Boston University CS 630 Fall 2024
Review Problems

These are some practice problems for the upcoming midterm exam. Note that the exam is only 75 minutes (conducted during lecture time), the number of problems here far exceed the length of the actual exam. To give you an idea we have marked some problems with (*). The set of marked problems together would constitute an exam of appropriate length. The marking does not mean that we won't ask practical or theoreticl questions on one particular topic. We intend to have a good mix of the two.

How to study? Here are some recommendations on how to go about preparing for the exam.

- Know all definitions.
- Be able to run/trace algorithms on specific instances.
- As you may have noticed, most of the problems in labs and the homework were closely related to some problem from lecture. Make yourself so familiar with the lecture problems that you can comfortable make the connection between those and the problem in the assignment. (e.g. the homework problems were related to Independent Set and Set Cover respectively) This will also help you reference or apply approximation factors.
- You might find reviewing TopHat questions useful.
- Review lab and homework assignments.
- Go over the reading material (see Piazza post)
- Do a thorough job preparing the "perfect" cheat sheet. Thinking about what should exactly go on it will help you organize your knowledge.

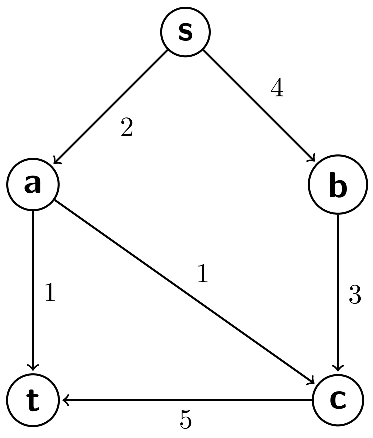
1 Network Flow

1. (*) Consider this weighted directed graph G with source s, sink t and edge capacities written on the edges.
   (a) List *all* the min-cuts in G.

   mincut1: $\{s, b\}$ mincut2: $\{s, a, b\}$

   (b) Suppose that the capacity on edge (c, t) is changed to 2 (from 5). First, list the min-cuts in this modified graph. Second, answer whether the maximum flow in this graph is unique. Explain why or why not.

   There is only one mincut: $\{s, a, b, c\}$, however there are two max flows depending whether edge (a, c) is assigned flow:
   flow1: $f(s, a) = 2, f(a, t) = 1, f(a, c) = 1, f(a, b) = 1, f(b, c) = 1, f(c, t) = 2.$
   flow2: $f(s, a) = 1, f(a, t) = 1, f(s, b) = 2, f(b, c) = 2, f(c, t) = 2$



2. (Jeff Erickson Algorithms, page.363, exercise 5)

   (a) (*) Describe a flow network that contains a *unique maximum (s, t)-flow* but does *not* contain a *unique minimum (s, t)-cut.*

   For example a path with capacity 1 on each edge

   (b) (*) Describe a flow network that contains a *unique minimum (s, t)-cut* but does *not* contain a *unique maximum (s, t)-flow.*

   Here are the edge capacities: $c(s, a) = 2, c(s, b) = 2, c(a, c) = 2, c(b, c) = 2, c(c, t) = 1$

   (c) Describe an efficient algorithm to determine whether a given flow network contains a *unique minimum (s, t)-cut.*

   We know that after conclusion of the Ford-Fulkerson algorithm nodes reachable from s in the residual graph are on the s-side of a min-cut. We can use the same argument to show that nodes that can still reach t are on the t-side of some min-cut. (To prove this you can for example reverse the edges in the input and run FF from t as the source.)

   If these two cuts are not identical, then clearly there are multiple min-cuts in G.

   If they are identical, then there can't be any other min-cut. To see this, suppose that there is a third min-cut C. Since C is different from the previous cut, there must be at least one vertex that was on the s-side in the original cut and the t-side in C. (or there is such a v from the t-side. The argument is the same.) We know that edges in any min-cut are saturated with the max flow. Since v is originally on the s-side it means that there is an augmenting path from v to s. Which means that non of the path's edges are saturated, hence they can't be in a min-cut. This contradicts the existence f C.

3. What is the tightest bound you can give on the running time of the Ford-Fulkerson algorithm when the input graph satisfies the following: all vertices have degree at most D_{max}, and all edges have capacity at most C_{max} (where C_{max} is an integer)? Your bound should depend on n, C_{max} and D_{max} but nothing else.

