Homework Assignment #3

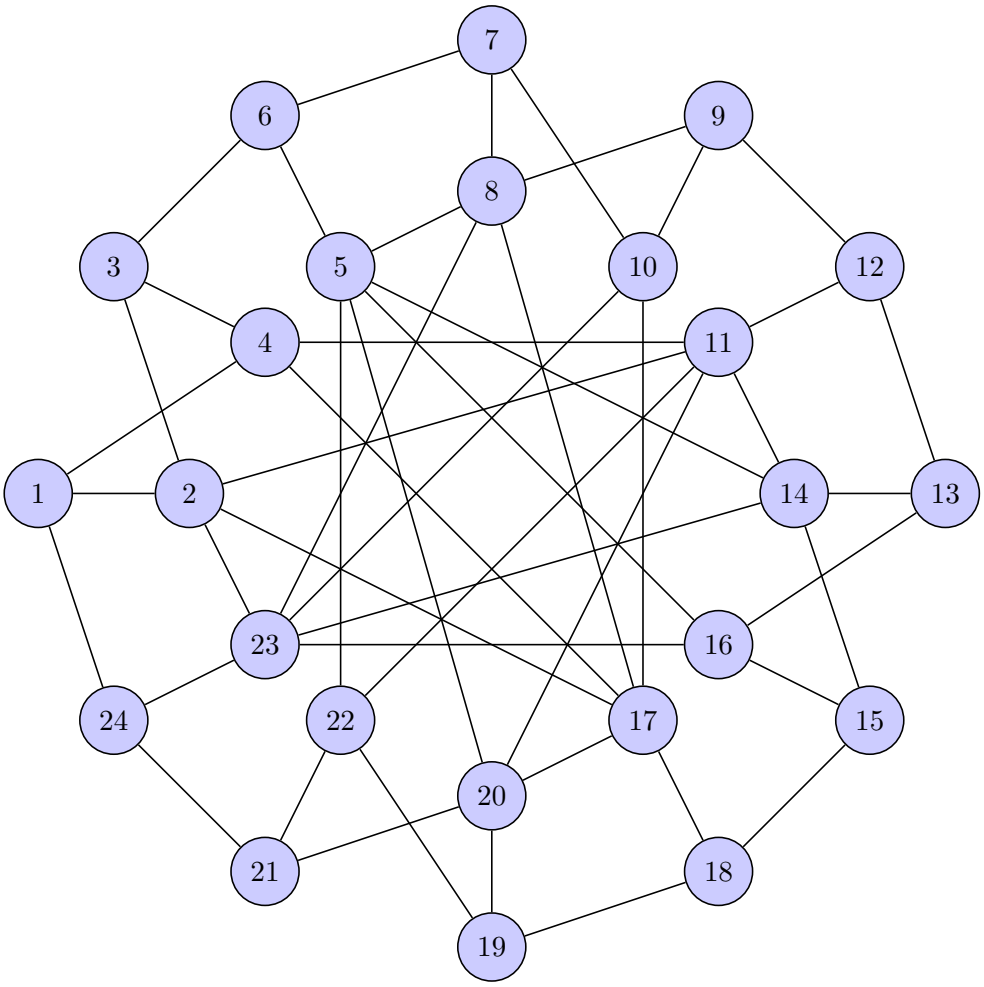
Due: Oct 18th, 2017

CMPUT 304
Department of Computing Science
University of Alberta

Note: All logarithms are in base 2 unless specified otherwise.

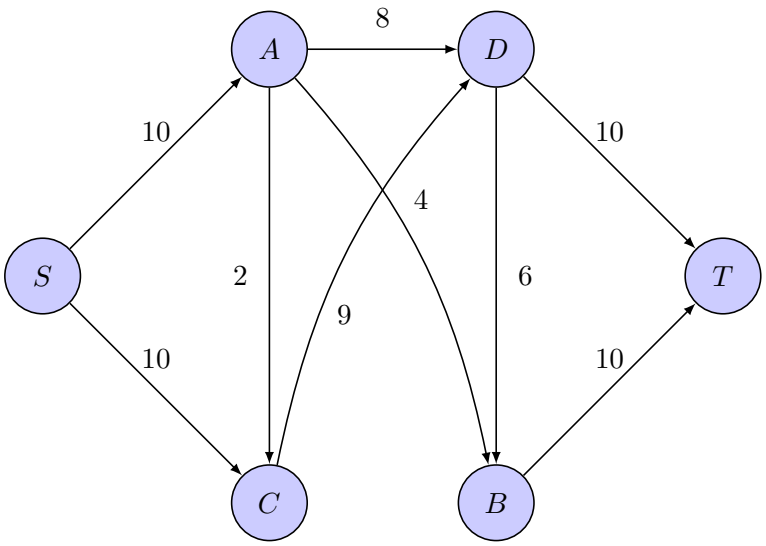
Exercises are for you and you alone. You need not submit them and they will not be graded. You may submit as many answers to as many problems as you like.

