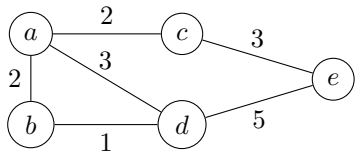
CS 161 Exam Review

March 14, 2021

Let G be a connected weighted undirected graph. Suppose that the maximum weight in G is m, and there is only one edge, e^*, that has that weight. For example, G might look like this, where $m = 5$ and $e^* = \{d, e\}$:

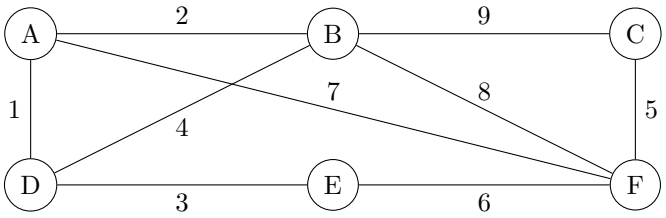


Prove the following statement:

If there is some spanning tree that does *not* contain the edge e^*, then *no* Minimum Spanning Tree can contain e^*.

[**We are expecting:** *A formal proof.*]

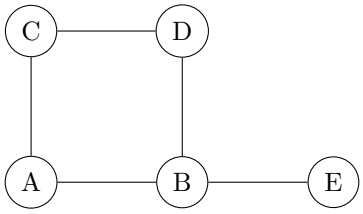
Consider the graph G below.



1. In what order does Prim's algorithm add edges to an MST when started from vertex C?
2. In what order does Kruskal's algorithm add edges to an MST?

[**We are expecting:** *For both, just a list of edges. You do not need to draw the MST, and no justification is required.*]

Consider the following graph:



1. What is the global mimimum cut of this graph?

   [**We are expecting:** *Just the global minimum cut. No explanation is required.*]

2. What is the probability that Karger's algorithm chooses an edge crossing the minimum cut with its first choice?

   [**We are expecting:** *Just the probability. No explanation is required.*]

3. What is the *exact* probability that one run of Karger's algorithm returns a minimum cut on this graph? How does it compare to the bound of $1/\binom{n}{2}$ that we saw in class?

   [**We are expecting:** *Your numerical answer (no justification required), as well as a statement about how this compares to* $1/\binom{n}{2}$.]

You join a nomadic tribe traveling through their fixed route of n ancient queendoms, going through queendoms in increasing order. This is a cool adventure, but also a nice opportunity to make money! In each queendom, you can sell any foreign currency and receive the local currency in exchange (but no other exchanges are allowed). You have an $n \times n$ table of the exchange rates between every pair of currencies. You start the journey with 1 unit of currency of the first queendom. Your goal is to buy and sell during the trip in order to maximize the money you'll have in the n-th queendom's currency, since you plan to stay there for a while.

There is one more restriction: in order to not attract too much attention to your side business, you should only exchange money in k queendoms.

Design an algorithm to determine the maximum of the n-th queendom's currency you can have.

Input: n: the number of queendoms; k: the number of exchanges allowed; A: an $n \times n$ table of exchange rates.
Output: the number of units of the n-th queendom's currency

Additional assumptions: You may assume that k is much smaller than n, but not as small as $O(1)$. Also, you do not exchange in the first queendom (since you already have that currency), and you always exchange in the last queendom.

Example 1 *In this example, the optimal strategy is to exchange in both the second and third queendom, and end up with* $2 \times 2 = 4$ *units of the last currency (exhanging only in the last one would give* 2.5*).*

Note: In the example $A[i][j] = c$ *means that exchanging 1 unit of currency* i *yields* c *units of currency* j*.*

```
Input:
n = 3
k = 2
A =  [[1.0  2.0  2.5]
      [0.5  1.0  2.0]
      [0.4  0.5  1.0]]
Output:
4
```

Example 2 *In this example, the optimal strategy is to exchange only in the third queendom, and end up with 2.5 units of the last currency (exhanging in both second and third one would give* $0.5 \times 2 = 1$*).*

```
Input:
n = 3
k = 2
A =  [[1.0  0.5  2.5]
      [2.0  1.0  2.0]
      [0.4  0.5  1.0]]
Output:
2.5
```

Give a short but clear English description of your algorithm.

[We are expecting: A clear yet thorough English description of your algorithm.]

Is your algorithm correct (Yes/No)?

[We are expecting: A clear answer of YES or NO. If YES, no explanation is necessary, if NO, explain why your algorithm is incorrect.]

Provide the pseudocode for your algorithm.

[We are expecting: Detailed pseudocode that matches your English description. You are free to use an interface of any of the algorithms covered in lecture.]

Provide an analysis of the runtime of your algorithm.

[We are expecting: A detailed analysis of the runtime of your algorithm including the Big-O time in terms of n and k.]

There are n mice and n holes along a line. Each hole can accommodate only 1 mouse. A mouse can stay at its position, move one step right from x to $x + 1$, or move one step left from x to $x - 1$. Any of these moves consumes 1 minute. Mice can move simultaneously. Assign mice to holes such that the time it takes for the last mouse to get to a hole is minimized, and return the amount of time it takes for that last mouse to get to its hole.

Example: Mice positions: 4, -4, 2
Hole positions: 4, 0, 5
Best case: The last mouse gets to its hole in 4 minutes ($\{4 \to 4, -4 \to 0, 2 \to 5\}$ and $\{4 \to 5, -4 \to 0, 2 \to 4\}$ are both possible solutions)