4. Kleinberg-Tardos, Chapter 7, exercises 1–9 (page 415+).

5. Kleinberg-Tardos, Chapter 7, exercise 15 (page 421).

6. Kleinberg-Tardos, Chapter 7, exercise 16 (page 422).

7. (*) Kleinberg-Tardos, Chapter 7, exercise 19 (page 425).

8. (*Note that if you were to get a problem like this, you don't have to do it from scratch, you could reference relevant ideas from the homework!*) Suppose you have already computed a maximum-value s-t-flow f for a graph $G = (V, E, c)$. Describe how, given $G, s, t, f, (u, v)$, you would update the flow in $O(m + n)$ time if the following changes were made to G:

    (a) First, how can you verify in time $O(n + m)$ that the flow is indeed maximum?

    Try to run one more iteration of Ford-Fulkerson, i.e. run BFS/DFS from s on the residual. If you can reach t, then the flow is not max, otherwise it is.

    (b) The edge (u, v) with capacity 1 is added to the graph.

    Run one more iteration of FF to find whether there is a new augmenting path.

    (c) The edge (u, v) with capacity 1 is deleted from the graph.

    this problem was on hw1.

    (d) The capacity of an existing edge (u, v) is *increased* by 1.

    Run one more iteration of FF to find whether there is a new augmenting path.

    (e) The capacity of an existing edge (u, v) is *decreased* by 1.

    this problem was on hw1.

9. During the pandemic, there is a global push to vaccinate as many people as quickly as possible. There are K different brands of vaccine available that need to be distributed to L countries. For each vaccine v_i there are q_i doses available. Further, we know the population p_j of each country c_j. However, not every vaccine is approved for use in every country. Currently, for each vaccine v_i there is a list ℓ_i of countries where it is approved.

    Find an algorithm to decide whether there is a way to immunize every person in every country with a vaccine approved by their country's health department.

    Your algorithm should use a reduction to network flow. Specify your algorithm by answering the questions below. Your answers should be clear enough that any other student could turn them in to working code. (Pictures are helpful.)

    (a) Design a directed graph $G(V, E)$ that is an instance of the max-flow problem. That is, G is directed, it has designated source s and sink t nodes and a capacity $c(e)$ on each directed edge. Describe each part of G:

    - Specify the vertices in G. *Hint: there should be nodes for both vaccines and countries.*

- Describe the edges in G. Make clear which direction they are.
- Describe the capacity on each edge.
- Make sure there are vertices named s (source) and t (sink)

We have s, t, one vertex for each vaccine brand and one for each country.
There are edges (s, v_i) for each vaccine with capacity $c(s, v_i = q_i$. There are edges (c_j, t) from each country to the sink with capacity $c(c_j, t) = p_j$. Finally, there is an edge from each vaccine v_i to country c_j if v_i is on the list of approved vaccines for c_j. The capacity on this edge is ∞.

(b) How many vertices does your graph have in terms of K and L (asymptotically)?

$\Theta(K + L)$ vertices and $O(k) + O(L) + O(DL) = O(K + DL)$ where D is the length of the longest list of approved vaccines. (we can upper bound D by K in which case we simply get $O(KL)$)

(c) What is the maximum number of edges your graph can have, in terms of K and L (asymptotically)?

$O(KL)$ (see explanation above)

(d) Suppose we found the maximum flow f in G. Describe how to find for each i, j the number of units of the vaccine v_i to be sent to country c_j given the value $f(e)$ along each edge.

The amount of flow $f(v_i, c_j)$ tells us how many doses of v_i were sent to c_j.

(e) Suppose that it is indeed possible to get each country its required amount of vaccine. In that case, what is the value of the flow that is returned by the max-flow subroutine?

Since each country receives vaccines for their full population we get
$value(f) = \sum_{c_j \ in \ countries} p_j$

2 NP, NP-C and polynomial time reductions

1. (* there would be one such question the exam.) Prove that the following problems are in NP. You will do this by (0.) Formulate the problem as a decision problem, i.e. it has a yes/no answer. (1.) Define the polynomial-length certificate and the polynomial-time verifier function that takes the problem and certificate as input. (2.) Show how the certificate correctly identifies the problem instances with "yes" answers.

(a) Perfect Matching

decision: Given graph G is there a perfect matching? certficate: the edges in the matching verifier: iterate over the edges to check whether they are indeed disjoint and verify whether every node is contained in one of the edges.

