The University of Newcastle
School of Information and Physical Sciences

COMP2230/COMP6230 Algorithms
Tutorial Week 8 Solutions
6^{th} – 10^{th} September 2021

Tutorial

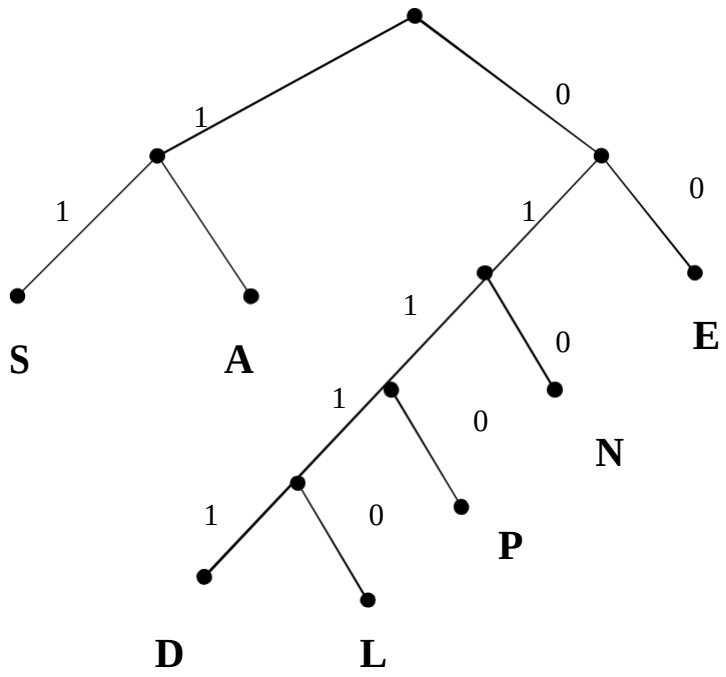
1. Construct Huffman code for the following data and calculate the compression ratio.

character	A	B	C	D	E
probability	0.2	0.3	0.15	0.25	0.1

Solution: A = 01, B = 10, D = 11, C = 001, E = 000.
The compression ratio is 0.75 (i.e. the new encoding saves 25% compared to a flat 3 bit code).

2. Decode each of the following bit strings using the following Huffman coding tree.

a. 01110100110
b. 01111001001110



Solution: The tree gave the code S = 11, A = 10, E = 00, N = 010, P = 0110, L = 01110, D = 01111. Then a. decodes to LAP, and b. to DEAL.

3. Consider the coin-changing algorithm and the following denomination: 1, 8, 12. Is the greedy algorithm presented in the lectures optimal for these denominations?

   Solution: No, take for example a total of 16, which is optimally made with two 8s, but the algorithm would give a 12 and four 1s.

4. Consider a possible divide-and-conquer approach to finding a minimum spanning tree in a connected, weighted graph G. Suppose that we divide the vertices of G into two disjoint subsets V_1 and V_2. We then find a minimum spanning tree T_1 for V_1 and a minimum spanning tree T_2 for V_2. Finally, we find a minimum weight edge e connecting T_1 and T_2. We then construct a minimum spanning tree T of G by taking e and all edges in T_1 and T_2.

   a. Is T always a spanning tree?

   b. If yes, is it always a minimum spanning tree?

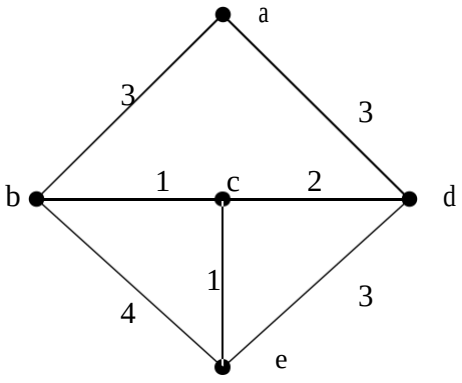
   Solution: Yes, it is always a spanning tree. The two subtrees only connect vertices in their subset, so there is no edges between the two sets, so when one it added to join them, there can be no cycles, thus as all vertices are used, it must be a spanning tree. However it is not necessarily a minimum spanning tree. Imagine a graph where the two subsets contain only edges of length 10, but the edges between the two subsets are length 1. Then it would give a "smaller" tree to use these edges, but the algorithm will only ever use one.

5. Let T be a minimum spanning tree for a graph G, let e be an edge in T and let T' be T with e removed. Show that e is a minimum weight edge between components of T'.

   Solution: If e is not a minimum weight edge between the components of T', then there is a smaller edge that we could use that would make T smaller. Thus T can't have been a minimum spanning tree, which is a contradiction, thus e must be a minimum weight edge between the components of T'.

