O(5n).\end{aligned}$$

Thus the average cost of each operation is $O(5n)/n = O(1)$.

Accounting method Charge 5 for each operation ($\hat{c}_i = 5$). If i is not a power of 2, pay 1 from credit. If i is a power of 2, pay i from credit. Since we've prooved that

$$\sum_{i=1}^{n} c_i < 5n - 1 < 5n = \sum_{i=1}^{n} \hat{c}_i,$$

such an amortized cost is valid, i.e. each operation costs $5 = O(1)$ on average.

Potential method Define a potential function

$$\Phi(D_i) = \begin{cases} 0, & i = 0, \\ 2i - (2^{1+\lfloor \lg i \rfloor} - 1), & i \geqslant 1. \end{cases}$$

If i is not a power of 2, then

$$\begin{aligned}\hat{c}_i &= c_i + \Phi_i(D_i) - \Phi(D_{i-1}) \\ &= 3 + 2 - 2^{1+\lfloor \lg i \rfloor} + 2^{1+\lfloor \lg(i-1) \rfloor} \\ &= 5.\end{aligned}$$

If i is a power of 2, i.e. $i = 2^j$ for some $j \in \{0, 1, 2, \cdots\}$, then

$$\begin{aligned}\hat{c}_i &= c_i + \Phi(D_i) - \Phi(D_{i-1}) \\ &= i + 2 - 2^{1+j} + 2^j \\ &= 2.\end{aligned}$$

Since $\hat{c}_i \leqslant 5$, the average cost of each operation is $O(1)$.

• 22.1-6

Assume that the adjacency matrix is M. Vertex k is a universal sink iff the k-th row of M is 0 and the k-th column of M is 1 (excluding $M[k,k]$).

We scan M from $M[1,1]$.

- If $M[i,j] = 1$, the target row $k > i$, so we increase i.
- If $M[i,j] = 0$ and $i \neq j$, the target column $k > j$, so we increase j.
- If $M[i,j] = 0$ and $i = j$, i is the potential target row (but we do not know whether column i is 1). We increase j to check whether the row is 0.

The scan is as follow:

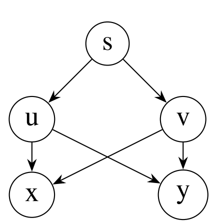
```
i = j = 1
while i < |V| and j < |V|
    if M[i, j] = 1
        i = i + 1
    else if M[i, j] = 0
        j = j + 1
```

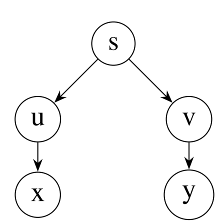
If there exists a universal sink, it must be vertex i. We just add a check at the end.

```
CONTAIN-UNIVERSAL-SINK(M)
    i = j = 1
    while i < |V| and j < |V|
        if M[i, j] = 1
            i = i + 1
        else if M[i, j] = 0
            j = j + 1
    for j = 1 to |V|
        if j ≠ i and M[j, i] = 0
            return false
    return true
```

The scan costs $O(V)$, and the check costs $O(V)$. The algorithm costs $O(V)$.

• 22.2-6

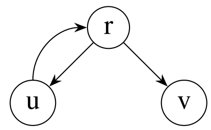




The G is shown left, and the tree is shown right. The shortest path from s to v is the same, but BFS will never generate such a tree.

- 22.3-9

  Such counterexample contains a loop back.



We start DFS from r and visit u before v. $v.d = 4$, $u.f = 3$. $v.d \nleq u.f$, but there exists a path $u \to r \to v$.