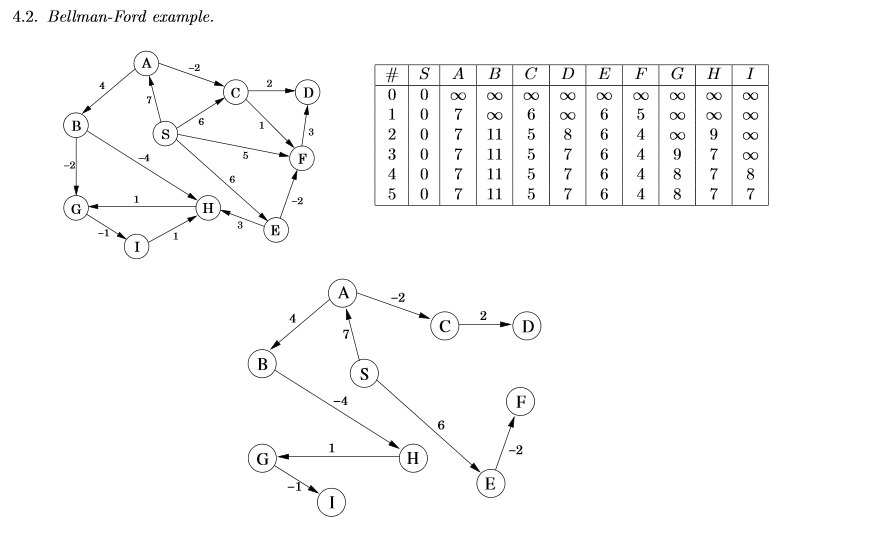
Allison Neyer

Algorithms Homework #4

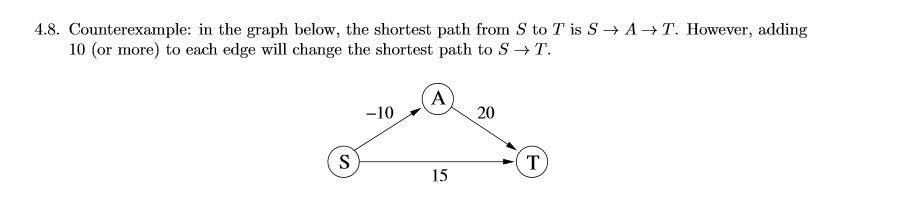
4.2a in the textbook)

<http://cseweb.ucsd.edu/~dasgupta/101/sol7.pdf>



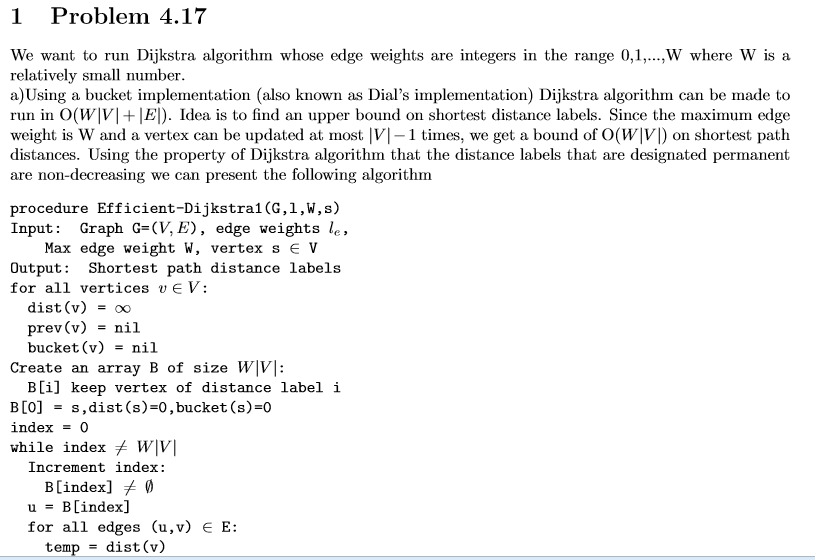
4,8 in the textbook)

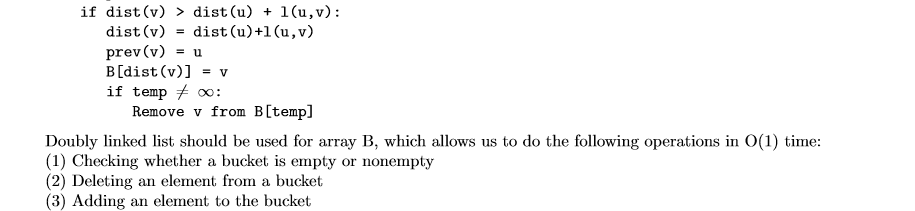
http://cseweb.ucsd.edu/~dasgupta/101/sol7.pdf



4.17a in the textbook)

<http://www.ece.northwestern.edu/~dda902/336/hw5-sol.pdf>

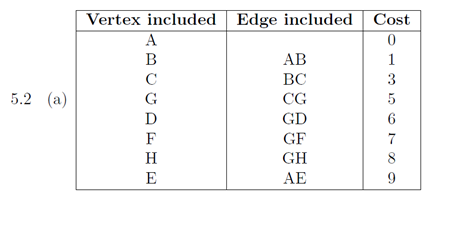




5.2a in the textbook)

<http://www.eecs.ucf.edu/~wocjan/Teaching/COT5405-Spring2011/> (homework 3 solutions link):

tinyurl.com/qa75gvn has the following solution for problem 5.2a)



In 5.2 (a), the last number should be 12 and not 9.

