Homework Assignment 12

Disjoint Sets and Huffman Coding

Assigned: Tuesday, April 19, 2022

Due: Monday, April 25, 2022 at 11:59pm

PROBLEM 1 2 POINTS

Complete all Homework Assignment 12 activities in the zyBook.

PROBLEM 2 10 POINTS

The union by rank and path compression heuristics are two ways to speed up a disjoint set forest data structure. Pseudocode for the disjoint set forest operations (Make-Set, Find-Set, and Union) using both heuristics is shown below.

```
MAKE-SET(x)
1  x.p = x
2  x.rank = 0

UNION(x, y)
1  LINK(FIND-SET(x), FIND-SET(y))

LINK(x, y)
1  if x.rank > y.rank
2       y.p = x
3  else x.p = y
4       if x.rank == y.rank
5            y.rank = y.rank + 1
```

```
FIND-SET(x)
1  if x ≠ x.p
2       x.p = FIND-SET(x.p)
3  return x.p
```

a) Implement a disjoint set forest to store disjoint sets of unique integer elements using the function headers provided in `disjointSetForest.cpp`. ***Your implementation should allow either of the two heuristics be used or not used.*** Submit your implementation along with your code for parts (b) and (c) in `disjointSetForest.cpp`. Sample output for the code block in the `main()` function is provided at the appendix at the end of the assignment. (4 points)

b) Devise a sequence of N Make-Set and $N-1$ Union operations that is representative of a worst-case scenario for a disjoint set forest that does not use the two heuristics. Compare the growth rate for the wall clock time for this sequence of operations on disjoint set forests with $N = \{10^2, 10^3, 10^4\}$ elements ***without using either heuristic and with using both heuristics***. (3 points)

c) To simulate an average case behavior, consider the following sequence of operations:

```
Make N sets of singletons (single elements).
for (i = 0; i < N; i++) {
   element1, element2 = Randomly selected elements
   Union(element1, element2)
}
```

Compare the growth rate of the average wall clock time for this sequence of operations for $N \in \{10^3, 10^4, 10^5, 10^6\}$ over 10 runs for each N ***using union by rank only, path compression only, and using both heuristics***. (Do not try this for large N without an optimization heuristic unless you enjoy waiting for a long time!). How does the average wall clock time grow as a function of N for each combination of optimization heuristics? (3 points)

PROBLEM 3 3 POINTS

You are given the following sequence of characters: AAATCCGAAGCCACATGGCA.

a) Construct a fixed-length bit code for this sequence and present a table listing the codeword for each character. Compute the number of bits used to represent this sequence. (1 points)

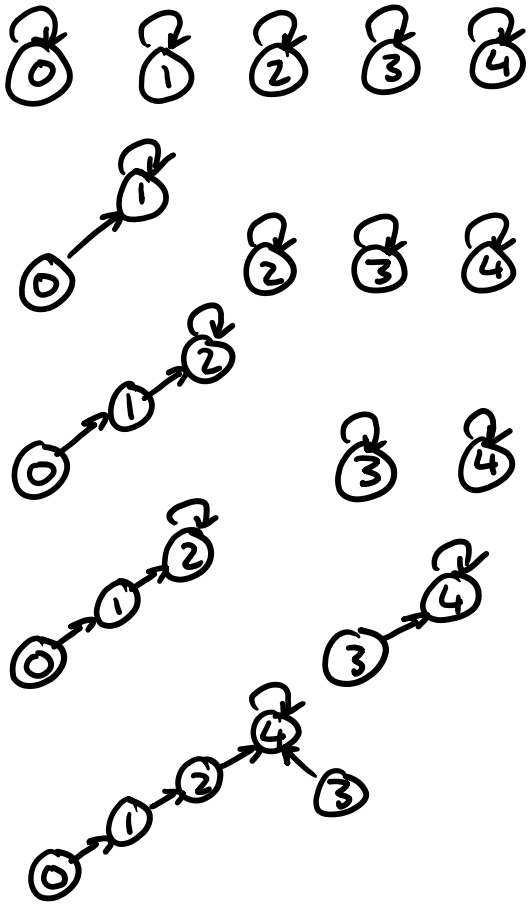
b) Construct a Huffman code for this sequence and present a table listing the codeword for each character. (You can do this either by hand or by writing a program.) Compute the number of bits used to represent this sequence. (1 points)

c) Compute a lower bound on the number of bits that must be used to compress this sequence in a lossless manner. How do the fixed-length and Huffman codes compare to this lower bound? (1 point)

APPENDIX

Output from `disjointSetForest.cpp` with `unionByRank = false, pathCompression = false`:

