CS130B- Data Structures and Algorithms II

Discussion Section Week 5

Midterm 1

- May 11th, 2017
- Sample exam: www.cs.ucsb.edu/~cs130b/mid.sample.pdf
- Links to more sample exams on the last slide

Problem 1(a)

Problem 1 (60%) An array A[1..n], with $n \geq 1$, is called *strictly unimodal* if there is some integer j, with $1 \leq j \leq n$, such that $A[1] < A[2] < \cdots < A[j] > A[j+1] > \cdots > A[n]$.

a. (15%) Design an algorithm for finding the index of the largest entry of a strictly unimodal array. Your algorithm must be more efficient than a brute-force one which linearly scans the array for the largest element. Give the pseudo-code and complexity of your algorithm.

Problem 1 – Example

$$A = \{1, 2, 4, 6, 3, 2\}$$

Problem 1 – Example

$$A = \{1, 2, 4, 6, 3, 2\}$$

Brute Force Solution O(n)

```
function maxIdxUnimodal(array A):
    maxVal = 0
    maxIdx = 0
    for i=0 to A.length:
        if A[i] > maxVal:
            maxVal = A[i]
            maxIdx = I
    return maxIdx
```

O(log(n)) Solution

```
function maxIdxUnimodal(array A, int low, int high):
    if low = high – 1:
        return index of max between (A[low], A[high])

    mid = ⌊(high + low)/2⌋

    if A[mid] < A[mid + 1]:
        return maxIdxUnimodal(A, mid + 1, high)
    else:
        return maxIdxUnimodal(A, low, mid)
```

Problem 1(b)

An array A[1..n], with $n \geq 1$, is called *unimodal* (not necessarily strictly) if there is some integer j, with $1 \leq j \leq n$, such that $A[1] \leq A[2] \leq \cdots \leq A[j] \geq A[j+1] \geq \cdots \geq A[n]$.

b. (15%) Give an example to show that your algorithm for the strictly unimodal case will not work for the unimodal case.

Problem 1(b)

For the array A={1, 2, 2, 2, 3, 2} our solution to 1.a will fail to find A[j]

Problem 1(c)

c. (30%) Design an algorithm for finding the index of the largest entry of an unimodal array. Give the pseudo-code and complexity of your algorithm. Please be clear but concise in your description.

If necessary, you may assume that the array A[1..n] is embedded in a larger array A[0..n+1], where $A[0] < A[1]$ and $A[n] > A[n+1]$ for strictly unimodal, and $A[0] \leq A[1]$ and $A[n] \geq A[n+1]$ for unimodal.

Problem 1(c)

What’s the difference with problem 1(a)? What case do we need to consider?

A[mid] == A[mid + 1]

Problem 1(c) Pseudocode

```
function maxIdxUnimodal(array A, int low, int high):
    if low = high – 1:
        return index of max between (A[low], A[high])

    mid = ⌊(high + low)/2⌋

    if A[mid] < A[mid + 1]:
        return maxIdxUnimodal(A, mid + 1, high)
    else if A[mid] > mid[mid + 1]:
        return maxIdxUnimodal(A, low, mid)
    else:
        return max( maxIdxUnimodal(A, mid + 1, high), …
                            maxIdxUnimodal(A, low, mid))
```