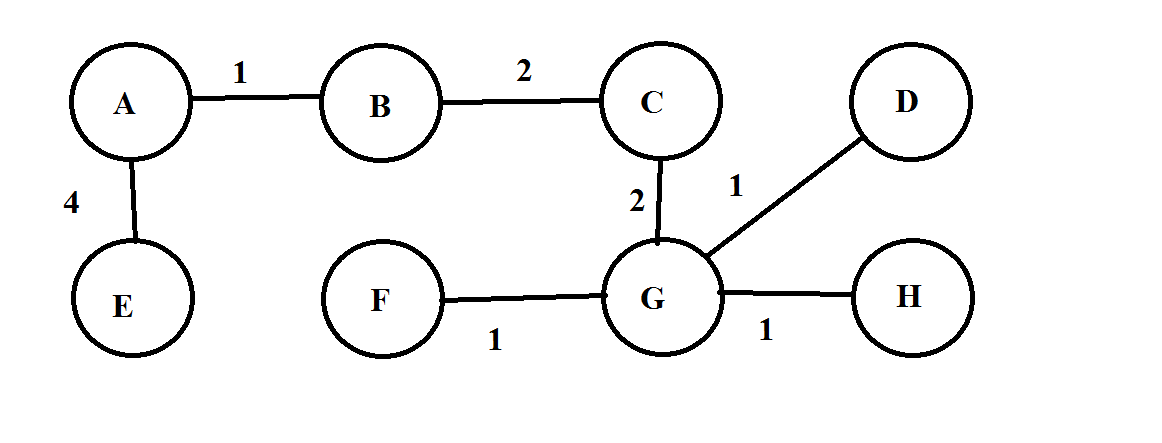
5.2b in the textbook)

Steps: list edges by increasing order of weight. Then add them unless they make a cycle.

total edges: A-B(1); G-D(1); G-H(1); F-G(1); B-C(2);C-G(2);C-D(3);D-H(4);A-E(4);E-F(5);B-G(6);B-F(6);A-F(8)

Included edges: A-B(1); G-D(1); G-H(1); F-G(1); B-C(2);C-G(2); A-E(4);

Edges that were NOT included because they made a cycle: C-D(3) [cycle]; D-H(4) [cycle]; E-F(5) [cycle];B-G(6) [cycle];B-F(6) [cycle];A-F(8) [cycle].



min spanning tree length: 12 = 4+1+2+2+1+1+1

how do you compress and know where to make the directed arrows to?

1. start from included edges.
2. pick ones that don’t make a cycle
3. point the root of the shortest subtree (defined by height of the tree)to the root of the tallest subtree.
4. Ties are resolved arbitrarily.

step 1) A-B(1): Have A point to B (height tie so where it points is arbitrary). **Disjoint-set data structure:** **(A)🡪(B)** where ( ) represents a node and 🡪 represents a directed edge.

step 2) G-D(1): Have G point to D (height tie so where it points is arbitrary). **Disjoint-set data structure:** (G)🡪(D) where ( ) represents a node and 🡪 represents a directed edge.

step 3) G-H(1): Have H point to G’s root (D) because G’s height (2) is bigger than H’s height (1). **Disjoint-set data structure:** (H)🡪(D)🡨(G) where ( ) represents a node and 🡪 represents a directed edge.

|  |  |
| --- | --- |
|  |  |
| height of 2 (D-G) | height of 1 (H) |

step 4) F-G(1): Have F point to G’s root (D) because G’s height (2) is bigger than F’s height (1).

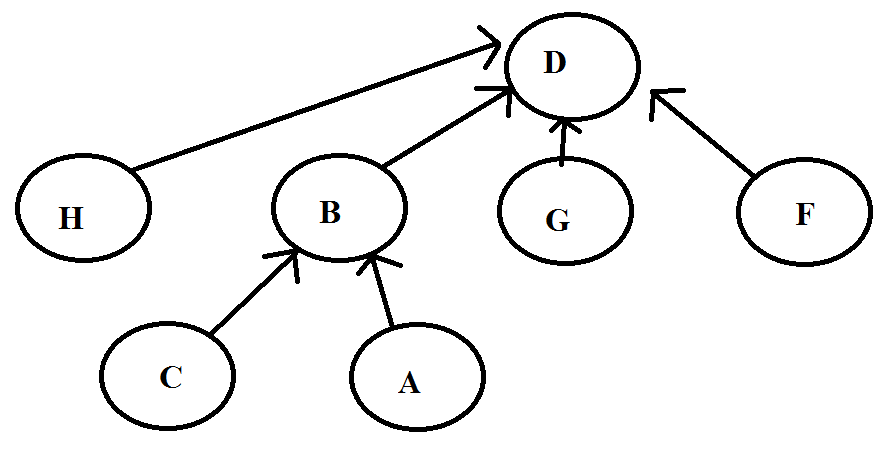
**Disjoint-set data structure:**

**Note: The lower levels of F, G, and H have arrows pointing up towards D.**

step 5) B-C(2): Have C point to B’s root (B) because B’s height (2) is bigger than C’s height (1). **Disjoint-set data structure:** **(C)🡪(B)🡨(A)** where ( ) represents a node and 🡪 represents a directed edge.