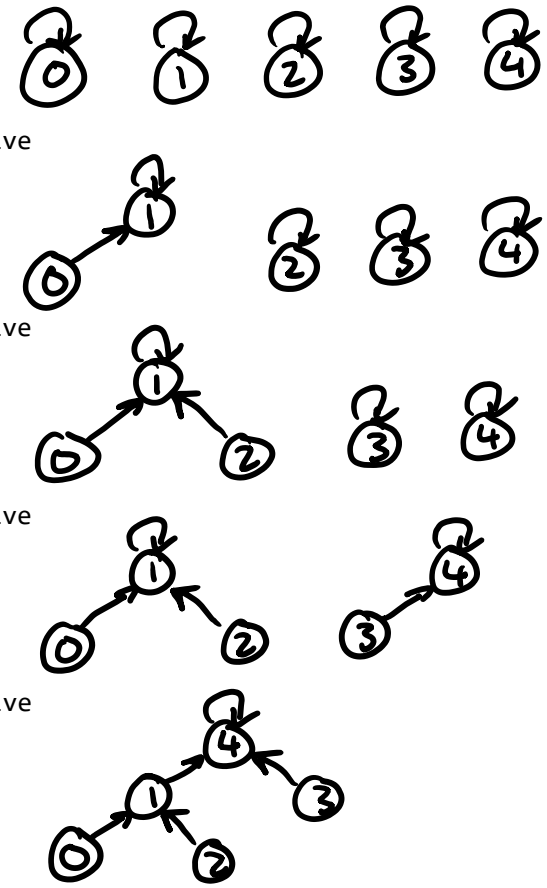
```
Element  Parent  Set Representative
0  0  0
1  1  1
2  2  2
3  3  3
4  4  4
Element  Parent  Set Representative
0  1  1
1  1  1
2  2  2
3  3  3
4  4  4
Element  Parent  Set Representative
0  1  2
1  2  2
2  2  2
3  3  3
4  4  4
Element  Parent  Set Representative
0  1  2
1  2  2
2  2  2
3  4  4
4  4  4
Element  Parent  Set Representative
0  1  4
1  2  4
2  4  4
3  4  4
```



```
4  4  4
```

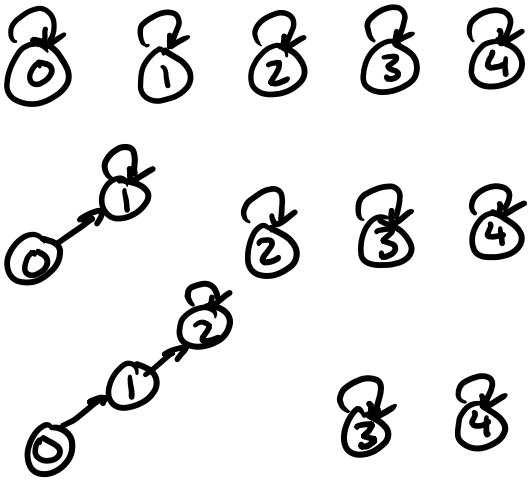
Output from `disjointSetForest.cpp` with `unionByRank = true, pathCompression = false`:

```
Element  Parent  Set Representative
0  0  0
1  1  1
2  2  2
3  3  3
4  4  4
Element  Parent  Set Representative
0  1  1
1  1  1
2  2  2
3  3  3
4  4  4
Element  Parent  Set Representative
0  1  1
1  1  1
2  1  1
3  3  3
4  4  4
Element  Parent  Set Representative
0  1  1
1  1  1
2  1  1
3  4  4
4  4  4
Element  Parent  Set Representative
0  1  4
1  4  4
2  1  4
3  4  4
4  4  4
```

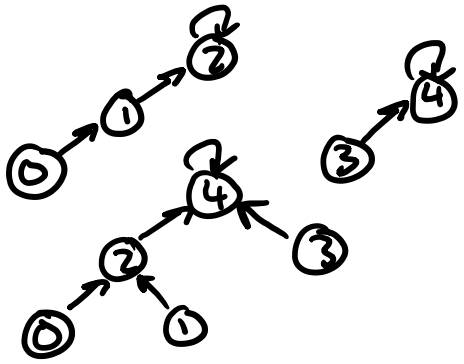


Output from `disjointSetForest.cpp` with `unionByRank = false, pathCompression = true`:

```
Element  Parent  Set Representative
0  0  0
1  1  1
2  2  2
3  3  3
4  4  4
Element  Parent  Set Representative
0  1  1
1  1  1
2  2  2
3  3  3
4  4  4
Element  Parent  Set Representative
0  1  2
1  2  2
2  2  2
3  3  3
4  4  4
```



```
Element  Parent  Set Representative
0  1  2
1  2  2
2  2  2
3  4  4
4  4  4
Element  Parent  Set Representative
0  2  4
1  2  4
2  4  4
3  4  4
4  4  4
```



Output from `disjointSetForest.cpp` with `unionByRank = true, pathCompression = true`:

```
Element  Parent  Set Representative
0  0  0
1  1  1
2  2  2
3  3  3
4  4  4
Element  Parent  Set Representative
0  1  1
1  1  1
2  2  2
3  3  3
4  4  4
Element  Parent  Set Representative
0  1  1
1  1  1
2  1  1
3  3  3
4  4  4
Element  Parent  Set Representative
0  1  1
1  1  1
2  1  1
3  4  4
4  4  4
Element  Parent  Set Representative
0  1  4
1  4  4
2  1  4
3  4  4
4  4  4
```