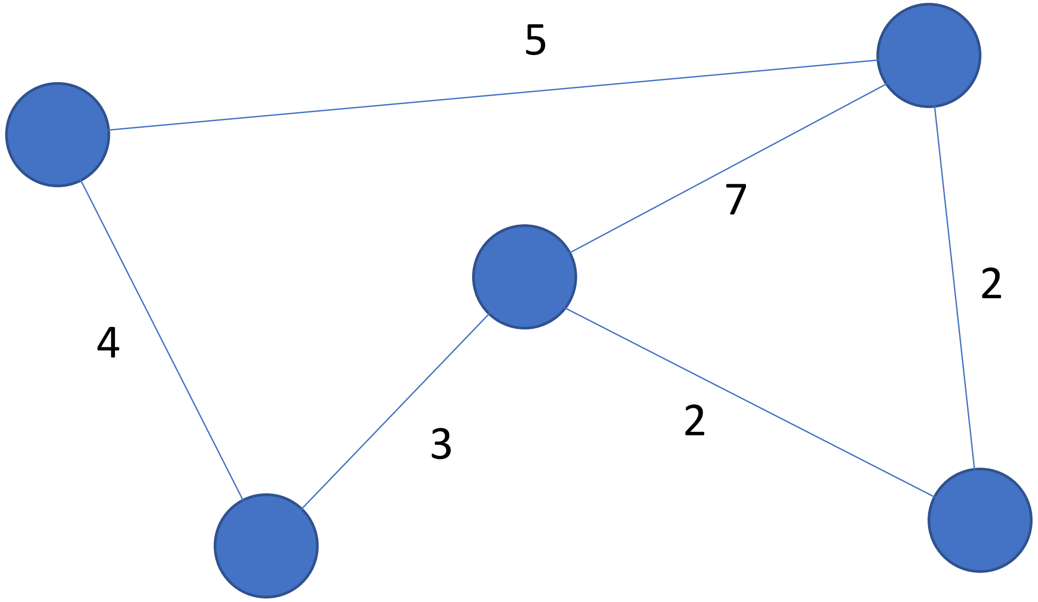
Problem 2

Problem 2 (40%) Recall that we discussed two greedy algorithms for finding the minimum cost spanning tree of a connected undirected graph of n vertices. One of the algorithms, Prim's algorithm, starts with a null tree, and adds one edge at a time until $n-1$ edges are added. At each step, the remaining edge of the smallest cost is added if the inclusion maintains the tree property (no cycle) of the partial solution.

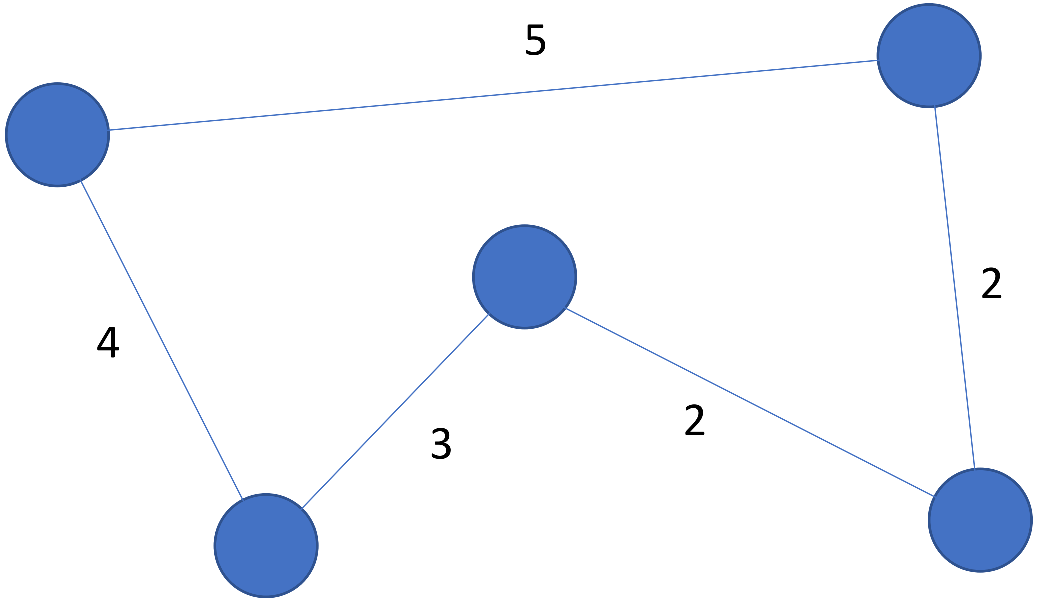
Now consider a variation of Prim's algorithm. Assume that all vertices in the input graph are connected (i.e., there exists at least one path between every pair of vertices). Instead of starting with a null tree, we start with the original graph. We examine edges in the order of *nonincreasing cost.* An edge is deleted if the deletion leaves the graph connected, otherwise, it is kept. We delete enough edges so that eventually only $n-1$ left. Hence, this greedy strategy says "removing edges of higher costs" instead of "retaining edges of lower costs" that is used in Prim's algorithm.