(b) Factoring

decision: Given an integer n,is there a non-trivial factoring of n into integers? The certificate are the integers that are factors of n. We can verify whether they are factors by performing the multiplication.

(c) Longest Path

decision: Given graph G and integer k, is there a path of length at least k? certificate: the edges in the path. verifier: iterate over the path to check whether the edges indeed exist and count the length.

(d) Hamiltonian-Path, Hamiltonian-Cycle, Traveling Sales Person (TSP)

see lecture slides

(e) Independent Set, Vertex Cover, Set Cover
see lecture slides

(f) Graph k-coloring (Given a graph G, can we assign at most k colors to its vertices, so that neighbors have different colors?)
decision: the problem is already in this form. Certificate: the color of each vertex. verifier: iterate over the edges to check that their two vertices are different.

(g) 3-SAT

decision: is there an assignment with k true literals that evaluates to true? certificate: the True/False Boolean values of the literals. Verifier: evaluate the formula.

2. (* we would ask 4-5 such questions on the exam.) For each of the following statements, indicate whether it is true, false, or "unknown", where "unknown" indicates that scientists have not conclusively determined whether the statement is true or false. For example, the correct answer for the statement "$P = NP$" is "unknown". But the correct answer for "the best algorithm for the maximum flow problem in n-vertex graphs takes time at least 2^n" is "false".

Give a short justification for your answer (i.e. explain why the statement is true, or false, or not known to be either true or false).

(a) If X is in NP, then there does not exist a polynomial-time algorithm for X.

False. Polynomial problems are by definition in NP.

(b) If X is NP-complete, then there is no polynomial-time algorithm for X.
We don't know whether there exsit poly algorithms for any of the NP-C problems.

(c) If X is NP-complete, Y is in NP, and X reduces to Y (thus $X \leq_p Y$), then Y is NP-complete.

True, we have seen this in class. In summary; since X is NP-C we know by definition that any problem in NP reduces to X. By tarnsitivity this yields a reduction from any problem in NP to Y, hence satisying the deifintion of NP-C.

(d) Shortest Paths is NP-complete.

False/Don't know. There are many polynomial algorithms for it. It's only true if P=NP.

(e) Shortest Paths is in NP.

True. a valid certificate would be the path itself.

(f) Shortest Paths is in P.

True. We know multiple polynomial algorithms for it.

(g) Longest Paths is NP-complete.

True. We have seen the proof in class.

(h) Longest Paths is in NP.

True, by definition of NP-C. We could also show the certificate and verifier.

(i) Longest Paths is in P.

We don't know, only if P=NP

(j) Every problem in P is in NP.

True. The polynomial solution to the problems serve as certificate.

(k) Every NP-complete problem is in NP.

true by definition of NP-C.

(l) Some NP-complete problems have polynomial time algorithms, but some others do not.

We don't know. But either all of them have pol algorithms or none.

(m) Suppose that X is polynomial and $X \leq_P Y$, then Y is a polynomial problem.

False/we don't know. Reducing a polynomial problem doesn't say anything about Y as a trivial reduction is to 1. solve X using their polynomial algorithm 2. call the solver to Y 0 times. a.k.a. ignore problem Y.

(n) Suppose that X is exponential, and $X \leq_P Y$, then Y is at least exponential.

True. If Y was less than exponential, then we would have a non-exponential algorithm for X through the reduction.

(o) Suppose that X is in NP, and $X \leq_P Y$, then Y is NP-C.

False/we don't know. The definition of NP-C is that every problem in NP can be reduced to it. The fact that one specific problem can is not sufficient.

(p) Suppose that Y is polynomial, and $X \leq_P Y$, then X is polynomial.

True. We have a polynomial algorithm for X through the reduction to Y.

3. In a directed graph G the Directed Hamiltonian-Path (DHP) problem decides whether there is a simple path in G that contains every vertex. Show that DHP is NP-C. (Don't forget to show that DHP is in NP and a reduction from some other known NP-C problem *to* DHP.)

   DHP is in NP as it has a polynomial size certificate and verifier. The path itself is a certificate and we can iterate over it to verify that it's simple and contains every node.

   To show that it is NP-C we can use for example the Directed Hamiltonian Cycle (DHC) problem. A directed graph has a DHC, if there is a directed cycle that contains every node once. We can reduce this to DHP in the following way: Iterate over the edges of G. For every edge (u, v), temporariliy remove it from G and decide whether there is a DHP from v to u in the remaining G. If yes, then the path+(u, v) form a DHC. If none of the edges yield a DHP, then there is no DHC.