6. Trace Kruskal's algorithm for the graph given below.

Solution:



Initially we have no edges so connected components are {a} {b} {c} {d} {e}
The smallest edge is (b,c) thus we have {a} {b,c} {d} {e}
The smallest edge is (c,e) thus we have {a} {b,c,e} {d}
The smallest edge is (c,d) thus we have {a} {b,c,e,d}
The smallest edge is (a,b) thus we have {a,b,c,e,d}

The edges are: {(b,c), (c,e), (c,d), (a,b)}

7. Trace Prim's algorithm for the same graph (Assume start vertex=a).

Solution: Prims algorithm begins with a start vertex. Add an edge of minimum weight that has one vertex in the current tree and other not in current tree.(No cycle)

1) Initially we have tree vertices {a(-,0)}, fringe vertices {b(a,3),d(a,3)} and unseen vertices {c(-,), e(-,)}.
2) Add b; tree vertices {a(-,0), b(a,3)}; fringe vertices { c(b,1), e(b,4),d(a,3)} and unseen vertices { }.
3) Add c: tree vertices {a(-,0), b(a,3), c(b,1)}; fringe vertices { d(c,2),e(c,1)} and unseen vertices { }.
4) Add e: tree vertices {a(-,0), b(a,3), c(b,1), e(c,1)}; fringe vertices { d(c,2)} and unseen vertices { }.
5) Add d: tree vertices {a(-,0), b(a,3), c(b,1), e(c,1), d(c,2)}; fringe vertices {} and unseen vertices { }.

The edges are: {(a,b), (b,c), (c,e), (c,d)}

Homework

8. Prove that the following Greedy Coin Changing Algorithm is optimal for denominations 1, 5, and 10. (Algorithm 7.1.1 and Theorem 7.1.2 from the text)

Greedy Coin Changing Algorithm: This algorithm makes change for an amount A using coins of denominations $denom[1] > denom[2] > \cdots > denom[n] = 1$.

Input Parameters: *denom,A*
Output Parameters: None

```
greedy_coin_change(denom,A) {
    i = 1
    while (A > 0) {
            c = A/denom[i]
            println("use " + c + " coins of denomination " +
                                        denom[i])
            A = A - c * denom[i]
            i = i + 1
    }
}
```

Solution:
Proof: We use induction on A to prove that to make change for an amount A, the output of algorithm 7.1.1 and the optimal solution are identical. The cases A= 1,2,3,4,5,10 are readily verified.

The inductive assumption is that to make change for an amount k, where k<A, the output of algorithm 7.1.1 and the optimal solution are identical. Suppose first that 5<A<10.

Let *Opt* be an optimal solution. Now *Opt* must use a coin of denomination 5. Now *Opt* with one coin of denomination 5 removed is optimal for A-5. By the inductive assumption, the output of algorithm 7.1.1 for A-5 and opt with one coin of denomination 5 removed are identical. Adding a coin of denomination 5 to the output of algorithm 7.1.1 And the optimal solution are identical for 5<A<10.

The argument for A>10 is similar. Let opt be an optimal solution. Now *Opt* must use a coin of denomination 10. Now opt with one coin of denomination 10 removed is optimal for A -10. By the inductive assumption, the output of algorithm 7.1.1 for A-10 and *Opt* with one coin of denomination 10 removed are identical. Adding a coin of denomination 10 to the output of algorithm 7.1.1 And the optimal solution are identical for A>10. The inductive step is complete

9. (Huffman Codes) Given a text that comprises characters from some n-character alphabet we need to construct a variable-length encoding so as to minimize the total length of the encoded text. This can be done by assigning short codes to frequent characters and longer codes to less frequent characters. Design a greedy algorithm to construct such encoding. Illustrate the algorithm on the following example:

character	a	b	c	d	E
probability	0.35	0.2	0.15	0.2	0.1

What is the compression ratio in this example?

Solution: Algorithm idea: The algorithm treats each character as a vertex, weighted by the probability. It then repeatedly picks the two smallest un -joined vertices, joins them to a new parent, whose weight is the sum of its children, until there are no un-joined vertices. The code is then determined by laying the tree out and assigning 0 to each left branch and 1 to the right branch and reading the code off the tree.

For this example the algorithm gives the following code:

A=01,B=11,C=001,D=10,E=000
The compression ratio is 0.75 (i.e. the new encoding saves 25% compared to a flat 3 bit code).

10. What is the time complexity for constructing Huffman code

a. if the frequencies of characters are sorted?
b. if they are not sorted?

Solution: If the characters are unsorted, and kept in a simple array or the like, the algorithm is quadratic; if they are sorted and kept in the same way, then when the new vertices are added back in, we have to re-sort, ultimately ending up with quadratic again. If however as the first step we add everything to a heap, then we reduce the time to T (n log n).

11. Let G be a connected weighted graph and let v be a vertex in G and let e be an edge of minimum weight incident on v. Show that e is contained in some minimum spanning tree.

