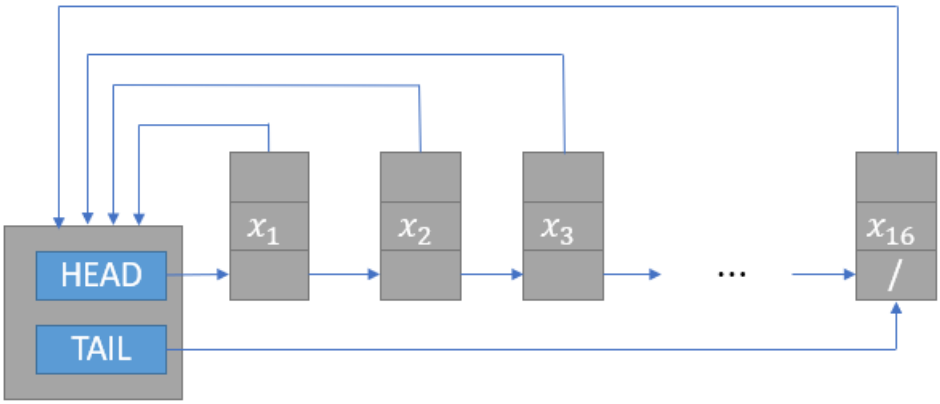
Sample Solution for Problem Set 1

Data Structures and Algorithms, Fall 2021

December 5, 2021

Contents

1 Problem 1



Find(x_2) = Find(x_9) = x_1.

2 Problem 2

Without loss of generality, we may assume that $n = 2^k$ for some $k \in \mathbb{N}$. Consider the following operation sequence:

- In the i-th round ($1 \leq i \leq k$), union x_{j*2^i} and $x_{(2j+1)*2^{i-1}}$ for all $0 \leq j \leq 2^{k-i}$.

It can be seen that the depth of the tree is $\Theta(\log n)$. Hence, it remains to query one of the leaves with $\Theta(\log n)$ depth $m - n$ times, achieving $\Omega(m \log n)$ time complexity.

3 Problem 3

(a) Let t_i denotes the number of times the $node[i]$ would be visited during the over procedure. If we implement FIND without path compression, the running time of the over procedure is linear in $\sum_{i=1}^{n} t_i$.

In the worst case, all the nodes form a chain. The node with depth d would be visited $n - d$ times.

$$\sum_{i=1}^{n} t_i = \sum_{i=1}^{n} i = \frac{n(n+1)}{2} = O(n^2)$$

(b) If we implement FIND with path compression, the running time of the over procedure is still linear in $\sum_{i=1}^{n} t_i$.

Nodes can be divided into two types:

- Root nodes: the number of times a root node would be visited during the over procedure is equal to the size of the tree rooted in it.
- Non-root nodes: the number of times a non-root node would be visited during the over procedure is equal to the number of its children plus one.

Let R be the set of root nodes. Then we have

$$\sum_{i=1}^{n} t_i = \sum_{i \in R} t_i + \sum_{i \in [n] \setminus R} t_i \leq n + 2n = O(n)$$

The above procedure runs in O(n) time in the worst case.

4 Problem 4

```cpp
struct String{
    deque <char> str;
    bool rev;
};
Strings* NewString(char c){return &(new String({deque <char>(1, c), 0}));}

String* reverse(String* S){
    S->rev ^= 1;
    return S;
}

char Lookup(String* S, int k){// 0 base for string
    if(S->str.size() >= k) return NULL;
    return S->rev ? S->str[S->str.size() - 1 - k] : S->str[k];
}
String* concat(String* S, String* T){
    if(S->str.size() > T->str.size()){
        for(int i = 0; i < T->str.size(); i ++){
            if(S->rev) S->str.push_front(Lookup(T, i));
            else       S->str.push_back (Lookup(T, i));
        }
        free(T);
        return S;
    }
    else{
        for(int i = 0; i < S->str.size(); i ++){
            if(T->rev) T->str.push_back (Lookup(S, i));
            else       T->str.push_front(Lookup(S, i));
        }
        free(S);
        return T;
    }
}
```

- Analysis:
  - We only analyze the time complexity of function $concat$.
    As well as $union - by - size$ of $DisjointSet$, for each element(character), whenever its string changes, its string size at least doubles.
    So each element's string changes $O(lgn)$ times, take $O(nlgn)$ time in total, and $O(lgn)$ for each merge operation in average.
  - It's obvious that other operations take $O(1)$ for one time.
  - Each character exists in memory for only one time, so the space complexity is $O(n)$.

5 Problem 5

- Proof:
  - I.B: For connected acyclic graph with $n = 2$ vertices, it's obvious that both vertices has 1 degree.
  - I.H: For $1 \leq n \leq k$, connected acyclic graph with n vertices has at least two vertices with degree 1.
  - I.S: For $n = k + 1$, we can cut one edge in the connected acyclic graph. Because it's acyclic, so the graph is divided into two connected acyclic subgraphs with less vertices. Each subgraph has at least two vertices, or only one isolated point. And we add the cut edge, each subgraph can provide at least one vertice with 1 degree.

6 Problem 6

Let $C = BB^T$, then

$$\begin{aligned} c_{ij} &= \sum_{k=1}^{|E|} b_{ik} b_{kj}^T \\ &= \sum_{k=1}^{|E|} b_{ik} b_{jk} \end{aligned}$$

- $i \neq j$
  If edge k is (i, j) or (j, i), $b_{ik}b_{jk} = 1 \times (-1) = -1$, otherwise $b_{ik}b_{jk} = 0$.
  So $c_{ij} = \sum_{k=1}^{|E|} b_{ik}b_{jk}$ is non-positive and $|c_{ij}|$ is the total number of edges connecting i and j.

- $i = j$
  If edge k is like $(i, *)$ or $(*, i)$, then $b_{ik}b_{jk} = b_{ik}^2 = 1$, otherwise $b_{ik}^2 = 0$.
  So $c_{ii} = \sum_{k=1}^{|E|} b_{ik}^2$ is non-negative and $c_{ii} = |\deg i|$