Prove or disprove the following statement: the above greedy strategy generates a minimum cost spanning tree of a connected undirected graph. Please be clear but concise in your proof. Writing a page of proof most likely means that your answer is wrong.

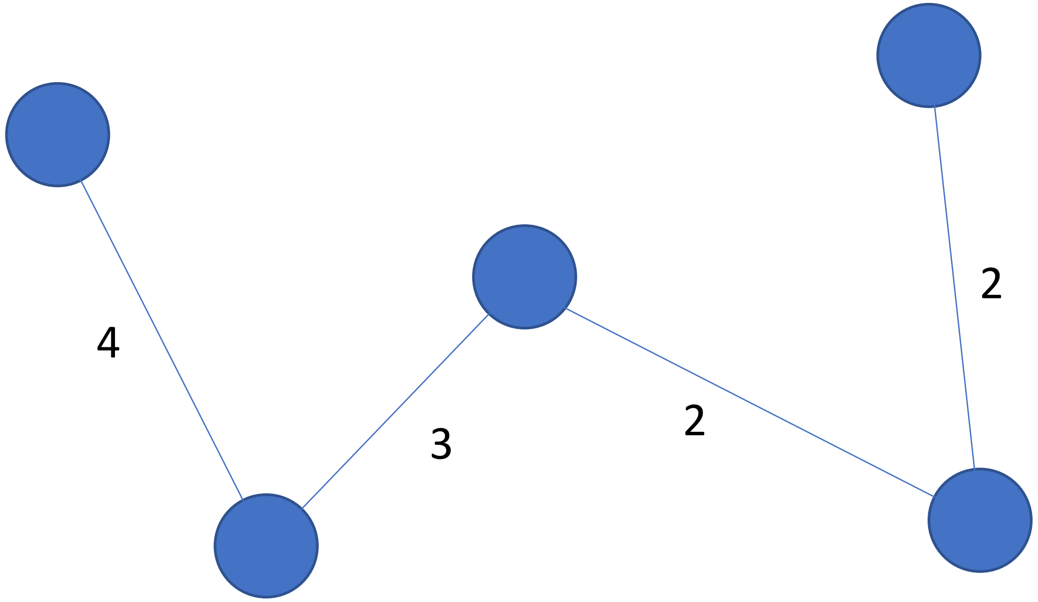
Example



Example



Example



Observation

We are always removing an edge in some cycle C; otherwise, the graph would be disconnected. If we show that the maximum weight edge in a cycle C will never appear in any minimum spanning tree of the graph G.

Proof

Lemma
Let C be any cycle in G, and let the edge $e = (u, v)$ be the most expensive edge in C. Then e does not belong to any minimum spanning tree of G.
Proof
Let T be a spanning tree that contains e. Deleting e partitions the vertices into two groups: a part S containing u, and $V - S$ containing v. The edges of C other than e form a path P with one end at u and the other at v, so there must be an edge e' on P that crosses from S to $V - S$. Adding the edge e' gives a graph (V, T') that is connected and has no cycles, so T' is a spanning tree of G, and is less expensive than T.

Additional Practice

https://hkn.eecs.berkeley.edu/exams/course/cs/170

- Fall 2014, David Wagner Midterm 1 Problem 10 and 12 (Kruskal's algorithm, binary search)
- Fall 2008, Satish Rao Midterm 1 Problem 5 (difficult divide and conquer problem)
- Fall 2007, Christos Papadimitriou Midterm 1 Problem 4 (divide and conquer on array) [look at the problem in the solution]
- Fall 2005, Michael Jordan Midterm 1 Problem 4 (divide and conquer)