4. The Clique problem in an undirected graph G finds the largest clique subgraph in G. (A clique is a complete graph, i.e. a graph such that every vertex is connected to every other vrtex.) Show that this problem is NP-C. (*Hint: you might want to look at the complement of this graph*)

   There is a one-to-one reduction to Independent Set, this was covered in lab.

3 Approximation Algorithms

1. (*) Answer the following questions and give a short explanation.

   (a) Suppose that some algorithm is a 3-approximation algorithm to a minimization problem. This algorithm outputs on some input a solution of value 120. What is the range of value for the optimal solution on this input?

   Let m be the output of the approximation algorithm and $m*$ be the optimal solution. Then the definition of the approximation ratio tells us that $m* \leq m \leq c \cdot m*$. Applying this we get that $m* \leq 120 \leq 3m*$. Which means that $40 \leq m* \leq 120$

   (b) Suppose that there is a c-approximation algorithm *myAlgo* for problem X. Then for any input to *myAlgo* the output is c-times worse than the optimal solution for this particular problem instance.

   False. The approximation ratio states that in worse case the output is c-times worse than the optimal. It is possible that we are lucky with the input and the output is in fact optimal.

   (c) There are no approximation algorithms for polynomial problems.

   False. The polynomial optimal algorithm itself is a 1-approximation. Also, sometimes there exist a polynomial exact algorithm, but it's inefficient and we might opt to use an inexact algorithm - that has some approximation guarantees - instead.

(d) Suppose that there is a polynomial time reduction from problem X to a problem Y. Further, we have a c-approximation algorithm for Y. Then this algorithm applied to (the reduced form of) problem X yield a c-approximation for X.

False. It's true that we can deduce an approximation algorithm for X through the reduction. But, depending on how the reduction works, it's possible that the approximation ratio is worse.

(e) If the c-approximation for Y in the above problem is tight, i.e. there is no algorithm for Y with a better approximation ratio, then this ratio is also tight for X.

False, because of the same argument.

2. Consider the following algorithm for Bin Packing; items are considered in the fixed input order. The next item will be assigned to the most recent bin, if it doesn't fit then a new bin is started. Show that this is a 2-approximation algorithm. *Hint: look at the total sum of elements and the number of bins used.*

   First, observe that there can't be two consecutive bins i and $i+1$ that are $\leq$ half full. If there were then the contents of $i+1$ would have fit in i in contradiction to the design of the algorithm. Let B be the number of bins returned by the greedy algorithm. By the previous argument we know that at least $\frac{B}{2}$ bins are needed (as we need ≥ 1 bins per consecutive pair). We can conclude that the optimal number of bins $B*$ can be bounded by $\frac{B}{2} \leq B* \leq B$ which means that we have a 2-approximation.

3. Describe a 2-approximation algorithm for the following Number Multi-Partitioning problem: We are given as input n positive numbers $x_1, x_2, \ldots, x_n$ and an integer m. Divide the numbers into m sets, such that the total value in the set with the largest sum is minimized. (*Hint: this should remind you very much of one of the problems that we studied.*)

   This problem is identical to scheduling on m machines, where the numbers x_i correspond to job lengths.

   The algorithm goes as follows; assign the numbers to the m sets one at a time. In each iteration assign x_j to the set with the current least sum.

   Let $X = \sum_{i=1}^{n} x_i$. Suppose that set s ends up with the maximum total sum, the value of which is X_s. Let x_j be the value last assigned to s. Finally, let X^* be the optimal maximum sum.

   Recall that x_j was assigned to s because this set has the current lowest sum. Which means that $X_s - x_j \leq X_k$ for any other set k. Since we have m sets we know that at least one of the sets ends up with a sum of at least $\frac{1}{m}X$. Thus we have

   $X_s - x_j \leq X_k \leq \frac{1}{m}X \leq X^*$

Finally we have $X* \leq X_s = X_s - x_j + x_j \leq X^* + X^* = 2X^*$

4. (*) Consider the Restricted Length Load Balancing problem; we want to run n jobs, each of them has a length t_j. We have unlimited machines available, however all the jobs have to be finished within 24 hours. Find an assignment of jobs to machines so that all jobs are finished on time and as few machines are used as possible. Find a 2-approximation algorithm.

   This is the same as the bin packing problem with bins having "capacity" of 24. Hence we can apply the algorithm studied in class.