Solution: Assume that e is the only edge of that weight incident on v. Assume that there is a minimum spanning tree that does not contain e. Then determine the component of the tree that the other vertex incident one is in. Then there is a path from v to this vertex, which we can disconnect by removing one of tree edges incident on v, and reconnect by adding e to the tree. This will not create a cycle, and will result in a spanning tree smaller than the posited one, which thus can't have been a minimum spanning tree, thus e must be contained in any minimum spanning tree in this case.
If e is not the only edge of that weight that connects v to the rest of the graph, then we may pick any the edge that joins v to the rest of the tree, and replace it with e with no loss. Thus e is in some minimum spanning tree in this case. For this second case you should be able to satisfy yourself that the tree includes one of these minimum edges.

12. Prove that the Greedy Coin Changing Algorithm from question 8 is optimal for denominations 1, 13, and 25.

Proof: We use induction on A to prove that to make change for an amount A, the output of algorithm 7.1.1 and the optimal solution are identical. The cases A= 1…..,13, 25 are readily verified.

The inductive assumption is that to make change for an amount k, where k<A, the output of algorithm 7.1.1 and the optimal solution are identical. Suppose first that 13<A<25. Let opt be an optimal solution.

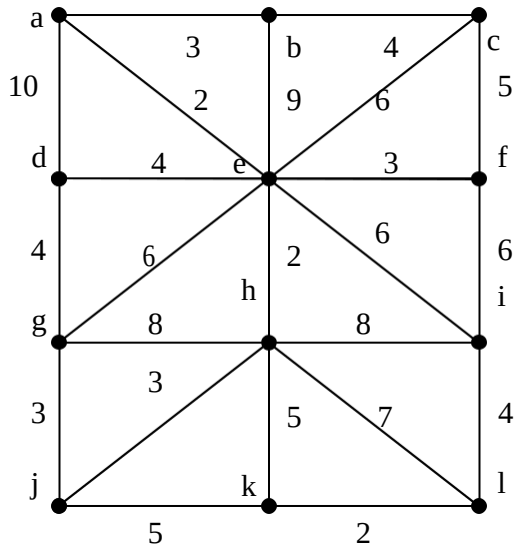
Now *Opt* must use a coin of denomination 13. Indeed, if it does not, then we need to use at least 13 coins of denomination 1, and 13 of them can be traded in for one coin of denomination 13, therefore it is not optimal.

Now *Opt* with one coin of denomination 13 removed is optimal for A-13. Indeed, if that is not the case, there is another solution for A-13 that uses fewer coins. Adding a coin of denomination 13 to the solution for A-13 problem produces a solution for A using fewer coins than *Opt*, which is not possible.

By the inductive assumption, the output of algorithm 7.1.1 for A-13 and opt with one coin of denomination 13 removed are identical. Adding a coin of denomination 13 to the output of algorithm 7.1.1 And the optimal solution are identical for 13<A<25.

The argument for A>25 is similar. Let *Opt* be an optimal solution. Now *Opt* must use a coin of denomination 25. Now opt with one coin of denomination 25 removed is optimal for A-25. By the inductive assumption, the output of algorithm 7.1.1 for A-25 and opt with one coin of denomination 25 removed are identical. Adding a coin of denomination 25 to the output of algorithm 7.1.1 And the optimal solution are identical for A>25 The inductive step is complete.

13. Trace Kruskal's algorithm for the following graph.
Short olution: Edges:{(a,e), (e,h), (k,l), (a,b), (h,j), (j,g), (e,f), (b,c), (d,e), (i,l), (h,k)}



14. Trace Prim's algorithm for the same graph (Assume **a** starting vertex)
Short solution: Final result – edges: {(a,e), (e,h), (a,b),(e,f),(h,j) , (j,g), (b,c), (g,d),(h,k), (k,l), (l,i)}

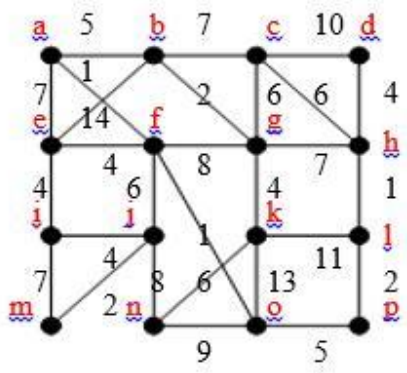
More exercises

15. Euro coin denominations are 1,2,5,10,20,50,100 and 200. Tell whether Greedy Coin Changing Algorithm from question 5 is optimal for this system. If it is optimal, give an argument to support your answer. If algorithm is not optimal give a counterexample.

Solution: Optimal.

16. Trace Kruskal algorithm for the following figure-3.

Solution: Follow the solution of question 13



17. Trace Prim's algorithm for the same graph.

Solution: Follow the solution of question 14