Exercise I. Does the following graph have a perfect matching?



Exercise II. Argue that in a graph where all edges have capacity 1, the Ford-Fulkerson method (without *any* restrictions on the type of augmenting path we find) concludes in poly-time.

Problem 1. (1 pts) Run Edmonds-Karp on the following network:



Problem 2. (1 pts) Given a flow f in a flow network (V, E, s, t, c), prove that for any s-t cut $(S, \bar{S})$ it holds that

$$|f| = f(S, \bar{S}) = \sum_{u \in S} \sum_{v \notin S} f(u, v) - \sum_{u \in S} \sum_{v \notin S} f(v, u)$$

Remember that in class we have proven this for one particular cut: $(S \setminus \{t\},\ \{t\})$ (the flow that comes out of the source is the flow that goes into sink).

Problem 3. (3 pts) Fix a graph G and two of its nodes u and v. Given two paths from u to v, denoted P_1 and P_2, we say the two paths are *edge-disjoint* if they share no edge, and we say the two paths are *vertex-disjoint* if the only vertices they share are u and v.

(i) (1 pts) Give a poly-time algorithm that takes as input a graph G and two nodes u and v in this graph, and returns a maximum number of edge-disjoint paths nodes u and v.

(ii) (2 pts) Give a poly-time algorithm that takes as input a graph G and two nodes u and v in this graph, and returns a maximum number of vertex-disjoint paths between u and v.

Problem 4. (2 pts) A bipartite graph $G = (L \cup R, E)$ is called d-regular if for any $v \in L \cup R$ we have that $\deg(v) = d$.

Show that in any d-regular bipartite graph $|L| = |R|$, and then use Hall's marriage theorem to infer that there must exist a perfect matching between L and R. Moreover, prove that any d-regular bipartite graph can be decomposed into d disjoint perfect matchings.

Problem 5. (2 pts) Image Foreground/Background Separation.

In this question, your input is composed of an image that has n pixels. Imagine that the picture is of a car driving on a road. Your goal is to separate the image's pixels into foreground (all the pixels that belong to the car) and background (all the pixels that belong to the road, the scenery, the sky etc). In other words, your goal is to find a partition of the n pixels into F (the foreground) and B (the background).

Luckily, most of the work has been done already for you: along with the n pixels, you get a "foreground likelihood" estimation f_i per each pixel i: This f_i is a number between 0 and 1 that intuitively represents how likely it is that pixel i belongs to the foreground, and thus $1 - f_i$ represents

how likely it is that pixel i belongs to the background. We assume these $f_1, f_2, f_3..., f_n$ are given to you. They are part of your input and your goal is to partition the n pixels into F and B such that you maximize $\sum_{i\in F} f_i + \sum_{i\in B}(1-f_i)$. Ideally, all of these $f_1, f_2, ..., f_n$ will be exactly 1 for any pixel belonging to the foreground and exactly 0 for every pixel belonging to the background. The trouble is however, that more often then not, they are numbers strictly between 0 and 1.

Clearly, if we only cared to work pixel-by-pixel, then in order to maximize our objective we decide on the foreground pixels as all pixels i such that $f_i \geq (1-f_i)$, namely $f_i \geq \frac{1}{2}$. However, an image also has a spacial component, and it stands to reason that nearby pixels will belong together to either the foreground or the background.

And so, in addition to the f_i-s, you are also provided a penalty $p_{i,j} \geq 0$ for every pixel i and any pixel j which is one of the ≤ 8 neighboring pixels next to i. If you decide on placing pixel i in the foreground while placing pixel j in the background (or vice versa) — you have to pay the $p_{i,j}$ penalty. Denoting this collection of penalties as E, you goal is thus to find the partition into F and B that maximizes:

$$\sum_{i\in F} f_i + \sum_{i\in B}(1-f_i) - \sum_{\{(i,j)\in E:\ \ (i\in F \wedge j\notin F) \text{ or } (i\notin F \wedge j\in F)\}} p_{i,j}$$

Give a $O(n^3)$-time algorithm that finds this partition.

Hint: Write the maximization objective as a complementary minimization problem. Once you have this minimization objective, the underlying graph problem is fairly immediate (especially considering this is an assignment about Max-Flow and therefore on **Min-Cut**).