|  |  |
| --- | --- |
|  |  |
| height of 2 (B-A) | height of 1 (C) |

step 6) C-G(2): Have C’s root (B) point to G’s root (D) because both subtrees have the same height and so there is a tie between these two subtree’s heights and thus where it points is arbitrary.

**Disjoint-set data structure:** .

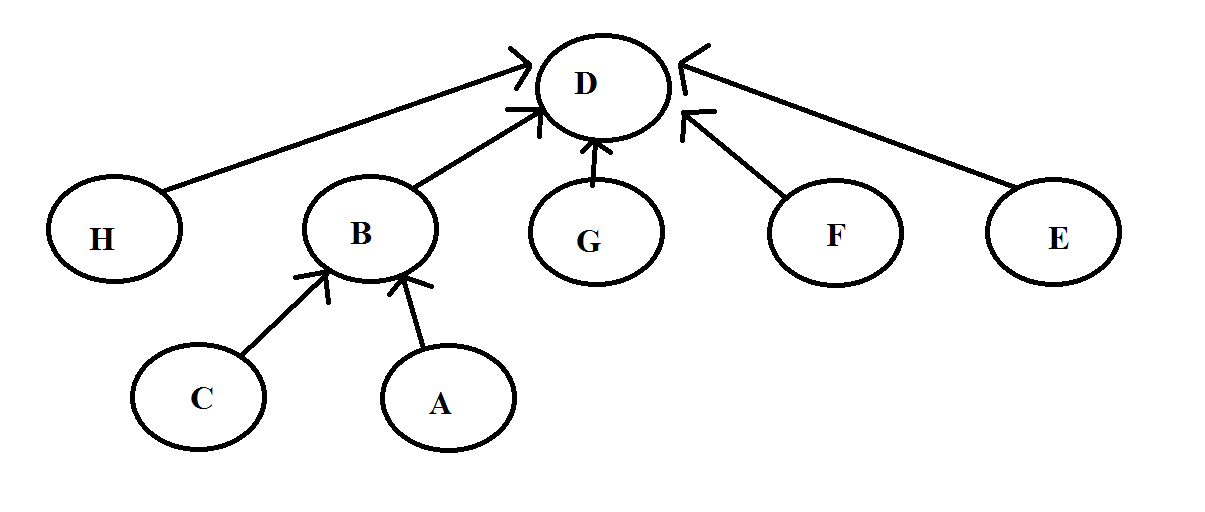
|  |  |
| --- | --- |
|  |  |
| height of C subtree is 2 | height of G subtree is 2 |

step 7) A-E(4): Have E’s root (node E) point to A’s root (node D) because A’s height (3) is bigger than E’s height (1).

|  |  |
| --- | --- |
|  |  |
| height of A subtree is 3 | height of E subtree is 1 |

Final minimum directed Spanning Tree:

**Final directed tree:**



5) dynamic programming to determine whether a numeric list has a subset of elements that sums to a given value:

**answer: see originalSubsetSum.c for dynamic programming solution answer.**

subsetSumTester-AllisonVersion.c is tester I used because assert was crashing the a.exe file for unknown reasons.

The originalSubsetSum.c program, given a set of non-negative integers, and a value sum, will determine if there is a subset of the given set with sum equal to given sum.

It will return true if there is a subset of set[] with sum equal to given sum

**Assumptions:**

Assumption1: function can't use one element repeatedly to make the sum. So, for example, the function can't repeatedly add one element of 3 to get 6.

Assumption 2: if sum is 0 it is always true because you just pick nothing (**The empty set is always a subset of every set**).

I assume that the function can't use one element repeatedly to make the sum. So, for example, the function can't repeatedly add 3 to get 6.

Since I assume that one cannot repeatedly use the same value in the subset, the function will return true given a set of {3,3} and a sum of 6. Given a set of {3} and a sum of 6, the function should return false because I’m assuming you can’t just say 3+3 =6.

Please note that I had to revise the code from the given link. I added missing libraries to properly run it. Also, the code uses c99, so use the following: “gcc -std=c99 subsetSumTester-AllisonVersion.c” to compile.

<http://www.geeksforgeeks.org/dynamic-programming-subset-sum-problem/>

6) backtracking solver for the Kirkman Schoolgirl Problem:

<http://www.delphiforfun.org/programs/Kirkman_Tabu.htm> (Not only does it have the solution but it’s a good resource for explaining the problem too)

See the “executable files” subfolder in the “KirkmanLittleSchoolGirlProblem” Folder and the “source files” subfolder for the program in Pascal.